COPPE/UFRJ

O PROBLEMA DE CLUSTERIZAÇÃO AUTOMÁTICA

Marcelo Dib Cruz

Tese de Doutorado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Doutor em Engenharia de Sistemas e Computação.

Orientadores: Adilson Elias Xavier
Luiz Satoru Ochi

Rio de Janeiro
Julho de 2010

O PROBLEMA DE CLUSTERIZAÇÃO AUTOMÁTICA

Marcelo Dib Cruz

TESE SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

Prof. Adilson Elias Xavier, D.Sc.

Prof. Luiz Satoru Ochi, D.Sc.

Prof. Marcia Helena Costa Fampa, D.Sc.

Prof. Nelson Maculan Filho, D. Habil.

Prof. Fabio Protti, D.Sc.

Prof. Victor Manuel Parada Daza, D.Sc.

RIO DE JANEIRO, RJ – BRASIL

JULHO DE 2010

Cruz, Marcelo Dib

O problema de Clusterização Automática/Marcelo Dib Cruz. – Rio de Janeiro: UFRJ/COPPE, 2010.

XIII, 120 p.: il.; 29, 7cm.

Orientadores: Adilson Elias Xavier

Luiz Satoru Ochi

Tese (doutorado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2010.

Referências Bibliográficas: p. 115 – 120.

1. Clusterização. 2. Algoritmos Evolutivos,GRASP,ILS. 3. Modelos Hibridos. I. Xavier, Adilson Elias *et al.*. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

A minha querida mãe

*(*in memoriam*).*

Agradecimentos

Primeiro, gostaria de agradecer aos meus orientadores Adilson Elias Xavier e Luiz Satoru Ochi pela amizade, atenção, ajuda, dedicação e carinho nesta longa caminhada; aos Professores Marcia Helena Fampa, Nelson Maculam Filho, Fabio Protti, Manuel Parada Daza por terem aceito participar desta avaliação;

A minha querida esposa Elaine e minha querida filha Giovanna pelo amor, carinho e por compreenderem minha ausência em momentos importantes de suas vidas;

A minha querida família , minha mãe (in memoriam), meu pai, minhas irmãs, meus cunhados e sobrinhos pelo amor, carinho e apoio de sempre em todos os momentos da minha vida.

A UFRRJ e principalmente ao DEMAT, nas figuras de seus professores e funcionários, que permitiram que este trabalho fosse realizado.

E finalmente, aos amigos do Labotim, aos professores e secretárias do PESC, pela ajuda, carinho e agradável convívio nos últimos anos.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

O PROBLEMA DE CLUSTERIZAÇÃO AUTOMÁTICA

Marcelo Dib Cruz

Julho/2010

Orientadores: Adilson Elias Xavier
Luiz Satoru Ochi

Programa: Engenharia de Sistemas e Computação

Clusterização é o processo em que elementos de um conjunto são alocados para grupos ou clusters de elementos similares. 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 são apresentados várias heurísticas, utilizando as metaheurísticas Algoritmos Evolutivos, GRASP e ILS, para a solução do PCA. São apresentados também, métodos híbridos novos, que utilizam modelos exatos para tentar melhorar as soluções obtidas pelas heurísticas. Resultados computacionais foram realizados para um conjunto de instâncias, incluindo uma comparação com um algoritmo recente da literatura, e mostram a eficiência e a robustez dos algoritmos propostos.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

THE AUTOMATIC CLUSTERING PROBLEM

Marcelo Dib Cruz

July/2010

Advisors: Adilson Elias Xavier
Luiz Satoru Ochi

Department: Systems Engineering and Computer Science

Clustering is the process by which elements of a set are assigned for groups or clusters of similar elements. In clustering algorithms, is usually assumed that the number of clusters is known or provided. Unfortunately, the optimal number of clusters is unknown for most applications. This problem is as Automatic Clustering Problem (ACP). In this work, we present some heuristics, using the metaheuristics Evolutive Algorithm, GRASP and ILS, to solve the PCA problem. We also present, a new hybrid methods, that uses exact models trying to provide the heuristics solutions. Computational results on a set of instances, including a comparison with a recent algorithm from the literature, illustrate the effectiveness and the robustness of the proposed method.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Clusterização (ou agrupamento) é o termo genérico para um processo que une objetos similares 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, quando o número ideal de cluster não é previamente conhecido, é denominado *Problema de Clusterização Automática (PCA)*. Ambos os problemas PC e PCA são classificados na literatura como problemas NP-Completo [52]. Para o problema PCA, o fato do valor do número de clusters k não ser conhecido a priori, torna-o mais complexo, pois aumenta muito o número de soluções possíveis.

Existem várias aplicações relacionadas à clusterização: Particionamento de Grafos, Problema de Manufatura Flexível, Formação de Anéis em Sistemas de Telecomunicações, Reconhecimento de Padrões. Essa ultima aplicação se apresenta em muitos aspectos como em Processamento de Imagens, Biologia Computacional, Pesquisa de Mercado, Classificação e agrupamento de Documentos, Mineração de Dados, entre outros.

Existem muitos algoritmos para resolver o problema na literatura, principalmente para o PC. O algoritmo mais conhecido é o *k-means* [31], que particiona os objetos e utiliza o conceito de centróide para representar os clusters.

Porém não existem muitos trabalhos relacionados ao PCA. Tratando-se de Metaheurísticas, só existem dois trabalhos, um de 2001 [17] e outro de 2004 [47]. Os dois utilizam Algoritmos Evolutivos.

Neste trabalho foram desenvolvidas novas heurísticas, utilizando conceitos das metaheurísticas *Algoritmos Evolutivos, ILS (Iterated Local Search) e GRASP (Greedy Randomized Adaptive Search Procedure).* Os algoritmos foram avaliados para verificar o desempenho. O objetivo neste contexto, foi verificar a qualidade das soluções resultantes, utilizando heurísticas com enfoques diferentes.

Para isso, foi criado um conjunto de procedimentos que são comuns a todas as heuristicas. Estes procedimentos incluem um pré-processamento, cujo objetivo é tentar reduzir as dimensões dos dados de entrada do problema e ao mesmo tempo, gerar soluções iniciais de boa qualidade. Os outros procedimentos comuns são as buscas locais. Além disso, foi definida uma estrutura de dados comum para os algoritmos aqui tratados.

Tinha-se como meta desenvolver um modelo exato para o PCA, utilizando a função Índice Silhueta, para poder verificar a qualidade das soluções obtidas. Isto não foi possível, pois a função Índice Silhueta não é facilmente linearizável. A partir daí, foi procurado na literatura, algum modelo exato para o PCA, o que também não foi conseguido. A terceira tentativa foi adaptar algum modelo exato do PC para o PCA. Também não foi possível, pois as funções utilizadas, como a função Diâmetro [38] e a Estrela [48], são funções decrescentes em relação ao número de clusters.

Então foram desenvolvidos alguns métodos híbridos de heurísticas e modelos exatos, conformado pela percepção de que os modelos exatos para o PC poderiam ser aproveitados para melhorar as soluções do PCA.

No final, os algoritmos desenvolvidos foram comparados com um dos algoritmos mais recentes da literatura, denominado CLUES.[51]

Este trabalho está organizado da seguinte forma: No capítulo 2 é descrito o problema; no capítulo 3 são descritas as heuristicas utilizando Algoritmo Evolutivo, ILS e GRASP acompanhado de algumas avaliaçẽs; No capítulo 4 são mostrados os métodos híbridos e realizadas novos testes computacionais; No Capítulo 5 são feitas as comparações das heurísticas propostas com o algoritmo CLUES, e finalmente, no capítulo 6 são feitas as conclusões e descritas algumas propostas para trabalhos futuros.

Capítulo 2

O Problema de Clusterização Automática

Clusterização é o termo genérico para um processo que une objetos similares 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, denominamos *Problema de K-Clusterização* ou simplesmente *Problema de Clusterização* (PC). Caso contrário é denominado *Problema de Clusterização Automática* (PCA).

O objetivo deste capítulo é introduzir o tema clusterização enfatizando o Problema de Clusterização Automática. Para isso, o tema é descrito e são mostradas algumas aplicações envolvendo clusterização. Um método para classificação dos algoritmos é apresentado e são mostradas as instâncias utilizadas para as avaliações dos algoritmos aqui propostos.

2.1 Descrição do Problema

Clusterização é a divisão de dados em grupos de objetos similares. Cada grupo é denominado um cluster. Cada cluster consiste em um grupo de objetos que são similares entre si e dissimilares com objetos de outros grupos.

Às vezes poderemos ter objetos da entrada que não são similares a nenhum dos clusters encontrados. Tais objetos são denominados *outliers* e ocorrem por erro na coleta de dados ou erro de digitação ou fraudes, etc. Os *outliers* são uma dificuldade

a mais na procura de soluções de boa qualidade.

Segundo Pierre Hansen e Brigitte Jaumard [21], para definir o problema de clusterização, é preciso seguir algumas etapas listadas abaixo:

- **Amostragem** – selecione um conjunto de m objetos $X = \{x_1, x_2, x_3....x_m\}$ onde estão os clusters;
- **Dados** – observe ou tire a medida das p características de cada objeto $x_i \in X$. Isto conduz a uma matriz de dados M_{mxp};
- **Similaridades** – calcule a partir de M_{mxp} , a matriz de similaridades $D_{mxm} = (d_{kl})$ entre os objetos de X. Estas similaridades devem satisfazer as propriedades $d_{kl} > 0$, $d_{kk} = 0$, $d_{kl} = d_{lk}$ onde *k,l = 1,2, ...m.* As similaridades não precisam ser necessariamente distâncias.
- **Restrições** – especifique o tipo de problema desejado (Subconjunto , Partição, Cobertura, etc. os tipos de problemas serão definidos posteriormente);
- **Critério** – escolha o critério (ou possivelmente mais de um critério) para expressar a homogeneidade e/ou separação dos clusters no problema a ser tratado;
- **Algoritmo** – defina um algoritmo para o problema. Codifique o algoritmo;
- **Computação** – aplique o algoritmo escolhido na matriz $D_{mxm} = (d_{kl})$ obtendo assim os clusters;
- **Interpretação** – aplique testes formais ou informais para selecionar os melhores clusters. Interprete os resultados;

Não existe somente uma maneira de definir o significado de clusters. É necessário especificar o tipo de problema desejado. Os algoritmos de clusterização são feitos para encontrar vários tipos de clusters numa base de dados X, como:

1. *Subconjunto S de X* ;

2. *Partição* $P_k = \{C_1, C_2, ...C_k\}$ de X em k clusters tal que:

$$(a) \quad C_i \neq \emptyset \ , \ para \quad i = 1, .., k$$
$$(b) \quad C_i \cap C_j = \emptyset \ , para \ i, j = 1, .., k \ e \ i \neq j$$
$$(c) \quad \bigcup_{i=1}^{k} C_i = X$$

3. *Empacotamento (Packing)* $PA_q = \{C_1, C_2, ...C_k\}$ de X em k clusters como em (2) , porém sem a condição (c);

4. *Cobertura (Covering)* $CO_k = \{C_1, C_2, ...C_k\}$ de X em k clusters como em (2), porém sem a condição (b) ;

5. *Hierarquia* $H = \{P_1, P_2, ...P_t\}$ com $t \leq m$ partições de X. O conjunto de partições $P_1, P_2, ...P_t$ são definidos tal que $C_i \in P_k, C_j \in P_l$ e $k > l$ implica que $C_i \subset C_j$ ou $C_i \cap C_j = \emptyset$ para i , j = 1,....t e $i \neq j$.

Para avaliar se um cluster é bom ou ruim, é necessário definir critérios que quantifiquem a homogeneidade de um cluster e a separação entre os clusters. Exemplos de critérios para a definição de separação são:

1. *Divisão (Split) s(C_j) de C_j*, ou a similaridade mínima entre um objeto de C_j e um outro objeto fora de C_j;

$$s(C_j) = \frac{Min}{k : x_k \in C_j; l : x_l \notin C_j} \quad d_{kl} \tag{2.1}$$

2. *Corte (Cut) c(C_j) de C_j*, a soma das similaridades entre objetos de C_j e objetos fora de C_j;

$$c(C_j) = \sum_{k:x_k \in C_j} \sum_{l:x_l \notin C_j} d_{kl} \tag{2.2}$$

Podemos ainda considerar o corte normalizado, para eliminar o efeito da cardinalidade dos clusters, dividindo o valor encontrado por $|C_j|$ / (m - $|C_j|$).

A homogeneidade de um cluster pode ser medida de várias maneiras diferentes. Para a definição de homogeneidade temos alguns critérios como:

1. *Diâmetro (Diameter) d(C_j) de C_j* , ou a similaridade máxima entre objetos de C_j.

$$d(C_j) = \frac{Max}{k, l : x_k, x_l \in C_j} \ d_{kl} \tag{2.3}$$

2. *Raio (Radius) r(C_j) de C_j*, ou o menor valor entre todos os objetos x_k de C_j da similaridade máxima do objeto x_k a um outro objeto de C_j;

$$r(C_j) = \frac{Min}{k : x_k \in C_j} \frac{Max}{l : x_l \in C_j} \; d_{kl} \tag{2.4}$$

3. *Estrela (Star) st(C_j) de C_j*, ou o menor valor entre todos os objetos x_k de C_j da soma de similaridades do objeto x_k a os outros objetos de C_j;

$$st(C_j) = \frac{Min}{k : x_k \in C_j} \sum_{l:\, x_l \in C_j} d_{kl} \tag{2.5}$$

4. *Clique (Clique) cl(C_j) de C_j*, ou a soma das similaridades entre objetos de C_j;

$$cl(C_j) = \sum_{k,l \,:\, x_k, x_l \in C_j} d_{kl} \tag{2.6}$$

É possível ter ainda a Estrela normalizada e o Clique normalizado, dividindo o valor encontrado por $|C_j|$-1 e por $|C_j|$ / ($|C_j|$ - 1) respectivamente.

Se os objetos x_j são pontos do espaço Euclidiano R^p ($x_i \in R^p$) então homogeneidade de C_j pode ser medida por referência ao centro ou centróide de C_j , que não é um ponto de C_j. Assim podemos definir homogeneidade como:

1. *Soma dos Quadrados (Sum-of-Squares)*

$$ss(C_j) = \sum_{k \,:\, x_k \in C_j} (\, ||x_k - \bar{x}||_2)^2 \tag{2.7}$$

onde $||\quad||_2$ define a distância euclidiana dos pontos e $\bar{x} = \frac{1}{|C_j|} \sum_{k : x_k \in C_j} x_k$ definido como o centro ou centróide de C_j.

2. *Variância (Variance) v(C_j) de C_j* definido como ss(C_j) dividido por $|C_j|$.

3. *Raio Contínuo (Continuous Radius)*

$$cr(C_j) = \frac{Min}{x \in R^p} \frac{Max}{k : x_k \in C_j} || \, x_k - x||_2 \tag{2.8}$$

4. *Estrela Contínua (Continuous Star)*

$$cst(C_j) = \frac{Min}{x \in R^p} \sum_{k:\, x_k \in C_j} || \, x_k - x||_2 \tag{2.9}$$

O tema clusterização é bastante vasto[6]. Existem muitos trabalhos na literatura tratando o assunto. A maioria dos trabalhos é para encontrar partições. Normalmente o número de clusters é um dado de entrada, ou seja, a maioria dos trabalhos é para o PC. O método mais conhecido é denominado *k-means* [31] que utiliza, na maioria dos casos, a função Soma dos Quadrados (equação 2.7) para a avaliação. Existem inúmeros trabalhos sendo publicados considerando esta função, com metodologias diferentes, como Suavização Hiperbólica [54], Algoritmos Genéticos [28], *Global Optimization* [5], Busca Tabu [29], entre outros.

Na literatura, não existem, do nosso conhecimento, modelos exatos sem restrições para o PCA. Os modelos exatos existentes possuem alguma restrição, como o número mínimo de pontos em cada cluster ou a capacidade máxima do cluster [25].

Para o PC, existem dois modelos exatos. O primeiro modelo, denominado Modelo Exato Diâmetro [38], utiliza a função Diâmetro, definida na equação (2.3), e minimiza o maior diâmetro de todos os clusters encontrados. O segundo modelo, denominado Modelo Exato K-Medianas [48], utiliza a função Estrela, definida pela equação (2.5), e minimiza a soma das distâncias dos pontos do cluster a um ponto mais ao centro deste.

No enfoque do Problema de Clusterização Automática, adotado no trabalho, é mais delicada a escolha de uma função de avaliação. Infelizmente, as funções e os algoritmos utilizados para o PC não funcionam bem na PCA. Por exemplo, as funções (ou critérios de definição de homogeneidade) Soma dos Quadrados (2.7), Estrela (2.5) e Diâmetro (2.3) são monotonamente decrescentes em relação ao número de clusters. Portanto, não podem ser utilizadas diretamente caso o número de clusters não seja conhecido a priori. Existem funções que podem ser utilizadas para o PCA [13], pois o valor de k não precisa ser conhecido a priori. Entre elas está a função Índice Silhueta, definida por Kaufman e Rousseeum [27], e que tem se mostrado eficiente em vários trabalhos, como em [44, 51].

Para este trabalho, os objetos estão associados a pontos no R^n, a similaridade é definida como a distância euclidiana entre os pontos e a função de avaliação é a função Índice Silhueta.

O PCA pode ser definido formalmente como: Seja $X = \{x_1, x_2, x_3....x_m\}$ um conjunto de m pontos no R^n. O objetivo deste problema é encontrar uma partição

$C = \{C_1, C_2, ...C_k\}$ onde k não é conhecido a priori. Cada subconjunto $C_i \subset C$ é denominado um cluster de X.

O Problema de Clusterização Automática pode ser reduzido ao problema de otimização da seguinte forma:

$$\underset{C}{Maximizar} \quad F(C) \quad = \quad \frac{1}{m} \sum_{i=1}^{m} s(x_i) \tag{2.10}$$

$$S.A. \qquad C \subset \underline{C} \tag{2.11}$$

onde $C = \{C_1, C_2, ...C_k\}$ é uma partição particular de clusters e $\underline{C}$ é o conjunto de todas as partições possíveis do conjunto X com k = *2,...,m-1* e onde $s(x_i)$ é o valor da silhueta de cada ponto $x_i \in X$ definido a seguir.

Seja x_i um ponto pertencente ao cluster $C_w \subset C$, com $|C_w| = V > 1$. A distância entre os pontos x_i e x_j é definido por d_{ij}. A distância média de x_i em relação a todos os pontos $x_j \in C_w$ é dada por $a(x_i)$ onde

$$a(x_i) = \frac{1}{V-1} \sum d_{i,j} \quad \forall x_j \neq x_i \quad , x_j \in C_w \tag{2.12}$$

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

$$d(x_i, C_t) = \frac{1}{T} \sum d_{i,j} \quad \forall x_j \in C_t \tag{2.13}$$

Seja $b(x_i)$o menor valor dentre todos os $d(x_i, C_t)$. Então,

$$b(x_i) = Min \;\; d(x_i, C_t) \;, \quad C_t \neq C_w \quad , \quad C_t \in C \tag{2.14}$$

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

$$s(x_i) = \frac{b(x_i) \quad - \quad a(x_i)}{\max(a(x_i), b(x_i))} \tag{2.15}$$

A função F na equação (2.10) utiliza simultaneamente dois critérios para a avaliação da clusterização. Como critério para a homogeneidade, F utiliza a soma das médias das distâncias entre os pontos de cada cluster (similar ao critério da equação (2.6)), e como critério de separação a média das distâncias entre cada ponto e os pontos do cluster mais próximo e diferente de onde ele está (similar ao critério da equação (2.2)).

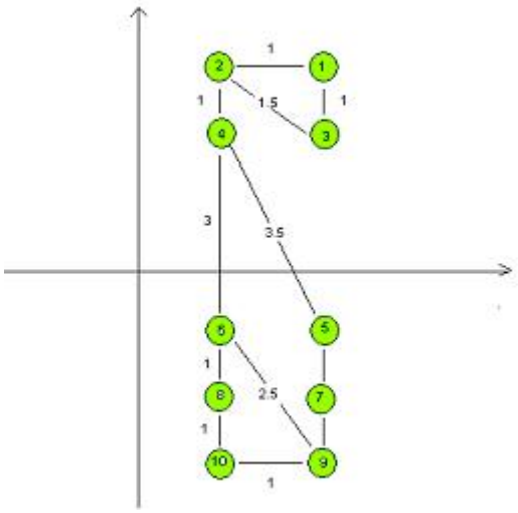

Figura 2.1: O conjunto de pontos

A função F possui valores no intervalo [-1,1] e quanto mais próximo estiver de 1(um) mais coesos e separados são os clusters.

Para exemplificar a função F, alguns exemplos são mostrados. Suponha que X=*{1,2,3,4,5,6,7,8,9,10}* seja um conjunto de pontos no R^2 dispostos como na figura 2.1. A distância entre dois pontos adjacentes que estão na mesma linha horizontal ou vertical é igual a 1. As únicas exceções são as distâncias entre os pontos 4 e 6 e 3 e 5 que tem valor igual a 3. As distâncias entre os pontos que

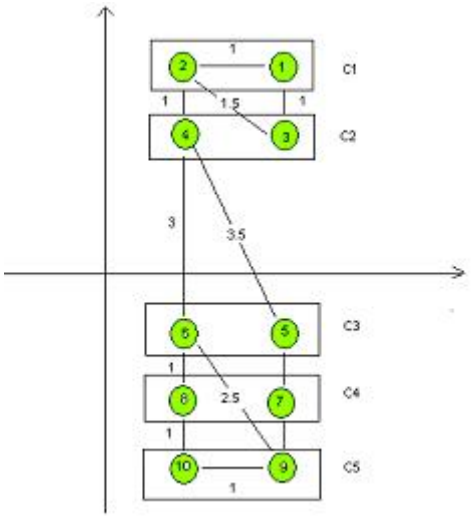

Figura 2.2: A partição $P^1 = \{C_1, C_2, C_3, C_4, C_5\}$ onde $C_1 = \{1, 2\}$, $C_2 = \{3, 4\}$, $C_3 = \{5, 6\}$ e $C_4 = \{7, 8\}$ e $C_5 = \{9, 8\}$

estão em diagonal tem seus valores decimais aproximados para 0.5 com o objetivo de simplificar os cálculos.

x_i	$a(x_i)$	$b(x_i)$	$s(x_i)$
1	1	1.25	0.2
2	1	1.25	0.2
3	1	1.25	0.2
4	1	1.25	0.2
5	1	1.25	0.2
6	1	1.25	0.2
7	1	1.25	0.2
8	1	1.25	0.2
9	1	1.25	0.2
10	1	1.25	0.2

Tabela 2.1: Calculo dos valores relativos a partição P^1

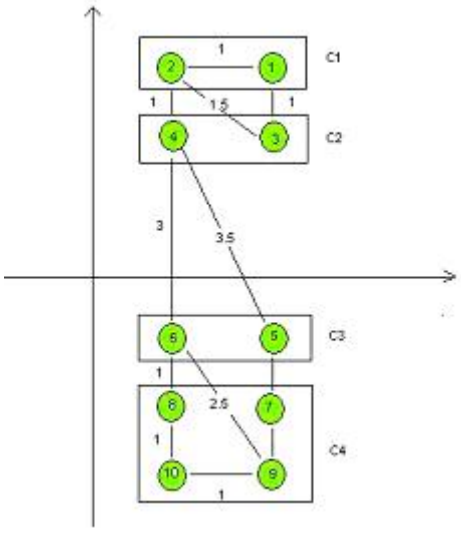

Figura 2.3: A partição $P^1 = \{C_1, C_2, C_3, C_4, C_5\}$ onde $C_1 = \{1, 2\}$, $C_2 = \{3, 4\}$, $C_3 = \{5, 6\}$ e $C_4 = \{7, 8\}$

Para melhorar o entendimento da função F, são exemplificados 4 partições desse conjunto. Para cada uma delas é calculado o valor de F. As partições são denominadas P^1, P^2, P^3 *e* P^4 e possuem 5 , 4 , 3 e 2 clusters respectivamente.

Na figura 2.2 é mostrada a primeira partição P^1 , que possui 5 clusters de dois pontos cada. A tabela 2.1 mostra os valores de $a(x_i)$, $b(x_i)$ *e* $s(x_i)$ de cada ponto

desta partição. As distâncias entre os pontos dentro de cada cluster e entre os pontos de clusters diferentes estão muito próximas. Tal fato ocasiona valores $s(x_i)$ pequenos e conseqüentemente, a função F também possui um valor pequeno: $F = \frac{1}{10} \sum_{i=1}^{10} s(x_i) = 0.2$

x_i	$a(x_i)$	$b(x_i)$	$s(x_i)$
1	1	1.25	0.2
2	1	1.25	0.2
3	1	1.25	0.2
4	1	1.25	0.2
5	1	1.75	0.2
6	1	1.75	0.2
7	1.16	1.75	0.42
8	1.16	1.75	0.42
9	1.16	2.25	0.48
10	1.16	2.25	0.48

Tabela 2.2: Calculo dos valores relativos a partição P^2

A figura 2.3 mostra a partição P^2 , que possui 4 clusters, sendo 3 clusters com 2 pontos e um cluster de 4 pontos. A tabela 2.2 mostra os valores $a(x_i)$, $b(x_i)$ *e* $s(x_i)$ de cada ponto desta partição. Neste caso, as distâncias entre os pontos dentro de cada cluster e entre os pontos de clusters diferentes continuam próximas, ocasionando valores de $s(x_i)$ pequenos. Porém, nesta partição os pontos do cluster C_4 estão mais distantes dos pontos do cluster C_3, e com isso o valor de F aumenta um pouco: $F = \frac{1}{10} \sum_{i=1}^{10} s(x_i) = 0.34$

A figura 2.4 mostra a partição P^3 que possui 3 clusters, sendo 2 clusters com 2 pontos e um cluster com 6 pontos. A tabela 2.3 mostra os valores $a(x_i), b(x_i) es(x_i)$ de cada ponto desta partição. Neste caso, os clusters C_2 e C_3 estão mais distantes, aumentando os valores de $b(x_i)$ *e* $s(x_i)$ e, conseqüentemente, o valor de F: $F = \frac{1}{10} \sum_{i=1}^{10} s(x_i) = 0.46$

A figura 2.5 mostra a partição P^4 que possui 2 clusters, com 4 e 6 pontos. A tabela 2.4 mostra os valores $a(x_i), b(x_i) es(x_i)$ de cada ponto desta partição. Neste caso, os dois clusters estão distantes aumentando os valores de $b(x_i)$ e ,conseqüentemente, os valores de $s(x_i)$ e da função F : $F = \frac{1}{10} \sum_{i=1}^{10} s(x_i) = 0.7$

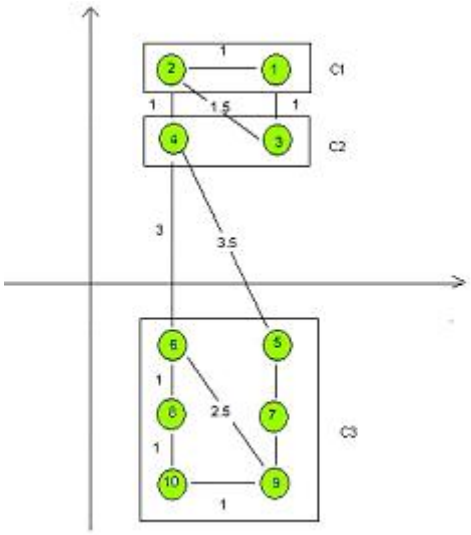

Figura 2.4: A partição $P^3 = \{C_1, C_2, C_3\}$ onde $C_1 = \{1, 2\}$, $C_2 = \{3, 4\}$ e $C_3 = \{5, 6, 7, 8, 9, 10\}$

O que é observado nestes exemplos, é que a função F aumenta à medida em que os pontos próximos se juntam num mesmo cluster e os pontos afastados ficam em clusters separados.

x_i	$a(x_i)$	$b(x_i)$	$s(x_i)$
1	1	1.25	0.2
2	1	1.25	0.2
3	1	1.25	0.2
4	1	1.25	0.2
5	1.6	3.25	0.5
6	1.6	3.25	0.5
7	1.2	4.25	0.71
8	1.2	4.25	0.71
9	1.6	5.25	0.69
10	1.6	5.25	0.69

Tabela 2.3: Calculo dos valores relativos a partição P^3

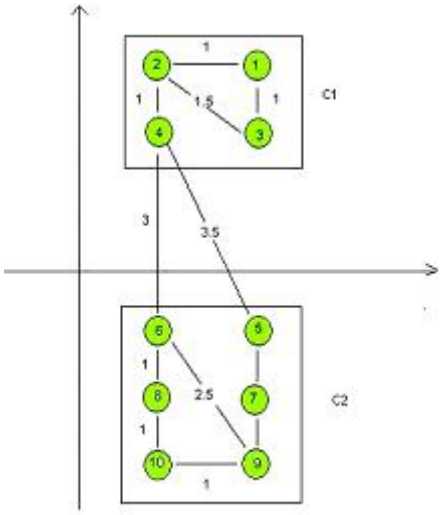

Figura 2.5: A partição $P^4 = \{C_1, C_2\}$ onde $C_1 = \{1, 2, 3, 4\}$ e $C_2 = \{5, 6, 7, 8, 9, 10\}$

2.2 Aplicações

Existem muitas aplicações envolvendo clusterização, nas mais diversas áreas. Cada área possui suas próprias especificações. Algumas das principais aplicações são mostradas a seguir:

x_i	$a(x_i)$	$b(x_i)$	$s(x_i)$
1	1.16	5.25	0.77
2	1.16	5.25	0.77
3	1.16	4.25	0.72
4	1.16	4.25	0.72
5	1.6	3.75	0.57
6	1.6	3.75	0.57
7	1.2	4.75	0.74
8	1.2	4.55	0.74
9	1.6	5.75	0.72
10	1.6	5.75	0.72

Tabela 2.4: Calculo dos valores relativos a partição P^4

2.2.1 Particionamento de Grafos

O problema pode ser definido como: dado um grafo $G = (V, E)$, onde V é o conjunto de vértices e E o conjunto de arestas, dividir os vértices em subconjuntos disjuntos, ou *clusters*, otimizando alguma função que mede a similaridade entre os vértices. Existem vários problemas associados ao particionamento de grafos [8]. No problema de particionamento *balanceado* de grafo a diferença de cardinalidade entre o maior *cluster* e o menor *cluster* deve ser de, no máximo, uma unidade. Quando o número de *clusters* é igual a dois, o problema é referenciado na literatura como problema de bisseção de grafos ou problema de bi-particionamento de grafos. Existe ainda o problema de edição de arestas onde se deseja encontrar conjuntos de vértices, de tal forma que o custo de inserir e deletar arestas para formar cliques seja mínimo [35]. Outro problema similar é agrupar os vértices do grafo em *clusters* de tal forma que seja maximizado o número total das arestas internas a cada *cluster*, ao mesmo tempo em que seja minimizado o número total de arestas entre pares de vértices que estejam em clusters diferentes [16].

2.2.2 Problema de Formação de Células de Manufatura (PCM)

Existem industrias que produzem uma pequena variedade de produtos e um alto volume de produção. Elas geralmente organizam o ambiente de produção em linhas de produção, sendo que cada linha de produção é composta de vários tipos de máquinas, dedicadas exclusivamente à produção de um único produto. Porém, em outras industrias é produzido uma grande variedade de produtos, com um volume médio de produção de cada item. Portanto, elas precisam de um modelo diferente de produção. Uma abordagem diferente é a formação de grupos (clusters) de máquinas com funcionalidades idênticas, formando departamentos especializados em uma função específica. Assim, uma parte de um produto que necessite de operações de manufatura de mais de um tipo de máquina, precisará percorrer todos os grupos que contenham os tipos de máquinas necessários para a sua formação. Nos últimos anos um novo modelo de organização destes sistemas vem sendo usado, denominado manufatura celular.

A manufatura celular [46] consiste em agrupar máquinas de diferentes funcionalidades para a manufatura de produtos com diferentes partes. O sistema de produção fica dividido em vários grupos ou clusters formados por células de máquinas e famílias de partes, denominados células de manufatura. A formação de clusters seguindo este modelo resulta em diversas vantagens para a gestão da produção, tais como:

1. Redução do transporte de material de produção;

2. Redução do tempo de produção dos produtos e conseqüente aumento da capacidade de produção;

3. Simplificação do gerenciamento e controle do sistema de produção, agora divididos em vários subsistemas independentes;

4. Simplificação do escalonamento das atividades, que agora será feito separadamente em cada cluster;

5. Aumento da segurança no trabalho com a minimização de manipulação de material no ambiente de produção;

2.2.3 Reconhecimento de Padrões

Reconhecimento de padrões é a área de pesquisa que tem por objetivo a classificação de objetos (padrões) em um número de categorias ou classes (clusters) [11]. Por exemplo, no reconhecimento de faces, as imagens das faces são os objetos e as classes são seus nomes ou identificações. Há também o problema de categorização de faces, que classificam as pessoas em categorias, discriminando por exemplo, gênero, faixa etária e etnia. Nesse caso, as classes são as categorias que as pessoas pertencem.

Um ponto em comum entre as aplicações de reconhecimento de padrões é que usualmente as características disponíveis nos padrões de entrada, que tipicamente são milhares, não são diretamente utilizadas. Normalmente são utilizadas características extraídas dos padrões de entrada otimizados, usando procedimentos orientados por dados. Dado um padrão, seu reconhecimento ou classificação consiste em uma das seguintes tarefas: *classificação supervisionada*, em que o padrão de entrada é identificado como um membro de uma classe pré-definida pelos padrões de treinamento, que são rotulados com suas classes; e *classificação não supervisionada*, em que o padrão é associado a uma classe que é aprendida com base na similaridade entre os padrões

de treinamento. O projeto de sistemas de reconhecimento de padrões essencialmente envolve três aspectos: aquisição de dados e pré-processamento, representação dos dados e tomada de decisões. Assim, geralmente o desafio encontra-se na escolha de técnicas para efetuar esses três aspectos. Em geral, acredita-se que um problema de reconhecimento de padrões bem definido e restrito, permitirá uma representação compacta e uma estratégia de decisão simples. As técnicas de reconhecimento de padrões podem ser aplicadas em vários domínios, dentre os quais temos:

- Mineração de Dados (*data mining*)
- Processamento de Imagens
- Análise de imagens de documentos para reconhecimento de caracteres (*Optical Character Rocognition - OCR*) ;
- Bio-informática, análise de seqüências de proteínas ou DNA;
- Busca e classificação em base de dados multimídia;
- Reconhecimento biométrico, incluindo faces, íris ou impressões digitais;
- Reconhecimento de fala.

2.3 Classificação dos Algoritmos de Clusterização

As classificações dos algoritmos de clusterização são gerais e não fazem distinção entre o PC e o PCA. Não existe uma única classificação destes algoritmos. Segundo [44] os algoritmos se encontram em dois grandes grupos: Hierárquicos e de Particionamento. Porém esta classificação não distingue todos os algoritmos. Adotaremos a classificação em [17] por ser simples e contemplar os algoritmos existentes na literatura. A classificação dos algoritmos de clusterização de acordo com a figura 2.6 é:

- **Hierárquico** - Os algoritmos Hierárquicos operam sobre o conjunto de entrada X de pontos e procuram construir uma árvore de clusters denominada dendograma. Nesta árvore cada nó é um cluster que pode ter outros clusters filhos, pais ou irmãos de forma que, cada ponto de X é associado a um único nó

da arvore em um dos seus níveis. A literatura apresenta ainda uma divisão dos algoritmos Hierárquicos em dois grupos de acordo como a árvore é construída. Quando a árvore é construída da raiz para as folhas (estratégia top-down) dizemos que o algoritmo é Hierárquico por Divisão. Quando é construído das folhas para a raiz (estratégia bottom-up) o algoritmo é dito Hierárquico por Aglomeração. O Hierárquico por Divisão começa com um único cluster contendo todos os pontos e recursivamente divide os clusters até um critério de parada. No Hierárquico por Aglomeração, cada ponto é um cluster e a cada momento eles são unidos para formar um novo cluster. Exemplos de algoritmos Hierárquicos para o PC são [19], [20], [26],[55]. Não encontramos exemplos para o PCA.

