UNIVERSIDADE FEDERAL DE JUIZ DE FORA
INSTITUTO DE CIÊNCIAS EXATAS
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Tatiane Martins Pacheco

ANTHILL: OTIMIZAÇÃO HEURÍSTICA POR COLÔNIA DE FORMIGAS PARA RESOLUÇÃO DO PROBLEMA DE AGRUPAMENTO AUTOMÁTICO

Juiz de Fora

2019

UNIVERSIDADE FEDERAL DE JUIZ DE FORA
INSTITUTO DE CIÊNCIAS EXATAS
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Tatiane Martins Pacheco

ANTHILL: OTIMIZAÇÃO HEURÍSTICA POR COLÔNIA DE FORMIGAS PARA RESOLUÇÃO DO PROBLEMA DE AGRUPAMENTO AUTOMÁTICO

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação, do Instituto de Ciências Exatas da Universidade Federal de Juiz de Fora como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Orientador: Victor Ströele de Andrade Menezes

Juiz de Fora

2019

Ficha catalográfica elaborada através do programa de geração
automática da Biblioteca Universitária da UFJF,
com os dados fornecidos pelo(a) autor(a)

Pacheco, Tatiane Martins.
Anthill: Otimização Heurística por Colônia de Formigas para Resolução do Problema de Agrupamento Automático / Tatiane Martins Pacheco. -- 2019.
91 p. : il.

Orientador: Victor Ströele de Andrade Menezes
Dissertação (mestrado acadêmico) - Universidade Federal de Juiz de Fora, Instituto de Ciências Exatas. Programa de Pós Graduação em Ciência da Computação, 2019.

1. Clusterização. 2. Agrupamento Automático. 3. Metaheurística. 4. Colônia de formigas. 5. Paralelismo. I. Menezes, Victor Ströele de Andrade , orient. II. Título.

Tatiane Martins Pacheco

"ANTHILL: Otimização Heurística por Colônia de Formigas para Resolução do Problema de Agrupamento Automático"

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Juiz de Fora como requisito parcial à obtenção do grau de Mestre.

Aprovada em 18 de março de 2019.

BANCA EXAMINADORA

Prof. Dr. Victor Ströele de Andrade Menezes – Orientador
Universidade Federal de Juiz de Fora

Prof. Dr. Heder Soares Bernardino
Universidade Federal de Juiz de Fora

Prof. Dr. José Elias Cláudio Arroyo
Universidade Federal de Viçosa

À memória de minha mamãe
Madalena, que permanece aqui,
tão longe dos olhos, mas dentro
do coração.

AGRADECIMENTOS

Agradeço a Deus por guiar meus passos, abençoar minhas escolhas e me dar perseverança nos momentos difíceis.

À minha filha amada Milena, por toda a compreensão e carinho, por entender que a mamãe precisava estudar e por fazer todo o possível para colaborar, mesmo que fosse com um abraço e um beijinho.

Ao meu amado Felipe, marido querido, carinhoso, colaborativo e compreensivo, que com sua calma e tranquilidade me transmitiu a paz necessária nos momentos mais difíceis.

Aos meus pais Genario e Madalena (in memoriam), pelo apoio incondicional e por todo amor que recebi.

Agradeço à todos os professores do programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Juiz de Fora e em especial ao meu orientador Victor pela paciência, e os professores Stênio e Luciana pela motivação, ajuda e compreensão.

Aos professores Heder Soares Bernardino e José Elias Claúdio Arroyo que fizeram parte da banca examinadora, agradeço as valiosas contribuições que enriqueceram este trabalho.

Aos integrantes do LAPIC pelo suporte computacional que me foi disponibilizado com muita boa vontade.

Aos amigos do PGCC que tornaram essa jornada mais leve e divertida, em especial a minha amiga Lenita, companheira para todas as horas.

E por fim, aos meus gêmeos Miguel e Mikael, ainda pequeninos, mas já participaram da defesa de mestrado da mamãe acolhidos em seu colinho.

"Lembre-se: seu foco determina a
sua realidade."
Qui-Gon Jinn, Star Wars

RESUMO

Métodos aproximativos são amplamente utilizados na resolução de problemas computacionais complexos, uma vez que são capazes de apresentar resultados significativos em relação à qualidade da solução em um tempo satisfatório. Esta dissertação aborda o problema do agrupamento automático, reconhecidamente NP-difícil. Para resolver o problema, o trabalho propõe uma metaheurística baseada em inteligência coletiva, inspirada no comportamento das formigas, denominado *Anthill*. O índice de silhueta foi utilizado para medir a qualidade dos *clusters* gerados. Os resultados obtidos pelo *Anthill* foram comparados com os resultados obtidos na literatura e os experimentos realizados indicam que o algoritmo é capaz de encontrar *clusters* que representam bem a distribuição original dos dados. Entretanto, durante a avaliação, foi identificado que o tempo de processamento exigido pelo *Anthill* pode impactar na sua escalabilidade, sendo um impeditivo para o seu uso na resolução de problemas de grande porte. Assim, foi proposta uma versão paralelizada do algoritmo e os experimentos foram reconduzidos. Os resultados finais foram analisados considerando as métricas de *Speedup* e *Eficiência*, mantendo os mesmos índices de silhuetas encontrados na versão sequencial.

Palavras-chave: Clusterização; Agrupamento Automático; Metaheurística; Colônia de formigas; Paralelismo.

ABSTRACT

Approximate methods are widely used in solving complex computational problems, since they are capable of presenting significant results regarding the quality of the solution in a satisfactory time. This dissertation addresses the problem of automatic grouping, admittedly NP-difficult. To solve the problem, the work proposes a metaheuristic based on collective intelligence, inspired by the behavior of the ants, called *Anthill*. The silhouette index was used to measure the quality of the generated *clusters*. The results obtained by the Anthill were compared with the results obtained in the literature and the experiments carried out indicate that the algorithm is able to find clusters which represent well the original distribution of the data. However, during the evaluation, it was identified that the processing time required by the *Anthill* can impact its scalability, being an impediment to its use in solving large problems. Thus, a parallel version of the algorithm was proposed and the experiments were carried out. The final results were analyzed considering the Speedup and Efficiency metrics, maintaining the same indexes of silhouettes found in the sequential version.

Keywords: Clustering. Automatic Grouping. Metaheuristic. Ant colony. Parallelism.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ACO *Ant Colony Optimization*

PCA Problema de Agrupamento Automático

ACC *Ant Colony Clustering*

FPSO *Fuzzy Particle Swarm Optimization*

RKMD *Rough K-Means Discernibility*

AECBL1 Algoritmo Evolutivo Construtivo com Busca Local

HH Heurística Híbrida

FCM *Fuzzy C-Means*

PSO *Particle Swarm Optimization*

FGP Formação de Grupos Parciais

JGP Junção Grupos Parciais

AG Algoritmo Genético

ILS *Iteraded Local Search*

DBSCAN *Density Based Spatial Clustering of Applications with Noise*

ILS-FJGP Junção da metaheuristica ILS com o método de Formação e Junção de Grupos Parciais

ILS-DBSCAN Junção da metaheuristica ILS com o método DBSCAN

IRACE *Iterated Racing for Automatic algorithm configuration*

ACP Analise de Componentes Principais

SUMÁRIO

1 INTRODUÇÃO

O crescente desenvolvimento tecnológico verificado desde a década de 1980 desencadeou um processo contínuo e irreversível de informatização nos diversos setores produtivos e governamentais, o que levou à geração de um grande volume de dados. Neste contexto, técnicas de Mineração de Dados vêm sendo desenvolvidas e utilizadas para extrair informação e conhecimento de grandes massas de dados.

Dentre as técnicas de mineração de dados existentes, destacam-se aquelas voltadas a problemas de agrupamento (*clustering*). Tais técnicas visam auxiliar o usuário na compreensão de estruturas naturais existentes em um conjunto de dados (ANKERST et al., 1999) e que, em geral, quanto maior o volume de dados mais complexa é esta tarefa. Transformar grandes volumes de dados em informação relevante proporciona vantagem competitiva no processo de tomada de decisão em diferentes domínios. O interesse principal ao utilizar tais técnicas é identificar grupos (*clusters*) representativos que permitam extrair conclusões úteis a respeito do agrupamento gerado a partir dos dados originais. Neste contexto, há duas situações distintas quanto à necessidade de agrupar os dados: quando já se conhece o número K de agrupamentos desejado, definido como Problema da K-Clusterização, e quando não se tem qualquer conhecimento quanto ao número de grupos existentes na base de dados. Neste último caso, tem-se o Problema de Agrupamento Automático - PCA.

Segundo Ankerst et al. (1999), um algoritmo de agrupamento pode ser usado como uma ferramenta autônoma para obter informações sobre a distribuição de um conjunto de dados e também como pré-processamento para outros algoritmos que operam sobre os *clusters* detectados. Por sua natureza não supervisionada, as técnicas de agrupamento se tornam especialmente aplicáveis a problemas onde há pouca ou nenhuma informação *apriori*.

Não existe uma técnica de agrupamento que seja universalmente aplicável ante à variedade de estruturas presentes em conjuntos de dados multidimensionais (JAIN et al., 1999). A tarefa de encontrar um agrupamento ótimo quanto a um dado critério a partir das combinações de objetos em grupos pode ser desencorajada em aplicações práticas com grande número de objetos e de atributos (NICHOLSON, 1998), dado o tempo com-

putacional requerido para a avaliação de cada uma das combinações possíveis. Assim, abordagens aproximativas são aplicadas em áreas diversificadas visando a obtenção de soluções em um tempo viável (HRUSCHKA et al., 2009).

O processo de exploração do espaço de busca a partir de qualquer abordagem aproximada pressupõe uma função de avaliação $f(.)$ definida sobre o espaço de busca em $\mathbb{R}^d$, $\forall$ d $\in \mathbb{N}^*$, que indica a qualidade das soluções exploradas ao longo do processo. Uma vez definida a função de avaliação $f(.)$, tem-se que o problema de agrupamento pode ser definido como um problema de otimização da função f. Devido sua complexidade natural, diversos métodos não exatos são aplicados em sua resolução. Quando tais métodos aproximativos para otimização não são de uso específico para um determinado problema, em geral recebem o nome de metaheurísticas (BLUM et al., 2011).

Metaheurísticas inspiradas em enxames são algoritmos aproximativos que imitam o comportamento de sistemas descentralizados e auto-organizados, que são aplicados a problemas computacionais difíceis na busca por soluções satisfatórias (HANDL; MEYER, 2007). Os comportamentos coletivos que mais inspiraram o desenvolvimento de algoritmos são os baseados em colônias de formigas (DORIGO et al., 1996), abelhas (KARABOGA, 2005) e partículas (KENNEDY, 2000).

Nesta dissertação é proposto o algoritmo *Anthill* para PCA, um método baseado em *Ant Colony Optimization* (ACO) inspirado no trabalho apresentado em Chen et al. (2005). Diferente da abordagem apresentada em Shelokar et al. (2004), em que a representação da solução é tal que cada formiga que integra a colônia codifica uma solução do problema, na abordagem proposta neste trabalho, uma solução do problema é obtida a partir da ação da colônia de formigas sobre o conjunto de dados da entrada. Neste sentido, pode-se dizer que a abordagem proposta consiste em um algoritmo que imita o comportamento de uma colônia de formigas, que constroem colaborativamente uma única solução ao término da execução do algoritmo. Para tal, cada elemento do conjunto de entrada é tomado como um nó de um digrafo ponderado nos arcos, onde o peso associado a cada arco é definido por uma medida de dissimilaridade entre o par de elementos em que o mesmo incide. Uma solução do problema é obtida a partir de um conjunto de componentes fortemente conexas geradas pela ação da colônia de formigas sobre o conjunto de arcos do referido digrafo.

Embora o algoritmo proposto possa ser facilmente adaptado ao caso em que o número

de *clusters* é dado de entrada, bastando para tal utilizar este número como critério de parada, o mesmo foi originalmente desenvolvido para solução do PCA, tendo sido submetido a bases de dados disponíveis em repositórios tradicionais deste tipo de problema. O *Anthill* foi comparado a outros algoritmos da literatura segundo os critérios de validação interna (silhueta), tempo computacional e inspeção visual.

1.1 OBJETIVOS

Este trabalho tem como principal objetivo o **desenvolvimento** de uma abordagem heurística baseada no comportamento de colônia de formigas para o Problema de Agrupamento Automático de forma a obter soluções de **boa qualidade** em um **tempo satisfatório** que representem significativamente a estrutura real dos dados e que possa ser utilizada em cenários reais.

Como objetivo secundário, pretende-se:

- Desenvolver e disponibilizar o *Anthill*;
- Avaliar a abordagem desenvolvida a partir da comparação da mesma com outras que utilizam diferentes métodos de Inteligência Computacional aplicados ao mesmo problema;
- Desenvolver uma versão paralelizada do *Anthill* para avaliar a escalabilidade do algoritmo.