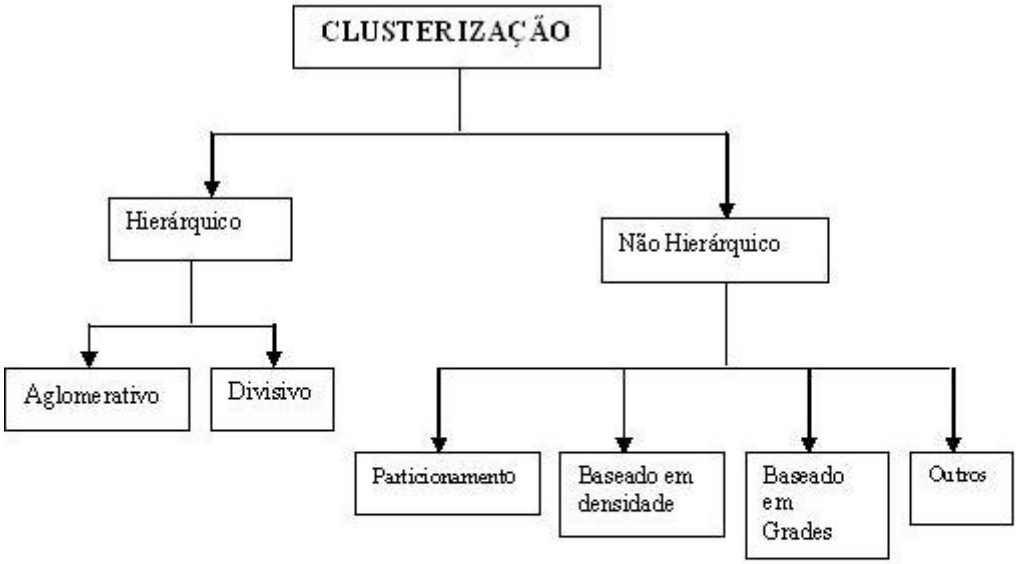

Figura 2.6: Classificação dos métodos de clusterização

- **Particionamento** - Os algoritmos de Particionamento consistem em classificar os m pontos do conjunto de entrada X em k grupos. Cada grupo contém pelo menos um ponto e cada ponto pertence à pelo menos um grupo. É uma

forma de otimização iterativa, que a cada iteração realoca os pontos entre os clusters. Os algoritmos de Particionamento são diferentes dos algoritmos Hierárquicos onde os clusters não são revisitados depois de construídos. Eles melhoram gradualmente os clusters. Como exemplos destes algoritmos para o PC, temos o *k-means* [31], que é o algoritmo mais conhecido da literatura e que a cada iteração constrói o centróide do cluster e avalia a clusterização através das distâncias entre os centróides. São encontradas também as variações do k-means, que utilizam metodologias diferentes como suavização hiperbólica [54], *Global Optimization* [5], arvores [39]. Tem ainda algoritmos que utilizam os k-medoids [12, 36] que são os pontos mais representativos de cada cluster. Existe ainda um grupo de algoritmos que utilizam as metaheurísticas como Busca Tabu [3, 45], Algoritmos Genéticos [9, 28, 30, 41], *Simulated Anealing* [37, 44], Colônia de Formigas [22, 43]. Para o PCA temos o [51, 53].

- **Baseado em Densidade**: os algoritmos Baseados em Densidade definem funções que utilizam conceitos de densidade e conectividade. A idéia é agrupar, no mesmo cluster, pontos que formam uma região densa. Essas regiões podem ter uma forma arbitrária e os pontos nessas regiões podem estar arbitrariamente distribuídos. Como exemplos destes algoritmos para o PC temos [4, 24, 33]. Para o PCA temos o [7, 10].

- **Baseado em Grades**: Os algoritmos Baseados em Grades dividem o espaço que contém os pontos em um determinado número de subespaços ou células. Células contendo um número relativamente grande de pontos são candidatos a clusters. Os resultados dos algoritmos dependem do tamanho das células, que normalmente são parâmetros de entrada. Exemplos de algoritmos Baseados em Grades para o PC são [42, 49, 50, 56]. Para o PCA temos [1, 34].

- **Outros**: nesta classe podemos citar os algoritmos Baseados em Modelos. Um modelo de dados é utilizado para cada cluster e o objetivo é encontrar os pontos que se adaptem a cada modelo de dados. Como exemplo para o PC temos o [14]. Não existem exemplos para o PCA.

  Existem ainda alguns algoritmos que tem características de mais de um grupo, como em [17, 44, 47]. Esses algoritmos utilizam duas etapas: a primeira utiliza

um procedimento Baseado em Densidade e a segunda um procedimento de Particionamento. Os algoritmos [44, 47] são para o PCA e o [17] para o PC.

2.4 O Conjunto de Instâncias

Neste trabalho são utilizadas algumas instâncias. Entre elas, seis são conhecidas na literatura como Ruspini data set [40] (Ruspini) , IrisDataSet [15] (Iris), Maronna data set [32] (Maronna), 200DATA [15], Vowel data set [23] (Vowel) e Broken Ring [51]. O número de instâncias da literatura é pequeno, e por isso, julgou-se desejável gerar outras, com cardinalidades diferentes. Para isso, foram construídas instâncias com 100 a 2000 pontos no espaço $\mathbf{R}^2$, com os números de clusters variando de 2 a 27. Assim é possível avaliar o comportamento dos algoritmos em problemas bem diferentes.

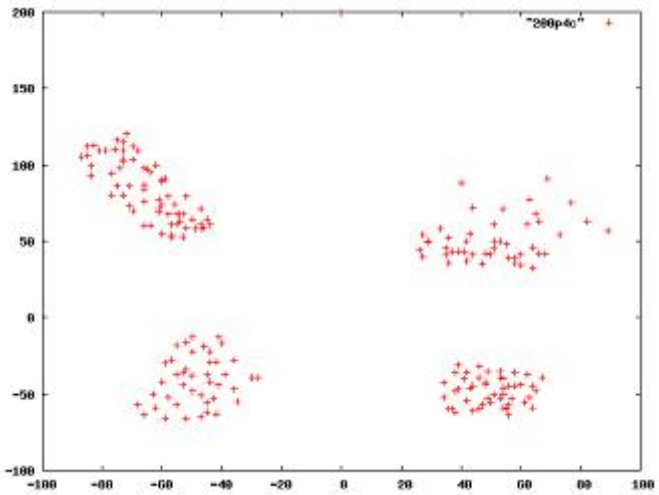

Figura 2.7: A instância comportada 200p4c

As instâncias foram construídas através de uma ferramenta gráfica denominada Dots, desenvolvida por Soares e Ochi em [44]. O aplicativo Dots constitui-se de uma interface gráfica sobre um sistema de eixos cartesianos. Nesta interface é possível construir um conjunto de pontos, onde cada cluster tem o formato desejável, utilizando somente o mouse. A cada clique do mouse, as coordenadas dos pontos são armazenadas. Desta forma, podemos construir instâncias com características variadas de densidade e formato.

Os nomes das instâncias foram definidos utilizando o número de pontos e o número de clusters. Por exemplo, a instância 200p4c possui 200 pontos e 4 clusters.

Foram definidos dois tipos de instâncias: as comportadas, onde os clusters são bem definidos e separados, como mostra a figura 2.7; e as não comportadas, que possuem muitos pontos entre os clusters, como mostra a figura 2.8. As instâncias não comportadas são caracterizadas por final 1 em seus nomes, como em 300p4c1. As instâncias comportadas terminam com c em seus nomes, como em 300p4c.

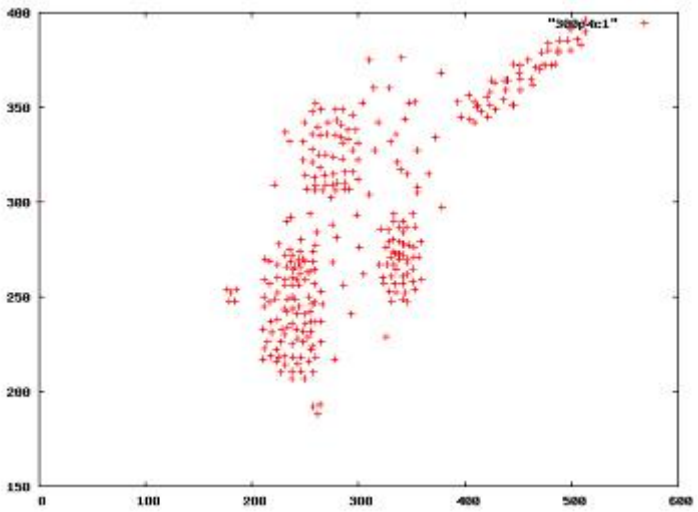

Figura 2.8: A instância não comportada 300p4c1

As instâncias são mostrados na tabela 2.5. Essa tabela possui 5 colunas, que descrevem as características de cada instância. A coluna *Nome* possui os nomes das instâncias, a coluna *Best* possui o melhor resultado encontrado na literatura, a coluna n^o *pontos* possui o número de pontos, a coluna *Dimensão* possui o número de coordenadas de cada ponto e a coluna n^o *clusters* possui o número de clusters. O número de clusters de cada instância não comportada é presumido, pois em vários casos os clusters não estão bem definidos.

Instância				
Nome	**Best**	n^o **pontos**	**Dimensão**	n^o **clusters**
Ruspini	0,7376	75	R^2	**4**
Iris	0.6862	150	R^4	**3**
Maronna	0.5745	200	R^2	**4**
200data	0,8231	200	R^2	**4**

Nome	**Melhor**	n^o **pontos**	**Dimensão**	n^o **clusters**
Vowel	0.4483	530	R^2	**12**
Broken Ring	0.4995	800	R^2	**5**
100p2c1	**?**	100	R^2	**2**
100p3c	**?**	100	R^2	**3**
100p3c1	**?**	100	R^2	**3**
100p5c1	**?**	100	R^2	**5**
100p7c	**?**	100	R^2	**7**
100p8c1	**?**	100	R^2	**8**
100p10c	**?**	100	R^2	**10**
200p2c1	**?**	200	R^2	**2**
200p3c1	**?**	200	R^2	**3**
200p4c	**?**	200	R^2	**4**
200p4c1	**?**	200	R^2	**4**
200p7c1	**?**	200	R^2	**7**
200p12c1	**?**	200	R^2	**12**
300p2c1	**?**	300	R^2	**2**
300p3c1	**?**	300	R^2	**3**
300p3c	**?**	300	R^2	**3**
300p4c1	**?**	300	R^2	**4**
300p6c1	**?**	300	R^2	**6**
300p13c1	**?**	300	R^2	**13**
400p3c	**?**	400	R^2	**3**
400p4c1	**?**	400	R^2	**4**
400p17c1	**?**	400	R^2	**17**
500p3c	**?**	500	R^2	**3**
500p4c1	**?**	500	R^2	**4**
500p6c1	**?**	500	R^2	**6**
600p3c1	**?**	600	R^2	**3**
600p15c	**?**	600	R^2	**15**
700p4c	**?**	700	R^2	**4**
700p15c1	**?**	700	R^2	**15**
800p4c1	**?**	800	R^2	**4**
800p10c1	**?**	800	R^2	**10**
800p18c1	**?**	800	R^2	**18**
800p23c	**?**	800	R^2	**23**
900p5c	**?**	900	R^2	**5**
900p12c	**?**	900	R^2	**12**

Nome	**Melhor**	n^o **pontos**	**Dimensão**	n^o **clusters**
1000p5c1	**?**	1000	R^2	**5**
1000p6c	**?**	1000	R^2	**6**
1000p14c	**?**	1000	R^2	**14**
1000p27c1	**?**	1000	R^2	**27**
1100p6c1	**?**	1100	R^2	**6**
1300p17c	**?**	1300	R^2	**17**
1500p6c	**?**	1500	R^2	**6**
1500p6c1	**?**	1500	R^2	**6**
1500p20c	**?**	1500	R^2	**20**
1800p22c	**?**	1800	R^2	**22**
2000p9c1	**?**	2000	R^2	**9**
2000p11c	**?**	2000	R^2	**11**

Tabela 2.5: Descrição das instâncias

Capítulo 3

Heurísticas para o PCA

Metaheurísticas são heurísticas genéricas para a solução aproximada de problemas, principalmente problemas de Otimização Combinatória de elevada complexidade. Entre elas, podemos citar, *Algoritmos Evolutivos, GRASP, ILS, Busca Tabu, VNS,* entre outras [18].

O PCA é um problema combinatório de elevada complexidade, pois o número soluções possíveis cresce exponencialmente à medida que o número de pontos cresce.

Não existem muitos trabalhos relacionados ao PCA. Tratando-se de Metaheurísticas, só existem dois trabalhos, de nosso conhecimento, um de 2001 [17] e outro de 2004 [47]. Os dois utilizam Algoritmos Evolutivos para a resolução do PCA.

O objetivo deste capítulo é propor algumas heurísticas para a solução do PCA. Para isso, são utilizadas as Metaheurísticas *Algoritmo Evolutivo , GRASP e ILS*. Estas Metaheurísticas se adequam ao problema e além disso, possuem enfoques diferentes.

Antes da definição das heurísticas, são definidas algumas características comuns aos algoritmos, e que ajudam na geração de soluções boas. Elas incluem um procedimento de construção de clusters parciais, a representação da solução, buscas locais e memória adaptativa.

No final, testes computacionais são realizados para verificar a qualidade das heurísticas propostas.

3.1 As características comuns aos Algoritmos

Nesta seção são descritas algumas características comuns aos algorimos. Primeiro é definido um procedimento denominado *Etapa de Construção*, que tem como objetivo tentar reduzir os dados de entrada do problema através da formação de clusters parciais. Estes clusters parciais são formados, agrupando em um mesmo cluster, pontos pertencentes a uma região densa. Desta forma, os algoritmos trabalham com conjunto de pontos e não necessariamente com pontos unitários,

Os algoritmos propostos neste trabalho utilizam a mesma estrutura de dados para representar e gerar a solução do problema.

Além disso, os algoritmos utilizam o conceito de memória adaptativa. Este conceito não está na definição original das metaheurísticas, mas pode melhorar as soluções obtidas.[18]

Finalmente, são definidas as buscas locais *Inversão Individual, Troca entre Pares e Reconexão por Caminhos* que tem como objetivo intensificar a busca de boas soluções. Os procedimentos são mostrados a seguir.

3.1.1 A Etapa de Construção

A Etapa de Construção é uma etapa inicial, que tem por objetivos tentar reduzir a cardinalidade dos dados de entrada do problema e facilitar a geração de soluções iniciais de boa qualidade para os algoritmos. Ela é composta por dois procedimentos: *O procedimento Gerar Clusters Parciais (GCP) e o Junção de Clusters Parciais Adjacentes (JCPA).*

O procedimento GCP tende a reduzir a cardinalidade do problema criando clusters parciais baseados no critério de densidade definido em [17, 47] . O *procedimento JCPA* tenta diminuir um pouca mais a cardinalidade do problema juntando clusters parciais adjacentes, ou seja, clusters parciais que estão muito próximos.

O *procedimento GCP* agrupa em um mesmo cluster os pontos pertencentes a uma região densa, como mostra o pseudocódigo da figura 3.1. Inicialmente, nas linhas 1, 2 e 3 , para cada ponto é definido a menor distância a outro ponto qualquer. Depois, na linha 4, é feito uma média destas distâncias, denominada d_{medio}. Então, na linha 5, cada ponto $x_i \in X$ é considerado o centro de um círculo cujo valor do raio é $r =$

Procedimento GCP (X,u)

1. *Para* $i = 1$ *ate* m *Faça*
2. $\quad d_{min}(x_i) = min||x_i - x_j||,\ i \neq j\ ,\ j = 1, ..., m$
3. *Fim Para*
4. $d_{medio} = \frac{1}{m} \sum_{i=1}^{m} d_{min}(x_i)$
5. $r = u * d_{medio}$
6. *Para* $i = 1$ *até* m *Faça*
7. $\quad N_i = circulo(x_i, r)$
8. $\quad T = T \cup N_i$
9. *Fim Para*
10. *Ordenar* T *em ordem decrescente*
11. $i = 1$
12. *Enquanto* $T \neq \emptyset$ *Faça*
13. $\quad B_i$ = *próximo*$(N_j \in T)$
14. $\quad T = T - N_j$
15. $\quad i = i + 1$
16. *Fim Enquanto*
17. *Retornar* $B = \{B_1, B_2, ..B_t\}$ *, os t clusters parciais.*
18. *Fim GCP*

Figura 3.1: O Pseudocódigo do procedimento GCP

u * d_{medio} , onde u é um parâmetro de entrada. Logo após, na linha 7, é calculado o conjunto de pontos contidos em cada círculo N_i = *circulo* (x_i, r), exemplificado na figura 3.2. Estes valores são colocados em uma lista T , indicado nas linhas 8 e 10, que é ordenada em ordem decrescente de cardinalidade. Entre as linhas 12 e 17, os elementos 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 seus pontos não podem mais entrar em outro círculo. Com este procedimento as regiões mais densas são selecionadas.

Após este procedimento inicial, é realizado um refinamento, denominado *JCPA* que tem como objetivo diminuir o número de clusters parciais. Assim, é efetuada uma agregação de *clusters parciais pequenos* (com até 4 pontos), que sejam próximos

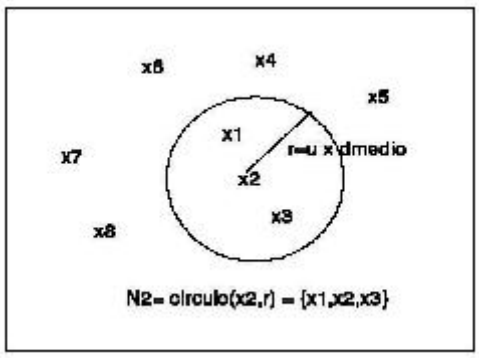

Figura 3.2: Cálculo dos pontos contidos no círculo de centro x_2 e raio $r = u\ x\ d_{medio}$

a algum outro *cluster parcial grande* (com mais de 10 pontos). Isto é realizado verificando se este cluster pequeno está a uma distancia d_{adj} do cluster grande, onde $d_{adj} = v * d_{medio}$ *(onde o valor v é um parâmetro de entrada)*

O psedocódigo de *procedimento JCPA* é descrito na figura 3.3. Os clusters parciais gerados no procedimento *GCP* são utilizados neste procedimento e está indicado na linha 1. Na linha 2 é calculado o d_{adj}. Para cada *cluster parcial pequeno*, é verificado qual o *cluster parcial grande* mais próximo em relação à distância entre os centróides (linhas 3 , 4 e 5). Se este cluster 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 6 e 7. O retorno deste procedimento é o conjunto de clusters $B = \{B_1, B_2, ..B_p\}$, onde $p \leq t$. Este procedimento reduz o número de clusters remanescentes do *procedimento GCP.*

Os clusters gerados na Etapa de Construção, após os *procedimentos GCP e JCPA,* são denominados clusters iniciais.

3.1.2 Representação da Solução

Considere os clusters iniciais gerados na Etapa de construção como $B = \{B_1, B_2, ...B_p\}$ e seja $v_i,\ i = 1, 2, ...p$ o centróide de cada cluster B_i. Para representar uma solução é utilizada uma cadeia binária de p posições. Por exemplo, se $p = 7$, então uma cadeia binária poderia ser {0110010}. Se o valor correspondente ao B_i na cadeia binária *for igual a 1*, 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*

Procedimento JCPA (B , v , d_{medio})

1. *Seja* $B = \{B_1, B_2, ..B_t\}$ *o conjunto de clusters parciais gerados pelo GCP*
2. $d_{adj} = v * d_{medio}$
3. *Para* $i = 1$ *até* t *Faça*
4. *Se* $cardinalidade(B_i) <= 4$ *Então*
5. $B_k = menor\_distancia\_centroide(B_i)$
6. *Se* $(\exists x_r \in B_i \; e \; \exists x_s \in B_k \; tq \; ||x_r - x_s|| < d_{adj})$ *Entao*
7. $B_k = B_k \cup B_i$
8. *Fim Se*
9. *Fim Se*
10. *Fim Para*
11. *Retornar* $B = \{B_1, B_{2,}, ..., B_p\}$ *clusters iniciais, onde* $p <= t$
12. *Fim JCPA*

Figura 3.3: O Pseudocódigo do procedimento JCPA

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 é recalculado. No final, todos os clusters filhos são unidos aos clusters pais para gerar uma solução. Portanto, o número de clusters pais gerados em cada solução não é alterado. Os clusters gerados após esse processo são denominados clusters finais $C = \{C_1, C_{2,}, ..., C_k\}$.

Para mostrar melhor o 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 $p = r + s$. Inicialmente cada B_i^1 é candidato único a cada cluster C_i. Então para cada $B_h^o \in B^o$, encontrar um $B_z^1 \in B^1$, tal que seus centróides satisfaçam a relação abaixo :

$$||v_z^1 - v_h^o|| = min||v_i^1 - v_h^o||, \;\; i = 1, ..., s. \tag{3.1}$$

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

definido como:

$$C_j = \bigcup_{i=1}^{q} B_i \;\; onde \; 1 \leq q \leq p \tag{3.2}$$

Após finalizar este procedimento, uma solução é encontrada.

3.1.3 Memória Adaptativa

As metaheurísticas utilizadas neste trabalho, não possuem em sua definição o conceito de Memória Adaptativa. As soluções são geradas, porém não são armazenadas para uso posterior.

A memória Adaptativa [18] utiliza um conjunto com as melhores soluções do algoritmo, que são atualizadas ao longo das iterações. Neste trabalho é utilizado um conjunto , denominado *ELITE* , que armazena a melhor solução de cada iteração do algoritmo. As soluções armazenadas são difrentes entre si.

O *conjunto ELITE* é utilizado em dois momentos diferentes em cada algoritmo proposto: no meio do algoritmo, para efetuar a busca local Reconexão por Caminhos e no final do algoritmo, para efetuar uma busca local (Troca entre Pares ou Inversão Individual, dependendo do algoritmo proposto), com o objetivo de tentar melhorar as melhores soluções encontradas.

3.1.4 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, ela permuta o valor de cada elemento da solução (1 por 0, ou 0 por 1), um por vez, e calcula o valor Índice Silhueta da nova solução. Porém o algoritmo só aceita a mudança, se o novo valor da função for melhor que o valor anterior.

Por exemplo, imagine que a solução corrente é $\{\boldsymbol{01}101101\}$. Primeiramente, é trocado o primeiro elemento da solução. Então é gerada a nova solução $\{\boldsymbol{1}101101\}$. Se esta solução tem valor da função maior que o anterior, então ela é a nova solução. E então, é trocado o segundo elemento. A solução agora é $\{1\boldsymbol{0}01101\}$. Se esta possuir o valor da função maior que o anterior, então a mudança é aceita. Caso contrario, a solução anterior é mantida. A busca acaba quando todos os elementos

da solução são trocados.

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 a solução corrente.

3.1.5 A Busca Local Troca Entre Pares

A busca local *Troca entre Pares* é uma busca intensiva e troca o status de dois elementos da solução com valores diferentes.

Por exemplo, suponha que a solução corrente é $\{\boldsymbol{10}111010\}$. Primeiro é trocado o primeiro e o segundo elemento da solução. Então a nova solução é $\{\boldsymbol{01}111010\}$. Se esta nova solução melhorar o valor Índíce Silhueta da solução anterior, então ela é aceita e é a nova solução corrente. A próxima troca é feita entre o primeiro e o terceiro elemento da solução corrente. Depois, entre o primeiro elemento e o quarto elemento, e assim, sucessivamente. A busca local termina quando todas as trocas entre dois elementos com valores distintos são testadas.

O objetivo desta busca local é tentar encontrar soluçãos melhores sem alterar o número de clusters pais obtidos pelas soluções anteriores.

3.1.6 Reconexão por Caminhos (RC)

O procedimento Reconexão por Caminhos (RC) foi proposto originalmente para os métodos *Tabu Search e Scatter Search [18]* como uma estratégia de intensificação, explorando trajetórias que conectavam soluções de boa qualidade obtidas pela heurística.

Este procedimento tem como objetivo procurar soluções intermediárias de boa qualidade entre duas soluções extremas. O princípio básico da RC é que, entre duas soluções de qualidade, pode existir uma terceira melhor que as outras.

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 de Caminhos utilizada, 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 solução de melhor qualidade é inserido um pedaço da solução de menor qualidade. Primeiro é inserido o último elemento de S_{pior} em S_{melhor}. Depois, são inseridos o último e o penúltimo elemento de S_{pior} em S_{melhor}. O processo continua até a troca de todos os elementos das soluções. A RC retorna a melhor solução obtida durante todo o processo.

A busca local *Reconexão por Caminhos* se justifica, pois encontrar o número ideal de clusters é um dos objetivos do problema, e entre duas soluções com números de clusters pais diferentes, existem outras soluções com diferentes números de clusters pais.

3.2 As Heurísticas Propostas

Uma vez definidos os procedimentos comuns, nesta seção são apresentadas as heurísticas propostas. Primeiro é apresentado um resumo de cada metahéuristica: *Algoritmo Evolutivo , GRASP e ILS.* Para cada uma delas, são propostas duas heurísticas. No final, experimentos computacionais são realizados para definir qual a proposta obtem os melhores resultados.

3.2.1 Algoritmos Evolutivos

A expressão Algoritmos Evolutivos ou Algoritmos Evolucionários [18] corresponde a classe de algoritmos para a solução de problemas de otimização que utilizam métodos computacionais baseados em teoria da evolução da espécie, proposta por Charles Darwin, e nos princípios básicos da herança genética, descritos por Gregor Mendel.

Darwin propoẽ um modelo de evolução em que uma população de indivíduos sofre um processo de evolução natural e estes são capazes de se adaptarem ao ambiente em que vivem através de processos de seleção natural, reprodução, recombinação sexual e mutação. Os indivíduos mais adaptados têm maiores chance de sobreviverem e gerarem descendentes. O processo de seleção privilegia os indivíduos com alta capacidade de sobrevivência e permite que a qualidade média da população melhore ao longo do processo evolutivo, levando a obtenção de um indivíduo totalmente adaptado ao ambiente.

Os Algoritmos Evolutivos (AEs) utilizam estas idéias através da manipulação de uma população de indivíduos (soluções) que evoluem ao longo de várias iterações do AE, chamados de gerações. Os AES são divididos em vários grupos, onde o principal representante é denominado Algoritmos Genéticos (AG) descrito a seguir.

Algoritmos Genéticos

O comportamento dos AGs corresponde a uma analogia com o comportamento dos indivíduos de uma população da natureza. Considerando uma população de indivíduos da natureza, estes competem entre si por diferentes recursos disponíveis ao seu meio ambiente (habitat) como água, comida e abrigo. Cada um dos indivíduos possuem caractecterísticas externas (fenótipo) relacionadas a sua constituiç ao geneética (genótipo), que os diferem entre si em relação à adaptação ao meio ambiente em que vivem. Esta adaptação afeta diretamente a capacidade de sobrevivência por período suficiente para se reproduzirem pelo acasalamento. Através do acasalamento, as caractecterísticas genéticas dos dois indivíduos envolvidos são combinadas e tranmitidas para a prole. Desta forma, as gerações futuras possuem uma grande probabilidade de serem formadas por indivíduos com características necessárias para um maior tempo de vida, em relação as gerações anteriores. Este processo é denominado seleção natural.

Em um AG tradicional, cada indivíduo corresponde à codificação de uma solução para o problema considerado. Para realizar esta codificação (ou representação da solução) , normalmente é utilizado um vetor de valores binários, inteiros ou reais, que constitui o próprio indivíduo. A população corresponde a um conjunto de soluções do problema.

O fenótipo de um indivíduo é obtido a partir da sua submissão a uma função que irá avaliar a qualidade do seu código genético. Esta função é denominada função de aptidão e representa a qualidade de cada indivíduo em relação ao problema modelado. Num AG, os códigos genéticos de indivíduos mais aptos, tem uma chance maior de serem tansmitidos para as gerações futuras, através dos processos de seleção e reprodução.

Um AG utiliza operadores genéticos sobre os indivíduos da população como o operador de cruzamanto e o operador de mutação.

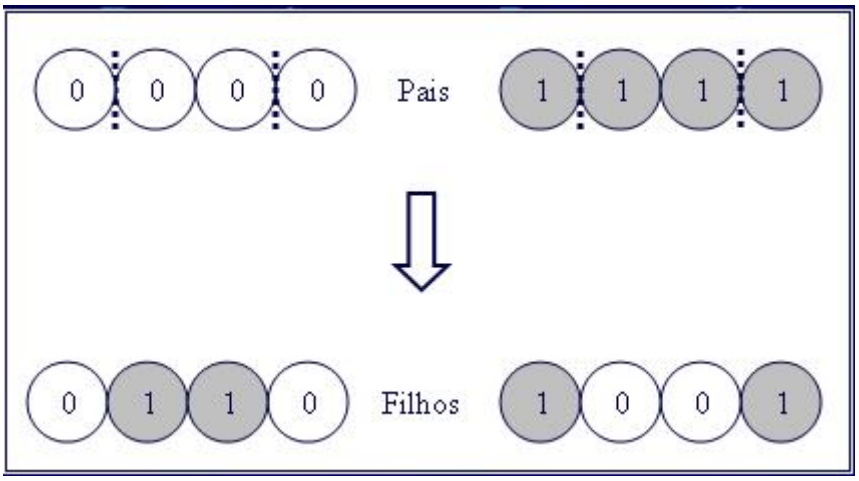

Figura 3.4: Exemplo de cruzamento de dois pontos

O operador de cruzamento combina partes do código genético de indivíduos diferentes. Existem diversas formas de utilização do operador de cruzamento, como o cruzamento de dois pontos.

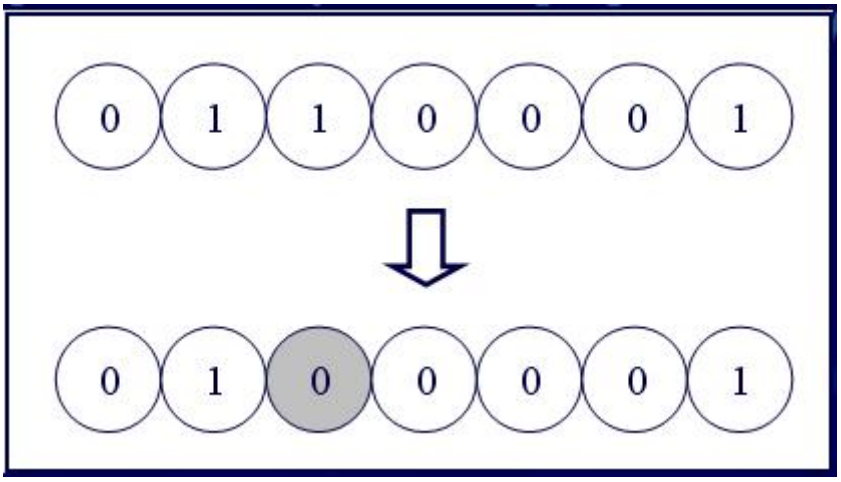

Figura 3.5: Exemplo de mutação

O cruzamento de dois pontos atua sobre dois indivíduos 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 são trocados. Os dois pontos de corte definem os segmentos dos vetores que serão trocados entre os mesmos para gerar novos indivíduos. A aplicação do operador de cruzamento a um par de indivíduos normalmente está sujeita a uma taxa de probabilidade, definida como parâmetro para a execução do AG. A figura 3.4 mostra este

procedimento.

O operador de mutação realiza trocas aleatórias de alguns valores dos indivíduos, e sua aplicação, também está sujeita a uma taxa de probabilidade, definida como parâmetro. A figura 3.5 mostra um exemplo de mutação.

Procedimento Algoritmo Genético Tradicional
1. i = 0
2. *Gerar a população inicial P(0)*
3. *Avaliar a população inicial P(0)*
4. *Enquanto não (condição de término) Faça*
5. *i = i + 1*
6. *selecionar P(i) de P(i-1)*
7. *Aplicar os operadores genéticos a P(i)*
8. *Avaliar os indivíduos de P(i)*
9. *Fim-enquanto*
10. *Fim Procedimento*

Figura 3.6: O psedocódigo do Algoritmo Genético Tradicional

Estes operadores permitem a exploração de novas características nos indivíduos e correspondem a evolução dos indivíduos. O objetivo dos operadores genéticos é permitir que diferentes áreas do espaço de busca possam ser exploradas, evitando a convergência prematura do algoritmo para um ótimo local, ainda distante do ótimo global. O pseudocódigo mostrado na figura 3.6 mostra o funcionamento de um AG tradicional.

Para modelar um problema específico utilizando um AG, é necessário considerar:

- Representação dos indivíduos: como representar as possíveis soluções para o problema.
- Função de aptidão : de que forma a função de aptidão pode representar, de forma precisa, a qualidade de cada solução obtida.
- Seleção e Reprodução: como será realizada a seleção de indivíduos de uma geração, para serem aplicados os operadores genéticos e com isso, constituírem

a população da geração seguinte.

- Operadores genéticos : quais os operadores genéticos devem ser aplicados, e de que forma.

- Outros paramêtros : quais os valores que devem ser utilizados para o tamanho da população, critério de parada, entre outros.

3.2.2 Os Algoritmos Evolutivos Propostos

Uma vez definidos os procedimentos comuns e a metaheurística Algoritmos Evolutivos, podemos definir as implementações utilizadas. Foram implementados duas versões para o AEs, denominados *Algoritmo Evolutivo Construtivo com Busca Local 1 (AECBL1) e AECBL2* . A diferença entre as implementações é a ordem que as buscas locais são chamadas.

O *AECBL1 e AECBL2* são métodos compostos de duas fases. A fase inicial corresponde a Etapa de Construção, definida anteriormente. A segunda etapa dos *AECBLs* possuem um algoritmo genético com busca local que utiliza conceitos de memória adaptativa para a busca da melhor configuração de uma solução possível. As duas fases dos *AECBLs* são denotadas por: fase de construção e o módulo evolutivo. As duas fases são mostradas a seguir.

A Fase de Construção

A fase de construção dos algoritmos evolutivos propostos é igual a Etapa de Construção, definida na seção 3.1.1.

O Módulo Evolutivo

O módulo evolutivo é composto de um Algoritmo Genético tradicional acrescido de 3 buscas locais. O algoritmo genético puro nem sempre consegue bons resultados. Então, é necessária a utilização de busca local para melhorar a qualidade das soluções. Por isso, foram utilizadas as buscas locais *Inversão Individual , Troca entre Pares e Reconexão por Caminhos.*

A racionalidade de utilizar algoritmos genéticos é gerar aleatoriamente um número qualquer de clusters pais. Como o número de clusters pais é um dos ob-

jetivos do problema, o algoritmo genético permite gerar a cada iteração, números diferentes de clusters pais.

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 de aptidão. Este procedimento permite começar o algoritmo genético com uma população de melhor qualidade.

Para efetuar a seleção dos indivíduos para cruzamento, são utilizados dois operadores. Esses operadores se alternam. O primeiro operador escolhe aleatoriamente os dois indivíduos dentre os 60% com melhores valores de aptidão. E o segundo operador, escolhe um indivíduo aleatoriamente entre os 60% com melhores valores de aptidão e o outro, entre os 40% restantes.

O operador de cruzamento utilizado é o cruzamento de dois pontos. Os indivíduos são submetidos ao cruzamento com probabilidade p_c.

O operador de mutação utilizado realiza a troca de um elemento do indivíduo, com probabilidade p_m.

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

A cada t iterações, os melhores indivíduos da população passam pela busca local (Inversão Individual no *AECBL1* e Troca entre Pares no *AECBL2*) . O objetivo é intensificar a procura de soluções diferentes no conjunto de soluções existentes.

A cada r iterações, o melhor indivíduo da população e o melhor indivíduo do conjunto *ELITE*, dos algoritmos *AECBL1 e AECBL2,* passam pelo Busca local Reconexão por Caminhos.

A cada iteração, a melhor solução é armazenada no conjunto *ELITE*. No final, o conjunto *ELITE* passa por uma segunda busca local (Troca entre Pares no *AECBL1* e Inversão Individual no *AECBL2).* O objetivo é tentar melhorar as melhores soluções encontradas.

Os Pseudocódigos dos Algoritmos

A figura 3.7 mostra o pseudocódigo do *AECBL1*.

O algoritmo utiliza os seguintes parâmetros de entrada: o conjunto de pontos

```
    Procedimento AECBL1 (X, Tpop , G_max , p_c , p_m , u , v , t , r)
1.  G = Fase de Construção (X,u,v)
2.  P = Gerar População Inicial(G , Tpop)
3.  Para k = 1 ate G_max   Faça
4.     i = 1
5.     Enquanto i < Tpop/2   Faça
6.        Seleciona (p1, p2)
7.        q = random (100)
8.        Se (q ≥ P_c * 100) Então
9.              i = i + 1
10.             Se Cruzamento (p1,p2,f1,f2) Então
11.                Mutacao(p1, p2, P_m)
12.             Fim Se
13.             Se(Avaliar Solução (f1,f2) ) Então
14.                Atualizar população (p1, p2, P)
15.             Fim Se
16.       Fim Se
17.    Fim Enquanto
18.    Se (k mod t = 0 ) Entao
19.       Inversão Individual (P)
20.    Fim Se
21.    Se ( k mod r) = 0 Entao
22.       Reconexao por Caminhos(P,ELITE)
23.    Fim Se
24.    Atualizar conjunto Elite (ELITE)
25. Fim Para
26. Troca entre Pares (ELITE)
27. Retornar (Melhor solução)
28. Fim AECBL1
```

Figura 3.7: O pseudocódigo do AECBL1

X , o tamanho da população definida *Tpop*, o número de iterações que o algoritmo irá executar G_{max}, as probabilidades de cruzamento p_c e mutação p_m , os valores de u *e* v utilizados na fase de Construção e os valores t *e* r , que representam a periodicidade que o algoritmo executa as buscas locais *Inversão Individual e Reconexão por Caminhos.*

Na linha 1, é executada a fase de construção no conjunto de pontos X. Este utiliza os parâmetros u *e* v e gera um conjunto de clusters iniciais G.

Na linha 2 é gerada a população P , tendo como entrada G e o tamanho da população *Tpop.* Os passos entre as linhas 3 e 17 mostram a execução das gerações do *AECBL1.* São realizadas G_{max} iterações no algoritmo, ou seja, são construídas G_{max} gerações no algoritmo genético.

Nesta figura, $p1$ *e* $p2$ representam indivíduos com respectivos valores de aptidão $f1$ *e* $f2$.

A cada geração, pares de indivíduos são selecionados para fazer o cruzamento. Porém, cada cruzamento só é realizado, dependendo da probabilidade p_c. O cruzamento é feito em indivíduos diferentes. A mutação é realizada em um dos indivíduos ($p1$ *ou* $p2$, escolhido aleatoriamente), dependendo da probabilidade p_m. Após o cruzamento e a mutação, é feita uma avaliação da função de aptidão para verificar se as novas soluções melhoram as soluções anteriores da população. Se isso acontecer, a população é atualizada. Isto está indicado entre as linhas 5 e 17.

A cada t iterações, nas linhas 18, 19 e 20, é realizada a busca denominada *Inversão Individual* nos melhores elementos da população.