1.2 ORGANIZAÇÃO DO TRABALHO

O restante desta dissertação está organizada como segue: o Capítulo 2 apresenta os fundamentos teóricos, onde algumas definições a respeito do problema de agrupamento e da otimização por colônia de formigas são apresentadas; os trabalhos relacionados são apresentados no Capítulo 3; no Capítulo 4 o algoritmo proposto é descrito e os resultados iniciais são apresentados no Capítulo 5; no Capítulo 6 é apresentado o algoritmo Anthill paralelizado, visando a escalabilidade da proposta e, no Capítulo 7 as métricas próprias de avaliação de aplicações paralelas são discutidas. Por fim, no Capítulo 8 são discutidas algumas conclusões e trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Segundo Arabie et al. (1996), agrupamento é a identificação de grupos homogêneos de **objetos**, o que consiste em inserir em um mesmo conjunto elementos que possuem uma alta similaridade intragrupo e alta dissimilaridade intergrupo. Como já apresentado no Capítulo 1, problemas de agrupamento podem ser classificados em duas classes distintas: (i) problema de K-clusterização, onde é necessário conhecer a *priori* o número de grupos; e (ii) problema de agrupamento automático, onde o número de grupos não é conhecido e precisa ser determinado pelo algoritmo de busca. Ambos os problemas pertencem à classe NP-difícil, quando $K > 3$ (COWGILL et al., 1999).

De acordo com Xu e Wunsch (2005), o processo de agrupamento pode ser dividido nas seguintes etapas:

1. Extração e seleção de características: extrair e selecionar as características representativas mais importantes do conjunto de dados original;

2. Projeto de algoritmo de agrupamento: projetar o algoritmo de agrupamento de acordo com as características do problema;

3. Avaliação do resultado: avaliar o resultado do agrupamento e julgar a validade do algoritmo;

4. Explicação do resultado: apresentar uma explicação prática para o resultado do agrupamento.

2.1 DEFINIÇÃO DO PROBLEMA

Agrupamento de dados, ou análise de *clusters*, é um processo de junção não supervisionada de padrões (observações, itens de dados, ou vetores de características) em grupos (*clusters*) (JAIN et al., 1999). Embora existam diferentes formas de avaliar uma solução do problema de agrupamento conforme o domínio de aplicação, em geral, dado um conjunto de soluções do problema, na escolha de uma boa solução deve-se buscar maximizar a dissimilaridade entre elementos de *clusters* diferentes (distância inter *clusters*) ao mesmo tempo em que maximiza a similaridade intra grupo.

O problema de agrupamento é definido sobre um conjunto de objetos ou padrões $X = \{X_1, X_2, \ldots, X_n\}$ como entrada, onde cada elemento X_j é definido como um vetor $X_j = (x_{j1}, x_{j2}, \ldots, x_{jd})$ no espaço $\mathbb{R}^d$, de forma que cada componente (x_{jd}) está associada a uma característica (atributo, dimensão ou variável) do elemento X_j.

O problema da K-clusterização consiste em obter K partições do conjunto X. Assim, uma clusterização $C = \{C_1, ..., C_K\}$, com $(k \leq n)$, sobre o conjunto X é solução do problema se:

$$\begin{aligned} & C_i \neq \emptyset, && \forall i = 1, \ldots, K; \\ & \bigcup_{i=1}^{K} C_i = X; && \\ & C_i \cap C_j = \emptyset, \quad \forall i, j = 1, \ldots K \text{ e } i \neq j. && \end{aligned} \tag{2.1}$$

Nota-se que uma solução do PCA apresenta a mesma estrutura de uma solução do problema da K-clusterização, mudando apenas o fato de que qualquer valor $2 \leq K \leq n$ gera solução viável.

2.2 AVALIAÇÃO DA QUALIDADE DO AGRUPAMENTO

Uma vez obtido um agrupamento sobre um conjunto de dados de entrada, a avaliação da qualidade da solução obtida pode ser dada segundo dois tipos de medida, conforme Rendón et al. (2011): (i) externa, quando a validação do agrupamento é feita a partir do conhecimento prévio de especialista sobre os dados de entrada; (ii) e interna, quando se considera as informações iguais intrínsecas dos dados (DEBORAH et al., 2010), o que pressupõe o estabelecimento de uma métrica específica.

Como métrica para aferir a qualidade das soluções encontradas pelos algoritmos propostos neste trabalho, utiliza-se o índice silhueta, conforme apresentado em Rousseeuw (1987). A justificativa para o uso desta métrica está no fato de que a mesma consiste em uma função monotônica, o que permite o emprego da mesma na otimização do número adequado de *clusters*. Assim, para qualquer solução do PCA, pode-se obter um valor de silhueta dentro do intervalo [-1,1], sendo que quanto maior o valor, melhor a qualidade da solução.

O índice silhueta é definido sobre uma solução do problema. Para tal, cada elemento do conjunto de dados da entrada tem o valor de silhueta calculado conforme o *cluster* ao

qual está associado. Assim, para um dado elemento, valores próximos a 1 indicam que o mesmo está bem agrupado, enquanto valores distantes de 1 indicam que o elemento está mal agrupado e talvez deva pertencer a outro *cluster*.

A silhueta da solução é obtida pela média aritmética das silhuetas de todos os elementos da entrada, calculada conforme a Equação 2.2, onde $a(i)$ índica a dissimilaridade média do elemento i em relação a todos os demais elementos pertencentes ao *cluster* ao qual i faz parte, e $b(i)$ indica a dissimilaridade média de i em relação aos elementos alocados em outros *clusters*.

$$SC = \frac{1}{n}\sum_{i=1}^{n} s(i)$$
$$s(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}} \tag{2.2}$$

Observa-se que o índice silhueta é uma função que estabelece um parâmetro comparativo entre duas soluções, mas o pressuposto da mesma é que se tenha uma métrica que defina a dissimilaridade entre pares de elementos do conjunto de entrada. Neste trabalho, utiliza-se a distância euclidiana.

2.3 METAHEURÍSTICAS

O termo metaheurística se refere a algoritmos aproximados de propósito geral, podendo ser aplicados a diferentes problemas de otimização, diferente das heurísticas que são algoritmos propostos especificamente para um dado problema (BLUM et al., 2011). Basicamente, metaheurísticas são desenvolvidas para obter soluções aproximadas para problemas de elevada complexidade computacional em diversas áreas, como aprendizado de máquina, mineração de dados, bioinformática, biologia computacional, finanças, otimização estrutural e topológica em eletrônicos, processamento de imagens e sinais, telecomunicações, roteamento, entre outros (TALBI, 2009).

Em Talbi (2009) é apresentada uma classificação das diversas metaheurísticas, que integram a área de Inteligência Computacional. Por tal classificação, uma metaheurística pode ser do tipo populacional ou baseada em uma única solução. Dentre as abordagens populacionais para problemas de agrupamento, destacam-se as abordagens evolucionistas (HRUSCHKA et al., 2009).

Técnicas inspiradas na natureza estudam o comportamento de sistemas descentralizados e auto-organizados, visando imitá-los e aplicá-los em problemas computacionais difíceis na busca por soluções satisfatórias (HANDL; MEYER, 2007). Um dos comportamentos coletivos que mais inspiraram o desenvolvimento de algoritmos são aqueles baseados em colônias de formigas (DORIGO et al., 1996), abelhas (KARABOGA, 2005) e partículas (KENNEDY, 2000).

2.3.1 OTIMIZAÇÃO POR COLÔNIA DE FORMIGAS

Na natureza, as formigas tendem a seguir o menor caminho entre o formigueiro e a fonte de alimento. Durante a busca pelo alimento, as formigas depositam no ambiente uma substância volátil e olfativa denominada feromônio. O feromônio é uma substância secretada por insetos e mamíferos com funções de atração sexual para acasalamento, demarcação de trilhas ou comunicação entre indivíduos. Ao caminhar, uma formiga demarca seu caminho com essa substância e, ao fazer isso, as demais formigas percebem a intensidade desta substância no ambiente e tomam esta informação como critério para definir seus próximos passos. Este processo é denominado estigmergia.

A ideia central da Otimização por Colônia de Formigas (do inglês *Ant Colony Optimization* - ACO), proposta por Dorigo (1992), é imitar o comportamento das formigas para resolver problemas de otimização. A proposta do ACO é utilizar agentes que simulam o comportamento das formigas para gerar as soluções, onde o feromônio é utilizado para guiar estes agentes ajudando a definir os atributos que serão incluídas na solução. Além do feromônio, informações específicas do problema que será resolvido também são consideradas durante a busca.

A estrutura básica do ACO, de acordo com o apresentado em Talbi (2009), pode ser vista no Algoritmo 1, onde é possível observar o comportamento iterativo em que cada iteração compreende, basicamente, a construção das soluções e atualização do feromônio. A atualização do feromônio, por sua vez, é dividida em duas etapas: Evaporação e Reforço.

Na linha 1 do algoritmo é possível observar que o feromônio inicial precisa ser definido. Na maioria das heurísticas da literatura, o feromônio inicial é zero, representando que as formigas ainda não caminharam neste ambiente, mas formas diferentes de inicialização podem ser utilizadas.

No ACO padrão, a construção das soluções implica que cada agente, que representa

Algoritmo 1: Otimização por Colônia de Formigas

Parâmetros: numFormigas
return **:** melhorSolucao;

```
1 inicializaFeromonio(ambiente);
2 while critério de parada não satisfeito do
3 |   for cada formiga do
4 |   |   constroiSolucao(ambiente);
5 |   |   evaporcaoFeromonio(ambiente);
6 |   |   reforcoFeromonio(ambiente);
7 |   end for
8 end while
```

uma formiga, constrói uma solução decidindo cada uma das características que farão parte ou não da solução. Para determinar as características das soluções, as formigas consideram dois tipos de informação: o feromônio (τ) e características específicas do problema (η). A probabilidade de uma formiga sair de um estado i e ir para j é igual a p_{ij}, que é definida de acordo com a Equação 2.3, onde α representa a influência relativa do feromônio e β a influência heurística associada ao problema, e S representa o conjunto de soluções ainda não exploradas pelas formigas.

$$p_{ij} = \frac{\tau_{ij}^{\alpha} \times \eta_{ij}^{\beta}}{\sum_{k \in S} \tau_{ik}^{\alpha} \times \eta_{ik}^{\beta}}, \quad \forall \ j \in S \tag{2.3}$$

Na atualização do feromônio, linhas 5 e 6 do algoritmo, são realizadas as etapas de Evaporação e Reforço. No reforço, as trilhas escolhidas pelas formigas têm seu feromônio aumentado, ou seja, o reforço depende dos caminhos percorridos pelas formigas. Já na evaporação, para simular o comportamento volátil do feromônio e evitar a convergência prematura do algoritmo, o feromônio depositado em todo o ambiente sofre redução.

Ao implementar um ACO para resolver um problema específico, entre as decisões de projeto, a mais importante é definir a política para atualização do feromônio, principalmente no que se refere ao reforço, já que a evaporação do feromônio é uma ação que alcança o ambiente como um todo, pois esta é a única informação compartilhada por todas as formigas, e a medida que esta taxa é atualizada, as soluções vão sendo geradas. Sobre o reforço, as estratégias mais comuns incluem: mesma forma de atualização dos caminhos percorridos por cada uma das formigas; atualização proporcional a qualidade da solução obtida pela formiga; atualização apenas dos caminhos relacionados às formi-

gas que obtiveram as melhores soluções; e, por fim, a diminuição do feromônio associado ao caminho que levou à pior solução. Neste trabalho todos os caminhos percorridos são reforçados do mesmo modo.

O ACO é uma metaheurística que vem sendo utilizada, com sucesso, para tratar diferentes problemas de otimização com variadas aplicações, como no controle de tráfego aéreo (BASKAN; OZAN, 2013), escalonamento em sistema de manufatura (DRABOWSKI; WANTUCH, 2013), previsão de congestionamento de tráfego (KURIHARA, 2013), roteamento de veículos elétricos (JOO; LIM, 2018) e para o clássico problema do caixeiro viajante (ELVEZIA, 1996). Além destas aplicações, em Parpinelli et al. (202) os autores apresentam um ACO para o problema de identificar regras de classificação a partir de um conjunto de dados de treinamento onde formigas geram um conjunto ordenado de regras a serem utilizadas no conjunto de testes. A grande variedade de aplicações de ACO em diferentes áreas, motivou a escolha desta técnica como instrumento para tratar o problema alvo deste trabalho.

3 TRABALHOS RELACIONADOS

Tradicionalmente, técnicas de agrupamento são divididas em hierárquicas e particionais (BERKHIN, 2002). Os métodos hierárquicos geram diversas partições, enquanto os particionais produzem apenas uma (JAIN et al., 1999). De modo complementar à forma como o algoritmo produz a solução, determina também se o tipo de agrupamento é divisivo (*cluster* único sendo dividido até o critério de parada) ou aglomerativo (N grupos *singleton* recombinados entre si). Além disso, o agrupamento também pode ser classificado em:

- **Exclusivo**: cada objeto pertence a um único *cluster*. Agrupamentos que atendem a essa propriedade não permitem que um objeto pertença a mais de um grupo;
- **Não exclusivo**: os objetos podem ser alocados em diferentes *clusters*. De forma que, um objeto pertence a um ou mais grupos e, em geral, essas abordagens não apontam a qual grupo o objeto está mais relacionado;
- ***Fuzzy***: onde existem graus de pertinência em cada grupo. Agrupamentos que permitem que os objetos pertençam a mais de um grupo, e também indicam a aderência desses objetos aos grupos que eles foram alocados.

Na literatura existem diversos algoritmos que se propõem a resolver problemas de agrupamento, entre eles vale citar: o clássico *K-means* (MACQUEEN et al., 1967), que é amplamente utilizado devido a sua simplicidade e reduzido custo computacional; os algoritmos baseados em densidade *DBSCAN* (ESTER et al., 1996) e *OPTICS* (ANKERST et al., 1999) ; os hierárquicos *SLINK* (SIBSON, 1973), *COBWEB*(FISHER, 1987) e *CURE* (GUHA et al., 1998); e os métodos baseados em grade *BANG* (SCHIKUTA; ERHART, 1997) e *STING*(WANG et al., 1997). Além desses, existem os algoritmos evolutivos, tais como Algoritmo Genético (BANDYOPADHYAY; MAULIK, 2001) e Evolução Diferencial (DAS et al., 2008), que possuem diversas variantes desenvolvidas para a resolução do problema de agrupamento. As metaheurísticas *Simulated Annealing* (BANDYOPADHYAY et al., 2001) e *Tabu Search* (PAN; CHENG, 2007) também possuem aplicações voltadas ao problema de agrupamento automático.

Metaheurísticas populacionais, são processos iterativos que tentam melhorar uma população de soluções, tais técnicas baseadas nos paradigmas de computação evolutiva, inteligência de enxame e sistemas imunológicos artificiais; tornaram-se populares para solucionar o problema de agrupamento automático (JOSÉ-GARCÍA; GÓMEZ-FLORES, 2016). Uma vez que permitem uma maior diversificação de soluções por explorarem mais o espaço de busca. Nesse trabalho, uma metaheurística populacional baseada em inteligência de enxame, que visa imitar o comportamento das formigas é aplicada ao problema de agrupamento automático de dados.

O primeiro trabalho relacionado ao uso de ACO para problemas de agrupamento foi proposto por Deneubourg et al. (1990), onde a função que define a mudança do estado i para o estado j leva em conta a ideia de distância média entre os elementos do conjunto de dados, muito embora a distância considerada não é estabelecida com espaço de atributos. Mais tarde, Lumer e Faieta (1994) apresentaram um modelo em que a dissimilaridade entre pares de objeto é definida no espaço de atributos, como no presente trabalho. Este tipo de abordagem é conhecido na literatura como ACC, do inglês *Ant Colony Clustering*. A partir deste trabalho, diversos outros surgiram, principalmente no sentido de utilizar o feromônio para reduzir o tempo computacional ou melhorar a convergência do algoritmo (BORYCZKA, 2009) e (MONMARCHÉ et al., 1999).

Em Kuo et al. (2005) os autores apresentaram o *Ant K-Means*, um algoritmo *K-Means* adaptado que emprega um ACO para calcular a probabilidade de um dado objeto pertencer a um dado *cluster*. O valor da probabilidade é dado conforme o feromônio. Já no trabalho apresentado em Tiwari et al. (2010) os autores propuseram um ACO combinado com uma estratégia de busca local para o problema de agrupamento onde cada formiga codifica uma solução. O algoritmo mantém uma matriz de feromônio e a posição de cada formiga em relação aos *clusters* da solução.

Nesta dissertação, o método proposto é comparado a sete algoritmos, onde 3 deles utilizam algum conceito de lógica fuzzy para realizar o agrupamento sendo: *Hybrid FCM-PSO* (IZAKIAN; ABRAHAM, 2011), *Fuzzy Particle Swarm Optimization* (FPSO) (PANG et al., 2004), *Rough K-Means Discernibility* (RKMD) (LINGRAS; WEST, 2004), Algoritmo Evolutivo Construtivo com Busca Local (AECBL1)(CRUZ, 2010)e as três heurísticas propostas por Semaan (2013), ILS-FJGP 3.5, ILS-DBSCAN 3.6 e a Heurística Híbrida (HH) 3.7. Como essas abordagens foram utilizadas para comparação de resultados, alguns

conceitos importantes a respeito dos algoritmos citados são apresentados nas próximas seções.

3.1 HYBRID FCM-FPSO

No *K-means*, de forma estocástica alguns objetos são designados como centroides iniciais. Os demais objetos são alocados nos *clusters* com base na distância em relação aos centroides. Então os centroides são recalculados e esse processo se repete até que os *clusters* formados se estabilizem.

Fuzzy C-Means (FCM) pode ser visto como uma variante do *K-means* que utiliza lógica fuzzy para a formação das partições. Nesta abordagem, cada objeto é vinculado a múltiplos *clusters* com diferentes graus de pertinência. O Hybrid FCM-FPSO (IZAKIAN; ABRAHAM, 2011) é uma junção dos métodos FPSO com FCM, onde o FPSO produz o particionamento e os centroides iniciais que alimentam o FCM.

3.2 FUZZY PARTICLE SWARM OPTIMIZATION (FPSO)

Particle Swarm Optimization (PSO) é um algoritmo populacional onde os indivíduos são representados por partículas que, por sua vez, simbolizam as possíveis soluções para o problema. A inicialização da população é feita de forma aleatória onde cada partícula possui velocidade e posição específica no espaço de soluções. Em um processo iterativo a busca pelo posicionamento ideal das partículas é feita com base em uma função de *fitness* onde a partícula tem sua posição atualizada.

Em Pang et al. (2004) um algoritmo híbrido que mescla PSO com o FCM, denominado *Fuzzy Particle Swarm Optimization* (FPSO), foi apresentado. Onde o posicionamento e a velocidade das partículas foram alteradas de forma a representar o relacionamento difuso entre os atributos (IZAKIAN; ABRAHAM, 2011). Esse algoritmo foi aplicado ao *Travelling Salesman Problem* e uma adaptação para agrupamento pode ser vista em Izakian e Abraham (2011).

3.3 ROUGH K-MEANS DISCERNIBILITY (RKMD)

O *Rough K-Means* (RKM) se concentra em distinguir objetos nítidos e vagos, onde o *cluster* é visto como um *cluster* de intervalos. O objeto é dividido na aproximação mais

baixa, onde o objeto é certamente um membro do *cluster* e a região do limite, onde o objeto é um membro de mais de um *cluster*. RKM é derivado da integração da *Rough Set Theory* com o *K-means* (LINGRAS; WEST, 2004).

O algoritmo *Rough K-Means Discernibility* (RKMD) aborda a otimização e vários tipos de formação de *cluster*, realizando o cálculo da partição aproximada para produzir uma partição melhor no conjunto de dados de sobreposição. Empregando primeiro o RKM, que é otimizado usando o conceito de semente inicial para gerar o intervalo apropriado, e depois efetuando o cálculo de discernibilidade, o *rough membership degree* é calculado como a probabilidade do objeto incerto pertencente a cada *cluster* (SETYOHADI et al., 2015).

3.4 ALGORITMO EVOLUTIVO CONSTRUTIVO COM BUSCA LOCAL (AECBL1)

O Algoritmo Evolutivo Construtivo com Busca Local proposto por Cruz (2010), é composto por duas etapas, a construtiva e a evolutiva. Onde, a fase de construção é composta por dois processos: O de Formação de Grupos Parciais (FGP) e a Junção Grupos Parciais (JGP) que tem a função de gerar*clusters* parciais com base em um critério de densidade estipulado, cuja a finalidade é reduzir a dimensão dos dados. Nesta etapa, também é realizada uma agregação de *clusters* parciais de menor cardinalidade.

Já o módulo evolutivo, consiste na implementação de um Algoritmo Genético que utiliza memória adaptativa em suas iterações, bem como diversos tipos de buscas locais tais como: Inversão Individual, Troca entre Pares e *Path Relinking* com a finalidade de refinar as soluções geradas pelo AG.

3.5 ILS-FJGP

O ILS (Iteraded Local Search) definido por Lourenço et al. (2010) é uma metaheurística baseada na ideia que, uma solução local ótima pode ser melhorada aplicando algum procedimento de pertubação que gere uma nova solução, tomando como base a anterior e aplicando a busca local nessa nova solução, assim sucessivamente até que algum critério de parada seja satisfeito.

O Algoritmo ILS-FJGP (ILS- Formação e Junção de Grupos Parciais) apresentado

por Semaan (2013) pode ser definido como uma junção do ILS com a etapa construtiva do 3.4. Onde a solução ótima local inicial é gerada pela etapa construtiva, definida no algoritmo AECBL1, de forma que o ILS atua nesse subespaço de soluções geradas pela fase de construção.

3.6 ILS-DBSCAN

Density Based Spatial Clustering of Application with Noise (DBSCAN) é um método baseado em densidade, capaz de identificar os ruídos dos dados. Proposto por Ester et al. (1996), a ideia chave é que, para cada ponto de um *cluster*, a vizinhança de um determinado raio deve conter pelo menos um número f de pontos, ou seja, a densidade na vizinhança tem que exceder um limiar.

ILS-DBSCAN apresentado por Semaan (2013) é uma junção da metaheurística ILS com o método DBSCAN, de forma que grupos parciais são formados pelo dbscan, onde, todos os objetos devem pertencer a um grupo, inclusive os objetos classificados como ruídos. Essas soluções são refinadas pelo ILS em busca de uma solução ótima global ou um ótimo local de excelente qualidade.

3.7 HEURÍSTICA HÍBRIDA (HH)

A Heurística Híbrida (HH) definida por Semaan (2013) utiliza a Estatística de *Hopkins*, para identificar se em determinado *dataset* existe uma tendência à formação de agrupamentos. Para *datasets* que possuem essa tendência, O HH aplica o algoritmo 3.6 sob os dados de entrada. Caso negativo, o HH utiliza o 3.4 para realizar o agrupamento dos dados. A ideia é que a HH direcione o *dataset* submetida a heurística mais adequada.

4 ALGORITMO ANTHILL

Neste capítulo é apresentado um método baseado no comportamento coletivo das formigas, aplicado ao problema de agrupamento de dados, denominado *Anthill*. Esse método foi baseado no trabalho de Chen et al. (2005), no qual os autores apresentam uma proposta de ACO para agrupamento, denominada *Ant-Cluster*. Como forma de acelerar o processo de obtenção de soluções, o algoritmo proposto neste trabalho inclui estratégias adaptativas, além de mecanismos que permitem a obtenção de várias soluções candidatas, de forma a prover ao especialista da área diferentes soluções, o que em muitas aplicações é essencial no processo de tomada de decisão.

Convém destacar que, a exemplo da abordagem apresentada em Chen et al. (2005), o algoritmo proposto difere do algoritmo de Colônia de Formigas original proposto por Dorigo (1992), onde uma formiga é um agente artificial que mapeia uma solução do problema. No algoritmo proposto, os dados de entrada são modelados de forma a simular o ambiente onde os agentes (formigas) atuarão de forma colaborativa no sentido de alterá-lo gradativamente até que o próprio ambiente se configure numa solução do problema. Neste sentido, o comportamento colaborativo das formigas no algoritmo proposto e no trabalho de Chen et al. (2005) é equivalente ao de grupos de castores, que atuam coletivamente alterando o ambiente ao construírem diques necessários à segurança de todo o grupo.

Na Tabela 4.1 é apresentada a simbologia utilizada nesse trabalho.

Tabela 4.1: Parâmetros utilizados no Anthill

Parâmetros	**Descrição**
$\tau_{ij}(t)$	Feromônio depositado em cada arco ij na iteração t
η_{ij}	Função heurística
α	Efeito da taxa de τ_{ij}
β	Proporção de η_{ij}
q	Valor aleatório [0 - 1]
$q0$	Probabilidade de escolha gulosa
Q	Reforço dos arcos onde as formigas passam
ρ	Taxa de evaporação do feromônio
$limiar$	Limiar de corte
$numFormigas$	Número de formigas

Para uma melhor compreensão do modelo adotado em Chen et al. (2005) e neste trabalho, considere $X = \{X_1, X_2, \ldots, X_n\}$ o conjunto de objetos da entrada do problema, onde cada objeto X_j é definido como um vetor $X_j = (x_{j1}, x_{j2}, \ldots, x_{jd})$ no espaço $\mathbb{R}^d$ de atributos. Seja $D = (X, A, w)$ um grafo orientado ponderado nos arcos no qual o conjunto de nós é dado pelo conjunto de objetos X e $w : X \times X \rightarrow \mathbb{R}$ é uma função que atribui a cada arco $(X_i, X_j) \in A$ um valor que indica o quanto os objetos X_i e X_j, nesta ordem, são similares.