A cada r iterações, nas linhas 21, 22 e 23, é realizada a busca local *Reconexão por Caminhos* entre o melhor elemento da população e o melhor elemento do conjunto *ELITE.*

A cada geração, na linha 24, a melhor solução encontrada é armazenada no conjunto *ELITE.* Este conjunto só armazena a solução se está for melhor que a pior solução do conjunto *ELITE* e diferente das outras.

A busca *Troca entre Pares* é realizada no conjunto *ELITE* e está indicada na linha 26. O retorno do *AECBL1* é a melhor solução encontrada após a execução de todos os procedimentos. Isto está indicado na linha 27.

O algoritmo *AECBL2* é semelhante ao anterior e as difenças estão nas linhas 19

```
    Procedimento AECBL2 (X, Tpop , G_max , p_c , p_m  , u , v , t , r)
1.  G = Fase de Construção (X,u,v)
2.  P = Gerar População Inicial(G , Tpop)
3.  Para k = 1 ate G_max   Faça
4.     i = 1
5.     Enquanto i < Tpop/2   Faça
6.        Seleciona (p1, p2)
7.        q = random (100)
8.        Se (q ≥ P_c * 100) Então
9.              i = i + 1
10.             Se Cruzamento (p1,p2,f1,f2) Então
11.                Mutacao(p1, p2, P_m)
12.             Fim Se
13.             Se(Avaliar Solução (f1,f2) ) Então
14.                Atualizar população (p1, p2, P)
15.             Fim Se
16.       Fim Se
17.    Fim Enquanto
18.    Se (k mod t = 0 ) Entao
19.       Troca entre Pares (P)
20.    Fim Se
21.    Se ( k mod r) = 0 Entao
22.       Reconexao por Caminhos(P,ELITE)
23.    Fim Se
24.    Atualizar conjunto Elite (ELITE)
25. Fim Para
26. Inversão Individual(ELITE)
27. Retornar (Melhor solução)
28. Fim AECBL2
```

Figura 3.8: O pseudocódigo do AECBL2

e 26, onde as buscas locais Inversão Individual e Troca entre Pares estão invertidas. O pseudocódigo do algoritmo *AECBL2* é mostrado na figura 3.8.

3.2.3 GRASP

A metaheurística *GRASP (Greedy Randomized Adaptative Search Procedure)* foi proposta por Feo e Rezende[18]. É um processo iterativo do tipo *multistart* para obter soluções para problemas de Otimização combinatória. É um método que consiste de duas fases: uma de construção e outra de busca local. A primeira fase constroi uma solução viável para o problema proposto. A busca local tenta melhorar a solução obtida na fase anterior.

O pseudocódigo do GRASP, na sua forma clássica, é mostrado na figura 3.9. Na linha 1, os dados do problema são lidos. Nas linhas de 2 à 8, são realizadas as iterações do GRASP. Estas iterações são realizadas por MaxIter iterações. Na linha 3 é executada a fase de construção e na linha 4 a busca local. Na linha 5 é verificado se a solução obtida na iteração é melhor a que solução encontrada até o momento. Na linha 6 é feita atualização da melhor solução.

	Procedimento GRASP
1.	*Ler os Dados do Problema*
2.	*Para k = 1 até MaxIter Faça*
3.	*Construir uma solução Randomizada (fase de construção)*
4.	*Encontrar y aplicando uma busca local em x (fase de busca local)*
5.	*Se f(y) < f^* então*
6.	$x^* = y \; ; \; f^* = f(y)$
7.	*Fim Se*
8.	*Fim Para*
9.	*Retornar x^**
10.	*Fim GRASP*

Figura 3.9: O pseudocódigo do GRASP

Na fase de construção, o conjunto de elementos candidatos é formado por todos os elementos que não foram incorporados á solução parcial em construção e que não inviabilizam esta, caso sejam incorporados. A escolha do próximo elemento a ser in-

corporado é determinada pela avaliação de todos os elementos candidatos de acordo com uma função gulosa (custo incremental). Esta função avalia os benefícios ganhos com a inserção deste elemento na solução em construção. A avaliação dos elementos leva a criação de uma lista restrita de candidatos (LRC) formado por um subconjunto dos melhores candidatos, isto é, aqueles cuja incorporação à solução parcial corrente resulta nos melhores custos incrementais (aspecto guloso do algoritmo). O elemento a ser incorporado à solução parcial é selecionado, aleatoriamente, dentro da LRC (aspecto probabilístico do algoritmo). Uma vez que o elemento selecionado foi incorporado á solução parcial, a LRC é atualizada e os custos incrementais são reavaliados (aspecto Adaptativo).

	Procedimento Construtivo
1.	$x = \emptyset$
2.	*Enquanto (x não for uma solução completa) Faça*
3.	*Avaliar os custos dos elementos candidatos*
4.	*Construir uma lista de candidados LRC*
5.	*Selecionar aleatoriamente um elemento* $s \in LRC$
6.	$x = x + s$
7.	*Fim-enquanto*
8.	*Retornar x*
9.	*FIM Procedimento Construtivo*

Figura 3.10: O pseudocódigo do procedimento construtivo do GRASP

O procedimento construtivo é mostrado na figura 3.10. Na linha 1 , a solução a ser construída é inicializada. A solução é construída entre as linhas 2 e 7. Nas linhas 3 e 4, os custos de cada candidato são avaliados e formam a LRC. Um elemento é escolhido na LRC aleatoriamente (linha 5), e na linha 6 , este elemento é incorporado a solução. O retorno deste procedimento é uma solução inicial viável.

A busca local tenta melhorar a solução corrente e encontrar um ótimo local. Para isto, a busca local substitui, iterativamente, a solução corrente por uma solução melhor, pertencente a sua vizinhança.

Vários conceitos e módulos de aperfeiçoamento foram propostos para tentar melhorar as soluções da Metaheurística GRASP. Alguns exemplos são GRASP rea-

tivo, Memória a Longo Prazo, entre outros. Porém existe um módulo, denominado GRASP com filtro, cujo conceito é utilizado neste trabalho. Este módulo é descrito a seguir.

GRASP com Filtro

Os métodos de construção em otimização combinatória, normalmente, têm a função de gerar uma solução viável para o problema. Métodos que utilizam construção e busca local se caracterizam por construir e posteriormente refinar a solução inicial.

O que é observado, é que as buscas locais utilizam a maior parte dos tempos totais dos métodos.

Uma forma de tentar reduzir os tempos de uma busca local é melhorar as soluções iniciais. Esta sugestão é bem justificada no caso do GRASP, onde a cada iteração é gerada uma solução inicial e posteriormente essa é refinada por uma busca local.

A proposta de usar um filtro na etapa de construção GRASP, simplesmente, se reduz a cada iteração GRASP gerar p soluções iniciais e selecionar somente as melhores para efetuar a busca local.

3.2.4 Algoritmos GRASP propostos

Uma vez definidos os procedimentos comuns e a metaheurística GRASP, podemos definir as implementações utilizadas. Foram implementados duas versões, denominados *Grasp com Busca Local Inversão Individual e Troca entre Pares e Reconexão por Caminhos 1 (GBLITRC1) e GBLITRC2* . Os algoritmos possuem duas fases : *a fase de construção e a fase de busca local.*

A fase de construção é comun para as duas implementações. Eles utilizam os procedimentos da Etapa de Construção e geram uma lista LRC com com o número de clusters pais das melhores soluções encontradas. Esta lista é utilizada posteriormente para gerar as soluções iniciais.

A fase de busca local utiliza as buscas locais Troca entre Pares, Inversão individual e Reconexão por Caminhos. A diferença entre os algoritmos é a ordem em que as buscas locais aparecem. Os procedimentos são mostrados a seguir.

A fase de Construção

A fase de construção do GRASP utiliza os procedimentos da Etapa de construção, definida na seção 3.1.1. Nesta fase, são geradas várias soluções, e o número de clusters pais das melhores, são armazenados em uma lista LRC.

Inicialmente, o procedimento *Gerar LRC* utiliza a Etapa de Construção para construir os clusters iniciais $B = \{B_1, ..., B_p\}$. Depois, a cada ietração é gerada uma solução com k clusters pais, *k variando entre 2 e p-1.* Os k clusters pais são escolhidos, aleatoriamente, entre os *p clusters iniciais possíveis.* Este procedimento é repetido *MaxIter* vezes. Cada solução encontrada é avaliada através da função Índice Silhueta e o número de clusters pais das melhores soluções são armazenados na LRC, cujos valores são diferentes.

Após a geração da lista LRC, os algoritmos tem uma boa estimativa do número ideal de clusters.

	Procedimento Gerar LRC (X , u , v , $MaxIter$)
1.	*G = Etapa de Construção (X, u , v)*
3.	*Para i = 1 ate MaxIter Faça*
4.	*Para k = 2 ate p-1 Faça*
5.	s^0= *Gerar Solucao(k , G)*
6.	*Atualizar LRC*(s^0)
7.	*Fim Para*
8.	*Fim Para*
9.	*Retornar LRC*
10.	*Fim Gerar LRC*

Figura 3.11: O pseudocódigo do procedimento construtivo dos algoritmos GRASP

O procedimento é mostrado na figura 3.11. Na linha 1 são gerados os clusters iniciais através da Etapa de Construção. Estes clusters são arnazenados no conjunto G. Depois é feito um processo iterativo, que a cada momento, gera uma solução com k clusters pais , $2 < k < p - 1$. Cada solução é avaliada e as melhores são armazenadas numa lista denominada LRC. Isto está indicado entre as linhas 4 e 7. Este processo iterativo é repetido por MaxIter vezes, como indica a linha 3. O retorno deste procedimento é a LRC.

Para gerar uma solução inicial, é escolhido um elemento $nc \in LRC$. Então é gerada uma solução contendo nc clusters pais, escolhidos, aleatoriamente, entre os p possíveis.

A Fase de Busca Local

O algoritmo *GBLITRC1* gera uma solução inicial e aplica a busca local Inversão Individual. A cada t iterações, o algoritmo também aplica a busca local Reconexão por Caminhos. No final, o algoritmo aplica a busca local Troca entre Pares, no conjunto ELITE gerado.

O algoritmo *GBLITRC2*, gera uma solução inicial e aplica a busca local Troca entre Pares. A cada t iterações, o algoritmo também aplica a busca local Reconexão por caminhos. No final, o algoritmo aplica a busca local Inversão individual, no conjunto ELITE gerado.

Procedimento GBLITRC1 (X, u, v , MaxIter , G_{max}, t)

1. *Gerar LRC (X,u,v,MaxIter)*
2. *Para k = 1 ate G_{max} Faça*
3. *Selecionar nc $\in$ LRC*
4. *s^0 = Gerar solucao inicial(nc)*
5. *s^1 = Inversao Individual(s^0)*
6. *Se (k mod t) = 0 Então*
7. *Reconexao por Caminhos (s^1, s^*)*
8. *Fim Se*
9. *Atualizar ELITE(s^1)*
10. *s^*= Retornar melhor (ELITE)*
11. *Fim Para*
12. *Troca entre Pares(ELITE)*
13. *s^*= Retornar melhor (ELITE)*
14. *Retornar (s^*)*
15. *Fim GBLITRC1*

Figura 3.12: O pseudocódigo do GBLITRC1

Os pseudocódigos dos Algoritmos

Os pseudocódigo do algoritmo *GBLITRC1* é mostrado na figura 3.12. O algoritmo utiliza os seguintes parâmetros de entrada: o conjunto de pontos X , os valores de u , v *e* $MaxIter$ utilizados no procedimento Gerar LRC, o número de iterações que o algoritmo executa G_{max} e o valor t, que representa a periodicidade que o algoritmo executa a busca local *Reconexão por Caminhos*.

	Procedimento GBLITRC2 (X, u, v , MaxIter, G_{max}*, t)*
1.	*Gerar LRC (X,u,v,MaxIter)*
2.	*Para k = 1 ate* G_{max} *Faça*
3.	*Selecionar* $nc \in LRC$
4.	s^0 *= Gerar solucao inicial(nc)*
5.	s^1 *= Troca entre Pares*(s^0)
6.	*Se (k mod t) = 0 Então*
7.	*Reconexao por Caminhos* (s^1, s^*)
8.	*Fim Se*
9.	*Atualizar ELITE*(s^1)
10.	s^**= Retornar melhor (ELITE)*
11.	*Fim Para*
12.	*Inversão Individual(ELITE)*
13.	s^**= Retornar melhor (ELITE)*
14.	*Retornar* (s^*)
15.	*Fim GBLITRC2*

Figura 3.13: O pseudocódigo do GBLITRC2

Na linha 1 , é gerada a LRC através do procedimento Gerar LRC. O processo iterativo do GRASP se repete por G_{max} vezes, e seleciona um elemento $nc \in LRC$, e gera uma solução s^0 com este número de clusters pais. Depois a solução s^0 passa pela busca local Inversao Individual gerando uma nova solução s^1. A cada t iterações, a solução s^1 passa por uma outra busca local denominada Reconexão por Caminhos, juntamente com s^* , que é a melhor solução do Conjunto ELITE. Isto está indicado entre as linhas 2 e 8. Nas linhas 9 e 10, o conjunto ELITE e s^* são atualizados. No final, na linha 12 , o conjunto ELITE passa pela busca local Troca Entre Pares. A

Melhor solução s^* é o retorno do algoritmo.

O algoritmo *GBLITRC2* é semelhante ao anterior e as difenças estão nas linhas 5 e 12, onde as buscas locais Inversão Individual e Troca entre Pares estão invertidas. O pseudocódigo do algoritmo *GBLITRC2* é mostrado na figura 3.13.

3.2.5 ILS

A metaheurística ILS (Iterated Local Search) foi proposta por Lourenço, Martin e Stutze em 2002 [18]. A ILS consiste, basicamente, na aplicação iterativa de um procedimento de busca local em uma solução inicial s^0. A solução inicial é obtida através de uma heurística de construção ou de um procedimento aleatório de construção. A busca local tenta melhorar a solução inicial, no primeiro momento, e posteriormente nas soluções perturbadas, com o objetivo de produzir soluções ótimas ou próximas a elas.

O desempenho da Metaheurística ILS usualmente está condicionado a escolha dos 3 procedimentos básicos: busca local , pertubação e critério de aceitação. Estes procediemntos estão intimamente ligados ao problema a ser resolvido.

Algumas considerações são necessárias para desecrever o ILS. Seja P o problema a ser resolvido e f a função associada ao problema. Seja S o conjunto de todas as soluções viáveis de P e seja $S^* \subset S$ o conjunto de todos os ótimos locais de P. O objetivo do ILS é trabalhar com S^* e não com S, para obter soluções de melhor qualidade, pois trabalha num conjunto mais restrito.

O objetivo é explorar S^* , considerando uma trajetória que possibilite a passagem de uma solução atual $s^* \in S^*$ para uma nova solução $s^{**} \in S^*$, independente desta solução estar próxima ou não da atual. Para isto é feito uma pertubação em s^*, o que conduz a uma solução intermediária $s^1 \in S$. Então é aplicado uma busca local a s^1 e encontra-se uma nova solução $s^{**} \in S^*$. Se esta solução for aceita no teste de aceitação, então passa a ser o novo elemento da trajetória, ou seja, $s^* = s^{**}$. Se s^{**} não for aceito no teste de aceitação, é feito uma nova perturbação e o processo coninua até um número MaxIter de iterações.

O pseudocódigo do algoritmo é mostrado na figura 3.14. Na linha 1 é gerado uma solução inicial s^0. Na linha 2, esta solução passa por uma busca local e gera uma nova solução $s^* \in S^*$. Depois começa um processo iterativo do ILS, que se repete

	Procedimento Iterated Local Search
1.	s^0 = *Gerar Solução Inicial*
2.	s^* = *Busca_local*(s^0)
3.	*Repita*
4.	s^1 = *Pertubação*(s^*)
5.	s^{**} = *Busca_local*(s^1)
6.	s^* = *Critério_aceitação* (s^*, S^{**})
7.	*Até (sejam efetuadas MaxIter iterações)*
8.	*Fim Procedimento*

Figura 3.14: O pseudocódigo do ILS

por $MaxIter$ vezes. Na linha 4 a solução s^* é perturbada e gera uma nova solução $s^1 \in S$. Na linha 5, s^1 passa por uma busca local e é tranformada em $s^{**} \in S^*$. Na linha 6 o critério de aceitação é aplicado e o processo iterativo continua.

3.2.6 Algoritmos ILS propostos

Uma vez definidos os procedimentos comuns e a metaheurística ILS, podemos definir as implementações utilizadas. Foram implementados duas versões para o ILS, denominadas *ILS com Busca Local Inversão Individual e Troca entre Pares e Reconexão por caminhos 1 (IBLITRC1) e IBLITRC2*. Os algoritmos possuem 4 componentes principais: o procedimento Gerar solução inicial, as buscas locais, a perturbação e o critério de aceitação. A diferença entre os algoritmos é a ordem que as buscas locais são chamadas.

O procedimento Gerar solução Inicial é comum para as duas implementações. Ele utiliza a Etapa de Construção (definido na seção 3.1.1) para gerar os clusters iniciais. O procedimento gera várias soluções e armazena em uma lista LRC, o número de clusters pais das melhores soluções encontradas. Além da LRC, o procedimento retorna a melhor solução encontrada.

As buscas locais utilizadas são Troca entre Pares, Inversão individual e Reconexão por Caminhos.

A perturbação altera o número de clusters pais da solução corrente e utiliza os dados da LRC.

```
    Procedimento Gerar Solução Inicial (X , u , v , Iter)
1.  G = Etapa de Construção (X, u, v)
3.  Para i = 1 ate Iter Faça
4.    Para k = 2 ate p-1 Faça
5.          s^0 = Gerar solucao(k , G)
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 Gerar Solução Inicial
```

Figura 3.15: O pseudocódigo do procedimento Gerar solução inicial

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

O Procedimento Gerar solução inicial

O procedimento Gerar solução inicial é muito parecido com o procedimento construtivo do GRASP, e gera, inclusive, uma lista LRC. Porém o retorno deste procedimento é a melhor solução encontrada.

O procedimento é mostrado na figura 3.15. Na linha 1 são gerados os clusters iniciais através da Etapa de Construção. Estes clusters são arnazenados no conjunto G. Depois é feito o processo iterativo que, a cada momento, gera uma solução com k clusters pais , $2 < k < p-1$. Cada solução é avaliada e o número de clustres pais das melhores são armazenadas numa lista denominada LRC. Além disso, a melhor solução fica armazenada em s^*. Isto está indicado entre as linhas 4 e 10. Este processo iterativo é repetido por *Iter* vezes, como indica a linha 3. O retorno deste procedimento é s^* e a LRC.

As Buscas Locais

O algoritmo *IBLITRC1* aplica a busca local Inversão Individual na solução inicial e nas soluções perturbadas. A cada t iterações, o algoritmo também aplica a busca local Reconexão por Caminhos. No final, o algoritmo aplica a busca local Troca entre Pares, no conjunto ELITE gerado.

O algoritmo *IBLITRC2* aplica a busca local Troca entre Pares na solução inicial e nas soluções perturbadas. A cada t iterações, o algoritmo também aplica a busca local Reconexão por Caminhos. No final, o algoritmo aplica a busca local Inversão individual, no conjunto ELITE gerado.

A Perturbação

O objetivo da perturbação é alterar o número de clusters pais da solução corrente. Dado uma solução inicial, esta pode ter um número de clusters pais menor ou maior que o ideal. Portanto pode ser necessário aumentar ou diminuir este número. A pertubação 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 obtida 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 2 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. O procedimento Perturbar é descrito abaixo:

Perturbar(s , l , l_i , l_s , flag) – Dado uma solução qualquer s, a perturbação aumenta ou diminui o número de clusters pais da solução em n unidades, dependendo do valor do flag (flag = n) ; O valor do flag pode ser positivo ou negativo. Se for positivo, aumenta o número de clusters pais em n unidades. Se for negativo, diminui o número de clusters pais em n unidades. Se flag = 1, aumenta em uma unidade

e se flag = -1 , diminui em uma unidade. Para isto, são escolhidos n clusters pais, aleatoriamente, do conjunto de clusters filhos. O numero de clusters pais na solução, após a perturbação, varia entre o menor valor da LRC (l_i) e o maior valor da LRC (l_s), ou seja , $l_i \leq n + l \leq l_s$, onde l é o número de clusters pais da solução antes da perturbação.

O Critério de Aceitação

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

Os pseudocódigos dos Algoritmos

A perturbação aumenta ou diminui o número de clusters pais dentro dos limites definidos pela LRC. Porém, quando a busca Reconexão de Caminhos é realizada, esta pode encontrar uma solução melhor, fora dos limites da LRC. Quando isto ocorre, os limites são atualizados e incluem este novo valor.

Os pseudocódigo do algoritmo *IBLITRC2* é mostrado na figura 3.16. O algoritmo utiliza os seguintes parâmetros de entrada: o conjunto de pontos X , os valores de u , v *e* *Iter* utilizados no procedimento Gerar Solução Inicial e o valor t, que representa a periodicidade que o algoritmo executa a busca local *Reconexão por Caminhos*.

Na linhas 1 é gerado uma solução inicial s^0. Na linha 2, esta solução passa pela busca local *Troca entre Pares*, gerando s^1. Na linha 3 é verificado o número de clusters pais (l) da solução s^1. Nas linhas 4, 5, 6 e 7 são definidos os limites para o número de clusters pais, através da lista LRC gerada no procedimento anterior. O conjunto I é definido e possui todos os valores inteiros entre os valores l_i e l_s. As iterações do ILS estão descritas entre as linhas 7 e 20. Primeiro a solução passa por uma perturbação e depois é realizada a busca local Troca entre Pares. A cada t iterações é realizada a busca local Reconexão por Caminhos entre a solução corrente s^3 e a melhor solução do conjunto ELITE. Isto está descrito entre as linhas 8 e 11. Nas linhas 12 e 13 são atualizados os novos limites do conjunto I. Nas linhas 15, 16 e 17 é atualizada a solução corrente. Nas linha 18 e 19 são atualizados o flag e o conjunto ELITE.

Na linha 21 é realizada a busca local Inversão Individual no conjunto ELITE e na linha 22, s^* recebe a melhor solução do conjunto ELITE. O algoritmo retorna s^*.

O psedocódigo do algoritmo IBLITRC1 é mostrado na figura 3.17. A diferença deste algoritmo para o anterior é a ordem em que as buscas locais são chamadas e estão indicados nas linhas 2, 9 e 21.

Procedimento IBLITRC2 (X, u, v, Iter, t)

---

1. s^0 = *Gerar Solução Inicial(X , u , v , Iter)*
2. s^1= *Troca entre Pares*(s^0)
3. l = *Retornar número clusters pais*(s^1)
4. l_i = *Mínimo(LRC)*
5. l_s = *Máximo(LRC)*
6. *Definir* $I = [l_i, l_s]$
7. *Enquanto* ($l_i \leq l \leq l_s$) *Faça*
8. s^2= *perturbar*(s^1, l, l_i, l_s, *flag)*
9 s^3=*Troca entre Pares* (s^2)
10. *Se (iteração mod t) = 0 Entao*
11. s^3 = *Reconexao por Caminhos*($s^3, ELITE$)
12. *l1 = Retornar número clusters pais*(s^3)
13. *Atualizar (I,l1)*
14. *Fim Se*
15. *Se* ($s^3 > s^1$) *Então*
16. $s^1 = s^3$
17. *Fim Se*
18. *Atualizar* ($flag$)
19. *Atualizar* $ELITE(s^3)$
20. *Fim Enquanto*
21. *Inversao Individual(ELITE)*
22. s^* = *retornar melhor* ($ELITE$)
23. Retornar (s^*)
24. *Fim IBLITRC2*

Figura 3.16: O pseudocódigo do IBLITRC2

3.3 Resultados Computacionais

Nesta seção são realizados alguns testes computacionais para verificar o desempenho dos algoritmos propostos. Primeiramente, são comparados os algoritmos que utilizam a mesma metaheurística : *AECBL1 e AECBL2; GBLITRC1 e GBLITRC2; IBLITRC1 e IBLITRC1.* Estas comparações vão indicar quais as versões de cada

---

Procedimento IBLITRC1 (X, u, v, Iter, t)

---

1. s^0 = *Gerar Solução Inicial (X , u , v , Iter)*
2. s^1= *Inversão Individual*(s^0)
3. l = *Retornar número clusters pais*(s^1)
4. l_i = *Mínimo(LRC)*
5. l_s = *Máximo(LRC)*
6. *Definir* $I = [l_i, l_s]$
7. *Enquanto* ($l_i \leq l \leq l_s$) *Faça*
8. s^2= *perturbar*(s^1, l, l_i, l_s, *flag)*
9 s^3= *Inversão Individual* (s^2)
10. *Se (iteração mod t) = 0 Entao*
11. s^3 = *Reconexao por Caminhos* ($s^3, ELITE$)
12. *l1 = Retornar número clusters pais*(s^3)
13. *Atualizar (I,l1)*
14. *Fim Se*
15. *Se* ($s^3 > s^1$) *Então*
16. $s^1 = s^3$;
17. *Fim Se*
18. *Atualizar* ($flag$)
18. *Atualizar* $ELITE(s^3)$
20. *Fim Enquanto*
21. *Troca entre Pares (ELITE)*
22. s^* = *retornar melhor* ($ELITE$)
23. Retornar (s^*)
24. *Fim IBLITRC1*

Figura 3.17: O pseudocódigo do IBLITRC1

metaheuística obtem os melhores resultados.

Depois, os melhores algoritmos serão comparados de várias formas diferentes. Primeiro, utilizando o mesmo tempo, depois verificando o tempo de convergência, e finalmente, fazendo distribuição Empírica de Probabilidade.

Todos os algoritmos foram implementados usando a linguagem C++ com o compilador gcc versão 4.1.2 no ambiente Linux Ubuntu 7.5. Nos testes de contagem de tempo foram utilizados computadores com processadores Intel Xeon Quad core onde cada processador tem 3.00 Ghz e com 16G de memória RAM.

Todos os algoritmos utilizam alguns parâmetros comuns, pois compartilham a Etapa de Construção. Foi utilizado o valor de *u variando entre 1 e 4.5* e o valor de v dependente do tamanho do problema. *Se* $m <= 200$ *então* $v = 0$ *e se* $m > 200$ *então* $v = 2$.

3.3.1 Comparação dos Algoritmos Evolutivos

Para comparar os algoritmos Evolutivos, é necessário definir os parâmetros. O *AECBL1 e AECBL2* utilizam os mesmos parâmetros para que as comparações entre eles sejam mais justas.

A taxa de cruzamento dos indivíduos de cada geração foi fixada como 80% do tamanho da população. O número de mutações foi fixado em 10% do tamanho da população. Portanto, as probabilidades de cruzamento P_c e mutação P_m foram fixados 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 indivíduos. O tamanho do conjunto ELITE foi fixado com cinco elementos. O número de iterações do algoritmo (G_{max})foi fixado como 50. O valor de t, que indica a periodicidade da busca local (Inversão Individual no AECBL1 e Troca entre Pares no AECBL2) foi fixado em 5 iterações. Esta busca é realizada somente nos 3 melhores elementos da população. A busca local reconexão de Caminhos é executada 4 vezes, nas iterações 18, 28 , 38, e 48. Estes valores de entrada foram alcançados após a execução de um conjunto de testes preliminares.

		AECBL1				AECBL2			
Nome	**Melhor**	**I. Silhueta**	**t(s)**	**NC**	**%**	**I. Silhueta**	**t(s)**	**NC**	**%**
Ruspini	0.7376	0.7376	0.9	4	**0.00**	0.7376	5.4	4	**0.00**
Iris	0.6862	0.6862	2.7	3	**0.00**	0.6862	5.9	3	**0.00**
Maronna	0.5745	0.5745	2.8	4	**0.00**	0.5745	4.9	4	**0.00**
200data	0.8231	0.8231	4.3	3	**0.00**	0.8231	5.2	3	**0.00**
Vowel	0.4246	0.4246	15.4	27	**0.00**	0.4174	42.3	2	**0.00**
Broken Ring	0.4995	0.4995	39.6	5	**0.00**	0.4995	267.3	5	**0.00**
100p2c1	0.7427	0.7427	1.8	2	**0.00**	0.7427	9.2	2	**0.00**
100p3c	0.7858	0.7858	2.2	3	**0.00**	0.7858	9.9	3	**0.00**
100p3c1	0.5802	0.5802	2.4	3	**0.00**	0.5802	8.2	3	**0.00**
100p5c1	0.6972	0.6958	1.6	7	-0.20	0.6972	5.7	8	**0.00**
100p7c	0.8338	0.8338	1.8	7	**0.00**	0.8338	10.7	7	**0.00**
100p7c1	0.4911	0.4911	1.9	7	**0.00**	0.4911	4.5	7	**0.00**
100p10c	0.8336	0.8336	1.8	10	**0.00**	0.8336	12.3	10	**0.00**
200p2c1	0.7642	0.7642	2.3	2	**0.00**	0.7642	11.2	2	**0.00**
200p3c1	0.6805	0.6797	3.1	3	-0.12	0.6805	7.1	3	**0.00**
200p4c	0.7725	0.7725	3.1	4	**0.00**	0.7725	11.1	4	**0.00**
200p4c1	0.7449	0.7449	3.2	4	**0.00**	0.7449	4.9	4	**0.00**
200p7c1	0.5759	0.5759	2.7	13	**0.00**	0.5741	14.4	14	-0.31
200p12c1	0.5770	0.5753	2.8	13	-0.29	0.5770	16.7	13	**0.00**
300p2c1	0.7764	0.7764	5.1	2	**0.00**	0.7758	20.6	2	-0.08
300p3c	0.7663	0.7663	7.2	3	**0.00**	0.7663	33.4	3	**0.00**
300p3c1	0.6768	0.6768	6.4	3	**0.00**	0.6768	34.5	3	**0.00**
300p4c1	0.6065	0.5910	5.7	2	-2.56	0.6065	16.1	2	**0.00**
300p6c1	0.6636	0.6636	5.4	8	**0.00**	0.6572	14.4	8	-0.96
300p13c1	0.5644	0.5644	5.3	13	**0.00**	0.5615	10.7	13	-0.51
400p3c	0.7985	0.7985	9.1	3	**0.00**	0.7985	36.1	3	**0.00**
400p4c1	0.5989	0.5989	6.2	4	**0.00**	0.5989	43.3	4	**0.00**
400p17c1	0.5138	0.5138	10.6	2	**0.00**	0.5138	45.6	2	**0.00**
500p3c	0.8249	0.8249	9.5	3	**0.00**	0.8249	65.4	3	**0.00**
500p4c1	0.6595	0.6595	8.1	5	**0.00**	0.6595	40.3	3	**0.00**
500p6c1	0.6287	0.6287	8.5	6	**0.00**	0.6287	44.8	6	**0.00**
600p3c1	0.7209	0.7209	18.3	3	**0.00**	0.7187	112.2	3	-0.31
600p15c	0.7812	0.7812	32.7	15	**0.00**	0.7812	99.7	15	**0.00**
700p4c	0.7969	0.7969	31.5	4	**0.00**	0.7969	130.7	4	**0.00**
700p15c1	0.6804	0.6804	23.4	15	**0.00**	0.6804	135.7	15	**0.00**
800p4c1	0.7021	0.7021	38.6	4	**0.00**	0.7021	326.8	4	**0.00**

Nome	**Melhor**	**I. Silhueta**	**t(s)**	**NC**	**%**	**I. Silhueta**	**t(s)**	**NC**	**%**
800p10c1	0.4681	0.4681	34.7	2	**0.00**	0.4642	234.6	10	-0.83
800p18c1	0.6914	0.6914	24.9	19	**0.00**	0.6894	120.6	19	-0.29
800p23c	0.7873	0.7873	55.4	23	**0.00**	0.7873	248.8	23	**0.00**
900p5c	0.7160	0.7160	71.2	5	**0.00**	0.7160	768.9	5	**0.00**
900p12c	0.8408	0.8408	70.8	12	**0.00**	0.8408	645.8	12	**0.00**
1000p5c1	0.6391	0.6391	71.5	5	**0.00**	0.6391	657.4	5	**0.00**
1000p6c	0.7356	0.7356	76.7	6	**0.00**	0.7356	879.7	6	**0.00**
1000p14c	0.8306	0.8306	84.7	14	**0.00**	0.8306	567.7	14	**0.00**
1000p27c1	0.5196	0.5186	112.3	25	-0.19	0.5196	896.5	29	**0.00**
1100p6c1	0.6717	0.6717	91.5	6	**0.00**	0.6717	765.4	6	**0.00**
1300p17c	0.8229	0.8229	121.3	17	**0.00**	0.8229	879.7	17	**0.00**
1500p6c	0.6941	0.6941	214.7	6	**0.00**	0.6941	1987.5	6	**0.00**
1500p6c1	0.6436	0.6436	205.6	6	**0.00**	0.6436	1876.8	6	**0.00**
1500p20c	0.8232	0.8232	243.5	20	**0.00**	0.8232	2298.9	20	**0.00**
1800p22c	0.8036	0.8036	305.1	22	**0.00**	0.8036	2768.8	22	**0.00**
2000p9c1	0.6230	0.6230	344.2	9	**0.00**	0.6229	2213.7	9	-0.02
2000p11c	0.7129	0.7129	354.7	11	**0.00**	0.7129	2342.6	11	**0.00**
Média Percentual					-0.06				-0.06

Tabela 3.1: Comparação entre os algoritmos AECBL1 e AECBL2

Os algoritmos Evolutivos foram executados 5 vezes para cada instância. Os resultados estão na tabela 3.1. Nessa tabela a coluna *Nome* contém os nomes dos instâncias. A coluna *Melhor* contém a maior média da função Índice Silhueta encontrado pelos dois algoritmos. A coluna *I. Silhueta* contém a média dos valores da função Índice Silhueta que cada algorimo encontrou. A coluna $t(s)$ contém a média dos tempos de execução de cada algoritmo e a coluna *NC* contém o número de clusters que a melhor solução de cada algoritmo encontrou. A coluna % contém a diferença percentual que a média das soluções encontradas está da melhor média encontrado pelos dois algoritmos. Se *o valor é negativo,* é por que a média das soluções encontradas está *pior* que a melhor média e, se *o valor é nulo,* significa que a média encontrada é *igual* a melhor média. Os melhores resultados estão realçados em negrito. Neste contexto, observando os resultados da tabela 3.1, verificamos que os algoritmos obtem resultados semelhantes. Isto é devido ao bom funcionamento

das Buscas Locais Inversão Individual e Troca entre Pares. Porém, a busca Troca entre Pares é muito demorada, o que acarreta um tempo de execução muito alto para o algoritmo *AECBL2*. O *AECBL1*, utilizando a busca local Inversão Individual consegue, praticamente, os mesmos resultados com um tempo de execução muito menor. Portanto, o algoritmo escolhido para futuras análises foi o AECBL1.

3.3.2 Comparação dos Algoritmos GRASP

Para comparar os algoritmos que utilizam a metaheurística *GRASP*, é necessário definir os parâmetros. O *GBLITRC1 e GBLITRC2* utilizam os mesmos parâmetros.