A ideia básica do algoritmo é utilizar a informação referente à similaridade entre pares de objetos como critério na escolha de cada formiga ao trilhar os arcos do digrafo. Assim, arcos entre nós com maior similaridade tendem a ser mais explorados, aumentando a concentração de feromônio, enquanto os arcos entre nós com menor valor de similaridade tendem a ser menos explorados e, no decorrer das iterações do algoritmo, são eliminados, gerando componentes fortemente conectadas[1] que caracterizarão os *clusters* da solução.

Para se obter a similaridade entre pares de objetos, inicialmente calcula-se a dissimilaridade *diff(i,j)* entre todos os pares (X_i, X_j) de vértices de X a partir da distância euclidiana, conforme a Equação 4.1.

$$diff\,(i,j) = \sqrt{\sum_{v=1}^{d} (x_{iv} - x_{jv})^2} \tag{4.1}$$

A similaridade entre dois objetos de dados X_i e X_j é obtida pela Equação 4.2 que, além da dissimilaridade entre X_i e X_j, considera também o maior valor de dissimilaridade dentre os pares de objetos de X.

$$Sim\,(i,j) = 1 - \frac{diff\,(i,j)}{max\ diff}, \tag{4.2}$$

onde *max diff* indica o maior valor de dissimilaridade dentre todos os pares de objetos do conjunto de entrada.

Como pode ser observado a partir da Equação 4.2, caso se utilize diretamente os valores de $Sim(i,j)$ para todo par de vértices (X_i, X_j) chegaria-se ao caso em que o $w\,(X_i, X_j) = w\,(X_j, X_i)$, já que nenhuma informação referente ao próprio nó em relação aos demais foi considerada no cálculo e isso levaria ao caso de se poder considerar o

[1] Uma componente fortemente conexa em um digrafo D=(V,A) é qualquer subconjunto maximal $C \subseteq V$ tal que, para qualquer par de vértices $i, j \in C$, existe um caminho direcionado de i para j e de j para i.

grafo não direcionado. Assim, o peso dos arcos do digrafo D é tal que também considera a similaridade entre pares de objetos, mas é calculada levando-se em consideração os valores individuais de similaridade de cada objeto em relação a todos os demais. Para tal, a similaridade média (*mediasim*) e máxima (*max sim*) de um dado objeto X_i são obtidas a partir das Equações 4.3 e 4.4, respectivamente.

$$mediasim(i) = \frac{1}{n} \sum_{j=1}^{n} Sim\,(i,\ j) \tag{4.3}$$

$$max\,sim\,(i) = \max_{1 \leq j \leq n} Sim\,(i, j) \tag{4.4}$$

Uma vez estabelecidos aos valores de $mediasim(i)$ e $max\ sim(i)$ para todos os vértices X_i do digrafo, o peso de cada arco (X_i, X_j) é calculado de forma a expressar o que o algoritmo considera como similares entre dois objetos do conjunto de entrada quando não se tem ainda nenhuma informação sobre os dados obtida coletivamente pelos agentes artificiais. Ou seja, para que as formigas iniciem a exploração das trilhas no digrafo é necessário que se defina quão interessante é cada arco para que cada formiga decida qual caminho deverá tomar. Assim, para um dado objeto X_i, o peso do arco (X_i, X_j) para todo $X_j \in X$, onde $i \neq j$, denotado *taxaAceitacao*(i,j), indica a similaridade do objeto (X_i) em relação ao objeto (X_j) e pode ser obtida pela Equação 4.5.

$$taxaAceitacao\,(i\,,j\,) = Sim\,(i\,,j\,) - \frac{1}{4}\,(mediasim\,(i) + max\,sim\,(i)) \tag{4.5}$$

Note que esta medida de similaridade não é simétrica. Estratégia similar foi adotada em Chen et al. (2005). Entretanto, os autores daquele trabalho, diferente do que se obtém na Equação 4.4 com a similaridade máxima do objeto X_i, consideram a similaridade mínima entre os dois objetos. Pela Equação 4.5, pode-se ver que o peso do arco (X_i, X_j) somente terá valor positivo se a similaridade $Sim(i, j)$ for maior que 25% do somatório da média das similaridades do objeto X_i com o maior valor de similaridade associado ao mesmo. Isto permite que na primeira iteração do algoritmo um número considerável de arcos seja removido, evitando-se trabalhar com o digrafo completo. O valor da constante foi fixado em 1/4 empiricamente a partir da avaliação da relação custo/benefício entre a qualidade das soluções e o tempo de processamento do algoritmo.

A Equação 4.5 indica que quanto mais similares forem dois objetos, maior será a

taxa de aceitação entre os pares de objetos. Esta taxa é utilizada como valor inicial do feromônio em cada arco do digrafo.

Como já mencionado, a ideia básica do algoritmo é permitir que um conjunto de formigas atue sobre o digrafo de forma a indicar ao algoritmo quais os arcos mais significativos na definição do que vem a ser dois objetos pertencerem a um mesmo cluster. Para tal, cada formiga é lançada aleatoriamente sobre um dado nó X_i do digrafo (objeto do conjunto de entrada) e deverá decidir a cada iteração qual arco utilizará para dirigir-se a um outro nó até que todo nó seja explorado. Esta decisão leva em consideração a quantidade de feromônio depositada em cada arco com origem em X_i, bem como uma informação heurística associada à similaridade entre o objeto X_i e cada objeto X_j para o qual exista arco com origem no mesmo.

Para guiar a formiga na escolha da trilha que deve seguir, uma função de probabilidade é utilizada. Assim, considere um conjunto $F = \{1, \ldots, m\}$ de formigas. Para cada formiga $k \in F$ que se encontra em dado vértice $X_i \in X$, um valor **q** no intervalo [0,1] é obtido aleatoriamente. O nó X_j para o qual a formiga k se movimentará é definido a partir da Equação 4.6, onde α e β são parâmetros que determinam o peso atribuído ao feromônio e à informação heurística na escolha do arco a ser tomado. Se o valor de q sorteado for menor que a constante q_0, então a formiga **k** se move para o arco com maior valor na relação entre feromônio e a informação heurística. Caso contrário, um arco de A com origem em X_i é obtido aleatoriamente.

$$j = \begin{cases} \arg\max_{u \in \text{ aos vizinhos } O_i} [\tau_{iu}^{\alpha}(t)\, \eta_{iu}^{\beta}], \text{ quando } q \leq q_0 \\ \\ \text{Caso contrário, seleciona aleatoriamente.} \end{cases} \tag{4.6}$$

Essa estratégia baseia-se na apresentada em Chen et al. (2005), pela qual, caso $q > q_0$, um sorteio ponderado é realizado e os arcos que apresentam maior quantidade de feromônio acumulado e uma alta taxa de aceitação possuem mais chance de serem escolhidos. Testes computacionais preliminares que inspiraram este trabalho, mostraram que pela abordagem dos autores, a probabilidade de descoberta de novas soluções ficava prejudicada devido ao caráter excessivamente guloso do algoritmo.

A função heurística, definida pela Equação 4.7, consiste no valor da similaridade entre dois objetos, dada pela Equação 4.2. Esta função tem por objetivo refletir o quão bom é

para a solução do problema a decisão de seguir por determinado caminho considerando as características do problema, neste caso, a similaridade entre os dois nós em que incide o arco. Estes valores são calculados durante a montagem do digrafo e mantém-se fixos durante toda a execução do algoritmo.

$$\eta_{ij} = Sim\,(i, j) \tag{4.7}$$

O processo de atualização do feromônio inclui a evaporação e o reforço. Ao fim de cada iteração, após todas as formigas percorrerem o digrafo depositando o feromônio nos arcos por onde passaram, os valores são atualizados conforme a Equação 4.8, onde ρ é o coeficiente de evaporação, definido no intervalo $[0, 1]$ e Δ será descrito na equação 4.9.

$$\tau_{ij}\,(t+1) = (1-\rho)\,\tau_{ij}\,(t) + \sum_{v=1}^{m} \Delta\tau_{ij}^{k}, \tag{4.8}$$

A cada iteração, o algoritmo mantém uma lista dos arcos que foram utilizados pela formigas na iteração anterior e o reforço do feromônio é aplicado apenas nesses arcos, o que diminui o tempo de processamento. Por outro lado, a evaporação é aplicada em todos os arcos, inclusive naqueles pelos quais nenhuma formiga tenha trilhado. A evaporação permite, ao longo do tempo, que o digrafo tenha os arcos de baixa intensidade de feromônio removidos, tornando-o a cada procedimento de remoção de arcos mais esparso. A medida que o digrafo vai perdendo arcos pouco utilizados pelo conjunto de formigas, menos caminhos direcionados entre pares de nós existem no mesmo. Neste sentido, o digrafo construído no início do algoritmo e que consistia em uma única componente conexa, ao longo do processo iterativo composto por reforço e evaporação, seguido da remoção de arcos, vai perdendo a conexidade e se tornando um conjunto de componentes fortemente conexas. Neste sentido, uma característica do algoritmo proposto é que pode-se estabelecer *apriori* quantas soluções diferentes o usuário deseja obter, já que cada conjunto de componentes conexas obtido após o procedimento de remoção de arcos consiste, na verdade, em um agrupamento em que cada componente é um *cluster* da solução gerada. De forma a indicar para o conjunto de formigas na próxima iteração quais os arcos mais favoráveis a permanecerem no digrafo, o algoritmo integra o procedimento de reforço do

feromônio nos arcos, o que é feito segundo a Equação 4.9.

$$\Delta\tau_{ij}^{k} = \begin{cases} Q.Sim\,(i,j)\,, & \text{Se a formiga k passou no arco } (i,j) \\ 0, & \text{caso contrário} \end{cases} \tag{4.9}$$

A atualização dos parâmetros α e β é feita adaptativamente. Então a quantidade média de feromônio no digrafo é calculado (Equação 4.10). Este valor é utilizado no cálculo do peso de distribuição de feromônio conforme a Equação 4.11.

$$\tau_{media} = \frac{\sum_{(i,j)\in A}\ \tau_{i,j}}{|A|} \tag{4.10}$$

$$\psi = \frac{1}{|A|}\left[\sum_{(i,j)\in A}\left(\tau_{media} - \tau_{ij}\right)^2\right]^{\frac{1}{2}} \tag{4.11}$$

Usando o peso de distribuição de feromônio, o algoritmo atualiza o valor de α e β da seguinte maneira:

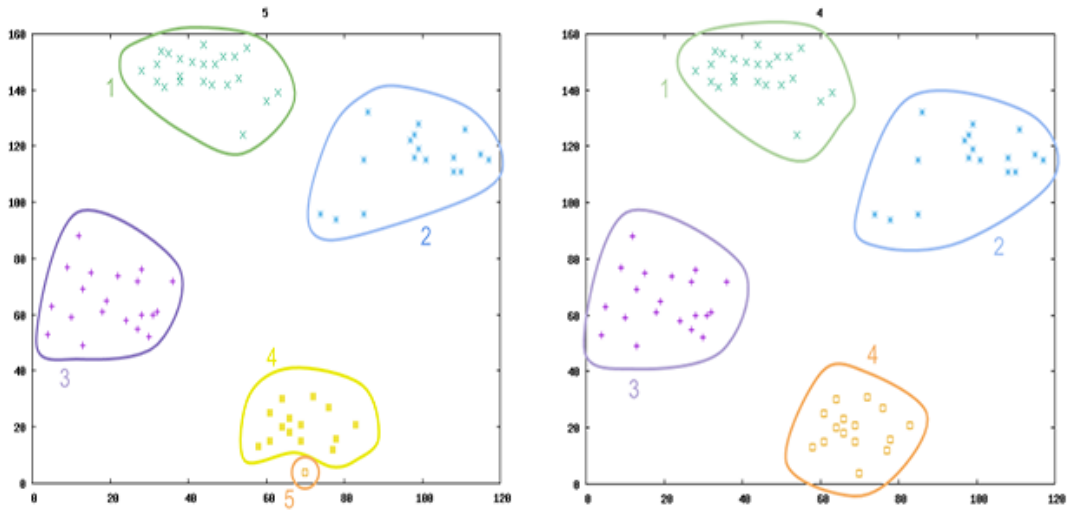
$$\alpha = e^{-\psi} \tag{4.12}$$

$$\beta = \frac{1}{\alpha} \tag{4.13}$$

Com a solução sendo formada de modo gradativo, a medida em que o digrafo vai se tornando mais esparso, o número de arcos que podem ser explorados a partir de caminhos direcionados torna-se cada vez mais dependente da posição inicial de cada formiga. Assim, um bom posicionamento inicial das formigas no digrafo aumenta as chances de melhoria na qualidade das soluções geradas. No início da execução do algoritmo, uma vez que o digrafo consiste em uma única componente fortemente conexa, a distribuição inicial das formigas é feita de forma totalmente aleatória. Entretanto, uma vez que a remoção de arcos tenha levado ao surgimento de um número de componentes maior que 1, no pior caso pode ocorrer de todas as formigas serem lançadas em uma única componente e isso levaria a uma situação em que o procedimento e reforço não seria aplicado nos arcos incidentes nos nós das demais componentes e os mesmos sofreriam apenas a ação do procedimento de evaporação, distorcendo assim a qualidade da solução. Para corrigir esta anomalia, ao término de cada iteração o algoritmo identifica todas as componentes fortemente conexas

do digrafo e, com base na quantidade de nós de cada uma, realiza a distribuição do total de formigas de forma que o número de formigas de uma componente é proporcional ao número de nós da mesma.