		GBLITRC1				GBLITRC2			
Nome	**Melhor**	**I. Silhueta**	**t(s)**	**NC**	**%**	**I. Silhueta**	**t(s)**	**NC**	**%**
Ruspini	0.7376	0.7376	0.8	4	**0.00**	0.7376	2.4	4	**0.00**
Iris	0.6862	0.6862	1.9	3	**0.00**	0.6862	3.2	3	**0.00**
Maronna	0.5745	0.5745	2.2	4	**0.00**	0.5745	4.6	4	**0.00**
200data	0.8231	0.8231	2.8	3	**0.00**	0.8231	5.6	3	**0.00**
Vowel	0.4183	0.4183	11.4	3	**0.00**	0.4174	9.1	3	-0.22
Broken Ring	0.4995	0.4995	25.2	5	**0.00**	0.4876	87.4	5	-2.38
100p2c1	0.7427	0.7427	1.2	2	**0.00**	0.7427	2.4	2	**0.00**
100p3c	0.7858	0.7858	1.8	3	**0.00**	0.7858	2.6	3	**0.00**
100p3c1	0.5802	0.5802	1.3	3	**0.00**	0.5802	2.8	3	**0.00**
100p5c1	0.6958	0.6958	1.4	8	**0.00**	0.6958	5.1	8	**0.00**
100p7c	0.8338	0.8338	1.3	7	**0.00**	0.8338	5.9	7	**0.00**
100p7c1	0.4868	0.4868	1.2	27	**0.00**	0.4738	2.5	2	-2.67
100p10c	0.8336	0.8336	1.4	10	**0.00**	0.8336	8.5	10	**0.00**
200p2c1	0.7642	0.7642	2.1	2	**0.00**	0.7642	7.6	2	**0.00**
200p3c1	0.6797	0.6797	2.5	3	**0.00**	0.6797	12.5	3	**0.00**
200p4c	0.7725	0.7725	2.4	4	**0.00**	0.7725	11.5	4	**0.00**
200p4c1	0.7449	0.7449	2.4	4	**0.00**	0.7449	13.5	4	**0.00**
200p7c1	0.5701	0.5701	2.3	8	**0.00**	0.5684	10.2	8	-0.30
200p12c1	0.5705	0.5695	2.3	13	-0.18	0.5705	12.2	8	**0.00**
300p2c1	0.7764	0.7764	4.1	2	**0.00**	0.7764	8.9	2	**0.00**
300p3c	0.7663	0.7663	4.2	3	**0.00**	0.7663	9.4	3	**0.00**
300p3c1	0.6768	0.6768	4.2	3	**0.00**	0.6768	12.5	3	**0.00**
300p4c1	0.5910	0.5910	4.3	2	**0.00**	0.5910	8.4	2	**0.00**
300p6c1	0.6607	0.6607	4.3	8	**0.00**	0.6534	27.5	9	-1.10

Nome	**Melhor**	**I. Silhueta**	**t(s)**	**NC**	**%**	**I. Silhueta**	**t(s)**	**NC**	**%**
300p13c1	0.5450	0.5450	4.2	2	**0.00**	0.5450	10.6	2	**0.00**
400p3c	0.7985	0.7985	5.6	3	**0.00**	0.7985	12.5	3	**0.00**
400p4c1	0.6018	0.6018	4.3	4	**0.00**	0.6015	18.2	4	-0.05
400p17c1	0.5138	0.5138	6.2	2	**0.00**	0.5138	18.5	2	**0.00**
500p3c	0.8249	0.8249	6.8	3	**0.00**	0.8249	17.2	3	**0.00**
500p4c1	0.6597	0.6597	6.7	3	**0.00**	0.6597	15.8	3	**0.00**
500p6c1	0.6287	0.6287	6.6	6	**0.00**	0.6281	27.2	6	-0.10
600p3c1	0.7209	0.7209	9.3	3	**0.00**	0.7209	17.2	3	**0.00**
600p15c	0.7812	0.7812	15.2	15	**0.00**	0.7812	91.6	15	**0.00**
700p4c	0.7969	0.7969	36.4	4	**0.00**	0.7969	62.7	4	**0.00**
700p15c1	0.6804	0.6804	21.8	15	**0.00**	0.6777	125.4	17	-0.40
800p4c1	0.7033	0.7033	26.8	4	**0.00**	0.7033	78.5	4	**0.00**
800p10c1	0.4681	0.4681	28.7	2	**0.00**	0.4681	36.7	2	**0.00**
800p18c1	0.6914	0.6914	19.2	19	**0.00**	0.6904	142.2	19	-0.14
800p23c	0.7873	0.7873	27.7	23	**0.00**	0.7549	192.5	27	-4.12
900p5c	0.7160	0.7160	33.2	5	**0.00**	0.7160	94.3	5	**0.00**
900p12c	0.8408	0.8408	47.9	12	**0.00**	0.8408	104.7	12	**0.00**
1000p5c1	0.6390	0.6390	55.4	5	**0.00**	0.6390	148.5	5	**0.00**
1000p6c	0.7356	0.7356	47.4	6	**0.00**	0.7356	179.2	6	**0.00**
1000p14c	0.8306	0.8306	63.2	14	**0.00**	0.7989	406.6	14	-3.82
1000p27c1	0.5188	0.5161	74.1	24	-0.52	0.5188	473.2	26	**0.00**
1100p6c1	0.6704	0.6704	71.3	6	**0.00**	0.6704	227.8	6	**0.00**
1300p17c	0.8229	0.8229	89.5	17	**0.00**	0.8179	702.4	17	-0.61
1500p6c	0.6941	0.6941	124.4	6	**0.00**	0.6941	320.2	6	**0.00**
1500p6c1	0.6436	0.6436	123.6	6	**0.00**	0.6436	334.2	6	**0.00**
1500p20c	0.7914	0.7914	146.3	22	**0.00**	0.7884	1148.4	20	-0.38
1800p22c	0.8036	0.8036	218.8	22	**0.00**	0.8036	1309.6	22	**0.00**
2000p9c1	0.6230	0.6230	204.8	9	**0.00**	0.6229	982.4	9	**0.00**
2000p11c	0.7129	0.7129	217.3	11	**0.00**	0.7043	1577.5	11	-1.21
Média Percentual					-0.01				-0.33

Tabela 3.2: Comparação entre os algoritmos GBLITRC1 e GBLITRC2

O tamanho da LRC é de 7 elementos. O número de iterações G_{max} realizadas é de 35. O valor de MaxIter é igual a 5. A busca local reconexão por Caminhos é

executada 4 vezes, nas iterações 15 , 20 , 25 e 30. O tamanho do conjunto *ELITE* foi fixado com cinco elementos. De modo similar ao ocorrido nos AEs, a escolha destes parâmetros foi baseada em testes preliminares.

Os algoritmos GRASP foram executados 5 vezes para cada instância. Os resultados estão na tabela 3.2. Nessa tabela a coluna *Nome* contém os nomes dos instâncias. A coluna *Melhor* contém a maior média da função Índice Silhueta encontrado pelos dois algoritmos. A coluna *I. Silhueta* contém a média dos valores da função Índice Silhueta que cada algorimo encontrou. A coluna $t(s)$ contém a média dos tempos de execução de cada algoritmo e a coluna *NC* contém o número de clusters que a melhor solução de cada algoritmo encontrou. A coluna % contém a diferença percentual que a média das soluções encontradas está da melhor média encontrado pelos dois algoritmos. Se o *valor é negativo*, é por que a média das soluções encontradas está *pior* que a melhor média e, se *o valor é nulo,* significa que a média encontrada é *igual* a melhor média. Os melhores resultados estão realçados em negrito.

As soluções obtidas pelos dois algoritmos dependem dos valores da LRC. É necessario que a busca local procure soluções com números de clusters pais próximos (mas não necessariamente iguais) a estes valores. Isto acontece com a busca local Inversão individual, utilizada pelo *GBLITRC1.* Além disso, esta busca é rápida e com isso, o tempo de execução total do algoritmo é pequeno. A busca local Troca entre Pares (utilizada pelo *GBLITRC2*) só procura soluções com o número de clusters pais identicos aos valores da LRC. Quando a LRC não encontra o número ideal de clusters , o algoritmo *GBLITRC2* não funciona bem. Além disso, como a busca local Troca entre Pares é muito demorada, o tempo total do algoritmo é muito alto. O algoritmo *GBLITRC1* obtem os melhores resultados com o menor tempo de execução e portanto, foi escolhido para análises futuras.

3.3.3 Comparação dos Algoritmos ILS

Para comparar os algoritmos que utilizam a metaheurística ILS, é necessário definir os parâmetros. O *IBLITRC1 e IBLITRC2* utilizam os mesmos parâmetros.

O tamanho da LRC é de 7 elementos. O número de iterações realizadas é dependente dos valores da LRC e varia entre o menor e o maior valor encontrados.

O valor de Iter é igual a 10. A busca local reconexão por Caminhos é executada 4 vezes, nas iterações que correspondem a 50%, 65% , 75% e 85% do número total de iterações. O tamanho do conjunto *ELITE* foi fixado com cinco elementos. De modo similar aos ocorridos nos AEs e GRASPs, os valores dos parâmetros foram alcançados após a execução de testes preliminares.

		IBLITRC2				IBLITRC1			
Nome	**Melhor**	**I. Silhueta**	**t(s)**	**NC**	**%**	**I. Silhueta**	**t(s)**	**NC**	**%**
Ruspini	0.7376	0.7376	0.8	4	**0.00**	0.7376	0.7	4	**0.00**
Iris	0.6862	0.6862	3.4	3	**0.00**	0.6862	2.2	3	**0.00**
Maronna	0.5745	0.5745	1.7	2	**0.00**	0.5745	1.2	2	**0.00**
200data	0.8231	0.8231	1.9	3	**0.00**	0.8231	1.3	3	**0.00**
Vowel	0.4341	0.4341	87.3	23	**0.00**	0.4183	23.2	3	-3.64
Broken Ring	0.4994	0.4994	17.5	5	**0.00**	0.4957	14.8	5	-0.74
100p2c1	0.7427	0.7427	9.5	2	**0.00**	0.7427	8.2	2	**0.00**
100p3c	0.7858	0.7858	0.9	3	**0.00**	0.7858	0.8	3	**0.00**
100p3c1	0.5802	0.5802	9.4	3	**0.00**	0.5802	7.1	3	**0.00**
100p5c1	0.6958	0.6958	1.1	8	**0.00**	0.6958	0.9	8	**0.00**
100p7c	0.8338	0.8338	23.2	7	**0.00**	0.8338	17.8	7	**0.00**
100p7c1	0.4835	0.4835	1.1	7	**0.00**	0.4835	0.9	7	**0.00**
100p10c	0.8336	0.8336	20.3	10	**0.00**	0.8336	18.8	10	**0.00**
200p2c1	0.7642	0.7642	1.8	2	**0.00**	0.7642	1.5	2	**0.00**
200p3c1	0.6797	0.6797	1.7	3	**0.00**	0.6805	1.1	3	**0.00**
200p4c	0.7725	0.7725	1.7	4	**0.00**	0.7725	1.2	4	**0.00**
200p4c1	0.7449	0.7449	1.9	4	**0.00**	0.7449	1.1	4	**0.00**
200p7c1	0.5394	0.5394	1.5	2	**0.00**	0.5394	1.1	2	**0.00**
200p12c1	0.5693	0.5693	1.9	8	**0.00**	0.5693	1.6	8	**0.00**
300p2c1	0.7764	0.7764	4.2	2	**0.00**	0.7758	3.3	2	-0.08
300p3c	0.7663	0.7663	1.9	3	**0.00**	0.7663	1.2	3	**0.00**
300p3c1	0.6768	0.6768	2.7	3	**0.00**	0.6768	2.1	3	**0.00**
300p4c1	0.5910	0.5910	2.6	2	**0.00**	0.5910	1.9	2	**0.00**
300p6c1	0.6636	0.6636	3.1	8	**0.00**	0.6636	2.9	8	**0.00**
300p13c1	0.5441	0.5441	2.8	2	**0.00**	0.5441	2.1	2	**0.00**
400p3c	0.7985	0.7985	5.2	3	**0.00**	0.7985	4.1	3	**0.00**
400p4c1	0.5989	0.5989	2.7	4	**0.00**	0.5989	2.1	4	**0.00**
400p17c1	0.5138	0.5138	8.5	2	**0.00**	0.5138	7.9	2	**0.00**
500p3c	0.8249	0.8249	2.3	3	**0.00**	0.8249	2.1	3	**0.00**

Nome	**Melhor**	**I. Silhueta**	**t(s)**	**NC**	**%**	**I. Silhueta**	**t(s)**	**NC**	**%**
500p4c1	0.6597	0.6597	2.6	3	**0.00**	0.6597	2.1	3	**0.00**
500p6c1	0.6287	0.6287	4.2	6	**0.00**	0.6287	3.3	6	**0.00**
600p3c1	0.7209	0.7209	4.8	3	**0.00**	0.7209	4.2	3	**0.00**
600p15c	0.7812	0.7812	37.4	15	**0.00**	0.7812	33.2	15	**0.00**
700p4c	0.7969	0.7969	9.7	4	**0.00**	0.7969	8.2	4	**0.00**
700p15c1	0.6804	0.6799	15.1	15	-0.07	0.6804	10.9	15	**0.00**
800p4c1	0.6643	0.6643	9.2	4	**0.00**	0.6643	7.2	4	**0.00**
800p10c1	0.4681	0.4681	10.2	2	**0.00**	0.4681	9.1	2	**0.00**
800p18c1	0.6914	0.6914	35.9	19	**0.00**	0.6837	23.5	23	-1.11
800p23c	0.7873	0.7873	44.9	23	**0.00**	0.7873	31.9	23	**0.00**
900p5c	0.7160	0.7160	36.4	5	**0.00**	0.7160	31.2	5	**0.00**
900p12c	0.8408	0.8408	51.2	12	**0.00**	0.8408	45.2	12	**0.00**
1000p5c1	0.6391	0.6391	31.3	5	**0.00**	0.6391	17.2	5	**0.00**
1000p6c	0.7356	0.7356	28.3	6	**0.00**	0.7356	22.3	6	**0.00**
1000p14c	0.8306	0.8306	166.5	14	**0.00**	0.8306	88.7	14	**0.00**
1000p27c1	0.4769	0.4769	43.6	2	**0.00**	0.4769	40.3	2	**0.00**
1100p6c1	0.6717	0.6717	50.7	6	**0.00**	0.6717	40.4	6	**0.00**
1300p17c	0.8229	0.8229	120.7	17	**0.00**	0.8229	72.6	17	**0.00**
1500p6c	0.6941	0.6941	67.4	6	**0.00**	0.6774	55.6	5	-2.41
1500p6c1	0.6436	0.6436	70.8	6	**0.00**	0.6436	60.3	6	**0.00**
1500p20c	0.8232	0.8232	311.2	20	**0.00**	0.8232	270.6	20	**0.00**
1800p22c	0.8036	0.8036	969.3	22	**0.00**	0.8036	876.8	22	**0.00**
2000p9c1	0.6230	0.6230	325.6	9	**0.00**	0.6230	284.3	9	**0.00**
2000p11c	0.7129	0.7129	438.4	11	**0.00**	0.7129	387.3	11	**0.00**
Média Percentual					0.00				-0.15

Tabela 3.3: Comparação entre os algoritmos IBLITRC2 e IBLITRC1

Os algoritmos ILS foram executados 5 vezes para cada instância. Os resultados estão na tabela 3.3. Nessa tabela a coluna *Nome* contém os nomes dos instâncias. A coluna *Melhor* contém a maior média da função Índice Silhueta encontrado pelos dois algoritmos. A coluna *I. Silhueta* contém a média dos valores da função Índice Silhueta que cada algorimo encontrou. A coluna $t(s)$ contém a média dos tempos de execução de cada algoritmo e a coluna *NC* contém o número de clusters que a melhor

solução de cada algoritmo encontrou. A coluna % contém a diferença percentual que a média das soluções encontradas está da melhor média encontrado pelos dois algoritmos. Se *o valor é negativo,* é por que a média das soluções encontradas está *pior* que a melhor média e, se *o valor é nulo,* significa que a média encontrada é *igual* a melhor média. Os melhores resultados estão realçados em negrito.

No ILS, para cada solução perturbada é necessário uma busca local intensiva para alcançar o ótimo local.

A busca local intensiva é a Troca entre Pares, e está no algoritmo *IBLITRC2*. Este algoritmo consegue os melhores resultados, porém, como esta busca é mais demorada, o algoritmo tem um tempo de execução maior que o *IBLITRC1*.

O *IBLITRC1* utiliza a busca local Inversão Individual, que procura soluções próximas a solução perturbada, porém não é uma busca intensiva e nem sempre encontra o ótimo local. Portanto, o algoritmo escolhido para análises futuras foi o *IBLITRC2*.

3.3.4 Comparação dos Melhores Algoritmos

Uma vez que os algoritmos que utilizam a mesma metaheurística foram comparados, nesta seção serão comparados as melhores versões de cada metaheurística, a saber: *AECBL1, GBLITRC1 e IBLITRC2*.

Os algoritmos utilizam os parâmetros definidos nas comparações anteriores.

		AECBL1			GBLITRC1			IBLITRC2		
Nome	**Melhor**	**%**	**t(s)**	**NC**	**%**	**t(s)**	**NC**	**%**	**t(s)**	**NC**
Ruspini	0.7376	**0.00**	0.9	4	**0.00**	0.8	4	**0.00**	0.8	4
Iris	0.6862	**0.00**	2.7	3	**0.00**	1.9	3	**0.00**	3.4	3
Maronna	0.5745	**0.00**	2.8	4	**0.00**	2.2	4	**0.00**	1.7	4
200data	0.8231	**0.00**	4.3	3	**0.00**	2.8	3	**0.00**	1.9	3
Vowel	0.4341	-2.19	15.4	27	-3.64	11.4	3	**0.00**	87.3	23
Broken Ring	0.4995	**0.00**	39.6	5	**0.00**	25.2	5	-0.02	17.5	5
100p2c1	0.7427	**0.00**	1.8	2	**0.00**	1.2	2	**0.00**	9.5	2
100p3c	0.7858	**0.00**	2.2	3	**0.00**	1.8	3	**0.00**	0.9	3
100p3c1	0.5802	**0.00**	2.4	3	**0.00**	1.3	3	**0.00**	9.4	3
100p5c1	0.6958	**0.00**	1.6	7	**0.00**	1.4	7	**0.00**	1.1	7
100p7c	0.8338	**0.00**	1.8	7	**0.00**	1.3	7	**0.00**	23.2	7

Nome	Melhor	%	t(s)	NC	%	t(s)	NC	%	t(s)	NC
100p7c1	0.4911	**0.00**	1.9	7	-0.88	1.2	27	-1.55	1.1	7
100p10c	0.8336	**0.00**	1.8	10	**0.00**	1.4	10	**0.00**	20.3	10
200p2c1	0.7642	**0.00**	2.3	2	**0.00**	2.1	2	**0.00**	1.8	2
200p3c1	0.6797	**0.00**	3.1	3	**0.00**	2.5	3	**0.00**	1.7	3
200p4c	0.7725	**0.00**	3.1	4	**0.00**	2.4	4	**0.00**	1.7	4
200p4c1	0.7449	**0.00**	3.2	4	**0.00**	2.4	4	**0.00**	1.9	4
200p7c1	0.5759	**0.00**	2.7	13	-1.01	2.3	8	-6.34	1.5	2
200p12c1	0.5753	**0.00**	2.8	13	-1.01	2.3	8	-1.04	1.9	8
300p2c1	0.7764	**0.00**	5.1	2	**0.00**	4.1	2	**0.00**	4.2	2
300p3c	0.7663	**0.00**	7.2	3	**0.00**	4.2	3	**0.00**	1.9	3
300p3c1	0.6768	**0.00**	6.4	3	**0.00**	4.2	3	**0.00**	2.7	3
300p4c1	0.5910	**0.00**	5.7	2	**0.00**	4.3	2	**0.00**	2.6	2
300p6c1	0.6636	**0.00**	5.4	8	-0.44	4.3	8	**0.00**	3.1	8
300p13c1	0.5644	**0.00**	5.3	13	-3.44	4.2	2	-3.60	2.8	2
400p3c	0.7985	**0.00**	9.1	3	**0.00**	5.6	3	**0.00**	5.2	3
400p4c1	0.6018	-0.48	6.2	4	**0.00**	4.3	4	-0.48	2.7	4
400p17c1	0.5138	**0.00**	10.6	2	**0.00**	6.2	17	**0.00**	8.5	2
500p3c	0.8249	**0.00**	9.5	3	**0.00**	6.8	3	**0.00**	2.3	3
500p4c1	0.6597	-0.03	8.1	5	**0.00**	6.7	5	**0.00**	2.6	5
500p6c1	0.6287	**0.00**	8.5	6	**0.00**	6.6	6	**0.00**	4.2	6
600p3c1	0.7209	**0.00**	18.3	3	**0.00**	9.3	3	**0.00**	4.8	3
600p15c	0.7812	**0.00**	32.7	15	**0.00**	15.2	15	**0.00**	37.4	15
700p4c	0.7969	**0.00**	31.5	4	**0.00**	36.4	4	**0.00**	9.7	4
700p15c1	0.6804	**0.00**	23.4	15	**0.00**	21.8	15	-0.07	15.1	15
800p4c1	0.7033	-0.17	38.6	4	**0.00**	26.8	4	-5.55	8.2	4
800p10c1	0.4681	**0.00**	34.7	2	**0.00**	28.7	2	**0.00**	10.2	2
800p18c1	0.6914	**0.00**	24.9	19	**0.00**	19.2	19	**0.00**	35.9	19
800p23c	0.7873	**0.00**	55.4	23	**0.00**	27.7	23	**0.00**	44.9	23
900p5c	0.7160	**0.00**	71.2	5	**0.00**	33.2	5	**0.00**	36.4	5
900p12c	0.8408	**0.00**	70.8	12	**0.00**	47.9	12	**0.00**	51.2	12
1000p5c1	0.6391	**0.00**	71.5	5	-0.02	55.4	5	**0.00**	31.3	5
1000p6c	0.7356	**0.00**	76.7	6	**0.00**	47.4	6	**0.00**	28.3	6
1000p14c	0.8306	**0.00**	84.7	14	**0.00**	63.2	14	**0.00**	166.5	14
1000p27c1	0.5186	**0.00**	112.3	25	-0.48	74.1	24	-8.04	43.6	2
1100p6c1	0.6717	**0.00**	91.5	6	-0.19	71.3	6	**0.00**	50.7	6
1300p17c	0.8229	**0.00**	121.3	17	**0.00**	89.5	17	**0.00**	120.7	17
1500p6c	0.6941	**0.00**	214.7	6	**0.00**	124.4	6	**0.00**	67.4	6

Nome	**Melhor**	**%**	**t(s)**	**NC**	**%**	**t(s)**	**NC**	**%**	**t(s)**	**NC**
1500p6c1	0.6436	**0.00**	205.6	6	**0.00**	123.6	6	**0.00**	70.8	6
1500p20c	0.8232	**0.00**	243.5	20	-3.86	146.3	22	**0.00**	311.2	20
1800p22c	0.8036	**0.00**	305.1	22	**0.00**	218.8	22	**0.00**	969.3	22
2000p9c1	0.6230	**0.00**	344.2	9	**0.00**	204.8	9	**0.00**	325.6	9
2000p11c	0.7129	**0.00**	354.7	11	**0.00**	217.3	11	**0.00**	438.4	11
Média Percentual		-0.05			-0.28			-0.50		

Tabela 3.4: Comparação entre os algoritmos AECBL1, GBLITRC1 e IBLITRC2

Os melhores algoritmos foram executados 5 vezes para cada instância. Os resultados estão na tabela 3.4. Nessa tabela a coluna *Nome* contém os nomes dos instâncias. A coluna *Melhor* contém a maior média da função Índice Silhueta encontrado pelos três algoritmos. A coluna *%* contém a diferença percentual que a média das soluções encontradas está da melhor média encontrado pelos três algoritmos. Se *o valor é negativo,* é por que a média das soluções encontradas está *pior* que a melhor média e, se *o valor é nulo,* significa que a média encontrada é *igual* a melhor média. Os melhores resultados estão realçados em negrito. A coluna $t(s)$ contém a média dos tempos de execução de cada algoritmo e a coluna *NC* contém o número de clusters que a melhor solução de cada algoritmo encontrou.

Os algoritmos *AECBL1, GBLITRC1 e IBLITRC2* obtiveram valores da função Índice Silhueta muito próximos, indicando que o desempenho dos algoritmos, na média, é muito semelhante.

O algoritmo *AECBL1* obteve os melhores resultados, seguido pelo *GBLITRC1*. O resultado é devido, possivelmente pelo fato, do *AECBL1* trabalhar com um conjunto de soluções que se combinam e formam novas soluções. Além disso a busca local Inversão Individual auxilia a convergência do algoritmo, buscando novas soluções próximas as pertencentes a população. Com isso, o espaço de busca é intensamente vasculhado. O algoritmo é auxiliado pelas buscas Reconexão por caminhos e no final pela Troca entre Pares que intensificam ainda mais a procura de novas soluções. O *GBLITRC1* sempre trabalha com apenas uma solução inicial a cada iteração, e portanto, tem mais dificuldade de vasculhar o espaço de busca. O *IBLITRC2* tra-

balha com uma busca local para cada número de clusters pais. Se a busca local não encontrar o ótimo local, os resultados podem não ser muito bons.

Em relação ao tempo de processamento, o *AECBL1* utiliza um tempo maior que os outros algoritmos. Porém, como mostra a tabela 3.5, mesmo fixando como o método de parada dos algoritmos o maior tempo de execução de todas as algoritmos, acrescidos de 20% deste valor, o *AECBL1* continua obtendo as melhores soluções.

		AECBL1		GBLITRC1		IBLITRC2	
Nome	**Melhor**	**%**	**NC**	**%**	**NC**	**%**	**NC**
Ruspini	0.7376	**0.00**	4	**0.00**	4	**0.00**	4
Iris	0.6862	**0.00**	3	**0.00**	3	**0.00**	3
Maronna	0.5745	**0.00**	4	**0.00**	4	**0.00**	4
200data	0.8231	**0.00**	3	**0.00**	3	**0.00**	3
Vowel	0.4341	-2.12	27	-3.64	3	**0.00**	23
Broken Ring	0.4995	**0.00**	5	**0.00**	5	-0.02	5
100p2c1	0.7427	**0.00**	2	**0.00**	2	**0.00**	2
100p3c	0.7858	**0.00**	3	**0.00**	3	**0.00**	3
100p3c1	0.5802	**0.00**	3	**0.00**	3	**0.00**	3
100p5c1	0.6958	**0.00**	7	**0.00**	7	**0.00**	7
100p7c	0.8338	**0.00**	7	**0.00**	7	**0.00**	7
100p7c1	0.4911	**0.00**	7	-0.88	27	-1.55	7
100p10c	0.8336	**0.00**	10	**0.00**	10	**0.00**	10
200p2c1	0.7642	**0.00**	2	**0.00**	2	**0.00**	2
200p3c1	0.6797	**0.00**	3	**0.00**	3	**0.00**	3
200p4c	0.7725	**0.00**	4	**0.00**	4	**0.00**	4
200p4c1	0.7449	**0.00**	4	**0.00**	4	**0.00**	4
200p7c1	0.5759	**0.00**	13	-1.01	8	-6.34	2
200p12c1	0.5753	**0.00**	13	-1.01	8	-1.04	8
300p2c1	0.7764	**0.00**	2	**0.00**	2	**0.00**	2
300p3c	0.7663	**0.00**	3	**0.00**	3	**0.00**	3
300p3c1	0.6768	**0.00**	3	**0.00**	3	**0.00**	3
300p4c1	0.5910	**0.00**	2	**0.00**	2	**0.00**	2
300p6c1	0.6636	**0.00**	8	**0.00**	8	**0.00**	8
300p13c1	0.5644	**0.00**	13	**0.00**	2	-3.60	2
400p3c	0.7985	**0.00**	3	**0.00**	3	**0.00**	3
400p4c1	0.6018	-0.48	4	**0.00**	4	-0.48	4
400p17c1	0.5138	**0.00**	2	**0.00**	2	**0.00**	2

Nome	**Melhor**	**%**	**NC**	**%**	**NC**	**%**	**NC**
500p3c	0.8249	**0.00**	3	**0.00**	3	**0.00**	3
500p4c1	0.6597	-0.03	5	**0.00**	5	**0.00**	5
500p6c1	0.6289	**0.00**	6	**0.00**	6	**0.00**	6
600p3c1	0.7209	**0.00**	3	**0.00**	3	**0.00**	3
600p15c	0.7812	**0.00**	15	**0.00**	15	**0.00**	15
700p4c	0.7969	**0.00**	4	**0.00**	4	**0.00**	4
700p15c1	0.6804	**0.00**	15	**0.00**	15	**0.00**	15
800p4c1	0.7033	-0.17	4	**0.00**	4	-0.17	4
800p10c1	0.4681	**0.00**	2	**0.00**	2	**0.00**	2
800p18c1	0.6914	**0.00**	19	**0.00**	19	**0.00**	19
800p23c	0.7873	**0.00**	23	**0.00**	23	**0.00**	23
900p5c	0.7160	**0.00**	5	**0.00**	5	**0.00**	5
900p12c	0.8408	**0.00**	12	**0.00**	12	**0.00**	12
1000p5c1	0.6391	**0.00**	5	**0.00**	5	**0.00**	5
1000p6c	0.7356	**0.00**	6	**0.00**	6	**0.00**	6
1000p14c	0.8306	**0.00**	14	**0.00**	14	**0.00**	14
1000p27c1	0.5186	**0.00**	25	-0.48	24	-8.04	2
1100p6c1	0.6717	**0.00**	6	**0.00**	6	**0.00**	6
1300p17c	0.8229	**0.00**	17	**0.00**	17	**0.00**	17
1500p6c	0.6941	**0.00**	6	**0.00**	6	**0.00**	6
1500p6c1	0.6436	**0.00**	6	**0.00**	6	**0.00**	6
1500p20c	0.8232	**0.00**	20	-3.86	22	**0.00**	20
1800p22c	0.8036	**0.00**	22	**0.00**	22	**0.00**	22
2000p9c1	0.6230	**0.00**	9	**0.00**	9	**0.00**	9
2000p11c	0.7129	**0.00**	11	**0.00**	11	**0.00**	11
Média Percentual		-0.05		-0.21		-0.40	

Tabela 3.5: Comparação entre os algoritmos AECBL1, GBLITRC1 e IBLITRC2 com o mesmo tempo

A próxima comparação determina a média dos tempos que os algoritmos utilizam para alcançar um determinado valor da função Índice Silhueta, denominado de ALVO. O ALVO utilizado é a menor média da função Índice Silhueta que os algoritmos encontram. Isto é mostrado na tabela 3.6.

Todos os algoritmos possuem um procedimento para gerar a solução inicial ou

as soluções iniciais. O procedimento do *AECBL1* é mais rápido, pois o algoritmo não precisa de alguma estimativa do número ideal de clusters. O *GBLITRC1 e IBLITRC2* precisam estimar o número ideal de clusters ou valores próximos a este. Por isso, utilizam a LRC, que precisa de mais tempo para ser gerada. O *IBLITRC2* possui um procedimento inicial mais demorado que o *GBLITRC1*.

Portanto, quando o ALVO é encontrado no procedimento que gera a solução inicial, o *AECBL1* é mais rápido, seguido pelo *GBLITRC1*. Porém, considerando a média dos tempos de todos os problemas para encontrar o ALVO, o *GBLITRC1* é mais rápido, seguido pelo *IBLITRC2*.

		AECBL1	*GBLITRC1*	*IBLITRC2*
Nome	**Alvo**	**t(s)**	**t(s)**	**t(s)**
Ruspini	0.7376	0.1	0.2	0.2
Iris	0.6862	0.4	0.5	0.6
Maronna	0.5745	0.4	0.5	0.6
200data	0.8231	1.2	0.4	0.5
Vowel	0.4183	7.3	9.8	8.5
Broken Ring	0.4995	39.2	24.5	46.3
100p2c1	0.7427	0.2	0.3	0.4
100p3c	0.7858	0.2	0.3	0.4
100p3c1	0.5802	0.2	0.3	0.4
100p5c1	0.6958	0.2	0.3	0.4
100p7c	0.8338	0.3	0.8	0.9
100p7c1	0.4835	0.4	0.9	0.6
100p10c	0.8336	0.2	0.3	0.4
200p2c1	0.7642	2.4	0.4	0.5
200p3c1	0.6797	3.5	1.2	0.7
200p4c	0.7725	0.3	0.4	0.7
200p4c1	0.7449	3.1	0.5	0.7
200p7c1	0.5394	0.3	0.4	0.5
200p12c1	0.5693	0.7	1.2	0.8
300p2c1	0.7764	4.8	0.9	0.8
300p3c	0.7663	0.5	0.6	0.7
300p3c1	0.6768	6.2	3.8	0.8
300p4c1	0.5910	5.3	3.9	0.8
300p6c1	0.6607	0.7	4.1	2.5

Nome	**Alvo**	**t(s)**	**t(s)**	**t(s)**
300p13c1	0.5441	0.3	0.4	0.5
400p3c	0.7985	0.4	0.8	2.7
400p4c1	0.5989	0.3	0.8	0.9
400p17c1	0.5138	0.4	0.7	0.9
500p3c	0.8249	0.4	0.8	0.9
500p4c1	0.6595	0.7	2.9	2.3
500p6c1	0.6287	0.8	6.1	2.3
600p3c1	0.7209	17.6	6.4	2.4
600p15c	0.7812	32.1	14.8	36.9
700p4c	0.7969	3.2	7.8	6.4
700p15c1	0.6799	4.7	20.5	14.2
800p4c1	0.7021	8.3	24.6	7.1
800p10c1	0.4681	2.8	6.5	5.3
800p18c1	0.6914	23.6	18.2	6.4
800p23c	0.7873	21.5	25.1	35.3
900p5c	0.7160	5.4	7.3	11.7
900p12c	0.8408	32.3	44.2	17.2
1000p5c1	0.6390	67.2	52.1	15.6
1000p6c	0.7356	55.2	10.6	18.8
1000p14c	0.8306	64.7	38.2	150.2
1000p27c1	0.4769	6.1	9.5	12.3
1100p6c1	0.6717	11.3	10.6	25.9
1300p17c	0.8229	14.2	89.7	73.4
1500p6c	0.6941	201.4	22.5	54.2
1500p6c1	0.6436	171.3	111.3	60.2
1500p20c	0.8232	27.3	139.4	71.3
1800p22c	0.8036	290.4	202.4	728.5
2000p9c1	0.6230	329.2	189.2	224.5
2000p11c	0.7129	332.1	201.3	103.4
Média dos tempos(s)		34.02	24.93	33.24

Tabela 3.6: Comparação entre os algoritmos AECBL1, GBLITRC1 e IBLITRC2 com um alvo definido

Em um outro experimento procurou-se verificar a velocidade de convergência com uma medida mínima de qualidade. O experimento, baseado no trabalho de

[2], considera a distribuição empírica de probabilidade de se atingir um determinado valor alvo na solução incubente, ou seja, considera-se o tempo necessário para produzir soluções com custo maior ou igual ao valor alvo. Neste trabalho foram avaliados os métodos *AECBL1, GBLITRC1 e IBLITRC2*, sendo que os tempos de processamento para cada uma das 100 execuções de cada método foram computados. Os resultados de cada método foram plotados associando-se o i-ésimo menor tempo de execução t_i com a probabilidade $p_i = (i - 1/2)/100$ gerando os pontos $z_i = (t_i, p_i)$, para i = 1,...,100.

Foram testados sete instâncias, Ruspini, Maronna , 200p7c1, 300p3c1, 800p18c1, 1000p27c1 e 1500p6c1 que possuem cardinalidades de 75 , 200, 200, 300, 800, 1000 e 1500 pontos respectivamente. Os valores alvos considerados nos experimentos deste trabalho foram definidos como as maiores médias encontradas por todos os métodos e 70% destes valores. Portanto, os valores são os seguintes: para Ruspini fixou-se os alvos em 0.7376 e 0.5163; para Maronna fixou-se os alvos em 0.5745 e 0.4021; para 200p7c1 fixou-se os alvos em 0.5759 e 0.4031; para 300p3c1 fixou-se os alvos em 0.6768 e 0.4737; para 800p18c1 fixou-se os alvos em 0.6914 e 0.4839; para 1000p27c1 fixou-se os alvos em 0.5186 e 0.3690; e para 1500p6c1 fixou-se os alvos em 0.6436 e 0.4505. Os resultados desse experimento são mostrados, respectivamente, nas figuras 3.18, 3.19, 3.20, 3.21, 3.22, 3.23, e 3.24.

Os resultados deste experimento confirmam os resultados na comparação anterior. Todos os algoritmos possuem um procedimento para gerar a solução inicial ou as soluções iniciais. O procedimento do *AECBL1* é mais rápido, pois o algoritmo não precisa de estimativa do número ideal de clusters. O *GBLITRC1 e IBLITRC2* precisam estimar o número ideal de clusters ou valores próximos a este. Por isso, utilizam a LRC, que precisa de mais tempo para ser gerada. O *IBLITRC2* gera somente uma solução e por isso, o tempo inicial de processamento é maior que o *GBLITRC1*, que gera uma solução a cada iteração. Portanto, quando o ALVO é encontrado no procedimento da geração da solução inicial, o *AECBL1* é mais rápido, seguido pelo *GBLITRC1*. Porém, quando o ALVO não é encontrado no procedimento inicial, o *IBLITRC2 e GBLITRC1* se alternam na tendência de encontrar o alvo mais rapidamente.

Normalmente, em problemas de menor cardinalidade, o *AECBL1* é mais rápido,

pois encontra o alvo no procedimento para gerar a solução inicial, seguidos pelos *GBLITRC1 e IBLITRC2*, como mostram as figuras 3.18, 3.19. Nos problemas de

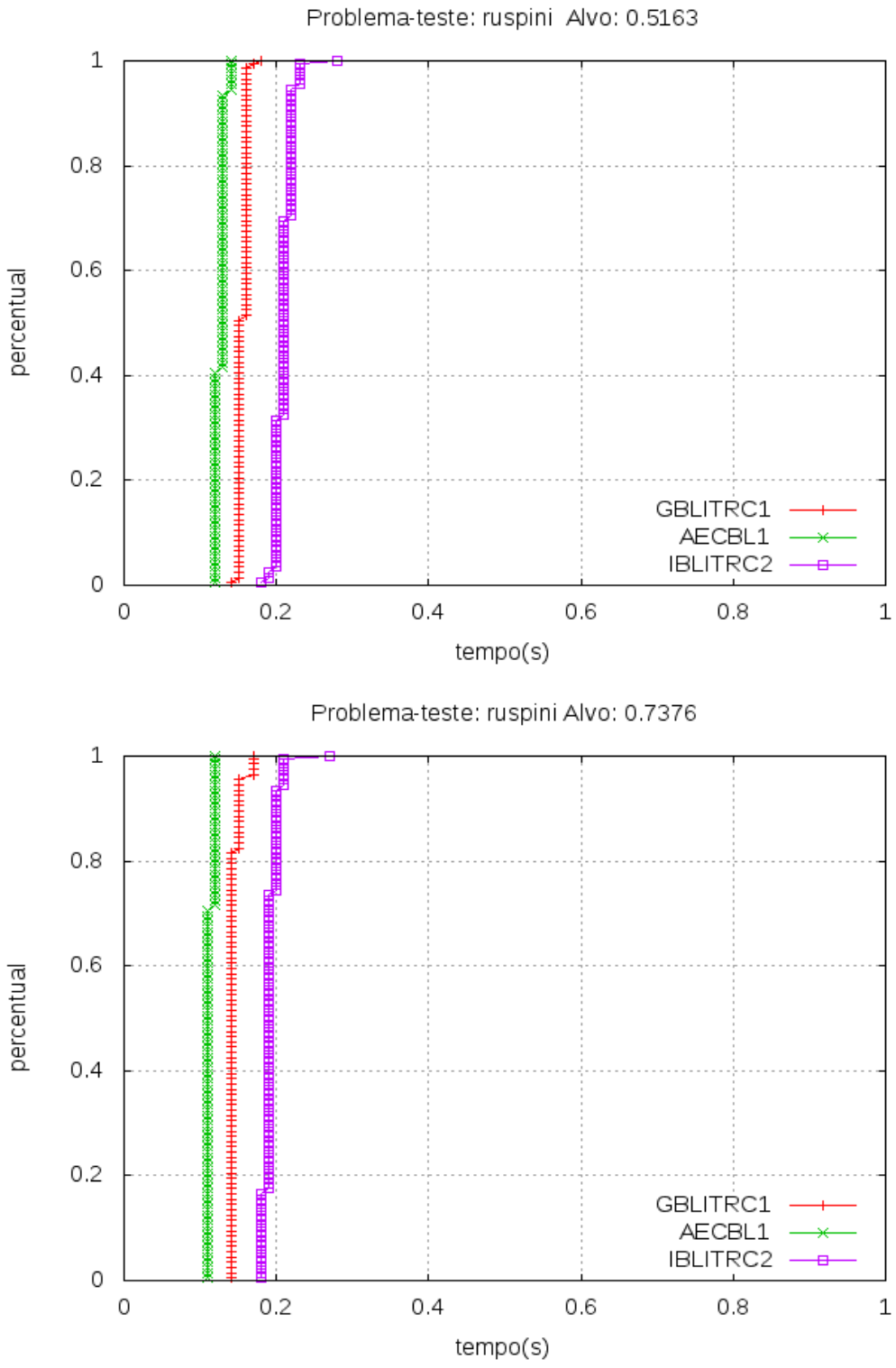

Figura 3.18: Distribuicao Empírica do problema ruspini com Alvo 0.5163 e alvo 0.7376

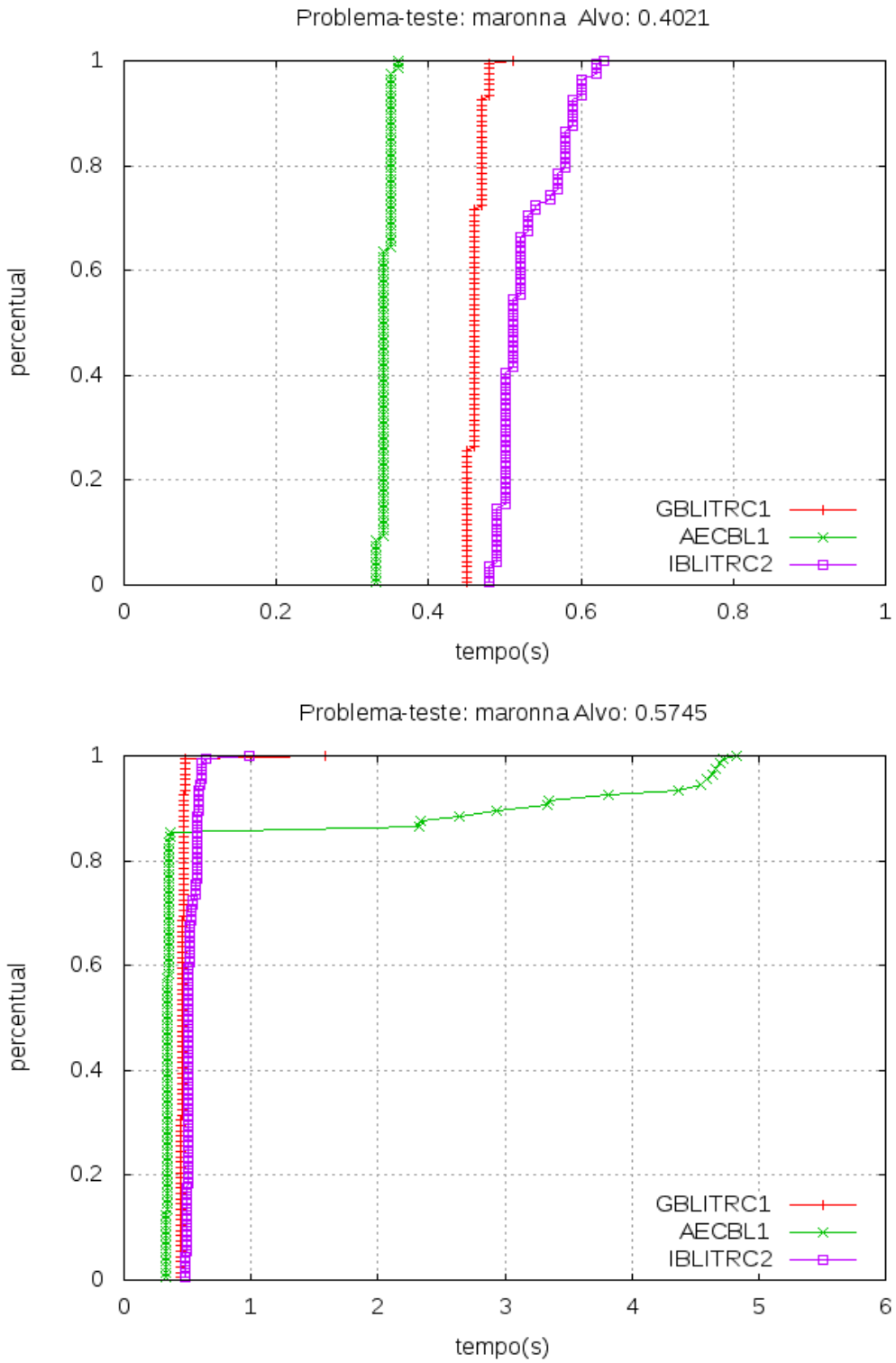

Figura 3.19: Distribuicao Empírica do problema Maronna com Alvo 0.4021 e alvo 0.5745

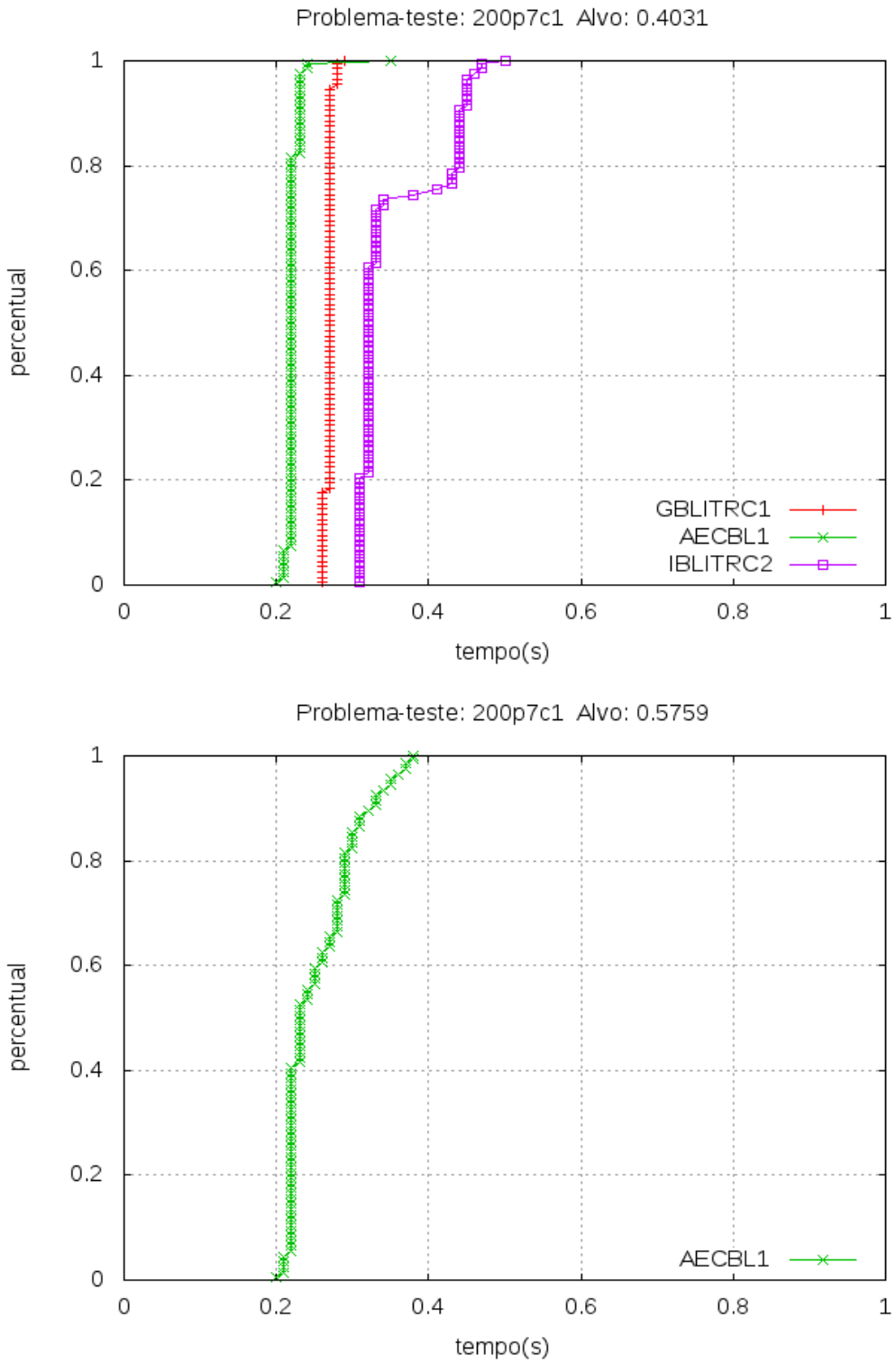

Figura 3.20: Distribuicao Empírica do problema 200p7c1 com Alvo 0.4031 e alvo 0.5759

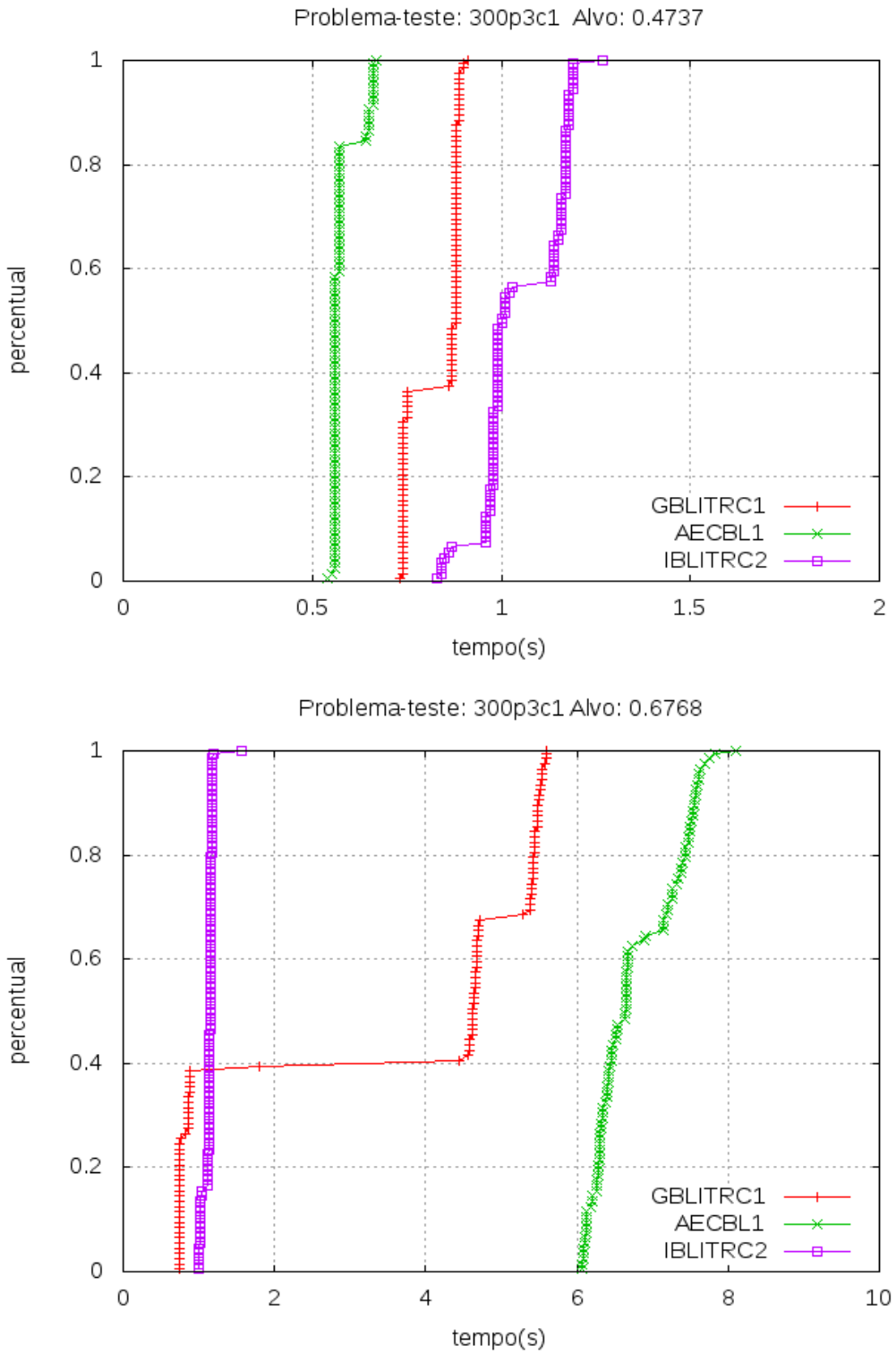

Figura 3.21: Distribuicao Empírica do problema 300p3c1 com Alvo 0.4737 e alvo 0.6768

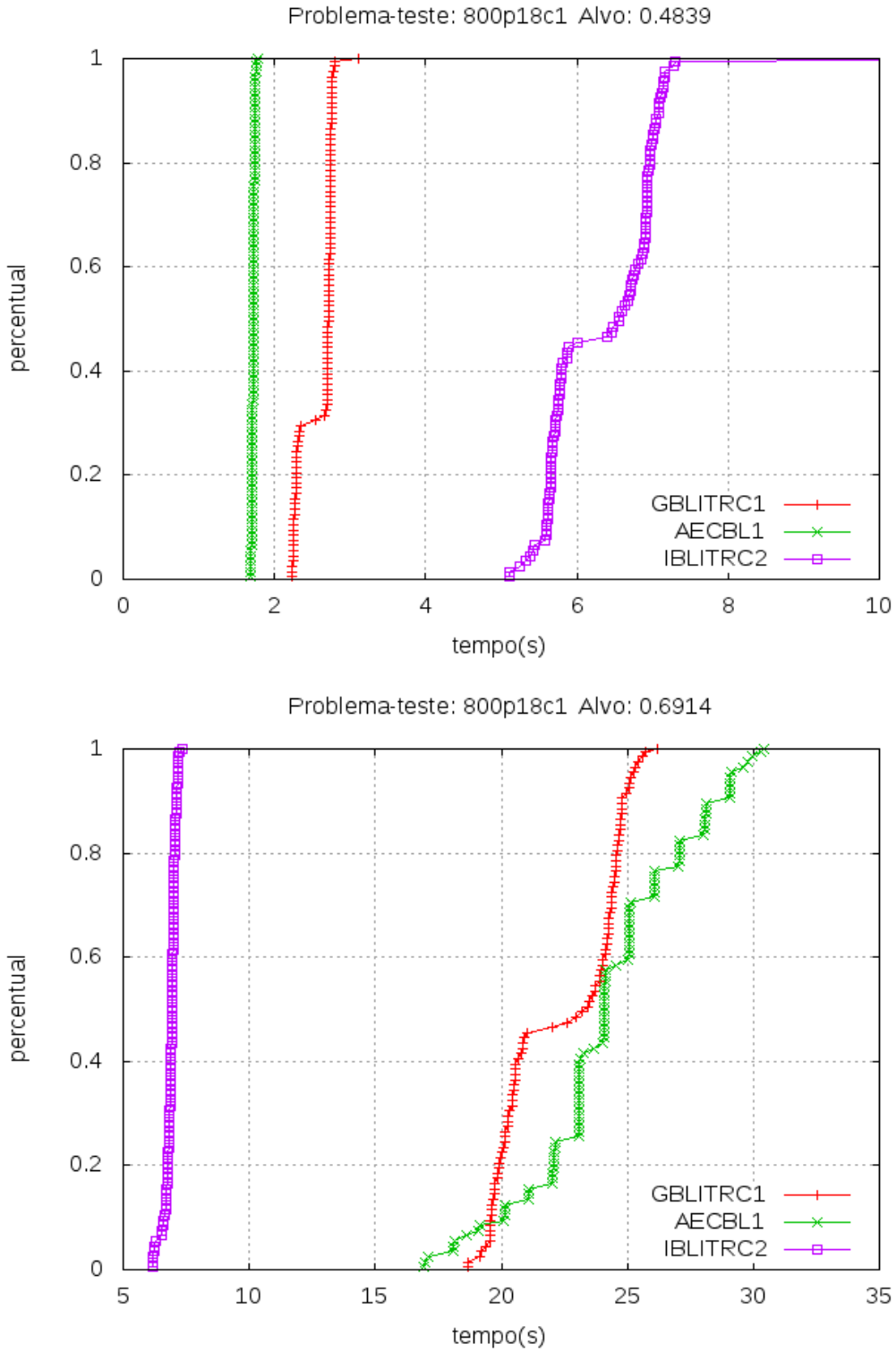

Figura 3.22: Distribuicao Empírica do problema 800p18c1 com Alvo 0.4839 e alvo 0.6914

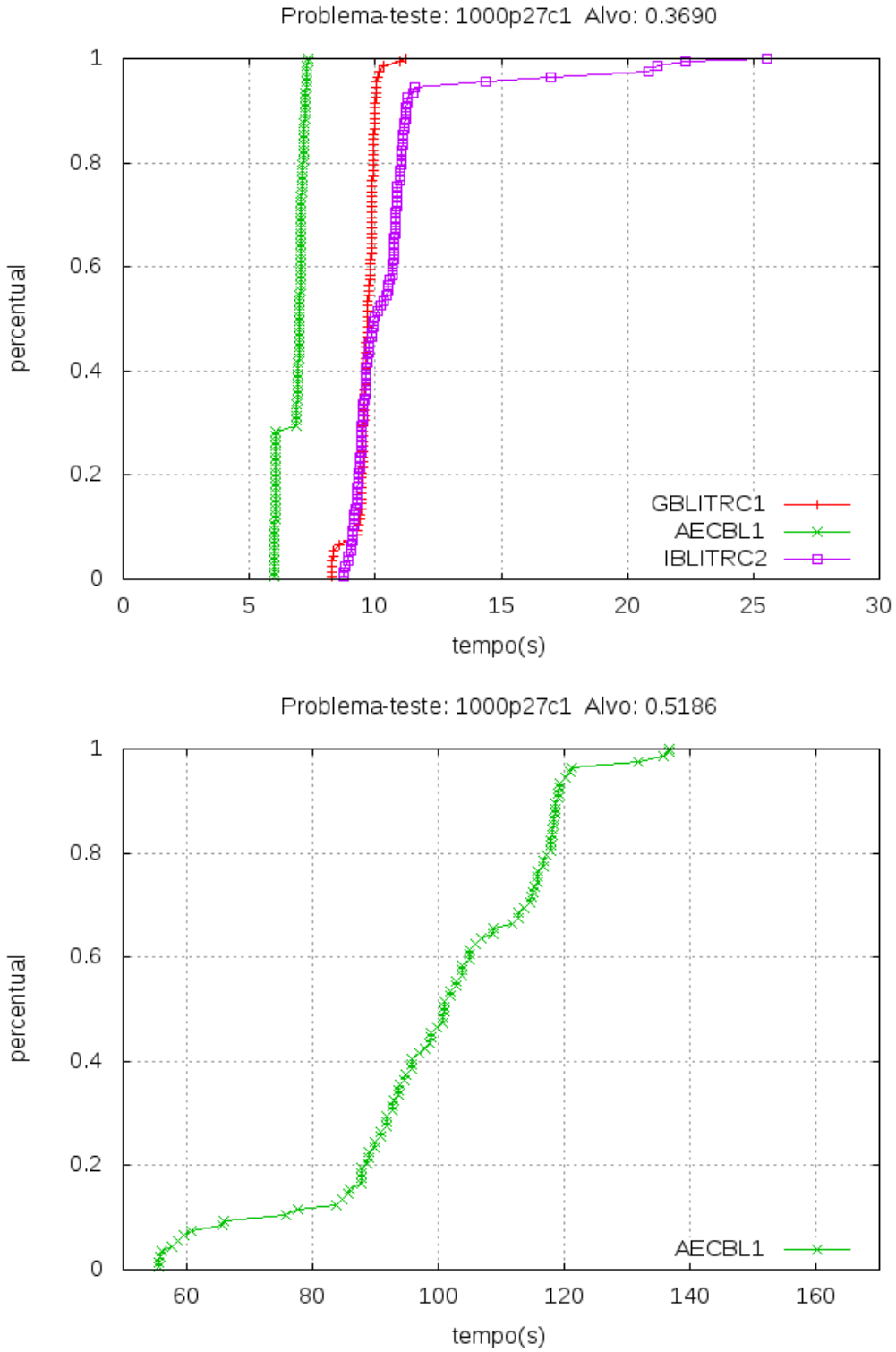

Figura 3.23: Distribuicao Empírica do problema 1000p27c1 com Alvo 0.3690 e alvo 0.5186

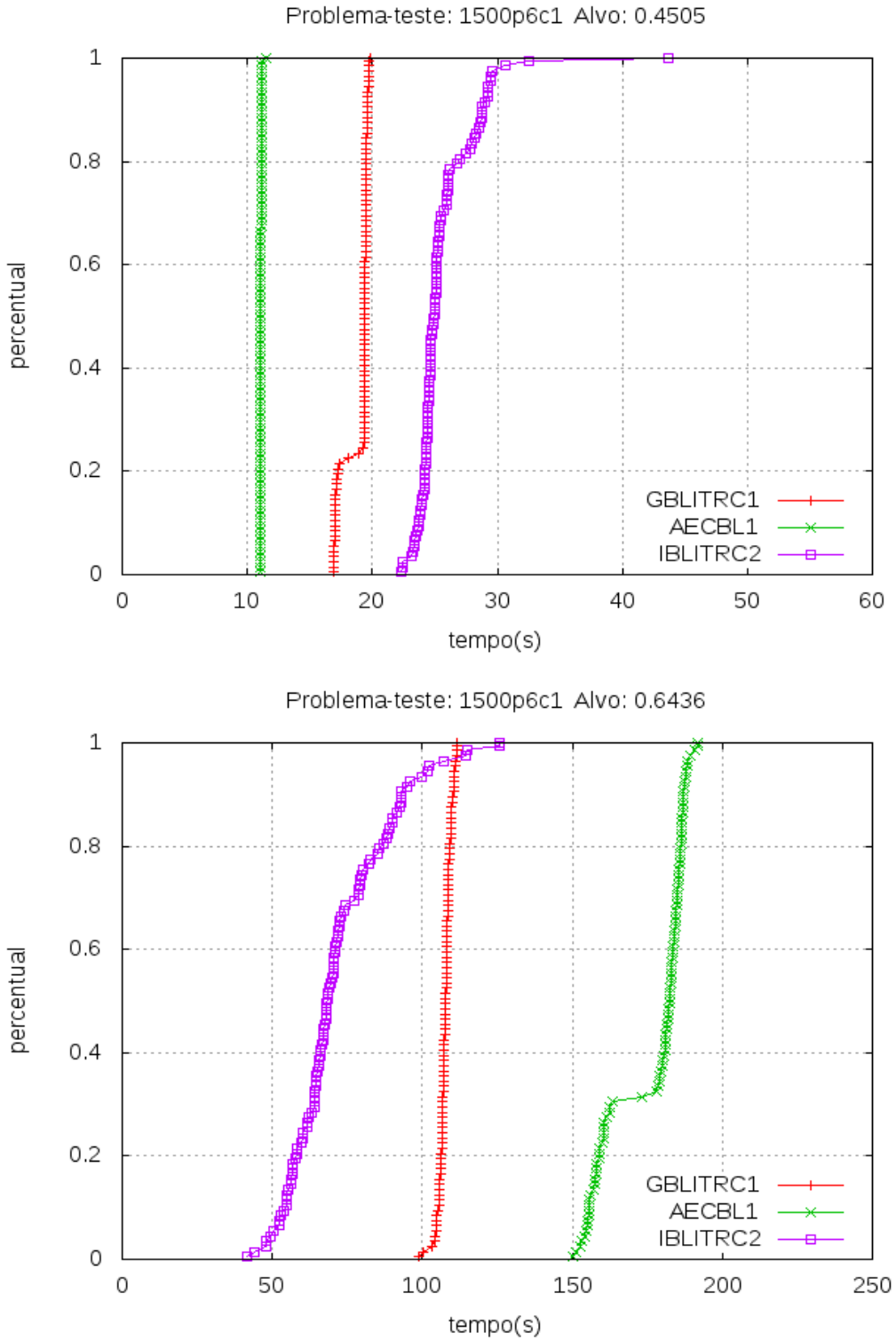

Figura 3.24: Distribuicao Empírica do problema 1500p6c1 com Alvo 0.4505 e alvo 0.6936

maior cardinalidade, com alvos mais difíceis, o *AECBL1* demora mais para alcançar o alvo, como mostram as figuras 3.21, 3.22 e 3.24. O *IBLITRC2* é mais rápido, seguido pelo *GBLITRC1*.

É importante ressaltar que, nas instâncias 200p7c1 e 1000p27c1, nos alvos mais difíceis, como mostram as figuras 3.20 e 3.23, somente o *AECBL1* conseguiu convergir, mostrando a sua robustez.

Capítulo 4

Métodos híbridos de Heurística com Modelo Exato para o PCA

A utilização de Programação Linear Inteira permite que alguns problemas sejam modelados (ou formulados) e resolvidos de maneira exata através da utilização de pacotes de *software*. Porém, nem todos os problemas combinatórios possuem funções lineares, o que restringe a utilização desta ferramenta em muitos problemas. Além disso, em alguns problemas do tipo NP-difícil, os *softwares* só conseguem resolver problemas de cardinalidade pequena.

Tinha-se como meta, desenvolver várias heurísticas de boa qualidade, e depois desenvolver um modelo exato para o PCA, utilizando a função Índice Silhueta, para poder verificar a qualidade das soluções obtidas. Isto não foi possível, pois a função Índice Silhueta não é facilmente linearizável. A partir daí, tentamos encontrar na literatura, algum modelo exato para o PCA, sem restrições, o que também não conseguimos. A terceira tentativa foi adaptar algum modelo exato do PC para o PCA. Também não foi possível, pois as funções utilizadas, como a função Diâmetro e a Estrela, são funções decrescentes em relação ao número de clusters.

Então, foi percebido que os modelos exatos para o PC poderiam ser aproveitados para melhorar as soluções do PCA, através de um método híbrido de heurística e modelo exato.

O objetivo deste capítulo é descrever os métodos híbridos propostos. Primeiro, são descritos os métodos exatos para PC. Depois são descritos os métodos híbridos e feitas algumas comparações para analisar a eficiência dos métodos.

4.1 Modelos Exatos para o PC

Neste trabalho, os problemas modelados como um problema de Programação Linear Inteira e resolvidos de modo exato serão denominados modelos exatos.

Na literatura, não existem modelos exatos sem restrições para o PCA. Os modelos exatos existentes possuem alguma restrição, como o número mínimo de pontos em cada cluster ou a capacidade máxima do cluster [25].

Existem dois modelos exatos para o PC. O primeiro modelo, denominado Modelo Exato Diâmetro, minimiza o maior diâmetro de todos os clusters encontrados. O segundo modelo, denominado Modelo Exato K-Medianas , minimiza a soma das distâncias dos pontos do cluster a um ponto mais ao centro deste. Os modelos são descritos a seguir.

4.1.1 O Modelo Exato Diâmetro

Este modelo foi definido por M. R. Rao em 1971 [38], e é um modelo linear inteiro 0-1. O objetivo deste modelo é minimizar a maior distância interna de todos os clusters, ou ainda, minimizar a maior diagonal de todos os clusters. Cada ponto só pode pertencer a um cluster e o número de clusters k é um dado de entrada. A grande dificuldade deste modelo é que o número de restrições cresce rapidamente, à medida que as quantidades de pontos m e de clusters k crescem. Portanto, o modelo exato Diâmetro é indicado somente para valores de k *e* m pequenos. A função deste modelo é a função Diâmetro, definida na equação (2.3). O modelo é descrito abaixo:

$$Min\ Z\ =\ D_{max} \tag{4.1}$$

$$S.A.\quad D_l \geq d_{ij}(x_{il} + x_{jl} - 1)\quad \forall i, j, l,\quad i, j = 1, \ldots, m,\quad l = 1, \ldots, k \tag{4.2}$$

$$\sum_{l=1}^{k} x_{il} = 1\ \ \forall i,\quad i = 1, \ldots, m \tag{4.3}$$

$$D_{\max} \geq D_l\ \ \forall l,\ \ l = 1, \ldots, k \tag{4.4}$$

$$x_{il} \in \{0,1\} \quad \forall i,l, \quad i = 1,...,m, \quad l = 1,...,k \tag{4.5}$$

$$D_l \geq 0 \qquad \forall l, \quad l = 1,...,k \tag{4.6}$$

onde

$$x_{ij} = \begin{cases} 1 & \text{se o ponto i pertence ao cluster j} \\ 0 & \text{caso contrario.} \end{cases}$$

$$d_{ij} = distancia\ entre\ o\ ponto\ i\ e\ o\ ponto\ j.$$

$$D_l = diametro\ do\ cluster\ l.$$

$$D_{max} = \ maior\ diametro\ de\ todos\ os\ clusters.$$

As restrições (4.2) garantem que a diagonal de cada cluster é maior ou igual a distância máxima entre dois pontos dentro do cluster. As restrições (4.3) garantem que cada ponto só pode ser alocado a um cluster e, finalmente, as restrições (4.4) garantem que D_{max} seja maior que todas as outras diagonais de todos os clusters.

4.1.2 O Modelo Exato K-Medianas

Este modelo foi definido por Hrishikesh D. Vinod em 1969 [48] e é um modelo linear inteiro 0-1.

Na descrição deste modelo é necessária a definição de *mediana* de um cluster, que é um ponto, que pertence ao cluster e que está mais próximo a todos os outros pontos do cluster. O objetivo deste modelo é selecionar k *medianas* do conjunto X, *uma mediana para cada cluster,* onde a soma das distâncias dos outros pontos do cluster a esta mediana é mínima. Cada mediana só pode pertencer a um cluster e o número de clusters k é um dado de entrada. A função deste modelo é a função Estrela, definida pela equação (2.5). O modelo é descrito abaixo.

$$Min \quad \sum_{i=1}^{m} \sum_{j=1}^{m} d_{ij} x_{ij} \tag{4.7}$$

$$S.A. \quad \sum_{j=1}^{m} x_{ij} = 1 \quad \forall i, \quad i = 1, ..., m \tag{4.8}$$

$$\sum_{j=1}^{m} x_{jj} = k \tag{4.9}$$

$$x_{ij} \leq x_{jj} \quad \forall i, j, \quad i = 1, ..., m \quad j = 1, ..., m \tag{4.10}$$

$$x_{ij} \in \{0, 1\} \quad \forall i, j, \quad i = 1, ..., m, \quad j = 1, ..., m \tag{4.11}$$

onde

$$x_{ij} = \begin{cases} 1 & \text{se o ponto i esta associado a mediana j} \\ 0 & \text{caso contrario.} \end{cases}$$

$$x_{jj} = 1, \ \textit{significa que o ponto } j \textit{ e a mediana.}$$

$$d_{ij} = \textit{distancia entre o pontos } \ i \textit{ e o ponto } j.$$

As restrições (4.8) garantem que cada ponto i seja associado a somente uma mediana j. As restrições (4.9) garantem que sejam escolhidos exatamente k medianas. Finalmente as restrições (4.10), garantem que o ponto i está associado ao ponto j, somente se o ponto j for uma mediana.

4.2 Métodos Híbridos

Uma alternativa na resolução de problemas de otimização é formular matematicamente o problema e conseguir um método exato, rápido e eficiente. Uma das formas de se conseguir isto é utilizando Programação Inteira. No problema de Clusterização Automática não é diferente. Porém, a função Índice Silhueta, não é facilmente linearizável. Além disso, não existe na literatura nenhum modelo exato, sem restrições, utilizando qualquer função, para resolver o PCA.

Existem modelos exatos para o PC. Porém, as funções utilizadas são diferentes da função Índice Silhueta. O que tentaremos, é relacionar as funções Diâmetro e Estrela com a função Índice Silhueta.

Como as funções citadas acima não resolvem o PCA, pois não conseguem encontrar o número ideal de clusters, é necessário utilizar a heurística para estimar este número. Portanto definiremos métodos híbridos, que utilizam heurística e modelos exatos. O objetivo é tentar melhorar as soluções obtidas pela heurística através de modelos exatos. A heurística escolhida foi o AECBL1, pois obteve os melhores resultados.

A função Índice Silhueta, definida nas equações (10 -15) é mostrada abaixo:

$$\underset{C}{Maximizar} \quad F(C) \quad = \quad \frac{1}{m}\sum_{i=1}^{m} s(x_i)$$

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

$$a(x_i) = \frac{1}{V-1}\sum d_{i,j} \quad \forall x_j \neq x_i \quad , \; x_j \in C_w$$

$$b(x_i) = Min \;\; d(x_i, C_t) \; , \quad C_t \neq C_w \quad , \quad C_t \in C$$

Analisando esta função, é verificado que uma das formas de tentar melhorar o seu valor é minimizando o valor de $a(x_i)$ para cada ponto de X. Este valor é mínimo quando os pontos pertencentes a cada cluster estão o mais próximo possível. Os métodos híbridos utilizam um modelo exato para minimizar as distâncias dos pontos dentro de cada cluster.

Os métodos híbridos funcionam da seguinte maneira: primeiro a heurística é executada utilizando a função Índice Silhueta e encontra um valor k para o número de clusters. Então, o modelo exato é executado, utilizando o valor k gerado anteriormente, e tenta minimizar as distâncias dos pontos dentro dos clusters. Assim uma nova solução é encontrada. Depois, é calculada a função silhueta desta nova solução. A solução com o maior valor da função Índice Silhueta é escolhida.

Para avaliar o desempenho dos métodos híbridos, testes computacionais foram realizados. Todos os algoritmos foram implementados usando compilador C++ no ambiente Linux Ubuntu 7.5. Os métodos híbridos chamam as bibliotecas do XPRESS, para a execução dos modelos exatos. Nos testes de contagem de tempo foram utilizados computadores com processadores Intel Xeon Quad core 3.00 Ghz com 16G de memória RAM.

4.2.1 Um Método Híbrido da Heurística AECBL1 com o Modelo Exato Diâmetro

O objetivo desta seção é descrever um método híbrido onde o modelo exato utilizado é o modelo Diâmetro.

Neste caso, o modelo exato Diâmetro gera clusters coesos minimizando a maior diagonal de todos os clusters. O *AECBL1* passa para o método híbrido apenas o valor de k.

O modelo exato funciona como uma busca local, que tenta melhorar a solução obtida pela heurística. Por isso, o modelo é definido como uma busca local e representado por BLD (Busca Local Diâmetro).

A tabela 4.1 mostra os resultados encontrados. Esta tabela mostra o nome da instância, o valor da função Índice Silhueta, o tempo em segundos e o número de clusters do *AECBL1*. Mostra também os dados do método híbrido *AECBL1 + BLD* como o valor da função Índice Silhueta, o tempo em segundos, o número de clusters e a variação percentual da função Índice Silhueta em relação à solução encontrada pelo *AECBL1*.

Nas instâncias comportadas (Ruspini, Iris , Maronna , 200DATA, 100p3c, 200p4c, 300p3c e 400p3c) foi observado que o *AECBL1* e o método híbrido convergiam para o mesmo valor da função Índice Silhueta.

	AECBL1			AECBL1+BLD			
Nome	**I. Silhueta**	**t(s)**	**NC**	**I. Silhueta**	**t(s)**	**NC**	**%**
Ruspini	0.7376	0.9	4	0.7376	78.5	4	0.00
Iris	0.6862	2.7	3	0.6862	128.2	3	0.00
Maronna	0.5745	2.8	4	0.5745	798.9	4	0.00
200data	0.8231	4.3	3	0.8231	344.2	3	0.00
100p2c1	0.7427	1.8	2	0.7427	17.2	2	0.00
100p3c	0.7858	2.2	3	0.7858	62.3	3	0.00
100p3c1	0.5802	2.4	3	0.5802	244.8	3	0.00
200p2c1	0.7642	2.3	2	0.7642	172.6	2	0.00
200p3c1	0.6797	3.1	3	0.6797	584.5	3	0.00
200p4c	0.7725	3.1	4	0.7725	879.3	4	0.00
200p4c1	0.7449	3.2	4	0.7449	2229.1	4	0.00
300p2c1	0.7764	5.1	2	0.7772	1073.7	2	**0.18**

300p3c	0.7663	7.2	3	0.7663	966.7	3	0.00
300p3c1	0.6768	6.4	3	0.6768	2161.4	3	0.00
300p4c1	0.5910	5.7	2	0.5910	1885.3	2	0.00
400p3c	0.7985	9.1	3	0.7985	3154.8	3	0.00

Tabela 4.1: Comparação entre os algoritmos AECBL1 e AECBL1+BLD

Quando as instâncias são não comportadas, é verificado que o método híbrido consegue melhorar o AECBL1 em apenas um caso, na instância 300p2c1.

Portanto, é percebido que o método híbrido testado melhora as solução de apenas uma instância e não é eficiente na maioria dos casos. Outro grande problema deste modelo é que apenas instâncias pequenas podem ser executadas. O tempo necessário para execução das instâncias é muito alto.

4.2.2 Um Método Híbrido da heurística AECBL1 com o Modelo Exato K-Medianas

O objetivo desta seção é descrever um método híbrido onde o modelo exato utilizado é o modelo K-Medianas. Este modelo é mais rápido que o anterior e consegue executar instâncias de cardinalidade maior.