Devido à natureza colaborativa do algoritmo, que leva à geração de soluções de forma coletiva, eventualmente pode acontecer a formação de alguns grupos com um único objeto (*singletons*). Este comportamento ocorre quando nós do digrafo tem todos os seus arcos de saída pouco explorados em muitas iterações, o que acaba levando à remoção destes arcos devido à quantidade de feromônio abaixo do limiar de corte e a consequente formação de nós sumidouros, que consistem em uma componente fortemente conexa com um único nó. Embora para algumas entradas isso possa acontecer devido a existência de *outliers*, a ocorrência de diversos *clusters* com um único objeto pode significar uma anomalia no processo de exploração dos arcos. De forma a incluir no algoritmo proposto um comportamento adaptativo que permita contornar o problema, foi desenvolvido o módulo denotado *clusterMin*, que identifica os elementos que se desprenderam de seus *clusters* originais de forma precoce e realocando os mesmos no *cluster* mais próximo. Um exemplo da aplicação do módulo de correção *clusterMin* pode ser visualizada na Figura 4.1, onde é possível observar a ocorrência de um *cluster* com um único objeto na Figura 4.1a e a configuração dos *clusters* após a correção na Figura 4.1b.



(a) Cluster singleton (b) Realocação do objeto desconectado

Figura 4.1: Exemplo da aplicação do módulo *clusterMin*

O pseudocódigo do Algoritmo *Anthill* pode ser observado no Algoritmo 2. O laço compreendido entre as linhas 2 e 22 controla o critério de parada do algoritmo. O cami-

nhamento de cada uma das formigas é realizado pelo laço entre as linhas 3 e 8, ao passo que na linha 9 ocorre a atualização do feromônio. Na estrutura de desvio condicional da linha 10 verifica-se se a iteração atual é múltipla de *tt*. Caso seja, o digrafo passa por uma limpeza com a remoção dos arcos cujo valor do feromônio seja inferior ao limiar. Em seguida, os parâmetros do algoritmo são ajustados e uma solução para o problema é verificada.

Algoritmo 2: AntHill

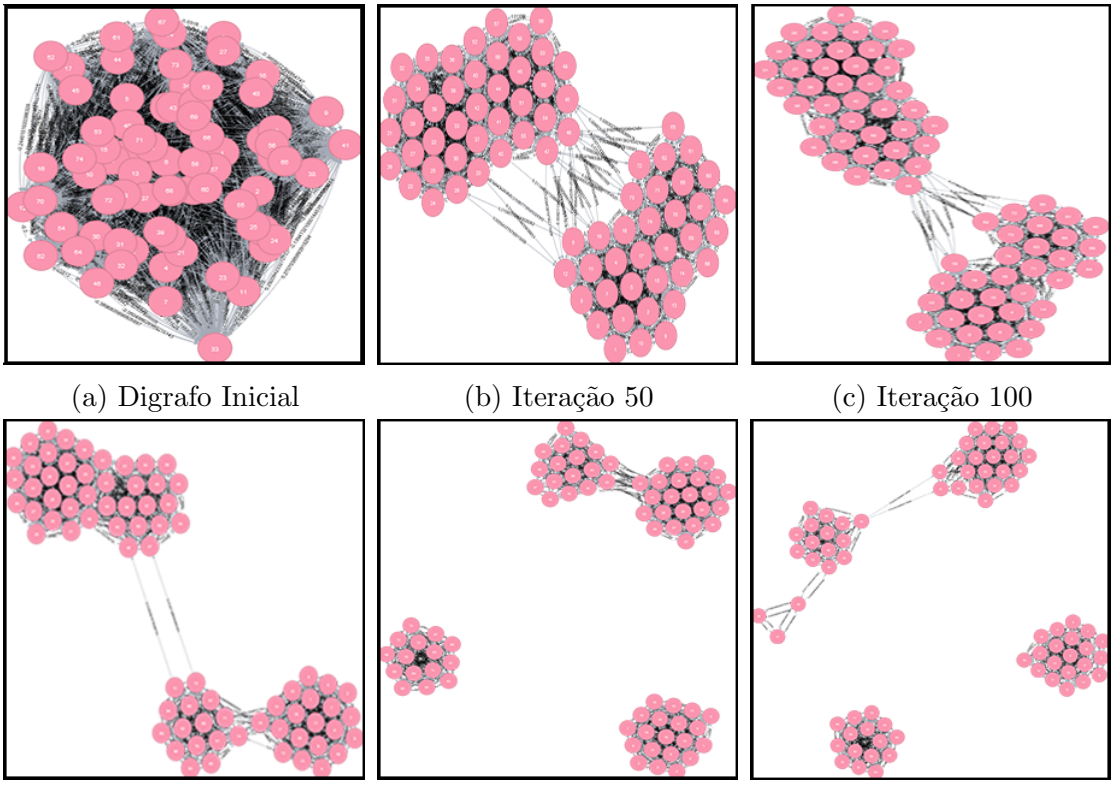
Parâmetros: alfa, beta, q, q0, Q, p, limiar, iterMax, numFormigas,tt

```
1  configInicialParametros(α, β, q, q0, Q, p, limiar, iterMax, numFormigas, tt);
2  while critério de parada não satisfeito do
3      for cada formiga k do
4          noInicial ← selecNoInicial();
5          while existe vizinho não visitado do
6              proximoNo ← selecionarProximoNo(Equação 4.6);
7          end while
8      end for
9      atualizaFeromonio(arcos);
10     if iterAtual múltiplo de tt then
11         for cada arco (i,j) ∈ D do
12             if consultaFeromonio(i,j) < limiar then
13                 removeArco(i,j);
14             end if
15         end for
16         atualizaParametros(Equações 4.10, 4.11, 4.12 e 4.13);
17         solucaoParcial ← obtemComponentes(D);
18         if Silhueta(solucaooParcial) > Silhueta(melhorSolucao) then
19             melhorSolucao ← solucaooParcial;
20         end if
21     end if
22 end while
   return      : melhorSolucao;
```

Para se evitar o gasto excessivo de tempo de execução sem a atualização da melhor solução, utilizou-se como critério de parada, além do número máximo de iterações, um segundo critério, pelo qual avalia-se a qualidade da nova solução segundo o índice silhueta em relação à qualidade da melhor solução até então obtida e a variação brusca de uma iteração para outra no número de clusters gerados.

O teste de validação inicial foi executado no *dataset* Ruspini e na Figura 4.2 é possível ver todo o processo executado pelo ACO. Para este *dataset*, o algoritmo obteve como resultado uma solução com k = 4 clusters, cujo valor de silhueta foi de **0,738**. Note, pela

Figura 4.2f, que ao final da execução do algoritmo ainda existem arcos ligando diferentes componentes conexas. Entretanto, tais arcos possuem apenas uma direção e, portanto, não é possível unir os nós destas componentes em um mesmo cluster, já que não há caminhos mutuamente direcionados entre pares de vértices de diferentes clusters.



(a) Digrafo Inicial (b) Iteração 50 (c) Iteração 100

(d) Iteração 150 (e) Iteração 200 (f) Iteração final

Figura 4.2: Teste de validação usando o dataset Ruspini

5 EXPERIMENTOS COMPUTACIONAIS DO ALGORITMO ANTHILL

Nessa seção são apresentados os resultados obtidos após a submissão do *Anthill* a testes computacionais. O algoritmo recebeu como entrada diversos *datasets* da literatura e seu desempenho é analisado sob três aspectos: custo computacional, validação interna (índice Silhueta) e inspeção visual.

O algoritmo *Anthill* foi desenvolvido utilizando a Linguagem Java. Os resultados preliminares foram obtidos através da execução do algoritmo em um computador dotado de 8GB de memória RAM, processador i5 de 2,53 GHZ e sistema operacional Windows 7 professional para arquiteturas de 64 bits.

Os experimentos podem ser divididos em duas nuances com diferentes objetivos, sendo eles: Resultados Computacionais Etapa 01 e Resultados Computacionais Etapa 02. Na primeira etapa de experimentos *Anthill* é avaliado quanto à capacidade de particionar dados, bem como o custo computacional envolvido . Nessa análise preliminar, foram utilizados 5 *datasets* clássicos obtidos do repositório da UCI Machine Learning Data, os resultados destes experimentos iniciais são apresentados na Seção 5.1. Já na segunda etapa, o objetivo dos experimentos realizados é permitir uma análise mais detalhada do comportamento do algoritmo com diferentes distribuições dos dados no espaço. Para tal, foram utilizados 80 *datasets* da literatura, os detalhes deste experimento são apresentados na Seção 5.2.

5.1 RESULTADOS COMPUTACIONAIS: ETAPA 01

Nesta primeira avaliação, o objetivo foi analisar o comportamento do algoritmo utilizando bases de dados clássicas para verificar se o mesmo é capaz de encontrar soluções compatíveis com as da literatura. Esses resultados foram publicados por Pacheco et al. (2018). Nesta etapa, foram utilizados os seguintes *datasets*:

- **Wine Dataset**: O conjunto de dados é o resultado de uma análise química dos vinhos cultivados em uma mesma região da Itália, mas derivados de três cultivos diferentes.

- **Iris Dataset**: O conjunto de dados contém 3 classes de 50 objetos cada, onde cada classe se refere a um tipo de planta de íris. Uma classe é linearmente separável das outras duas, que não são linearmente separáveis entre si.

- **Breast Cancer Wisconsin (Original) DataSet**: O conjunto de dados apresenta informações sobre casos de câncer de mama e foram obtidas dos Hospitais da Universidade de Wisconsin, Madison. O conjunto de dados é constituído de 9 atributos que caracterizam duas classes de tumor: maligno, ou benigno.

- **Pima Indians Diabetes Dataset**: O conjunto de dados, contém casos de estudos realizados sobre pacientes com pelo menos 21 anos de idade e com diabetes mellitus hereditária, sendo constituído de dois grupos de dados.

- **Haberman's Survival Dataset**: O conjunto de dados, contém casos de estudos realizados sobre a sobrevivência de pacientes submetidos a cirurgia para câncer de mama.

As características dos conjuntos de dados utilizados são mostradas na Tabela 5.1, que apresenta informações sobre o número de atributos, classes e objetos que estão contidos em cada dataset.

Tabela 5.1: Datasets utilizados no experimento inicial

Dataset	**Nº Atributos**	**Nº Classes**	**Nº Objetos**
Haberman	3	2	306
Iris	4	3	150
Pima	8	2	768
Wine	13	3	178
Winsconsin	9	2	683

5.1.1 EXPERIMENTOS

O desempenho do *Anthill* foi analisado em dois quesitos: inspeção visual e validação interna (silhueta). Na validação interna, o índice silhueta é calculado baseado na coesão e na separação de cada *cluster*. Uma interpretação possível para a métrica é encontrada em (KAUFMAN; ROUSSEEUW, 2009), onde valores de silhueta maiores que 0,7 indicam

que uma estrutura forte foi encontrada, valores no intervalo [0,51 - 0,7] indicam um particionamento razoável, já para valores entre [0,26 – 0,50] a estrutura é fraca e valores de silhueta menores que 0,25 apontam a ausência de um agrupamento substancial.

O algoritmo foi executado 100 vezes para cada *dataset* e em cada execução foi utilizada uma semente de randomização diferente. A Tabela 5.2 apresenta os resultados comparativos entre o algoritmo proposto (coluna *Anthill*) e outras três abordagens (colunas FPSO, FCM-FPSO e RKMD) em relação à média e desvio padrão do índice silhueta, onde os maiores valores médios de silhueta estão destacados em negrito. Pode-se observar que o *Anthill* encontrou uma estrutura forte em dois, dos cinco *datasets* (Wisconsin e Pima), com valores superiores á 0,7. Em 4 observações, encontramos valores acima de 0,6 e o *Anthill*, obteve os maiores valores de silhueta em pelo menos 80% dos testes realizados. Observa-se ainda que apenas para o *dataset* Wine o algoritmo proposto não obteve solução melhor que todos os demais algoritmo.