A função Estrela, utilizada neste modelo, é definida como a soma dos pontos de cada cluster a um ponto mais ao centro deste mesmo cluster, denominado mediana. Portanto, o objetivo desta função é tornar os clusters coesos. Quanto mais coeso for cada cluster, menor é essa soma.

Este modelo pode melhorar as soluções obtidas pelo *AECBL1*, pois quanto mais coesos forem os clusters, mais próximos estão os pontos de cada cluster e menor é o valor de cada $a(x_i)$ de cada ponto.

Supondo que o *AECBL1* encontre boas soluções, é interessante que algumas informações sejam passadas para o modelo exato. Além do valor do k, é importante também passar alguns clusters encontrados. O modelo exato deve escolher as medianas pertencentes a estes clusters. Para isso, algumas restrições são incluídas no

modelo. Estas restrições obrigam o modelo exato a escolher algumas medianas pertencentes a estes clusters. Por exemplo, se o cluster passado para o modelo exato é o C_3 e este é definido contendo os seguintes pontos C_3= {1,3,5,7}, então a restrição utilizada é $x_{11} + x_{33} + x_{55} + x_{77} = 1$, ou seja, o modelo exato tem que escolher um dos quatro pontos como uma mediana do problema.

O objetivo de passar somente alguns clusters encontrados ao modelo exato, é deixar o modelo flexível, ou seja, o modelo exato utiliza informações da solução encontrada pelo *AECBL1* e tem liberdade de procurar por suas próprias informações no problema. Neste trabalho, a metade dos clusters encontrados pelo *AECBL1* é passada para o modelo exato.

A função Índice Silhueta também utiliza, no critério de avaliação, a distância entre pontos de clusters diferentes, definida como $b(x_i)$. Quando uma solução é modificada, os valores de $a(x_i)$ e $b(x_i)$ também são modificados. Portanto, o que pode ocorrer é que a mudança, além de diminuir o valor de $a(x_i)$, pode diminuir também o valor de $b(x_i)$. Se a diminuição dos valores $b(x_i)$ for maior que a diminuição dos valores $a(x_i)$, então o valor do índice Silhueta da nova solução é pior que o valor da solução anterior. Com isso, não podemos garantir que a solução gerada pelo modelo exato K-medianas melhore a função Índice Silhueta.

O modelo exato funciona como uma busca local, que tenta melhorar a solução obtida pela heurística. Por isso, o modelo é definido como uma busca local e representado por *BLM* (Busca Local K-medianas).

A tabela 4.2 mostra os resultados. Esta tabela mostra o nome da instância, o valor da função Índice Silhueta, o tempo em segundos e o número de clusters do *AECBL1.* Mostra também, os dados do método híbrido *AECBL1 + BLM* como o valor da função Índice Silhueta, o tempo em segundos, o número de clusters e a variação percentual da função Índice Silhueta em relação à solução encontrada pelo *AECBL1.*

Foi observado que na maioria das instâncias comportadas (Ruspini, Iris, Maronna, 200DATA, 100p3c, 100p7c, 100p10c, 200p4c, 300p3c, 400p3c, 500p3c, 600p15c, 700p4c, 800p23c, 900p5c, 900p12c, 1000p6c, 1000p14c, 1300p17c, 1500p6c, 1800p22c e 2000p11c), as soluções da heurística e do modelo híbrido possuem o mesmo valor da função Índice Silhueta. As instâncias comportadas possuem clus-

ters coesos e separados.

	AECBL1			AECBL1+BLM			
Nome	**I. Silhueta**	**t(s)**	**NC**	**I. Silhueta**	**t(s)**	**NC**	**%**
Ruspini	0.7376	0.9	4	0.7376	3.3	4	0.00
Iris	0.6862	2.7	3	0.6862	6.1	3	0.00
Maronna	0.5745	2.8	4	0.5745	9.5	4	0.00
200data	0.8231	4.3	3	0.8231	9.8	3	0.00
Vowel	0.4246	15.4	27	0.4246	82.6	27	0.00
Broken Ring	0.4995	39.6	5	0.4995	256.5	5	0.00
100p2c1	0.7427	1.8	2	0.7427	5.2	2	0.00
100p3c	0.7858	2.2	3	0.7858	5.6	3	0.00
100p3c1	0.5802	2.4	3	0.5802	5.8	3	0.00
100p5c1	0.6958	1.6	8	0.6958	4.3	8	0.00
100p7c	0.8338	1.8	7	0.8338	4.7	7	0.00
100p7c1	0.4911	1.9	7	0.4950	4.9	7	**0.79**
100p10c	0.8336	1.8	10	0.8336	6.3	10	0.00
200p2c1	0.7642	2.3	2	0.7642	7.1	2	0.00
200p3c1	0.6797	3.1	3	0.6797	8.7	3	0.00
200p4c	0.7725	3.1	4	0.7725	8.5	4	0.00
200p4c1	0.7449	3.2	4	0.7449	7.7	4	0.00
200p7c1	0.5759	2.7	13	0.5775	7.2	13	**0.28**
200p12c1	0.5753	2.8	13	0.5753	10.5	13	0.00
300p2c1	0.7764	5.1	2	0.7774	18.3	2	**0.13**
300p3c	0.7663	7.2	3	0.7663	19.2	3	0.00
300p3c1	0.6768	6.4	3	0.6768	23.4	3	0.00
300p4c1	0.5910	5.7	2	0.5910	21.8	2	0.00
300p6c1	0.6636	5.4	8	0.6636	20.1	8	0.00
300p13c1	0.5644	5.3	13	0.5644	29.8	13	0.00
400p3c	0.7985	9.1	3	0.7985	31.2	3	0.00
400p4c1	0.5989	6.2	4	0.6065	32.3	4	**1.27**
400p17c1	0.5138	10.6	2	0.5138	42.4	2	0.00
500p3c	0.8249	9.5	3	0.8249	63.7	3	0.00
500p4c1	0.6595	8.1	3	0.6597	62.2	3	**0.03**
500p6c1	0.6287	8.5	6	0.6317	63.3	6	**0.48**
600p3c1	0.7209	18.3	3	0.7209	100.2	3	0.00
600p15c	0.7812	32.7	15	0.7812	87.9	15	0.00
700p4c	0.7969	31.5	4	0.7969	186.5	4	0.00

Nome	**I. Silhueta**	**t(s)**	**NC**	**I. Silhueta**	**t(s)**	**NC**	**%**
700p15c1	0.6804	23.4	15	0.6917	145.3	15	**1.66**
800p4c1	0.7021	38.6	4	0.7044	266.3	4	**0.33**
800p10c1	0.4681	34.7	2	0.4777	246.8	2	**2.05**
800p18c1	0.6914	24.9	19	0.6914	210.2	19	0.00
800p23c	0.7873	55.4	23	0.7873	230.7	23	0.00
900p5c	0.7160	71.2	5	0.7160	353.3	5	0.00
900p12c	0.8408	70.8	12	0.8408	317.8	12	0.00
1000p5c1	0.6391	71.5	5	0.6431	522.6	5	**0.63**
1000p6c	0.7356	76.7	6	0.7356	445.5	6	0.00
1000p14c	0.8306	84.7	14	0.8306	440.3	14	0.00
1000p27c1	0.5186	112.3	25	0.5186	495.7	25	0.00
1100p6c1	0.6717	91.5	6	0.6733	651.2	6	**0.24**
1300p17c	0.8229	121.3	17	0.8229	970.9	17	0.00
1500p6c	0.6941	214.7	6	0.6941	1945.8	6	0.00
1500p6c1	0.6436	205.6	6	0.6458	1798.3	6	**0.34**
1500p20c	0.8232	243.5	20	0.7884	1601.5	20	0.00
1800p22c	0.8036	305.1	22	0.8036	3882.2	22	0.00
2000p9c1	0.6230	344.2	9	0.6270	5407.8	9	**0.64**
2000p11c	0.7129	354.7	11	0.7129	4640.3	11	0.00

Tabela 4.2: Comparação entre os algoritmos AECBL1 e AECBL1+BLM

Uma vez que nas instâncias comportadas, as soluções encontradas pelo *AECBL1* e o Método Híbrido obtiveram os mesmos valores para a função Índice Silhueta, as instâncias não comportadas foram testadas. Nestes problemas, é mais difícil encontrar o número ideal de clusters. Porém, é observado que, uma vez que o numero de k é fixado e é o melhor possível, o modelo exato K-Medianas tende a colocar os pontos nas melhores posições relativas a cada cluster. Isto faz que o modelo exato melhore também a função Índice Silhueta destas instâncias. Isso acontece em 13 instâncias.

4.2.3 Um Método Híbrido da Heurística AECBL1 com o Modelo Exato K-medianas Utilizando Busca Local

Agora tentaremos melhorar o resultado final do método, através de um refinamento do método anterior. O objetivo deste novo módulo é fazer uma busca local no valor de k. Esta busca é denominada *Busca Local em k (BLk).*

Foi verificado, empiricamente, através da execução de várias instâncias com valores de k diferentes, que a função Índice Silhueta cresce à medida que o valor k se aproxima de seu melhor valor e diminui à medida que k se afasta. A figura 4.1 mostra este comportamento da instância 1000p5c1.

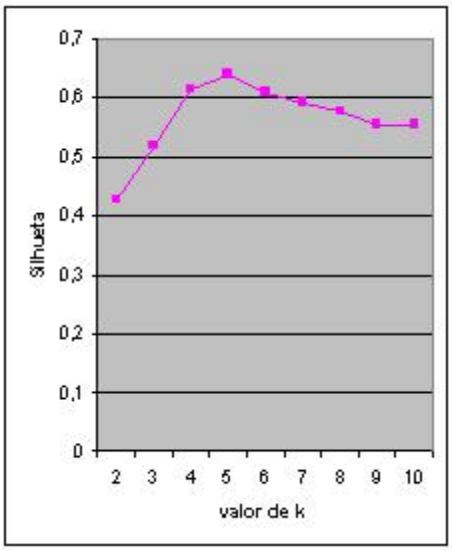

Figura 4.1: Os valores da função Índice Silhueta para a instância 1000p5c1

Portanto, supondo que o *AECBL1* não encontre o valor ideal de k, é possível fazer um busca local através do modelo exato K-medianas, para tentar encontrar o melhor valor.

O método funciona da seguinte maneira: Inicialmente o *AECBL1* é executado e gera um valor k. Depois, o método híbrido executa o modelo exato K-Medianas com os valores de k no intervalo [k-3, k+3]. Para cada k, é gerada uma nova solução e calculado o Índice Silhueta. O maior valor do Índice Silhueta entre as soluções é escolhido.

A *BLk* só é executada nos casos em que os valores da função Índice Silhueta dos algoritmos AECBL1 *e AECBL1+BLM* são diferentes.

	AECBL1		AECBL1+BLM			AECBL1+BLM+BLK			
Nome	**I. Silhueta**	**t(s)**	**I. Silhueta**	**t(s)**	**NC**	**I. Silhueta**	**t(s)**	**NC**	**%**
Ruspini	0.7376	0.9	0.7376	3.3	4	0.7376	3.3	4	0.00
Iris	0.6862	2.7	0.6862	6.1	3	0.6862	20.3	3	0.00
Maronna	0.5745	2.8	0.5745	9.5	4	0.5745	9.5	4	0.00
200data	0.8231	4.3	0.8231	9.8	3	0.8231	9.8	3	0.00
Vowel	0.4246	15.4	0.4246	82.6	27	0.4337	302.4	30	**2.14**
Broken Ring	0.4995	39.6	0.4995	256.5	5	0.4995	1467.7	5	0.00
100p2c1	0.7427	1.8	0.7427	5.2	2	0.7427	5.2	2	0.00
100p3c	0.7858	2.2	0.7858	5.6	3	0.7858	5.6	3	0.00
100p3c1	0.5802	2.4	0.5802	5.8	3	0.5802	5.8	3	0.00
100p5c1	0.6958	1.6	0.6958	4.3	8	0.6958	12.4	8	0.00
100p7c	0.8338	1.8	0.8338	4.7	7	0.8338	4.7	7	0.00
100p7c1	0.4911	1.9	0.4950	4.9	2	0.4950	12.5	2	0.00
100p10c	0.8336	1.8	0.8336	6.3	10	0.8336	6.3	10	0.00
200p2c1	0.7642	2.3	0.7642	7.1	2	0.7642	21.8	2	0.00
200p3c1	0.6797	3.1	0.6797	8.7	3	0.6797	22.6	3	0.00
200p4c	0.7725	3.1	0.7725	8.5	4	0.7725	8.5	4	0.00
200p4c1	0.7449	3.2	0.7449	7.7	4	0.7449	28.3	4	0.00
200p7c1	0.5759	2.7	0.5775	7.2	13	0.5775	27.4	13	0.00
200p12c1	0.5753	2.8	0.5753	10.5	13	0.5796	32.7	12	**0.75**
300p2c1	0.7764	5.1	0.7774	18.3	2	0.7774	58.7	2	0.00
300p3c	0.7663	7.2	0.7663	19.2	3	0.7663	19.2	3	0.00
300p3c1	0.6768	6.4	0.6768	23.4	3	0.6768	71.3	3	0.00
300p4c1	0.5910	5.7	0.5910	21.8	2	0.5910	75.5	2	0.00
300p6c1	0.6636	5.4	0.6636	20.1	9	0.6636	74.9	9	0.00
300p13c1	0.5644	5.3	0.5644	29.8	13	0.5672	73.3	12	**0.50**
400p3c	0.7985	9.1	0.7985	31.2	3	0.7985	31.2	3	0.00
400p4c1	0.5989	6.2	0.6065	32.3	4	0.6065	174.6	4	0.00
400p17c1	0.5138	10.6	0.5138	42.4	2	0.5138	221.4	2	0.00
500p3c	0.8249	9.5	0.8249	63.7	3	0.8249	63.7	3	0.00
500p4c1	0.6595	8.1	0.6597	62.2	3	0.6597	291.3	3	0.00
500p6c1	0.6287	8.5	0.6317	67.3	6	0.6317	331.9	6	0.00
600p3c1	0.7209	18.3	0.7209	100.2	3	0.7209	505.8	3	0.00
600p15c	0.7812	32.7	0.7812	87.9	15	0.7812	87.9	15	0.00
700p4c	0.7969	31.5	0.7969	186.5	4	0.7969	186.5	4	0.00
700p15c1	0.6804	23.4	0.6917	145.3	15	0.6917	916.7	15	0.00
800p4c1	0.7021	38.6	0.7044	266.3	4	0.7044	1968.3	4	0.00

Nome	**I. Silhueta**	**t(s)**	**I. Silhueta**	**t(s)**	**NC**	**I. Silhueta**	**t(s)**	**NC**	**%**
800p10c1	0.4681	34.7	0.4777	246.8	2	0.4777	1416.2	2	0.00
800p18c1	0.6914	24.9	0.6914	210.2	19	0.6991	1370.4	18	**1.11**
800p23c	0.7873	55.4	0.7873	230.7	27	0.7873	230.7	27	0.00
900p5c	0.7160	71.2	0.7160	353.3	5	0.7160	353.3	5	0.00
900p12c	0.8408	70.8	0.8408	317.8	12	0.8408	317.8	12	0.00
1000p5c1	0.6391	71.5	0.6431	522.6	5	0.6431	4654.7	5	0.00
1000p6c	0.7356	76.7	0.7356	445.5	6	0.7356	445.5	6	0.00
1000p14c	0.8306	84.7	0.8306	440.3	14	0.8306	440.3	14	0.00
1000p27c1	0.5186	112.3	0.5186	495.7	25	0.5229	3215.2	24	**0.83**
1100p6c1	0.6717	91.5	0.6733	651.2	6	0.6733	4608.8	6	0.00
1300p17c	0.8229	121.3	0.8229	970.9	17	0.8229	970.7	17	0.00
1500p6c	0.6941	214.7	0.6941	1945.8	6	0.6941	1945.8	6	0.00
1500p6c1	0.6436	205.6	0.6458	1798.3	6	0.6458	15101.5	6	0.00
1500p20c	0.8232	243.5	0.7884	1601.5	20	0.7884	1601.5	20	0.00
1800p22c	0.8036	305.1	0.8036	3882.2	22	0.8036	3882.2	22	0.00
2000p9c1	0.6230	344.2	0.6270	5407.8	9	0.6270	40876.7	9	0.00
2000p11c	0.7129	354.7	0.7129	4640.3	11	0.7129	4640.3	11	0.00

Tabela 4.3: Comparação entre os algoritmos AECBL1 e AECBL1+BLM e AECBL1+BLM+BLK

A tabela 4.3 mostra os resultados obtidos. Esta tabela mostra o nome da instância, o valor da função Índice Silhueta, o tempo em segundos e o número de clusters dos algoritmos *AECBL1, AECBL1+BLM e AECBL1+BLM+BLk.* Mostra também, a variação percentual da função Índice Silhueta do método híbrido *AECBL1+BLM+BLk*, em relação ao melhor valor encontrado pelo método *AECBL1+BLM.*

Neste caso, observamos que o método proposto *AECBL1+BLM+BLk* consegue melhorar o método híbrido *AECBL1+BLM* anterior em 5 instâncias, onde o *AECBL1* não encontrou o número ideal de clusters. A grande desvantagem deste método é o tempo, pois é necessário executar o modelo exato sete vezes, aumentando significativamente o tempo final deste método.

Capítulo 5

Comparação das Heurísticas com o Algoritmo da Literatura

O PCA é um problema recente e não possui muitos trabalhos na literatura. Os algoritmos existentes utilizam várias funções diferentes para avaliar a qualidade das partições.

Além disso, não existe na literatura, um conjunto significativo de instâncias com resultados conhecidos para a função Índice Silhueta, que possam ser utilizados para testes e comparações.

Foi encontrado na literatura, um algoritmo denominado CLUES (CLUstEring based on local Shirinking), que resolve o PCA e utiliza a função Índice Silhueta. O CLUES é um algoritmo recente que foi publicado em 2007. O código fonte foi disponibilizado e utilizado para executar as instânciass propostas. O código fonte foi utilizado sem qualquer alteração.

O objetivo deste capítulo é comparar os algoritmos propostos com o CLUES. No final, são mostradas algumas imagens das partições encontrados pelo CLUES e pelo AECBL1, cujo objetivo é visualizar as diferenças entre as soluções.

5.1 O Algoritmo CLUES

O algoritmo CLUES foi desenvolvido por Xiaogang Wang, Weiliang Qiu e Ruben H. Zamar e foi publicado na revista *Computacional Statistics & Data Analysis em 2007.[51].*

O CLUES é um algoritmo iterativo que une pontos próximos através de um procedimento, denominado procedimento de Encolhimento *(Shirinking procedure)*, que é baseado em K-vizinhos próximos. Após este procedimento, o CLUES faz um particionamento e avalia o resultado através de uma função. O CLUES possui dois módulos independentes com funções diferentes. Um módulo utiliza a função Índice Silhueta e o outro a função CH index. O CLUES comparou as soluções encontradas pelas duas funções e concluiu que a função Índice Silhueta gera as melhores soluções. Neste trabalho, é utilizado somente o módulo da função Índice Silhueta.

A estimativa do melhor valor de K e a partição final são obtidos simultaneamente. Para isso, o algoritmo começa com valores pequenos de K e os incrementa até a função ser maximizada. No final, o CLUES retorna o valor da função, o valor de g (número de clusters encontados) e a imagem da partição.

O algoritmo CLUES possui 3 etapas: O Procedimento de Encolhimento, O Procedimento de Particionamento e o Procedimento que determina o valor de K ótimo. Os procedimentos são mostrados a seguir.

5.1.1 O procedimento de Encolhimento

Neste procedimento, cada ponto vai ser encolhido para um região densa. Isto é realizado através do conceito de K-vizinhança do ponto.

Seja $Y = \{y_1, y_2, y_3, ...y_n\}$ o conjunto de todos os pontos do problema. Seja K um valor dado , ϵ uma distância dada e M o número máximo de iterações utilizadas. O procedimento é mostrado na figura 5.1.

Procedimento de Encolhimento
Passo 1: t = 1 ; $Y^{[t]} = Y$; $Y^{[t+1]} = Y$
Passo 2: $y_i^{[t+1]} = media\ y_j^{[t]},\ y_j^{[t]} \in N_K(y_i^{[t+1]}),\ i = 1, ..n$
Passo 3: $d = \max_{i=1}^{n} \max_{j=1}^{p} \|y_{ij}^{[t]} - y_{ij}^{[t+1]}\|$
Passo 4:
Passo 4.1: se $d < \epsilon$ ou $t > M$ então va para o passo 5
Passo 4.2: caso contrário, $y_{ij}^{[t]} = y_{ij}^{[t+1]}, t = t + 1$ e vá para o passo 2
Passo 5: sejam $y_i^{[t+1]}$, i = 1,..n , os grupos de pontos resultantes

Figura 5.1: O pseudocódigo do Procedimento de Encolhimento

Este procedimento é um processo iterativo que depende do valor de K. Para cada K dado, o procedimento junta os K vizinhos póximos a cada ponto. Este procedimento é executado para todos os pontos.

5.1.2 O procedimento de Particionamento

Uma vez que os pontos são agrupados, é necessário fazer o particionamento. Este particionamneto é feito utilizando uma lista com as menores distâncias entre os pontos encolhidos.

Para definir este procedimento é necessário definir dois conjuntos: S_u ó conjunto de índices dos pontos já incluídos na lista e S_r é o conjunto de índices a serem incluídos.

Inicialmente $S_u = \emptyset$ e $S_r = 1, ..., n$, ou seja, S_r contém todos os índices de todos os pontos do problema. Primeiro, é escolhido um ponto aleatóriamente, por exemplo a. Seja $\tilde{y}_a$ o conjunto de pontos gerados no procedimento anterior. Depois é encontrado o ponto mais próximo a ele, por exemplo $\tilde{y}_b$. O processo é repetido até que todos os pontos entrem na lista. Distâncias grandes entre pontos sugere o início de um novo cluster.

Neste procediemnto é definido um valor que auxilia a descoberta de novos clusters, denominado *IQR (Inter Quartile Range)*. Ele é definido como a diferençã entre as médias das maiores distâncias (aquelas com valores maiores que 75% do total) e as menores distâncias (aquelas com valores menores que 25% do total). O pseudocódigo do procedimento é descrito na figura 5.2.

5.1.3 O procedimento para encontrar o K ótimo

Os procedimentos anteriores dependem do valor de K. O ideal para este método é fazer K variar de 2 até n-1 e, para cada valor, executar o algoritmo e avaliar a partição através da função definida. Porém, isto faria o algoritmo ficar muito lento.

Para acelerar o processo, é introduzido um fator de aceleração α, $0 < \alpha < 1$. O valor inicial de K e o incremento são definidos como $\alpha * n$. Por exemplo, se $|Y| = 1000$ e $\alpha = 0.05$ então K inicial é 50, e depois 100, 150, etc.

O *CLUES* possui um módulo de refinamento para melhorar a qualidade das partições obtidas, definindo um novo intervalo de valores de K para fazer uma nova

Procedimento de Particionamento

Passo 1: $t = 1$, $S_U = 0$ *e* $S_R = 1, 2, ...n$

Passo 2: *seja* $a \in S_R$

Passo 3: *Se* $S_R \neq \emptyset$ então

Encontrar $b \in S_R$ *tal que* $b = argmind(y_a, y_b)$, *ou seja, o vizinho mais proximo de* y_a

$d_b = d(y_a, y_b)$

$S_R = S_R - b$ *e* $S_U = S_U + b$

$t = t + 1$ *e* $a = b$

Va para o passo 3

Passo 4: $R = \sum_{i=1}^{n-1} d_i/n - 1 + 1.5 * IQR$

Passo 5: *Inicialize* $C_i = 1$, $i = 1...n$, $g = 1$ *e* $t = 1$

Passo 6: *Se* $d_t > R$ *entao* $g = g + 1$

Passo 7: $C_{S_U[t+1]} = g$ *onde* $S_U[t + 1]$ *e o* $(t + 1)^{th}$ *elemento de* S_U

Passo 8: Se $t < n$ entao t = t + 1 e vá para o passo 6. Senão vá para o passo 9

Passo 9: *Retornar* $C_i, i = 1, ..., n$

Figura 5.2: O pseudocódigo do Procedimento de Particionamento

busca. Por exemplo, suponha que $|Y| = 1000$ e que o algoritmo encontre K igual a 250. Porém, suponha ainda que o K ótimo é 257. Então é feita uma busca nos valores de K entre (250-$\alpha * n$) e (250 + $\alpha * n$), ou seja, o algoritmo agora é executado com os valores de K variando entre 200 e 300.

5.2 Resultados Computacionais

Nesta seção são realizados alguns testes computacionais para verificar o desempenho dos algoritmos propostos. Primeiramente, são comparados os algoritmos híbridos com o *CLUES* e depois são comparadas as melhores heurísticas com o *CLUES.*

Todos os algoritmos propostos foram implementados usando a linguagem C++ com o compilador gcc versão 4.1.2 no ambiente Linux Ubuntu 7.5. O CLUES foi implemntado utilizando a linguagem R. Nos testes de contagem de tempo foram utilizados computadores com processadores Intel Xeon Quad core onde cada processador tem 3.00 Ghz e com 16G de memória RAM.

Os algoritmos propostos neste trabalho utilizam os parametros definidos anteriormente.

No *CLUES*, alguns parâmetros são definidos, como o $\alpha = 0.05$ e o $\epsilon = 0.0001$. Porém, não é necessário entrar com qualquer parâmetro para a execução do CLUES.

		AECBL1+BLM+BLK			AECBL1+BLM			CLUES		
Nome	**Melhor**	**%**	**t(s)**	**NC**	**%**	**t(s)**	**NC**	**%**	**t(s)**	**NC**
Ruspini	0.7376	**0.00**	3.3	4	**0.00**	3.3	4	**0.00**	0.6	4
Iris	0.6862	**0.00**	20.3	3	**0.00**	6.1	3	-18.01	1.3	3
Maronna	0.5745	**0.00**	9.5	4	**0.00**	9.5	4	**0.00**	4.2	4
200data	0.8231	**0.00**	9.8	3	**0.00**	9.8	3	-43.85	2.8	3
Vowel	0.4483	-3.26	302.4	30	-5.29	82.6	27	**0.00**	26.8	9
Broken Ring	0.4995	**0.00**	1467.7	5	**0.00**	256.5	5	**0.00**	31.8	5
100p2c1	0.7427	**0.00**	5.2	2	**0.00**	5.2	2	-29.81	3.7	5
100p3c	0.7858	**0.00**	5.6	3	**0.00**	5.6	3	**0.00**	1.6	3
100p3c1	0.5966	-2.75	5.8	3	-2.75	5.8	3	**0.00**	1.8	3
100p5c1	0.7034	-1.08	12.4	8	-1.08	4.3	8	**0.00**	3.1	6
100p7c	0.8338	**0.00**	4.7	7	**0.00**	4.7	7	**0.00**	3.7	7
100p7c1	0.5511	-10.18	12.5	7	-10.18	4.9	7	**0.00**	4.1	7
100p10c	0.8336	**0.00**	6.3	10	**0.00**	6.3	10	**0.00**	2.7	10
200p2c1	0.7642	**0.00**	21.8	2	**0.00**	7.1	2	-22.64	3.1	3
200p3c1	0.6797	**0.00**	22.6	3	**0.00**	8.7	3	-0.84	3.9	3
200p4c	0.7725	**0.00**	8.5	4	**0.00**	8.5	4	**0.00**	3.6	4
200p4c1	0.7544	-1.26	28.3	4	-1.26	7.7	4	**0.00**	3.4	4
200p7c1	0.5775	**0.00**	27.4	8	**0.00**	7.2	8	-3.84	8.8	9
200p12c1	0.5796	**0.00**	32.7	8	-0.74	10.5	8	-2.97	8.5	9
300p2c1	0.7774	**0.00**	58.7	2	**0.00**	18.3	2	-36.98	4.7	5
300p3c	0.7663	**0.00**	19.2	3	**0.00**	19.2	3	-27.97	9.2	3
300p3c1	0.6768	**0.00**	71.3	3	**0.00**	23.4	3	-12.47	4.4	4
300p4c1	0.5924	-0.24	75.5	2	-0.24	21.8	2	**0.00**	4.6	4
300p6c1	0.6636	**0.00**	74.9	9	**0.00**	20.1	9	-12.78	11.9	7
300p13c1	0.5994	-5.37	73.3	2	-5.84	29.8	2	**0.00**	10.8	9
400p3c	0.7985	**0.00**	31.2	3	**0.00**	31.2	3	**0.00**	9.6	3
400p4c1	0.6204	-2.24	174.6	4	-2.24	32.3	4	**0.00**	12.8	4
400p17c1	0.5524	-6.99	221.4	2	-6.99	42.4	2	**0.00**	20.2	15
500p3c	0.8249	**0.00**	63.7	3	**0.00**	63.7	3	**0.00**	10.1	3
500p4c1	0.6597	**0.00**	291.3	3	**0.00**	62.2	3	-21.93	9.9	3

Nome	**Melhor**	**%**	**t(s)**	**NC**	**%**	**t(s)**	**NC**	**%**	**t(s)**	**NC**
500p6c1	0.6684	-5.49	331.9	6	-5.49	67.3	6	**0.00**	16.7	6
600p3c1	0.7209	**0.00**	505.8	3	**0.00**	100.2	3	-2.51	9.6	3
600p15c	0.7812	**0.00**	87.9	15	**0.00**	87.9	15	-3.66	37.1	16
700p4c	0.7969	**0.00**	186.5	4	**0.00**	186.5	4	**0.00**	12.9	4
700p15c1	0.6917	**0.00**	916.7	15	**0.00**	145.3	15	-4.66	22.9	17
800p4c1	0.7143	-1.39	1968.3	4	-1.39	266.3	4	**0.00**	15.9	4
800p10c1	0.5071	-5.80	1416.2	2	-5.80	246.8	2	**0.00**	29.7	8
800p18c1	0.6991	**0.00**	1370.4	18	-1.10	210.2	19	-0.72	42.2	19
800p23c	0.7873	**0.00**	230.7	23	**0.00**	230.7	23	-6.17	47.7	22
900p5c	0.7160	**0.00**	353.3	5	**0.00**	353.3	5	-14.40	24.4	7
900p12c	0.8408	**0.00**	317.8	12	**0.00**	317.8	12	-5.15	63.1	10
1000p5c1	0.6431	**0.00**	4654.7	5	**0.00**	522.6	5	-13.23	38.0	7
1000p6c	0.7356	**0.00**	445.5	6	**0.00**	445.5	6	**0.00**	38.3	6
1000p14c	0.8306	**0.00**	440.3	14	**0.00**	440.3	14	-7.61	51.6	11
1000p27c1	0.5631	-7.14	3215.2	24	-7.90	495.7	24	**0.00**	67.4	14
1100p6c1	0.6847	-1.66	4608.8	6	-1.66	651.2	6	**0.00**	39.9	6
1300p17c	0.8229	**0.00**	970.7	17	**0.00**	970.9	17	-10.33	80.4	15
1500p6c	0.6941	**0.00**	1945.8	6	**0.00**	1945.8	6	-1.38	73.6	5
1500p6c1	0.6597	-2.11	15101.5	6	-2.11	1798.3	6	**0.00**	61.7	6
1500p20c	0.8232	**0.00**	1601.5	20	**0.00**	1601.5	20	-16.50	92.9	13
1800p22c	0.8036	**0.00**	3882.2	22	**0.00**	3882.2	22	-19.95	136.1	18
2000p9c1	0.6270	**0.00**	40876.7	9	**0.00**	5407.8	9	-14.61	123.2	7
2000p11c	0.7129	**0.00**	4640.3	11	**0.00**	4640.3	11	-14.97	93.6	8
Média Percentual		-1.07			-1.17			-7.27		

Tabela 5.1: Comparação entre os algoritmos AECBL1+BLM+BLK, AECBL1+BLM e CLUES

A Tabela 5.1 mostra a comparação entre o *CLUES* e os Métodos híbridos. Os algoritmos foram executados 5 vezes para cada instância. Nas duas tabelas a seguir, as colunas tem os seguintes significados : a coluna *Nome* contém os nomes das instâncias. A coluna *Melhor* contém a maior média da função Índice Silhueta encontrado pelos algoritmos. A coluna % contém a diferença percentual que a média das soluções encontradas está da melhor média encontrado pelos 3 algoritmos. Se *o valor é negativo*, é por que a média das soluções encontradas está *pior* que a me-

lhor média e, se *o valor é nulo,* significa que a média encontrada é *igual* a melhor média. Os melhores resultados estão realçados em negrito. A coluna $t(s)$ contém a média dos tempos de execução de cada algoritmo e a coluna *NC* contém o número de clusters que a melhor solução de cada algoritmo encontrou.

Nesta comparação, é obervado que os métodos híbridos obtem os melhores resultados. O *AECBL1+BLM+BLK* tem a melhor média percentual , seguido pelo *AECBL1+BLM* e pelo *CLUES*. É importante observar que a diferença entre as médias é muito grande.

Além das melhores médias, eles também alcançam os melhores valores em uma uma quantidade maior de instâncias. O *AECBL1+BLM+BLK* chega nos melhores valores em 39 instâncias, num total 53. O *AECBL1+BLM* chega nos melhores valores em 37 instâncias, enquanto *CLUES* chega em 25 instâncias. Portanto, podemos concluir que os mëtodos híbridos funcionam muito melhor que o *CLUES.*

Porém o tempo dos métodos híbridos são muito maiores que o tempo do *CLUES*. Então, o *CLUES* é comparado com as heurísticas, que possuem os tempos mais próximos. Isto é mostrado na tabela 5.2.

		AECBL1			GBLITRC1			IBLITRC1			CLUES		
Nome	**Melhor**	**%**	**t(s)**	**NC**	**%**	**t(s)**	**NC**	**%**	**t(s)**	**NC**	**%**	**t(s)**	**NC**
Ruspini	0.7376	**0.00**	0.9	4	**0.00**	0.8	4	**0.00**	0.8	4	**0.00**	0.6	4
Iris	0.6862	**0.00**	2.7	3	**0.00**	1.9	3	**0.00**	3.4	3	-18.01	1.3	3
Maronna	0.5745	**0.00**	2.8	4	**0.00**	2.2	4	**0.00**	1.7	2	**0.00**	4.2	4
200data	0.8231	**0.00**	4.3	3	**0.00**	2.8	3	**0.00**	1.9	3	-43.85	2.8	3
Vowel	0.4483	-5.29	15.4	27	-6.69	11.4	3	-3.17	87.3	23	**0.00**	26.8	9
Broken Ring	0.4995	**0.00**	39.6	5	**0.00**	25.2	5	**0.00**	17.5	5	**0.00**	31.8	5
100p2c1	0.7427	**0.00**	1.8	2	**0.00**	1.2	2	**0.00**	9.5	2	-29.81	3.7	5
100p3c	0.7858	**0.00**	2.2	3	**0.00**	1.8	3	**0.00**	0.9	3	**0.00**	1.6	3
100p3c1	0.5966	-2.75	2.4	3	-2.75	1.3	3	-2.75	9.4	3	**0.00**	1.8	3
100p5c1	0.7034	-1.08	1.6	7	-1.08	1.4	7	-1.08	1.1	7	**0.00**	3.1	6
100p7c	0.8338	**0.00**	1.8	7	**0.00**	1.3	7	**0.00**	23.2	7	**0.00**	3.7	7
100p7c1	0.5511	-10.89	1.9	7	-11.67	1.2	27	-12.27	1.1	7	**0.00**	4.1	7
100p10c	0.8336	**0.00**	1.8	10	**0.00**	1.4	10	**0.00**	20.3	10	**0.00**	2.7	10
200p2c1	0.7642	**0.00**	2.3	2	**0.00**	2.1	2	**0.00**	1.8	2	-22.64	3.1	3
200p3c1	0.6797	**0.00**	3.1	3	**0.00**	2.5	3	**0.00**	1.7	3	-0.84	3.9	3
200p4c	0.7725	**0.00**	3.1	4	**0.00**	2.4	4	**0.00**	1.7	4	**0.00**	3.6	4
200p4c1	0.7544	-1.26	3.1	4	-1.26	2.4	4	-1.26	1.9	4	**0.00**	3.4	4
200p7c1	0.5759	**0.00**	2.7	8	-1.01	2.3	14	-6.34	1s	1.5	-3.58	8.8	9
200p12c1	0.5753	**0.00**	2.8	13	-1.01	2.3	13	-1.04	1.9	8	-2.24	8.5	9
300p2c1	0.7764	**0.00**	5.1	2	**0.00**	4.1	2	**0.00**	4.2	2	-36.90	4.7	5
300p3c	0.7663	**0.00**	7.2	3	**0.00**	4.2	3	**0.00**	1.9	3	-27.97	9.2	3

Nome	Melhor	%	t(s)	NC	%	t(s)	NC	%	t(s)	NC	%	t(s)	NC
300p3c1	0.6768	**0.00**	6.4	3	**0.00**	4.2	3	**0.00**	2.7	3	-12.47	4.4	4
300p4c1	0.5924	-0.24	5.7	2	-0.24	4.3	2	-0.24	2.6	2	**0.00**	4.6	4
300p6c1	0.6636	**0.00**	5.4	8	-0.44	4.3	8	**0.00**	3.1	8	-12.78	11.9	7
300p13c1	0.5944	-5.84	5.3	13	-9.08	4.2	13	-9.23	2.8	2	**0.00**	10.8	9
400p3c	0.7985	**0.00**	9.1	3	**0.00**	5.6	3	**0.00**	5.2	3	**0.00**	9.6	3
400p4c1	0.6204	-3.47	6.2	4	-3.00	4.3	4	-3.47	2.7	4	**0.00**	12.8	4
400p17c1	0.5524	-6.99	10.6	2	-6.99	6.2	17	-6.99	8.5	2	**0.00**	20.2	15
500p3c	0.8249	**0.00**	9.5	3	**0.00**	6.8	3	**0.00**	2.3	3	**0.00**	10.1	3
500p4c1	0.6597	-0.03	8.1	3	**0.00**	6.7	5	**0.00**	2.6	3	-21.93	9.9	3
500p6c1	0.6684	-5.94	8.5	6	-5.94	6.3	6	-5.94	4.2	6	**0.00**	16.7	6
600p3c1	0.7209	**0.00**	18.3	3	**0.00**	9.3	3	**0.00**	4.8	3	-2.51	9.6	3
600p15c	0.7812	**0.00**	32.7	15	**0.00**	15.2	15	**0.00**	37.4	15	-3.66	37.1	16
700p4c	0.7969	**0.00**	31.5	4	**0.00**	36.4	4	**0.00**	9.7	4	**0.00**	12.9	4
700p15c1	0.6804	**0.00**	23.4	15	**0.00**	21.8	15	-0.07	15.1	15	-3.07	22.9	17
800p4c1	0.7143	-1.71	38.6	4	-1.54	26.8	4	-7.00	8.2	4	**0.00**	15.9	4
800p10c1	0.5071	-7.69	34.7	2	-7.69	28.7	10	-7.69	10.2	2	**0.00**	29.7	8
800p18c1	0.6941	-0.39	24.9	19	-0.39	19.2	19	-0.39	35.9	19	**0.00**	42.2	19
800p23c	0.7873	**0.00**	55.4	23	**0.00**	27.7	23	**0.00**	44.9	23	-6.17	47.7	22
900p5c	0.7160	**0.00**	71.2	5	**0.00**	33.2	5	**0.00**	39.4	5	-14.40	24.4	7
900p12c	0.8408	**0.00**	70.8	12	**0.00**	47.9	12	**0.00**	51.2	12	-5.15	63.1	10
1000p5c1	0.6391	**0.00**	71.5	5	**0.00**	55.4	5	**0.00**	31.3	5	-12.69	38.0	7
1000p6c	0.7356	**0.00**	76.7	6	**0.00**	47.4	6	**0.00**	28.3	6	**0.00**	38.3	6
1000p14c	0.8306	**0.00**	84.7	14	**0.00**	63.2	14	**0.00**	16.5	14	-7.61	51.6	11
1000p27c1	5631	-7.90	112.3	24	-8.35	74.1	25	-15.31	43.6	2	**0.00**	67.4	14
1100p6c1	0.6847	-1.90	91.5	6	-2.09	71.3	6	-1.90	50.7	6	**0.00**	39.9	6
1300p17c	0.8229	**0.00**	121.3	17	**0.00**	89.5	17	**0.00**	120.7	17	-10.33	80.4	15
1500p6c	0.6941	**0.00**	214.7	6	**0.00**	124.4	6	**0.00**	67.4	6	-1.38	73.6	5
1500p6c1	0.6597	-2.44	205.6	6	-2.44	123.6	6	-2.44	70.8	6	**0.00**	61.7	6
1500p20c	0.8232	**0.00**	243.5	20	-3.86	146.3	20	**0.00**	311.2	20	-16.50	92.9	13
1800p22c	0.8036	**0.00**	305.1	22	**0.00**	218.8	22	**0.00**	969.3	22	-19.95	136.1	18
2000p9c1	0.6230	**0.00**	344.2	9	**0.00**	204.8	9	**0.00**	325.6	9	-14.06	123.2	7
2000p11c	0.7129	**0.00**	354.7	11	**0.00**	217.3	11	**0.00**	438.4	11	-14.97	93.6	8
Média Percentual		-1.24			-1.46			-1.67			-7.07		

Tabela 5.2: Comparação entre os algoritmos AECBL1, GBLITRC1, IBLITRC1 e CLUES

As heurística possuem uma média percentual muito próxima, indicando que possuem um desempenho muito parecido. Porém quando comparado com o *CLUES,* a diferença das médias % aumenta muito, passando de -1.67 para -7.07, que é uma diferença muito grande.