Tabela 5.2: Comparação dos valores médios do índice silhueta e desvio padrão

Dataset	**FPSO**	**FCM-FPSO**	**RKMD**	**Anthill**
Haberman	0.0061	0.3508	0.4089	**0.6712**
	±(0.0030)	±(0.0649)	±(0.0135)	±(0.0089)
Iris	-0.0276	0.4671	0.5465	**0.6870**
	±(0.0091)	±(0.0688)	±(0.0000)	±(0.0000)
Pima	0.0025	0.4084	0.5099	**0.7456**
	±(0.0014)	±(0.0900)	±(0.0002)	±(0.0002)
Wine	-0.0333	0.4296	**0.5632**	0.5390
	±(0.0120)	±(0.0121)	±(0.0000)	±(0.0001)
Wisconsin	0.0030	0.4383	0.5192	**0.7023**
	±(0.0025)	±(0.1335)	±(0.0000)	±(0.0000)

5.1.2 ANÁLISE ESTATÍSTICA

Para efeitos de comparação, foi utilizado o teste de *Wilcoxon*, que se baseia nos postos das diferenças intra pares para dados pareados e tem como característica estabelecer uma ideia aproximada do significado das diferenças em experimentos desse tipo (WILCOXON, 1945), possibilitando assim, quantificar a qualidade das soluções apresentadas pelo algoritmo *Anthill*, em relação aos algoritmos da literatura Hybrid FCM-PSO, FPSO

e (RKMD), descritos nas Seções 3.1, 3.2 e 3.3, respectivamente.

Para tal, foi utilizado a linguagem de programação R, desenvolvida especificamente para cálculos estatísticos e visualização gráfica de resultados. Para esta análise comparativa entre os algoritmos utilizou-se os mesmos dados apresentados na Tabela 5.2. Os resultados obtidos de tais comparações é denominado por p-value, e para determinar se a diferença entre a mediana dos dados comparados e a mediana hipotética são estatisticamente significativas, o nível de significância (α) utilizado, foi de 0,05 ou 5%, valor este que nos permite comparar o resultado de p-value de como segue:

- Se p-value $\leqslant \alpha$, a diferença entre as medianas é significativa;
- Caso contrário, não há diferença significativa entre elas.

Para esta análise estatística, tomou-se como hipótese a premissa de que os valores de silhueta encontrados pelo algoritmo proposto possuem relevância estatística. Assim, uma vez que o valor de p-value obtido foi de 0.03125, pode-se concluir que as diferenças dos valores médios de silhueta obtidos pelo *Anthill* foram estatisticamente significativos em relação ao Hybrid FCM-PSO e ao FPSO, uma vez que o p-value foi menor que (α) em ambos os testes. Por outro lado, ao se comparar o algoritmo proposto com o algoritmo RKMD o valor de p-value obtido foi de 0.06215, portanto maior que (α), mostrando assim que não há uma diferença significativa entre estes algoritmos.

5.1.3 CONFIGURAÇÃO IDEAL DE PARÂMETROS

Definir parâmetros de metaheurísticas de forma empírica não é recomendado, visto que os parâmetros podem não ser os melhores, prejudicando o desempenho do algoritmo. Para evitar esse problema, neste trabalho utilizou-se um calibrador de parâmetros denominado *Iterated racing for automatic algorithm configuration* (Irace) (LÓPEZ-IBÁNEZ et al., 2011). O objetivo de sua utilização é gerar configurações candidatas a partir dos parâmetros de entrada utilizados no algoritmo a ser configurado. Para tal, o calibador é guiado por uma função de custo e testes estatísticos, fornecendo como saída, uma lista de configurações candidatas ordenada segundo a qualidade das soluções obtidas, onde cada elemento da lista consiste numa combinação de parâmetros que melhor se adaptaram ao algoritmo. Após a execução do Irace, os parâmetros obtidos para a execução do *Anthill*

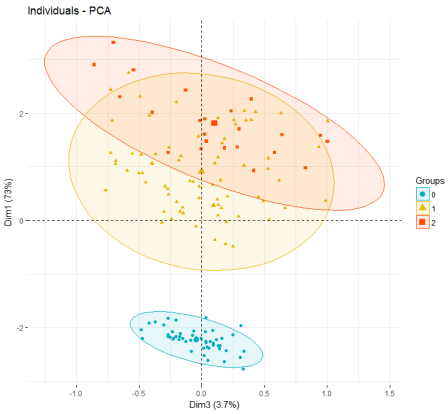
foram: $\alpha = 1.0$, $\beta = 3.0$, $q0 = 0.95$, $Q = 0.3$, $\rho = 0.01$, $limiar = 0.5$, $iterMax = 1000$, $numFormigas = 100$.

5.1.4 INSPEÇÃO VISUAL DE AGRUPAMENTOS

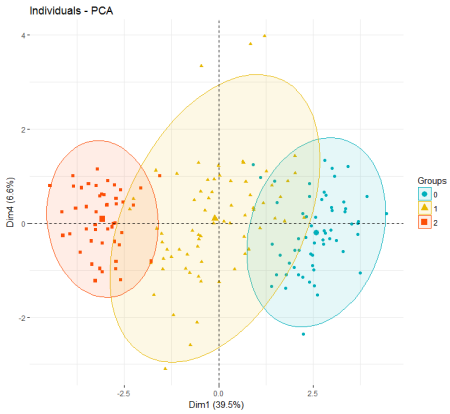
O objetivo primário da análise de componentes principais (ACP) é reduzir as dimensões do espaço de observação. A redução é obtida criando novas combinações lineares de variáveis que caracterizam os objetos estudados (MACKIEWICZ; RATAJCZAK, 1993). Dados de alta dimensionalidade, comuns em nossos dias atuais, são difíceis de serem analisados visualmente e, por isso, técnicas que permitam manipular dados multidimensionais são de grande valia, principalmente no domínio da Mineração de Dados. Dado um conjunto de dados, o ACP encontra a representação linear das observações, de forma que a variância dos dados reconstruídos seja preservada.

Com o uso do ACP, gerou-se os gráficos em hiperplanos de baixa dimensão para facilitar a visualização dos agrupamentos obtidos. A partir daí, analisou-se visualmente os grupos formados pelo *Anthill*. Na Figura 5.1 pode-se observar que, apesar dos *datasets* utilizados para fins de validação da proposta presentes na Tabela 5.1 possuírem conjuntos não linearmente separáveis e com algumas sobreposições em seus elementos, o *Anthill* foi capaz de particionar seus elementos de forma satisfatória.

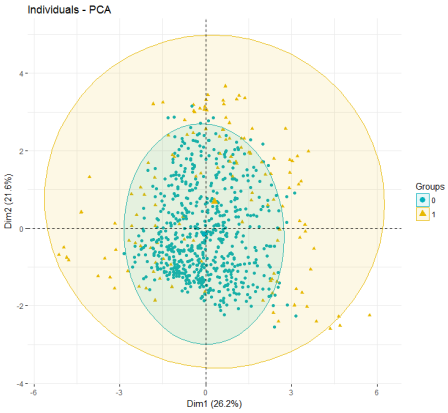
As cores na Figura 5.1 indicam os diferentes *clusters* gerados pelo *Anthill*. Pode-se observar que para todos os textitdatasets o algoritmo identificou o número K de *cluster* ideal. Convém destacar que regiões de sobreposição de dados, como no *dataset* clássico *Iris*, dificultam o particionamento, o que pode ser observado na Figura 5.1a, onde as classes sobrepostas são representadas pelas cores laranja e vermelho. Para os *datasets* das Figuras 5.1b e 5.1e, além de identificar corretamente o número k de clusters, as regiões de sobreposição foram bem separadas. A área de dados sobrepostos dos *datasets* das Figuras 5.1d e 5.1c é vasta mas o particionamento gerado dá indícios visuais de ter mantido a distribuição original dos dados, onde as classes majoritárias possuem respectivamente 224 e 500 objetos, sendo representadas nas imagens pela cor azul, cuja densidade de pontos é predominante.



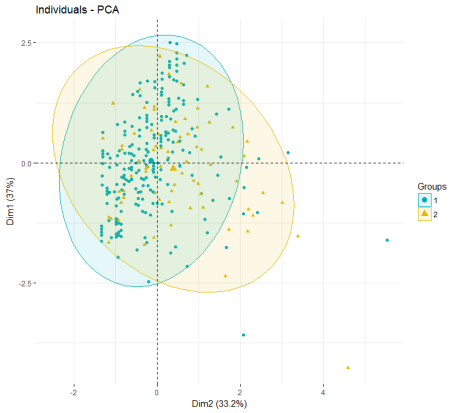
(a) Iris dataset



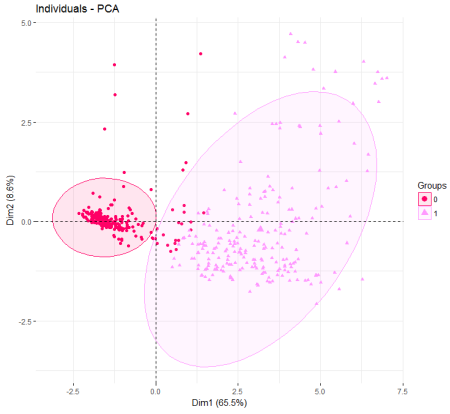
(b) Wine dataset



(c) Pima dataset



(d) Haberman dataset



(e) Wisconsin dataset

Figura 5.1: Redução de dimensionalidade

5.2 RESULTADOS COMPUTACIONAIS: ETAPA 02

Nessa segunda etapa de avaliação do algoritmo *Anthill* foram utilizados outros 80 *datasets* da literatura, cujos dados são estruturados de formas diversas. O objetivo dessa segunda avaliação é verificar o comportamento do algoritmo quando apresentado a conjuntos de dados com diferentes cardinalidades e distribuídos de diferentes formas no espaço, considerando a silhueta obtida e o tempo de processamento do algoritmo. Para tal, os *datasets* foram divididos em 3 conjuntos, classificados como:

1. **DS1**: conjunto que contém sete *datasets* clássicos da literatura, complementando os resultados da Etapa 01; Onde o número de atributos variam de dois a treze, com cardinalidades compreendidas no intervalo de 75 a 1484 objetos. A Tabela B.1, contém informações detalhadas sobre esse conjunto.

2. **DS2**: conjunto com 51 *datasets* construídos por Cruz (2010), que são nomeadas relacionando a quantidade de objetos, grupos e a dispersão dos dados no espaço, todos pertencentes ao espaço R^2 com cardinalidades variando de 100 a 2000 . Este conjunto reuniu as bases cujos grupos são coesos e bem separados (o nome dos *datasets* terminam em 'c') e as não comportadas, que possuem muitos pontos entre os *clusters* (o nome dos *datasets* terminam em 'c1'); Informações adicionais como número de atributos e quantidade de objetos, podem ser vistos na Tabela B.2.

3. **DS3**: conjunto que contém 22 *datasets* com diferentes cardinalidades e distribuição dos dados no espaço. Nesse conjunto de *datasets* possuem a quantidade de objetos de 30 a 2000, todas apresentando duas dimensões (SOARES; OCHI, 2004) e (SOARES, 2004). As características detalhadas de cada *dataset* pertencente ao conjunto DS3 são apresentadas na Tabela B.3.

Os melhores resultados disponibilizados pelos seus respectivos autores, para o índice silhueta e para o número de *clusters* obtidos pelos algoritmos AECBL1, ILS-FJGP, ILS-DBSCAN e HH; podem ser observados na Tabela 5.3, sendo que, as colunas AECBL1-K e ANTHILL-k se referem aos *clusters*.

foram disponibilizados pelos seus respectivos autores. Já o número de *clusters*, foram disponibilizados apenas para os algoritmos AECBL1 e Anthill, sendo que, para os demais,

essa informação não foi localizada. Sendo assim, na Tabela 5.3 as colunas AECBL1-K e ANTHILL-k se referem ao número de *clusters* obtidos pelos re

Os melhores resultados do índice silhueta obtidos pelos algoritmos AECBL1, ILS-FJGP, ILS-DBSCAN e HH foram disponibilizados pelos seus respectivos autores. Já o número de *clusters*, foram disponibilizados apenas para os algoritmos AECBL1 e Anthill, sendo que, para os demais, essa informação não foi localizada. Sendo assim, na Tabela 5.3 as colunas AECBL1-K e ANTHILL-k se referem ao número de *clusters* obtidos pelos respectivos algoritmos.

Tabela 5.3: Comparação dos valores do índice silhueta (DS1)

Dataset	**AECBL1-K**	**AECBL1**	**ILS-FJGP**	**ILS-DBSCAN**	**HH**	**ANTHILL-K**	**ANTHILL**
200Data	3	**0.823**	0.455	0.762	**0.823**	3	**0.823**
gauss9	9	**0.482**	0.289	0.106	**0.482**	7	0.399
iris	2	**0.687**	0.643	0.102	**0.687**	2	**0.687**
maronna	4	0.575	0.197	0.218	0.575	4	**0.584**
ruspini	4	0.738	0.522	**0.746**	0.738	4	0.740
spherical_4d3c	4	**0.689**	0.223	0.132	**0.689**	4	**0.689**
vowel2	74	**0.451**	0.242	0.417	**0.451**	4	0.380
wine	2	**0.660**	0.539	0.641	**0.660**	3	0.539
yeast	2	**0.628**	-0.055	0.380	**0.628**	2	**0.628**

As Tabelas 5.3, 5.4 e 5.5 consolidam os resultados obtidos pelo *Anthill* e por outras abordagens apresentadas em (SEMAAN, 2013) para os *datasets* dos conjuntos DS1, DS2 e DS3, respectivamente. Os grupos gerados pelo *Anthill* podem ser visualizados no Apêndice A.