Além das melhores médias, as heurísticas também alcançam os melhores valores em uma uma quantidade maior de instâncias. O *AECBL1* chega nos melhores

valores em 36 instâncias, num total 53. O *GBLITRC1* chega nos melhores valores em 33 instâncias, e o *IBLITRC1* chega nos melhores valores em 34 instâncias. O *CLUES* chega nos melhores valores em 27 instâncias.

Analisando o desempenho dos melhores métodos propostos, verificamos que o *AECBL1+BLM+BLK* conseguiu os melhores resultados, seguidos por *AECBL1+BLM, AECBL1, GBLITRC1 e IBLITRC2.* Todos os métodos apresentados neste trabalho conseguiram a média dos valores da função Índice Silhueta, executados em 53 instâncias, melhores que a média do *CLUES.*

Os algoritmos *AECBL1, GBLITRC1 e IBLITRC2* obtiveram valores da função Índice Silhueta muito próximos, indicando que o desempenho dos algoritmos, na média, é muito semelhante. O algoritmo *AECBL1* obteve os melhores resultados, seguido pelo *GBLITRC1.* O resultado é devido, possivelmente pelo fato, do *AECBL1* trabalhar com um conjunto de soluções que se combinam e formam novas soluções. Além disso a busca local Inversão Individual auxilia a convergência do algoritmo, buscando novas soluções próximas as pertencentes a população. Com isso, o espaço de busca é intensamente vasculhado. O algoritmo é auxiliado pelas buscas Reconexão por caminhos e no final pela Troca entre Pares que intensificam ainda mais a procura de novas soluções. O *GBLITRC1* sempre trabalha com apenas uma solução inicial a cada iteração, e portanto, tem mais dificuldade de vasculhar o espaço de busca. O *IBLITRC2* trabalha com uma busca local para cada número de clusters pais. Se a busca local não encontrar o ótimo local, os resultados podem não ser muito bons.

Os algoritmos *AECBL1, GBLITRC1 e IBLITRC2* trabalham com conjuntos de pontos, e estes são combinados para formarem uma solução. Esta combinação pode acarretar que alguns pontos não fiquem em suas melhores posições relativas a cada cluster. No método híbrido *AECBL1 + BLM* foi observado que, uma vez que o *AECBL1* encontra o valor de k e este é o melhor possível, o modelo exato *K-Medianas* tende a colocar os pontos, individualmente, nas melhores posições relativas a cada cluster. Isto faz que o modelo exato melhore o valor da função Índice Silhueta em algumas soluções geradas pelo *AECBL1.* Porém, quando o valor de k não é o melhor possível, o método híbrido *AECBL1+BLM+BLK* pode encontrar um valor melhor para k e com isso, melhorar as soluções geradas pelo método *AECBL1+BLM.*

Em relação ao tempo de processamento, é verificado que o algoritmo mais rápido é o CLUES seguidos por *GBLITRC1, AECBL1, IBLITRC2, AECBL1+BLM e AECBL1+BLM+BLK* . As médias dos tempos de execução de todas as intâncias são respectivamente 27.29s, 34.56s, 53.03s, 58.66s, 487.07s e 1758.48s. Os tempos de processamento dos métodos híbridos são mais altos, pois executam o modelo exato *K-medianas*, que demanda muito tempo de execução. É importante ressaltar que, em instâncias de cardinalidade pequena e média (de 100 a 800 pontos), os algoritmos *CLUES, GBLITRC1, AECBL1 e IBLITRC2* possuem tempos de processamento semelhantes, alternando o algoritmo que possui o menor tempo.

5.2.1 Imagens das Soluções

Nesta seção são mostradas algumas imagens das soluções encontradas. As imagens são do *AECBL1 e do CLUES.*

O *AECBL1* foi utilizado pois é a melhor heurística e possui um tempo de execução próximo ao *CLUES.* Primeiro, na parte de cima de cada figura, é mostrada a partição gerada pelo *CLUES.* Na parte de baixo da figura, é mostrada a partição do *AECBL1.*

O objetivo desta seção é ilustrar algumas soluções geradas pelos algoritmos. As instâncias escolhidas possuem soluções diferentes entre os algoritmos. As instâncias escolhidas foram Ruspini , Maronna , 200Data , Vowel , Broken Ring ,200p2c1 , 300p2c1 , 300p4c1, 500p4c1 , 700p15c1, 1000p27c1, 1800p22c e 2000p9c1. As figuras entre 5.3 e 5.15 mostram, respectivamente, as imagens das soluções encontradas pelos algoritmos. Cada cor diferente na imagem indica um cluster diferente.

É observado que as partições obtidas pelo *AECBL1* são mais definidas que as partições do *CLUES*, justificando os melhores valores obtidos pelo *AECBL1* para a função Índice Silhueta.