Os resultados mostram que o *Anthill* obteve agrupamentos considerados fortes para 45.12% dos *datasets*, o que significa que os valores de silhuetas encontrados neste caso são superiores a 0.7. Além disso, para 14.3% dos *datasets*, os valores encontrados foram maiores que 0.8. Em 90.24% dos casos o algoritmo foi capaz de gerar soluções razoáveis, onde os valores de silhuetas pertencem ao intervalo compreendido entre [0.5 e 0.7].

Para o conjunto DS1, o *Anthill* (Tabela 5.3) obteve resultados melhores ou iguais aos

apresentados pelos algoritmos da literatura em 4 dos 9 *datasets*. O valor médio de silhueta foi de 0,607, com desvio padrão de 0,140. O maior valor de silhueta para esse conjunto foi de 0,823, enquanto o menor valor foi de 0,380. Para o conjunto DS2 (Tabela 5.4), a silhueta média apresentada pelo algoritmo proposto foi de 0,671, sendo a maior 0,842 e a menor 0,411, com desvio padrão de 0,140. Já para o conjunto DS3, a Tabela 5.5 mostra que o valor médio de silhueta foi de 0,705, com desvio padrão de apenas 0,094, com valores médios variando de 0,854 a 0,515.

O algoritmo proposto alcançou a melhor solução para 55.55% dos *datasets* do DS1, 33.33% do DS2 e 36.36% para o DS3. Os algoritmos AECBL1 e HH obtiveram o melhor agrupamento segundo o índice silhueta para 77.77% dos casos do conjunto DS1, O ILS-FJGP não encontrou a melhor solução para nenhum dos *datasets* do DS1 e o ILS-DBSCAN conseguiu alcançar o melhor resultado para 1 *dataset*.

Tabela 5.4: Comparação dos valores de índice silhueta (DS2)

Dataset	**AECBL-K**	**AECBL1**	**ILS-FJGP**	**ILS-DBSCAN**	**HH**	**ANTHILL-k**	**ANTHILL**
100p10c	10	0.834	**0.838**	**0.838**	0.834	10	0.835
100p2c1	2	**0.743**	0.551	**0.743**	**0.743**	2	0.673
100p3c	3	0.786	0.502	**0.792**	0.786	3	0.784
100p3c1	3	0.579	0.391	0.559	0.579	3	**0.589**
100p7c	7	0.834	**0.838**	**0.838**	0.834	7	0.804
100p8c1	7	**0.528**	0.479	0.442	**0.528**	6	0.502
100p5c1	8	**0.700**	0.655	0.654	**0.700**	5	0.681
100p7c1	7	0.491	0.423	0.470	0.491	7	**0.494**
200p2c1	2	**0.764**	0.561	0.758	**0.764**	2	**0.764**
200p3c1	3	0.681	0.440	0.672	0.681	3	**0.684**
200p4c	4	0.773	0.441	**0.776**	0.773	4	0.773
200p4c1	4	0.773	0.441	**0.776**	0.773	4	0.745
200p7c1	14	**0.579**	0.381	0.487	**0.579**	7	0.550
200p8c1	9	**0.576**	0.365	0.505	0.576	5	0.457
200p12c1	13	**0.577**	0.344	0.441	0.577	9	0.566
300p13c1	13	0.566	0.331	0.456	0.566	11	0.573
300p10c1	10	**0.609**	0.337	0.589	**0.609**	7	0.545
300p2c1	10	0.776	0.621	0.777	0.776	2	**0.778**
300p3c	3	**0.766**	0.431	0.772	**0.766**	3	**0.766**
300p3c1	3	0.676	0.435	**0.679**	0.676	3	**0.679**
300p4c1	2	0.607	0.478	**0.614**	0.607	4	0.570
300p6c1	8	**0.662**	0.470	0.633	**0.662**	4	0.590
400p3c	3	0.799	0.486	0.808	0.799	3	**0.817**
400p4c1	4	0.602	0.264	0.427	0.602	4	**0.626**
400p17c1	2	**0.513**	0.278	0.427	**0.513**	14	0.461
500p19c1	20	**0.483**	0.265	0.382	**0.483**	12	0.411
500p3c	3	0.825	0.604	0.830	0.825	3	**0.842**
500p4c1	5	**0.661**	0.383	**0.661**	**0.661**	3	0.661
500p6c1	6	**0.630**	0.210	0.610	**0.630**	6	0.589
600p15c	15	0.781	0.262	**0.782**	0.781	15	**0.782**
600p3c1	3	0.721	0.393	0.710	0.721	3	**0.743**
700p4c	4	0.797	0.332	**0.804**	0.797	4	0.789

Continua na próxima página

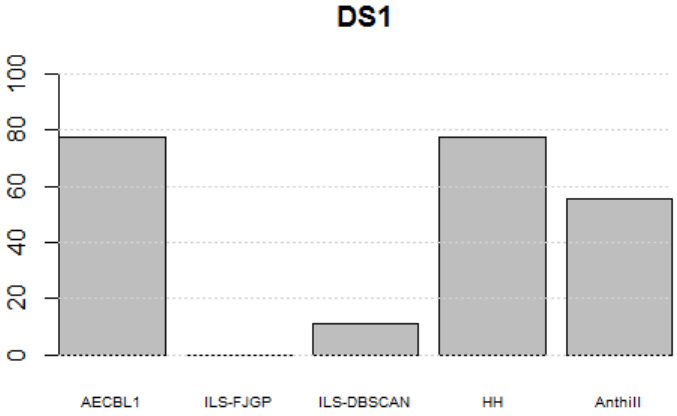
Tabela 5.4 – *Continuação da tabela*

Dataset	**AECBL-K**	**AECBL1**	**ILS-FJGP**	**ILS-DBSCAN**	**HH**	**ANTHILL-k**	**ANTHILL**
700p15c1	15	**0.680**	0.322	0.574	0.680	13	0.591
800p10c1	2	**0.468**	0.275	0.409	**0.468**	8	0.424
800p18c1	19	**0.692**	0.208	0.598	**0.692**	15	0.617
800p4c1	4	**0.702**	0.279	0.652	**0.702**	4	0.695
800p23c	23	0.787	0.301	**0.794**	0.787	23	0.781
900p12c	12	0.841	0.289	**0.842**	0.841	12	**0.842**
900p5c	5	0.716	0.302	**0.722**	0.716	5	0.698
1000p14c	14	0.831	0.231	**0.832**	0.831	14	0.828
1000p5c1	5	**0.639**	0.202	0.628	**0.639**	5	0.624
1000p6c	6	0.736	0.142	0.742	0.736	6	**0.751**
1000p27c1	28	**0.523**	0.225	0.326	**0.523**	14	0.412
1100p6c1	6	0.671	0.168	0.649	0.671	6	**0.688**
1300p17c	17	0.823	0.142	**0.824**	0.823	15	0.786
1500p6c1	6	0.644	0.050	0.618	0.644	6	**0.661**
1800p22c	22	0.798	0.145	**0.810**	0.802	22	0.801
1900p24c	24	**0.799**	0.288	**0.799**	**0.799**	24	0.797
2000p11c	11	0.713	-0.086	**0.716**	0.713	11	0.713
2000p26c	26	0.779	0.268	0.808	0.799	26	0.800
2000p9c1	9	0.624	-0.026	0.169	0.624	8	0.595

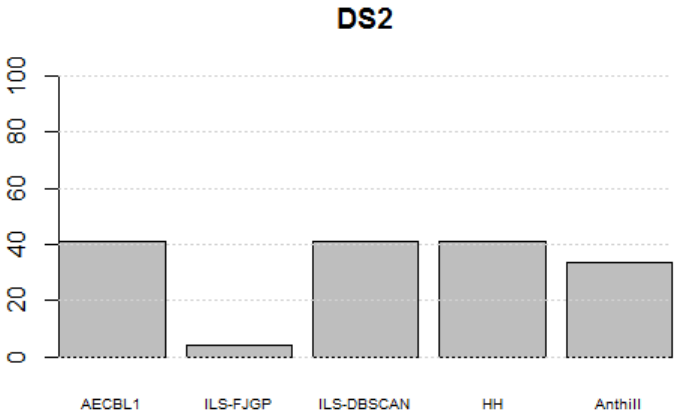
Já para os *datasets* do bloco DS2 Os algoritmos AECBL1, ILS-DBSCAN e HH encontraram 41.17% das melhores soluções e o ILS-FJGP 3.92% o que corresponde a dois *datasets*. Para o conjunto DS3 o ILS-DBSCAN alcançou 59.09% das maiores silhuetas, o AECBL1 18.18% , HH 13.63% e por último o ILS-FJGP com 9.09% de alcance para as melhores soluções. Esses resultados discutidos aqui foram condensados nos gráficos das Figuras 5.2a, 5.2b e 5.2c.

Podemos observar que os resultados obtidos pelo *Anthill* são similares aos da literatura. Em vários casos, observa-se que o *Anthill* alcançou resultados iguais ou melhores em termos de valores de silhuetas encontradas. Esses resultados apontam para a viabilidade do algoritmo para gerar soluções de qualidade.

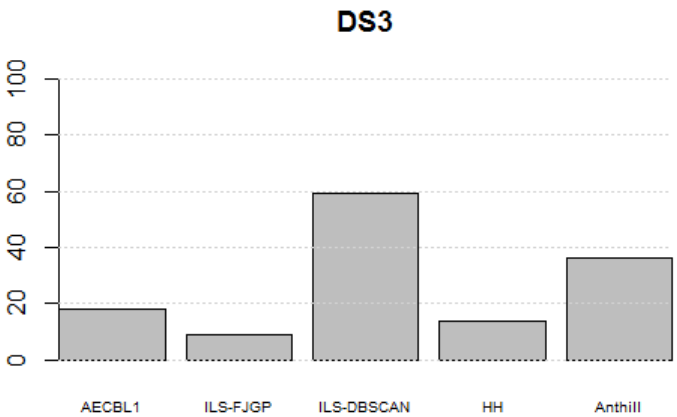
Com relação à análise do tempo de execução, vale salientar que o *Anthill* teve seu



(a) Percentual das melhores soluções para o *dataset* DS1



(b) Percentual das melhores soluções para o *dataset* DS2



(c) Percentual das melhores soluções para o *dataset* DS3

Figura 5.2: Análise dos algoritmos em relação ao percentual de melhores soluções obtidas.

desempenho comparado à apenas dois algoritmos da literatura, sendo estes o AECBL1 e HH, uma vez que os demais algoritmos não foram avaliados nessa perspectiva por seus respectivos autores. O desempenho temporal do *Anthill* e das demais abordagens da literatura, pode ser visto nas Tabelas 5.6, 5.7 e 5.8 e no Gráfico 5.3, onde os tempos de execução são apresentados em segundos.

O algoritmo *Anthill* mostrou-se competitivo em termos de tempo de execução, alcançando os menores tempos em 55.55% dos *datasets* do conjunto DS1, enquanto os algoritmos AECBL1 e HH apresentaram, cada um, os menores tempos em 22.22% dos casos. Para o conjunto DS2 o algoritmo proposto e o HH empataram e alcançaram os menores tempo de execução em 43.13% do total, enquanto o AECBL1 foi mais rápido em 13.72%. O algoritmo HH atingiu os melhores tempos em 72.72% dos *datasets* do conjunto DS3, *Anthill* alcançou 37.5% dos melhores resultados e o AECBL1 não foi o mais rápido para nenhum dos *datasets* em questão.

Observa-se que, durante os experimentos iniciais, embora o *Anthill* tenha alcançado valores significantes de silhueta em relação aos da literatura, verifica-se que o algoritmo tem alto custo computacional. O tempo de processamento para grandes *datasets* pode inviabilizar o uso do algoritmo na resolução de problemas com grandes volumes de dados. Neste sentido, visando a redução do tempo de execução do algoritmo, foi desenvolvida uma versão paralelizada do algoritmo, descrita no próximo capítulo.