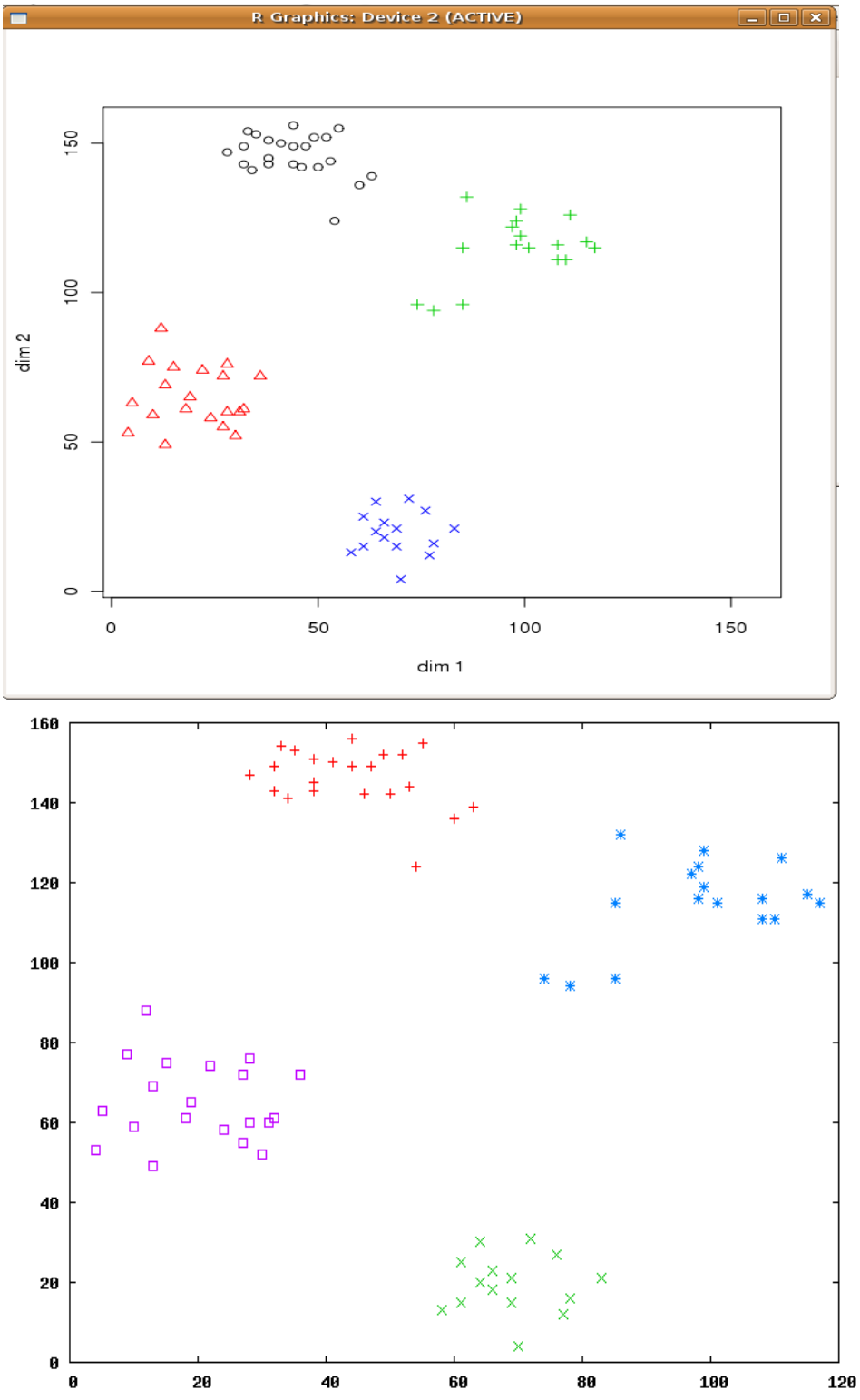

Figura 5.3: Particoes geradas para Ruspini: CLUES (em cima) e o AECBL1

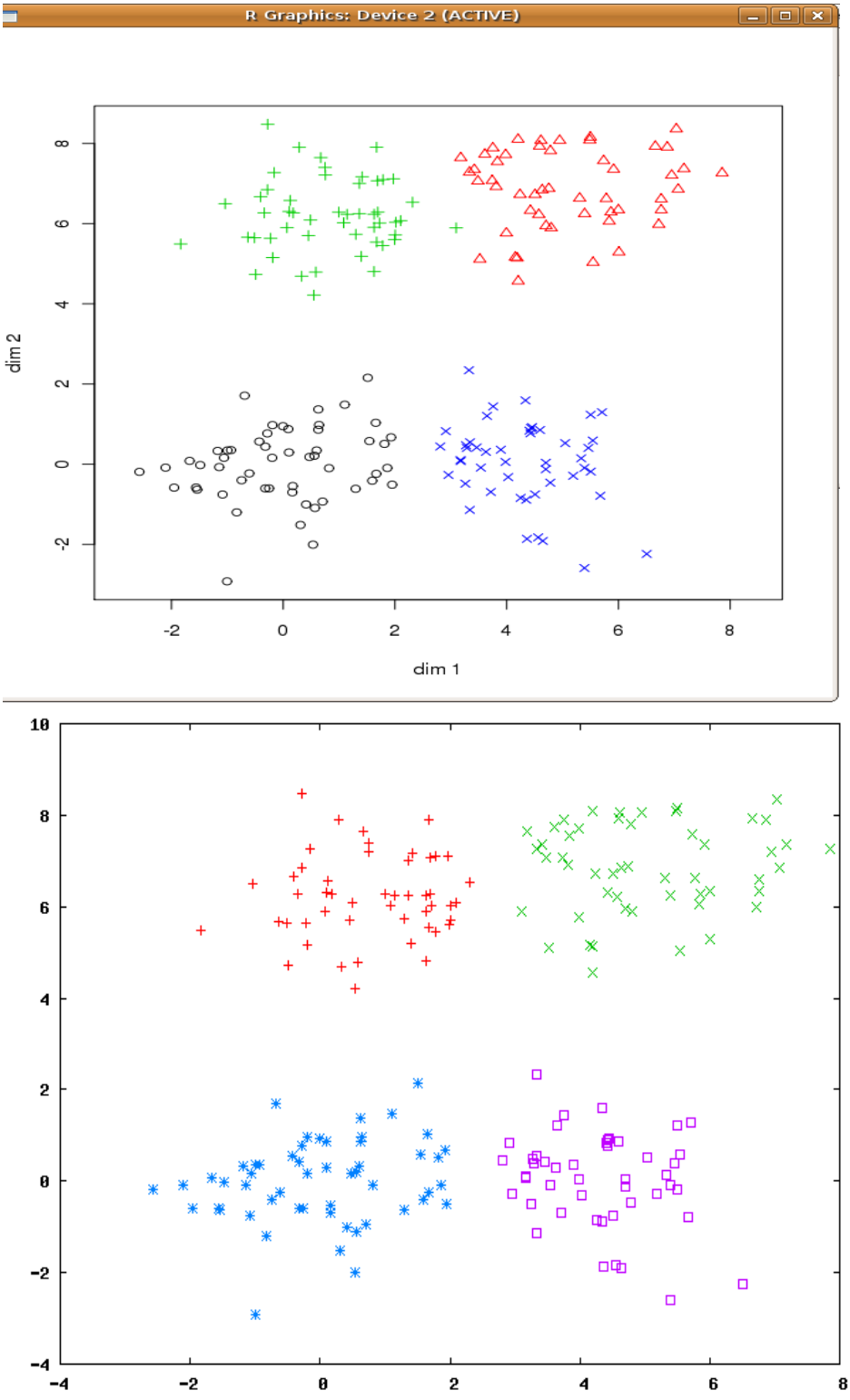

Figura 5.4: Particoes geradas para Maronna: CLUES (em cima) e o AECBL1

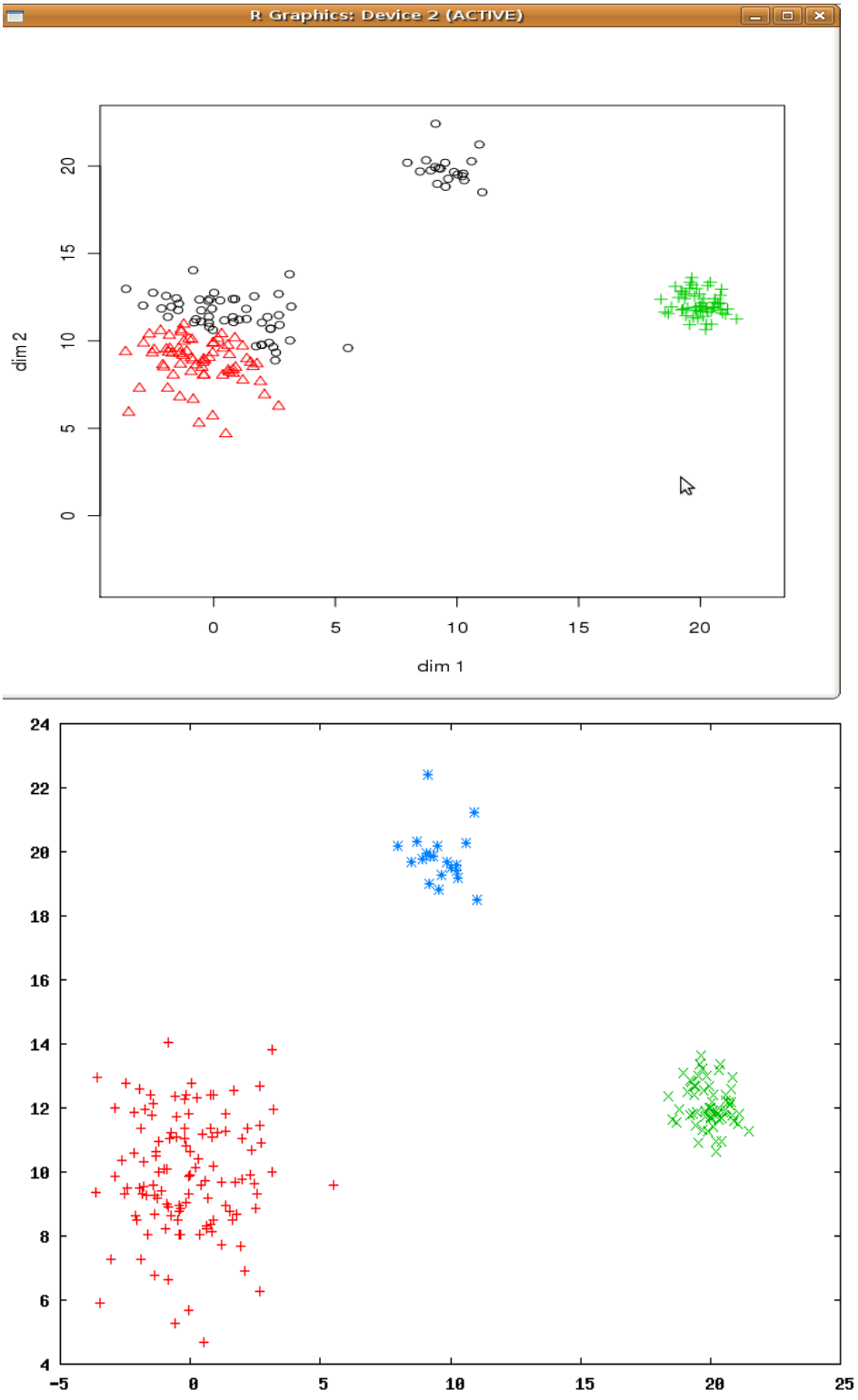

Figura 5.5: Particoes geradas para 200DATA: CLUES (em cima) e o AECBL1

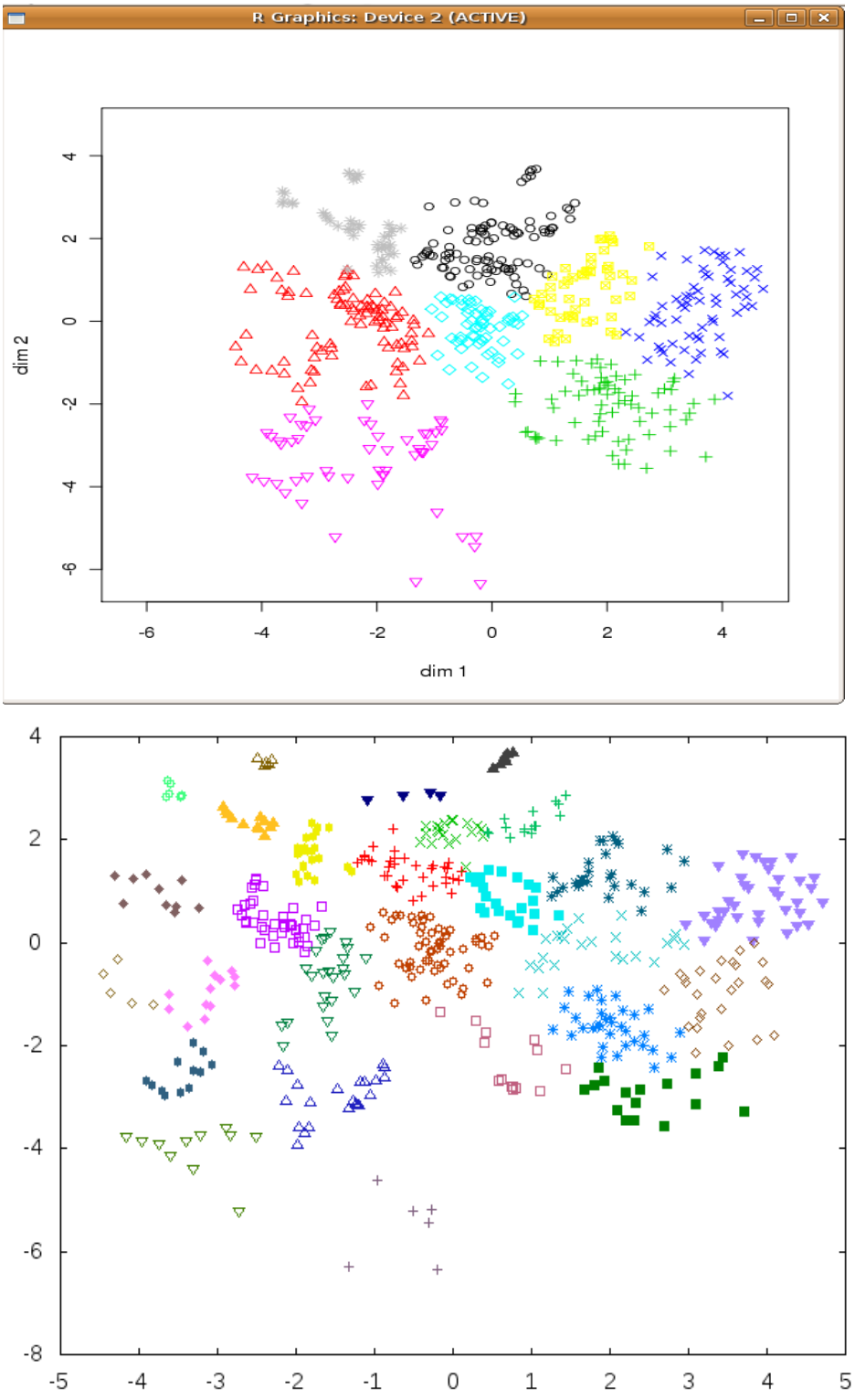

Figura 5.6: Particoes geradas para vowel: CLUES (em cima) e o AECBL1

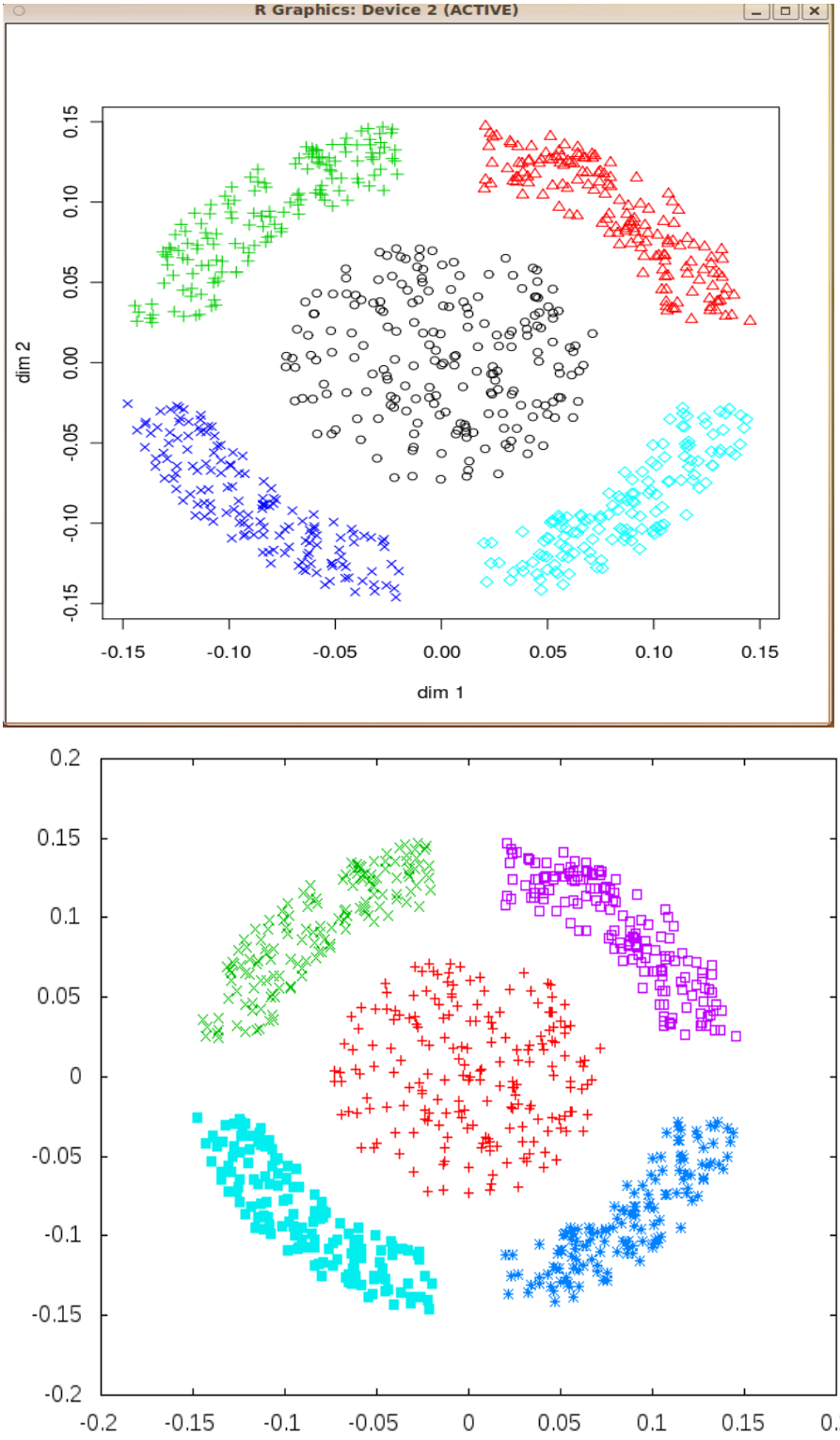

Figura 5.7: Particoes geradas para Broken Ring: CLUES (em cima) e o AECBL1

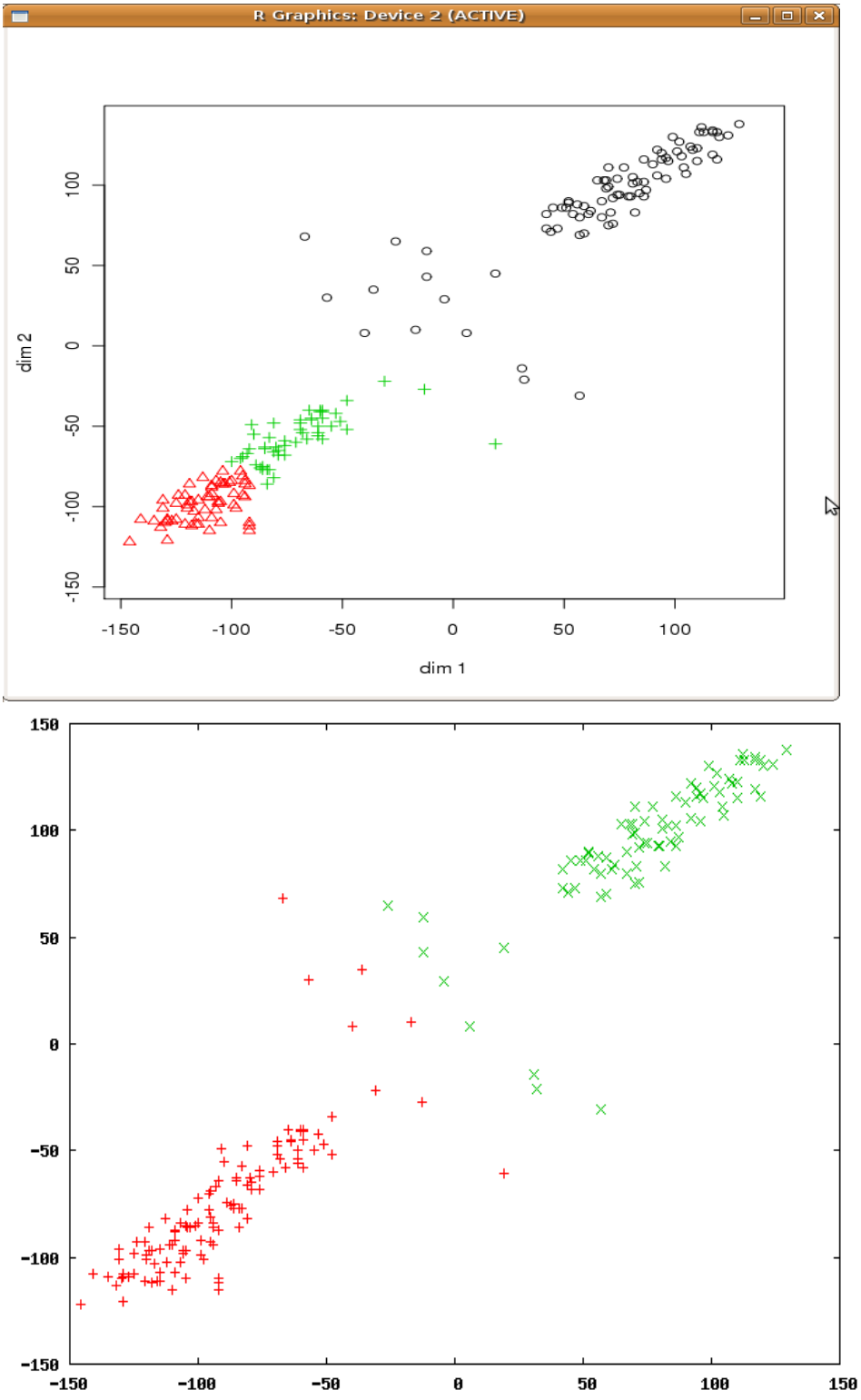

Figura 5.8: Particoes geradas para 200p2c1: CLUES (em cima) e o AECBL1

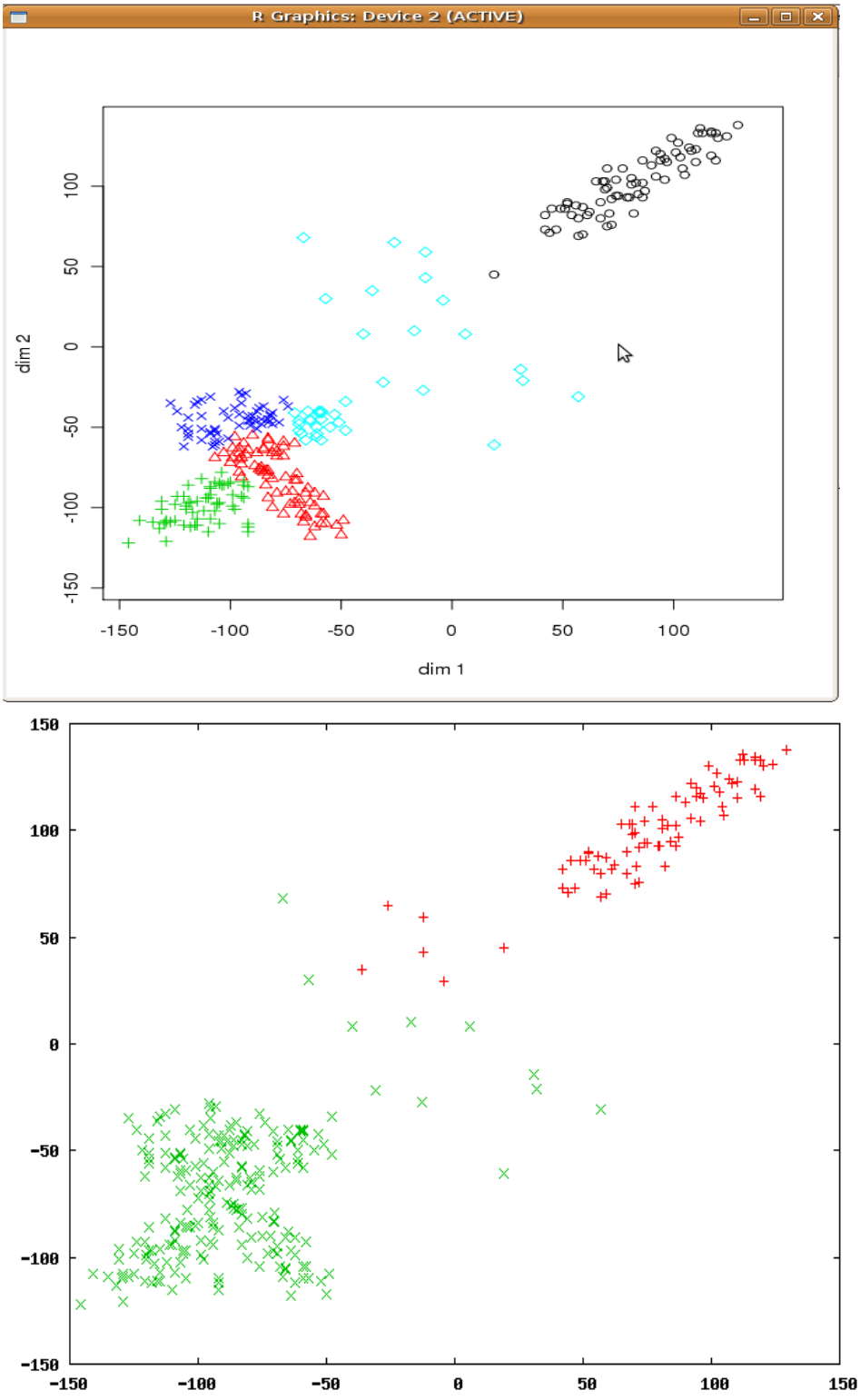

Figura 5.9: Particoes geradas para 300p2c1: CLUES (em cima) e o AECBL1

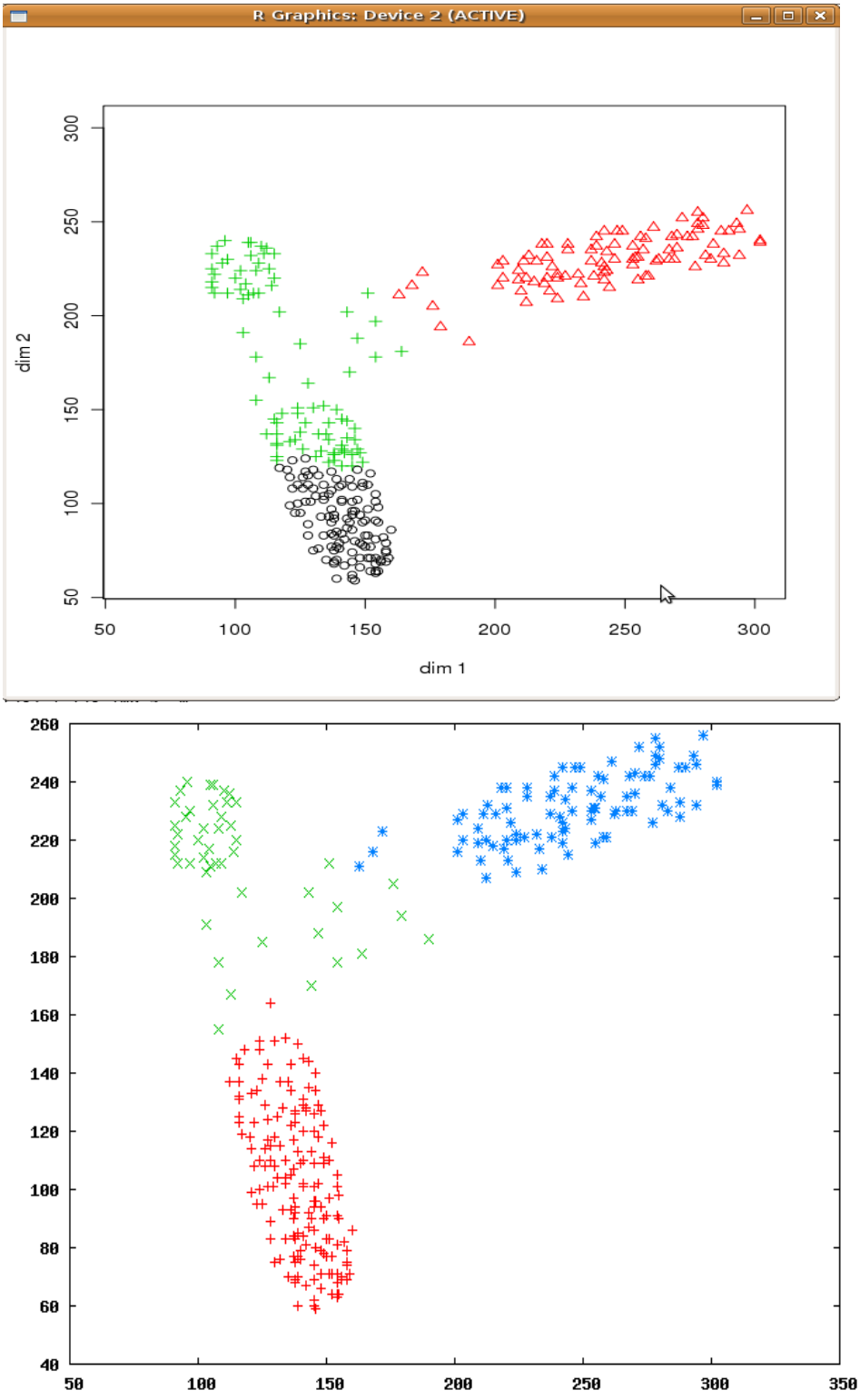

Figura 5.10: Particoes geradas para 300p3c1: CLUES (em cima) e o AECBL1

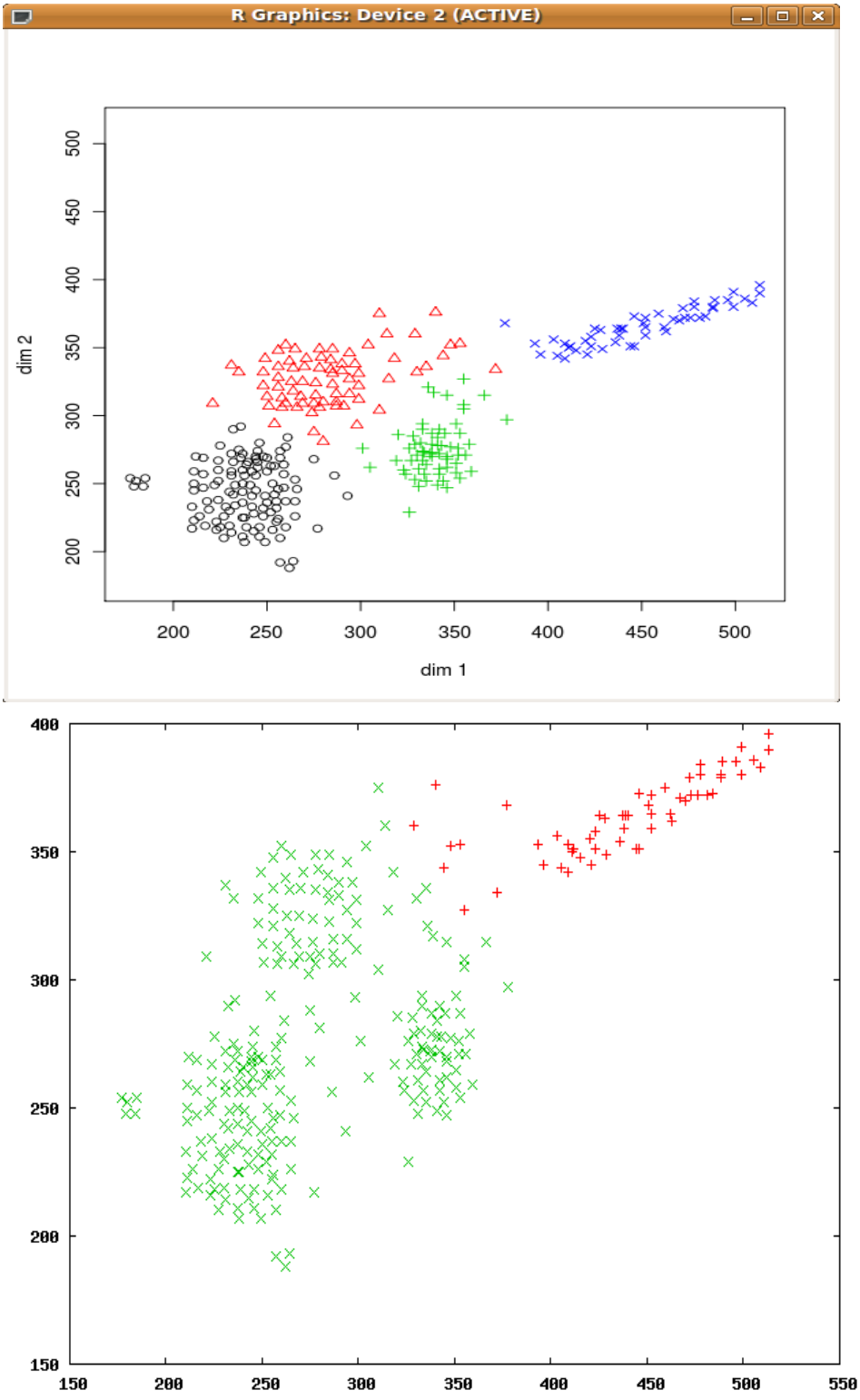

Figura 5.11: Particoes geradas para 300p4c1: CLUES (em cima) e o AECBL1

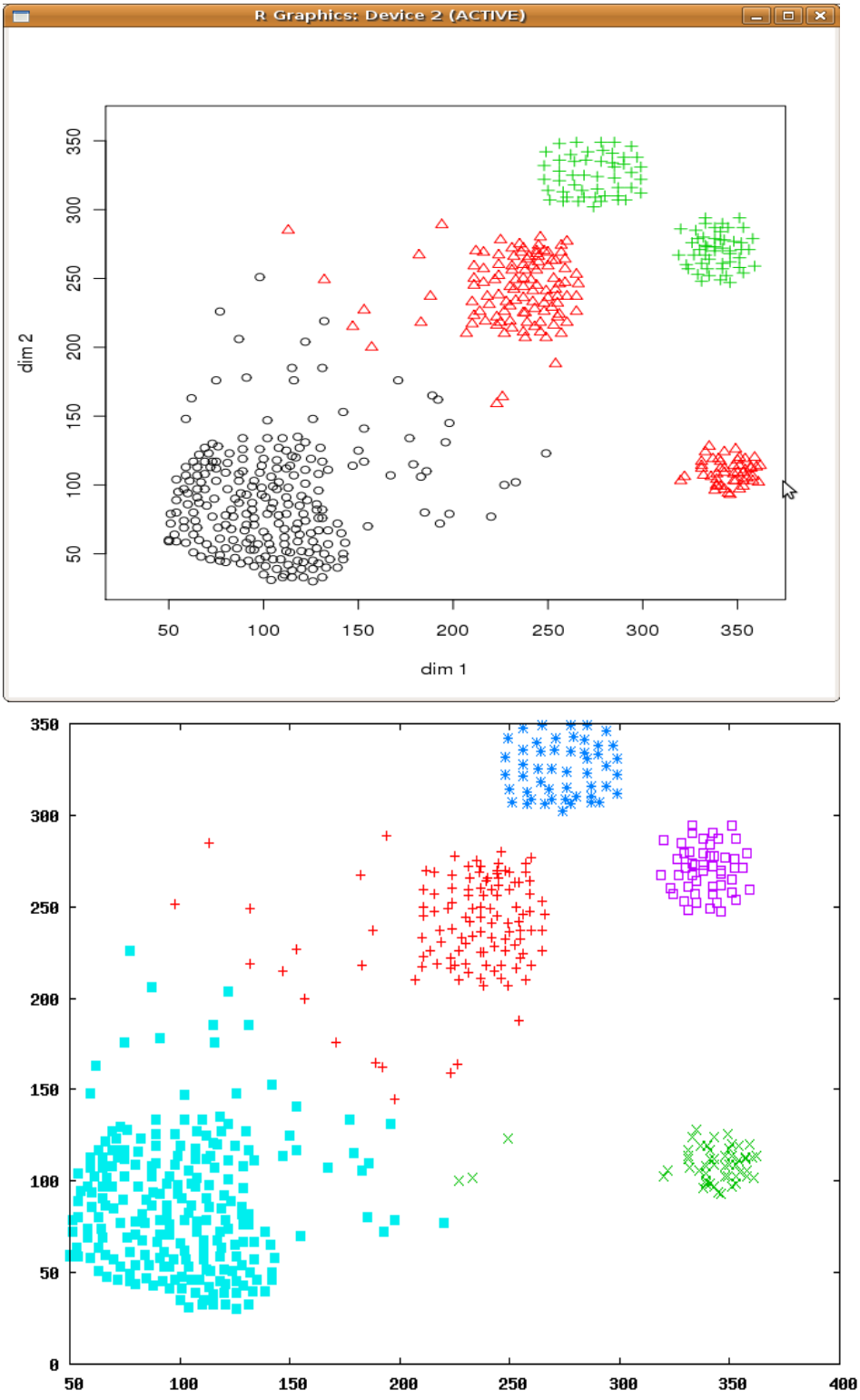

Figura 5.12: Particoes geradas para 500p4c1: CLUES (em cima) e o AECBL1

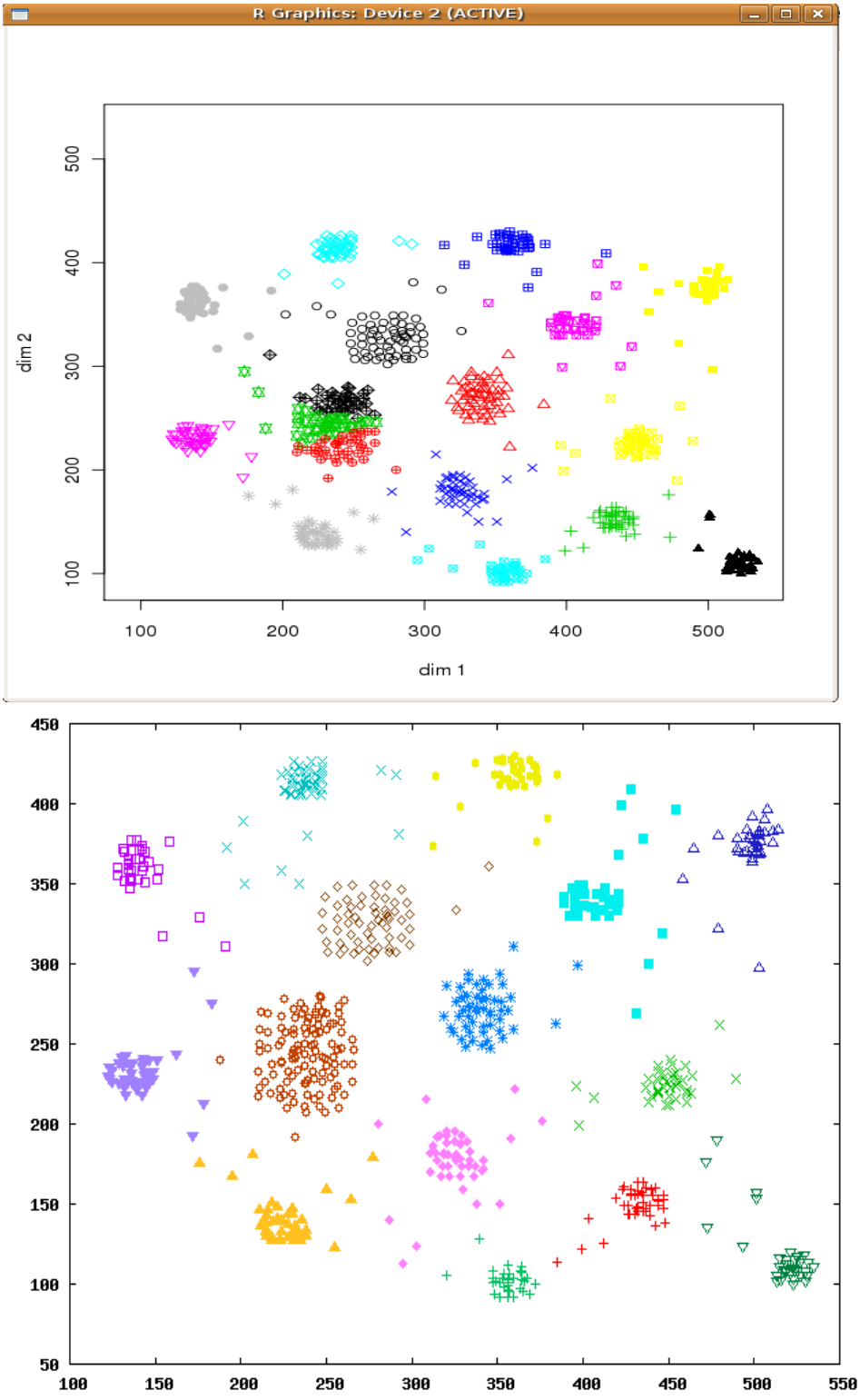

Figura 5.13: Particoes geradas para 700p15c1: CLUES (em cima) e o AECBL1

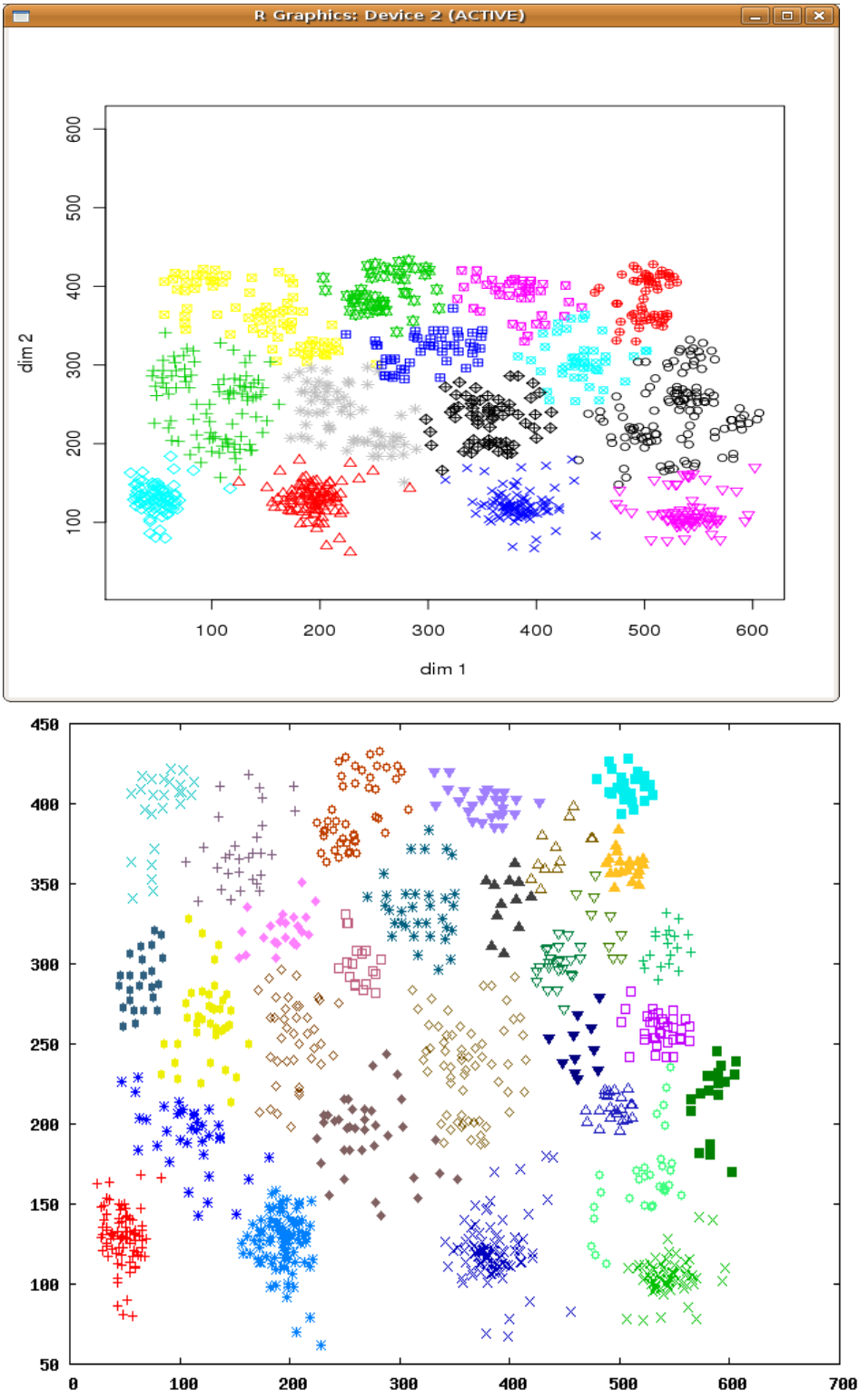

Figura 5.14: Particoes geradas para 1000p27c1: CLUES (em cima) e o AECBL1

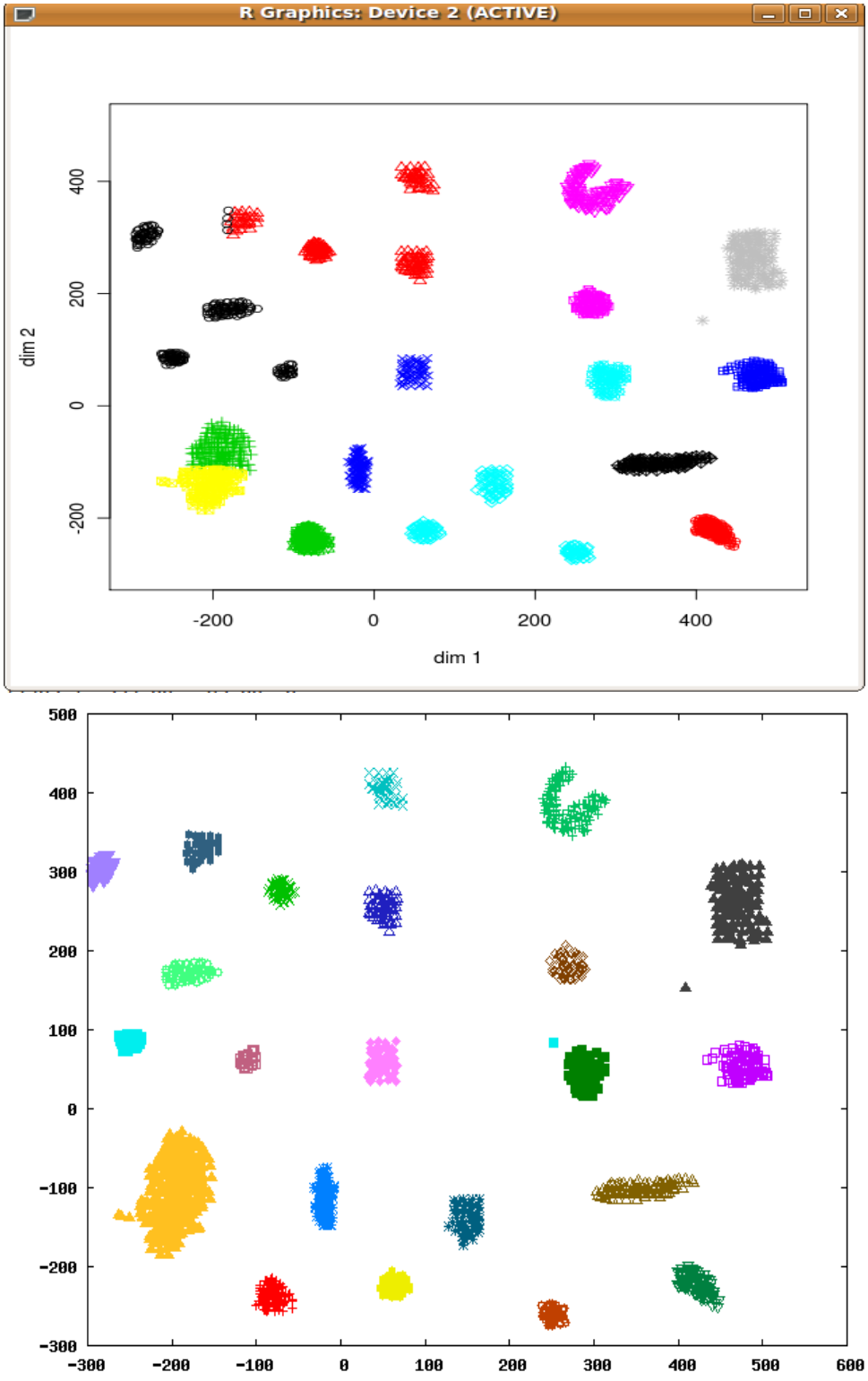

Figura 5.15: Particoes geradas para 1800p22c: CLUES (em cima) e o AECBL1

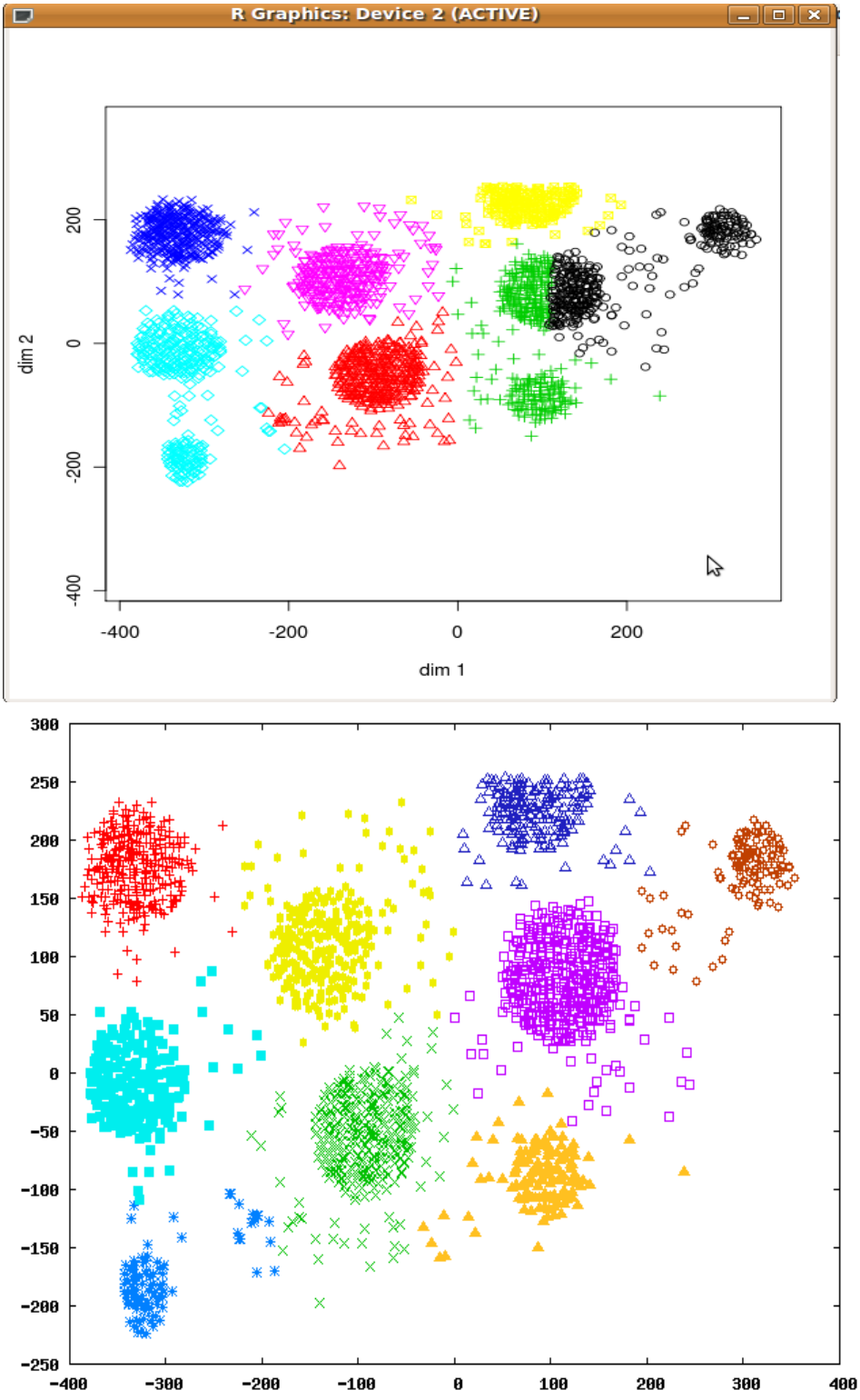

Figura 5.16: Particoes geradas para 2000p9c1: CLUES (em cima) e o AECBL1

Capítulo 6

Conclusão

Neste trabalho, apresentamos novas heurísticas para resolver o PCA. As heurísticas existentes, que utilizam metaheurísticas, usam na maioria dos casos, Algoritmos Evolutivos. As nossas contribuições são novas heurísticas utilizando *Algoritmos Evolutivos, GRASP e ILS.* Além disso, propomos um novo procedimento para juntar clusters parciais e novas buscas locais. Foram propostos também, novos métodos híbridos que utilizam modelos exatos para melhorar as soluções obtidas pelas heurísticas. Finalmente, os melhores algoritmos propostos neste trabalho foram comparados com um algoritmo recente da literatura.

Foi observado que as implementações dos *AE's, GRASP's e ILS's* que utilizavam características comuns, como o Etapa de Construção, a representação da solução e as buscas locais, obtiveram um desempenho semelhante, com vantagem para a heurística que utiliza Algoritmo Evolutivo. Os *AE's*, acrescidos de buscas locais, melhoram muito o seu desempenho.

Os algoritmos *AECBL1, GBLITRC1 e IBLITRC2* obtiveram valores da função Índice Silhueta muito próximos, indicando que o desempenho dos algoritmos, na média, é muito semelhante. O algoritmo *AECBL1* obteve os melhores resultados, seguido pelo *GBLITRC1.* O resultado é devido, possivelmente pelo fato, do *AECBL1* trabalhar com um conjunto de soluções que se combinam e formam novas soluções. Além disso a busca local Inversão Individual auxilia a convergência do algoritmo, buscando novas soluções próximas as pertencentes a população. Com isso, o espaço de busca é intensamente vasculhado. O algoritmo é auxiliado pelas buscas Reconexão por caminhos e no final pela Troca entre Pares que intensificam ainda mais a

procura de novas soluções. O *GBLITRC1* sempre trabalha com apenas uma solução inicial a cada iteração, e portanto, tem mais dificuldade de vasculhar o espaço de busca. O *IBLITRC2* trabalha com uma busca local para cada número de clusters pais. Se a busca local não encontrar o ótimo local, os resultados podem não ser muito bons.

Os métodos híbridos conseguiram melhorar alguns valores da função Índice Silhueta obtidas pelas heurísticas. Estes métodos utilizam um modelo exato para auxiliar a busca por soluções melhores. O que foi observado é que, uma vez que o número de k é fixado e é o melhor possível, o modelo exato tende a colocar os pontos nas melhores posições relativas a cada cluster, quando comparado com as soluções obtidas pela heurística. Isto aconteceu em 40 das 53 instâncias.

Na comparação dos métodos propostos com um algoritmo da literatura, denominado CLUES, percebemos que todos obtiveram resultados melhores. Os algoritmos propostos obtiveram uma média dos valores da função no conjunto das instâncias testadas melhores que o CLUES e também, alcançaram o melhor valor em um número maior de instâncias.

Como propostas para trabalhos futuros temos:

- Desenvolver outras metaheurísticas como VNS, Busca Tabu utilizando as características comuns, definidas neste trabalho, para verificar os seus desempenhos;
- Melhorar os métodos híbridos através de uma melhor integração entre a heurística e o modelo exato K-Medianas.
- Tentar melhorar o modelo exato K-Medianas, para diminuir os tempos de execução dos métodos híbridos.

Referências Bibliográficas

[1] AGRAVAL, R. ET AL ; Automatic Subspace Clustering of High Dimensional Data, *Data Mining and Knowledge Discovery 11*, pp. 5–33, 2005.

[2] ALEX, R., BIRATO, S., RESENDE, M.; Parallel GRASP with path relinking for job scheduling . *Parallel Computing (29)*, pp. 393-340. 2003

[3] ALJABER,N.; BAEK, W.; CHEN, C. L. A Tabu Search Approach to the Cell Formation Problem, *Computers and Industrial Engineering*, pp. 169-185. 1997.

[4] ANKERST, M.; BREUNIG,M. M.; KRIEGEL,K. P.; SANDER,J. Optics: ordering points to identify the clustering structure, *Proceeding of International Conference on Management of Data*, pp. 49–60, 1999.

[5] BAGIROV, A. M.; YEARWOOD, J. A new Nonsmooth Optimization Algorithm for Minimum Sum-of-squares Clustering Problems; *European Journal of Operational Research, Vol. 170 issue 2*, 2006.

[6] BERKHIN, P. Survey of Clustering Data Mining Techniques. *Accrue Software*, 2002.

[7] BIRANT, D.; KUT, A. ST-Dbscan: An algorithm for clustering spatial–temporal data. *Data & Knowledge Engineering 60*, pp. 208–221, 2007.

[8] DIAS, C. R. (Dissertação de Mestrado) Algoritmos Evolutivos para o Problema de Clusterização em Grafos Orientados: Desenvolvimento e Análise Experimental. Orientador: Luiz Satoru Ochi. Programa de Pós Grad. em Computação, IC/UFF, 2004.

[9] DOVAL, D., MANCORIDIS, S. E MITCHELL, B. S. Automatic Clustering of Software Systems using a Genetic Algorithm, *1999 International Conference on SoftwareTools and Engineering Practice*, 1999.

[10] DUAN, L. ET AL. A local-density based spatial clustering algorithm with noise, *Information Systems*, 2006.

[11] DUDA, R. and HART, P. ; Pattern Classification and Scene Analysis. *John Wiley & Sons*, New York, NY. 1973.

[12] ESTER, M.; KRIEGEL, H. P.; XU, X. A database interface for clustering in large spatial databases. In Proc. of the 1st Int'l Conference on Knowledge Discovery in Databases and Data Mining, Montreal, Canada, 1995.

[13] FACELI, K. , CARVALHO, A.C.P.L.F , SOUTO, M. C. P. Validação de Algoritmos de Agrupamento; Relatório técnico do ICMC. 2005.

[14] FISHER, L. Knowledge acquisition via incremental conceptual clustering, *Machine Learning 2*, pp. 139–172. 1987.

[15] FISHER, R. The use of multiple measurements in taxonomic problems. *Annual Eugenics 7*, pp. 179-188. 1936.

[16] FOULDS, L.R.; NEUMANN, K. A network flow model of group technology; *Mathematical and computer modeling, 38*, 2003.

[17] GARAI, G. , CHAUDHURI, B. B. ; A novel genetic algorithm for automatic clustering. *Pattern Recognition Letters*. pp 173-187. 2004.

[18] GLOVER, F.; KOCHENBERGER, G. A. Handbook of Metaheuristics. *Kluwer Academic Publishers.* 2003.

[19] GUHA, S.; RASTOGI, R.; SHIM, K. Cure: An efficient clustering algorithm for large databases, *Proc.of ACM SIGMOD Conference*, pp. 73-84. 1998.

[20] GUHA S.; RASTOGI R.; SHIM, K. Rock: a robust clustering algorithm for categorical attributes, *Proceedings of* 15^{th} *International Conference on Data Engineering*, pp. 512–521, 1999.

[21] HANSEN, P.; JAUMARD, B. Cluster Analysis and Mathematical Programming; *Mathematical Programming no. 79*. pp.191-215. 1997.

[22] HAN,Y.; SHI,P. An improved ant colony algorithm for fuzzy clustering in image segmentation; *Neurocomputing 70*.pp 665–671. 2007.

[23] HASTIE, T.;TIBSHIRANI, R.; FRIEDMAN, J. The Elements of Statistical Learning. *Data Mining , Inference, and prediction. Springer.* 2001.

[24] HINNEBURG, A.; KEIM, D. A. An efficient approach to clustering in large multimedia databases with noise, *Proceedings of 4^{th} International Conference on Knowledge Discovery and Data Mining,* New York City, NY, pp. 58–65. 1998.

[25] JI, X.; MITCHEL, J. Branch-and-price-and-cut on the clique partitioning problem with minimum clique size requirement. *Discrete Optimization, vol.4*. pp. 87-102. 2005.

[26] KARYPIS, G.; HAN, E. H.; KUMAR, V. Chamaleon: A Hierarchical Clustering Algorithm Using Dynamic Modeling, *IEEE Computer*, 32 , pp. 68–75. 1999.

[27] KAUFMAN,L.; ROUSSEEUM, P. J. Finding Groups in Data : An introduction to cluster Analysis. A Wiley-Intercience publication. 1990.

[28] LASZLO, M.; MUKHERJEE, S. A Genetic Algorithm that exchanges neighboring centers for k-means clustering; *Pattern Recognition Letters*, 2007.

[29] LIU, Y. ET AL. A tabu search approach for the minimum sum-of-squares clustering problems. *Information Sciences*. doi. 10.1016./j.ins.2008.01.022. 2008.

[30] LORENA, L. A. N.; FURTADO, J. C. Constructive Genetic Algorithm for Clustering Problems. *Evolutionary Computation*, vol. 9, no. 3, pp. 309-327, 2001.

[31] MACQUEEN,J. B. Some Methods for classification and Analysis of Multivariate Observations, Proceedings of 5-th Berkeley Symposium on Mathema-

tical Statistics and Probability, Berkeley, University of California Press 1, 1967.

[32] MARONNA, R.; JACOVKIS, P. M. Multivariate clustering procedures with variable metrics. *Biometrics30*, pp. 499-505. 1974.

[33] MA, S.; WANG, T. J.; TANG,S. W. A New Fast Clustering Algorithm Based on Reference and Density, *Lectures Notes in Computer Science*, vol. 2762, Springer, Berlin, pp. 214–225, 2003.

[34] MA, W. M.; EDEN,C.; TOMMY, W.S. A new shifting grid clusting algorithm, *Pattern Recognition 37*, pp. 503–514, 2004.

[35] MEHROTRA, A.; TRICK, M. A. Cliques and clustering: a combinatorial approach. *Operations Research Letters, 22*, 1998.

[36] NG, R. T.; HAN, J. Eficient and Efective clustering methods for spatial data mining. In Proc. of the VLDB Conference, Santiago, Chile, September 1994.

[37] OSMAN, I. H. ; CHISTOFIDES, N. Capacitated clustering problems by hibrid simulated annealing and tabu search, *International Transactions in Operational research*, pp 317-336, 1994.

[38] RAO, M. R. Cluster Analysis and Mathematical Programming; *Journal of the American Statistical Association, vol. 66*, pp.622-626. 1971.

[39] REDMOND, S. J. , HENEGGHAN , C. A method for initialing the k-means clustering algorithm using kd-trees. *Pattern Recognition Letters 28*, 2007.

[40] RUSPINI, E. H. Numerical methods for fuzzy clustering. Information Science. pp. 319-350. 1970.

[41] SANGHAMITRA, B.; UJJWAL, M. An evolutionary technique based on K-Means algorithm for optimal clustering, *Information Sciences*,146, pp. 221–237. 2002.

[42] SHEIKHOLESLAMI,G.; CHATTERJEE,S.; ZHANG, A. WaveCluster: a wavelet-based clustering approach for spatial data in very large data bases, VLDB J. 8, pp. 289–304, 2000.

[43] SHELOKAR, P.S; JAYARAMAN, V.K.; KULKARNI, B.D. An ant colony approach for clustering. *Analytica Chimica Acta 509*, pp. 187–195. 2004.

[44] SOARES, S. S. F. Dissertação de Mestrado) Metaheurísticas para o Problema de Clusterização Automática . Orientador: Luiz Satoru Ochi. Programa de Pós Graduação em Computação,IC/UFF, 2004.

[45] SUNG, C. S.; JIN, H. W. A tabu-search based heuristic for clustering, *Pattern Recognition 33*, pp. 849-858, 2000.

[46] TRINDADE, A. R. (Dissertação de Mestrado) Metaheuristicas para o Problema de Clusterização de Células de Manufatura . Orientador: Luiz Satoru Ochi. Programa de Pós Grad. em Computação, IC/UFF, 2004.

[47] TSENG., L. Y.; YANG, S. B. A genetic approach to the automatic clustering problem. *Pattern Recognition 34* .pp 415-424. 2001.

[48] VINOD, H. D. Integer Programming and Theory of Groups; *J. American Statistic Association Vol. 64*, pp. 506-519. 1969.

[49] WANG, W.; YANG,J.; MUNTZ, R. Sting: a statistical information grid approach to spatial data mining, *Proceedings ofthe International Conference on Very Large Data Bases*, pp. 186–195. 1997.

[50] WANG,W.; YANG,J.; MUNTZ,J; Sting+: an approach to active spatial data mining, *Proceedings of15th International Conference on Data Engineering*, pp. 116–125, 1999.

[51] WANG, X. ET AL. Clues: A non-parametric clustering method based on local shrinking, *Computational Statistics & Data Analysis*. pp.286–298, 2007.

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

[53] YUJIAN, L. A clustering algorithm based on maximal θ-distant subtrees, *Pattern Recognition*, 2006.

[54] XAVIER , A. E. The Hyperbolic Smoothing Clustering Method; *Pattern Recognition Letters. 2009*, dói 10.1016/j.patcog.2009.06.018. 2009.

[55] ZHANG, T.; RAMAKRISHNAN, R.; LIVNY, M. Birch: An efficient data clustering method or very large databases. *SIGMODRec. 25*, 2, pp. 103–114. 1996.

[56] ZHAO,Y. C.; SONG,J. Gdilc: a grid-based density-isoline clustering algorithm, *Proceedings ofInternational Conferences on Info-tech and Info-net,* Vol. 3, pp. 140–145, 2001.