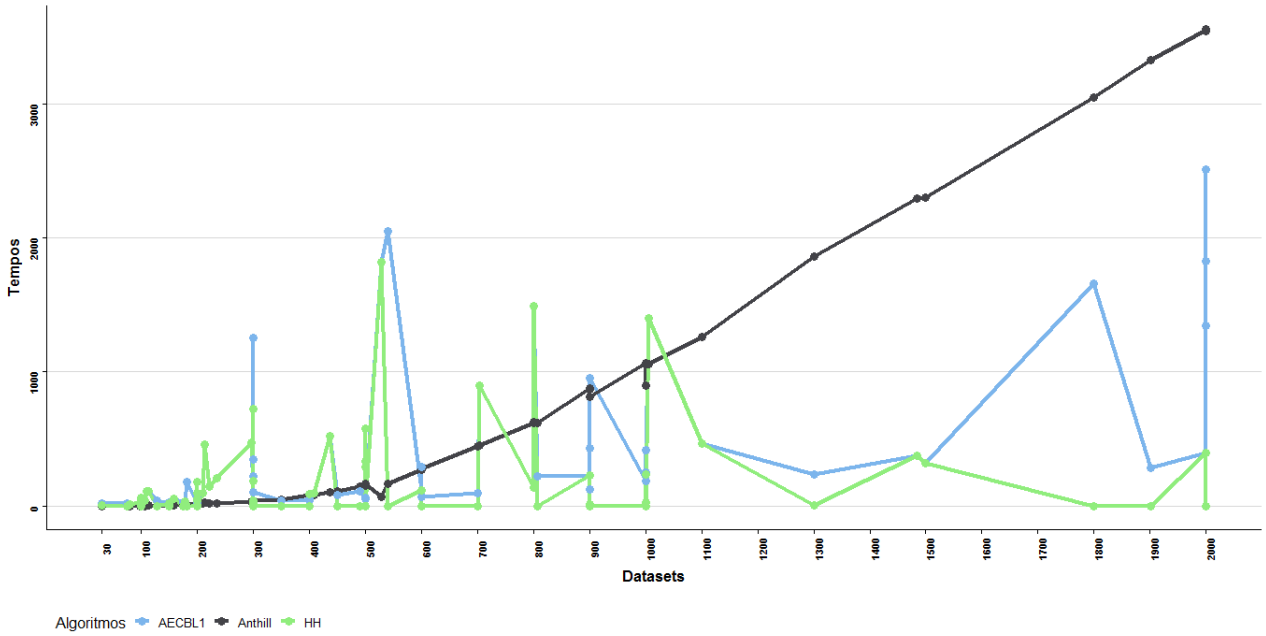


Figura 5.3: Gráfico comparativo tempos de execução

Tabela 5.5: Comparação dos valores índice silhueta (DS3)

Dataset	**AECBL-K**	**AECBL1**	**ILS-FJGP**	**ILS-DBSCAN**	**HH**	**ANTHILL-k**	**ANTHILL**
30p	9	0.724	0.564	0.791	0.787	12	**0.821**
outliers_args	7	0.748	0.742	0.742	0.748	7	**0.754**
97p	3	0.711	0.419	0.711	0.711	3	**0.715**
3dens	2	0.762	**0.764**	**0.764**	0.762	2	0.762
Outliers	2	0.785	0.786	0.786	0.785	7	0.528
157p	4	**0.666**	0.338	0.664	**0.666**	4	0.665
convdensity	3	0.854	0.577	**0.858**	0.854	3	0.854
181p	6	0.737	0.599	**0.744**	0.737	4	0.698
convexo	3	0.668	0.618	0.676	**0.668**	3	0.656
face	16	**0.527**	0.203	0.306	0.526	3	0.515
2face	2	**0.667**	0.548	0.674	**0.667**	2	**0.667**
300p4c	4	0.750	0.251	0.750	0.750	4	**0.764**
350p5c	5	0.759	0.250	**0.768**	0.759	5	0.759
numbers	10	**0.581**	0.258	0.561	**0.581**	10	0.520
450p4c	4	0.766	0.427	**0.772**	0.766	4	0.766
moreshapes	6	0.732	0.434	**0.734**	0.732	7	0.728
500p3c	3	0.825	0.604	0.830	0.825	3	**0.842**
numbers2	10	**0.600**	0.267	**0.600**	**0.600**	10	**0.600**
600p3c	3	0.751	0.331	**0.752**	0.751	3	**0.752**
900p5c	5	0.716	0.302	**0.722**	0.716	5	0.698
1000p6c	6	0.736	0.206	0.742	0.736	6	**0.751**
2000p11c	11	0.713	0.202	**0.716**	0.713	11	0.713

Tabela 5.6: Comparação tempos de execução (em segundos) (DS1)

Dataset	**AECBL1**	**HH**	**Anthill**
200Data	54.19	54.21	**13.199**
gauss9	**228.87**	228.89	875.935
iris	17.90	17.92	**5.304**
maronna	178.69	178.70	**14.045**
ruspini	22.20	**0.17**	0.375
spherical_4d3c	43.18	**0.69**	82.776
vowel2	1821.61	1821.63	**66.706**
wine	35.76	35.77	**11.053**
yeast	**372.59**	372.61	2294.758

Tabela 5.7: Comparação tempos de execução (em segundos) (DS2)

Dataset	***AECBL1***	***HH***	***Anthill***	***Dataset***	***AECBL1***	***HH***	***Anthill***
100p10c	17.71	**0.05**	1.14	500p19c1	332.52	332.54	**155.70**
100p2c1	17.93	17.94	**1.54**	500p3c	60.53	**0.06**	150.55
100p3c	17.79	**0.03**	35.87	500p4c1	575.66	575.67	**158.76**
100p3c1	58.65	58.67	**1.38**	500p6c1	290.66	290.67	**153.18**
100p5c1	113.17	113.19	**1.88**	600p15c	291.67	**0.19**	277.69
100p7c	13.30	**0.03**	1.05	600p3c1	**115.72**	115.74	271.53
100p7c1	107.64	107.65	**1.96**	700p4c	94.64	**0.09**	442.66
100p8c1	39.62	39.64	**1.27**	700p15c1	901.18	901.19	**451.07**
200p2c1	40.30	40.32	**12.45**	800p10c1	**136.96**	136.97	622.07
200p3c1	69.72	93.59	**18.86**	800p18c1	1490.96	1490.98	**626.95**
200p4c	36.99	**0.03**	11.83	800p4c1	**136.26**	136.28	621.55
200p4c1	53.63	**6.48**	17.86	900p12c	428.56	**13.21**	871.88
200p7c1	95.49	95.50	**19.26**	900p5c	125.45	**0.14**	867.94
200p8c1	461.31	461.33	**23.78**	1000p14c	247.62	**25.66**	897.65
200p12c1	145.30	145.31	**18.07**	1000p5c1	**232.16**	232.18	1056.55
300p13c1	209.06	209.08	**22.42**	1000p6c	188.67	**0.16**	1067.19
300p10c1	1252.57	722.72	**39.11**	1000p27c1	1398.75	1398.77	**1060.04**
300p2c1	104.38	189.56	**37.42**	1100p6c1	**467.11**	467.13	1261.73
300p3c	43.10	**0.04**	35.87	1300p17c	234.17	**2.39**	1860.59
300p3c1	221.70	**3.96**	38.47	1500p6c1	**320.22**	320.24	2302.27
300p4c1	38.91	38.93	**36.54**	1800p22c	1658.86	**0.52**	3045.17
300p6c1	347.78	**19.62**	37.72	1900p24c	281.13	**0.78**	3327.56
400p3c	51.80	**0.05**	81.47	2000p11c	1342.98	**0.55**	3539.95
400p4c1	85.94	85.96	**82.96**	2000p26c	2507.53	**0.64**	3552.76
400p17c1	95.18	95.20	**80.06**	2000p9c1	**393.65**	393.66	3547.48

Tabela 5.8: Comparação tempos de execução (em segundos) (DS3)

Dataset	**AECBL1**	HH	**Anthill**
2face	58.51	**0.03**	16.421
3dens	36.74	**0.03**	0.971
30p	16.70	6.57	**0.415**
97p	24.90	24.91	**1.342**
157p	52.81	52.83	**6.193**
181p	177.86	**0.07**	7.139
300p4c	105.77	**0.04**	37.901
350p5c	38.96	**0.05**	44.218
450p4c	79.10	**0.06**	106.851
500p3c	50.63	**0.06**	163.168
600p3c	69.33	**0.08**	275.802
900p5c	953.11	**0.14**	813.703
1000p6c	414.03	**0.17**	1056.768
2000p11c	1826.71	**0.55**	3545.063
convexo	28.97	**0.03**	27.828
convdensity	24.93	**0.82**	7.364
face	472.43	472.44	**36.071**
moreshapes	112.62	**1.01**	144.081
numbers	521.06	521.07	**102.982**
numbers2	2047.18	**0.27**	163.172
outliers	28.08	**0.40**	5.916
outliers_ags	12.46	12.48	**1.106**

6 ANTHILL-PARALELO

Os testes preliminares apresentados no capítulo anterior 5.1 indicam que o *Anthill* é capaz de encontrar boas soluções, vide os valores de silhuetas apresentados. Entretanto, o algoritmo consome um tempo muito maior de processamento quando comparado aos algoritmos da literatura, sendo esta observação mais expressiva em *datasets*s com maior dimensionalidade.

Técnicas de *profiling* são amplamente utilizadas para identificar gargalos em programas e estimar os tempos de execução, para otimização de código (BALL; LARUS, 1994). Sendo assim, buscando identificar blocos de instruções suscetíveis de serem aprimorados, tais técnicas foram utilizadas. Inicialmente foram encontrados dois pontos de melhoria a serem trabalhados: (i) a construção do digrafo completo, que demanda $n(n-1)$ comparações e (ii) o caminhamento das formigas durante a construção das soluções. Neste sentido, esses dois quesitos foram considerados no processo de paralelização do algoritmo *Anthill*.

6.1 CONSTRUÇÃO DO DIGRAFO COMPLETO

A construção do digrafo envolve a criação de todos os nós, arcos, cálculos da distância entre pares de todos os objetos de dados, bem como a similaridade entre esses objetos. Por fim, calcula-se a taxa de aceitação, que vem a se tornar a informação heurística associada ao trabalho colaborativo da colônia de formigas. Sendo esses valores imutáveis durante a execução do algoritmo, não há a necessidade de recalcular tais valores em diferentes execuções para o mesmo *dataset*. Sendo assim, uma vez construído o digrafo, a estrutura é serializada e armazenada. Esse processo faz com que o digrafo seja mantido ocupando pouco espaço em disco. Nas próximas execuções, o processo inverso denominado desserialização é realizado, trazendo para a memória a estrutura completa de forma muito mais rápida.

Entende-se por serialização o processo no qual a instância de um objeto é transformada em um array de bytes com a finalidade de facilitar o armazenamento de dados ou transmissão pela rede. Desserialização é o processo inverso que transforma esse array de bytes em um objeto utilizável.

Como dito anteriormente, são necessárias $n(n-1)$ comparações para construção do

digrafo. Entretanto, como a distância entre dois objetos o_i e o_j é simétrica, ou seja, $D(o_i, o_j) = D(o_j, o_i)$, essas comparações podem ser reduzidas pela metade.

Em geral, algoritmos seguem passos sequenciais executando as instruções na ordem em que aparecem. Com isso, recursos computacionais acabam ficando ociosos, principalmente nos computadores atuais que possuem diversos processadores. Diante disso, o *Anthill* foi alterado de forma a permitir que trechos de códigos fossem executados de forma paralela e independente do fluxo principal.

O digrafo inicial é um dos parâmetros de entrada para o *Anthill*, preservando o estado atual do objeto faz com que o custo computacional diminua, uma vez que, é necessário apenas, o carregamento desse objeto em memória. Além disso, a tarefa de construção do digrafo é dividida proporcionalmente entre os vários processadores disponíveis no *hardware* utilizado. De forma que, dado o número total de objetos em determinado *dataset* e o número de núcleos disponíveis, associa-se igualmente o número de operações a serem realizadas por cada núcleo. Por exemplo: em um *dataset* com 100 objetos de dados, com 4 núcleos disponíveis, cada núcleo de processamento ficará responsável por 25% do processamento.

Resumidamente as instruções sequencialmente executadas pelo algoritmo podem ser divididas em três etapas, sendo:

1. Pré-processamento:
   (a) Leitura do dataset;
   (b) Criação do digrafo (nós e arcos);
   (c) Cálculo das similaridades;
   (d) Cálculo taxa de aceitação.
2. Processamento
   (a) Caminhamento no digrafo, formigas percorrem sua vizinhança marcando as arestas por onde passaram;
   (b) Atualização do feromônio;
   (c) Atualização dos parâmetros (a cada 10 iterações);
3. Pós-processamento:

(a) Cálculo das componentes fortemente conectados;

(b) Cálculo silhueta;

(c) Realocação de objetos em grupos vizinhos mais próximos.

Em relação ao Pré-processamento (Etapa 1), o item corresponde a criação do digrafo (nós e arcos) é a etapa do algoritmo que demanda maior esforço computacional. Neste sentido, com a execução sequencial do algoritmo, recursos como processadores e memória acabam subutilizados. Assim, analisando o algoritmo que realiza o cálculo dessas distâncias entre pares e preenche o estrutura de vizinhos, percebe-se que este cálculo pode ser executado em paralelo. Tirando proveito desta característica, foram criadas Tarefas (*threads*) que podem ser executadas concorrencialmente para dividir o custo computacional desta etapa.

De forma simplória *Threads* são linhas de execução independentes e concorrentes dentro de um mesmo processo, que é um programa em execução. As *Threads* permitem dividir o processo principal de um programa e a execução de várias tarefas executadas simultaneamente. Um processo é fundamentalmente um contêiner que armazena todas as informações necessárias para executar um programa (TANENBAUM; FILHO, 1995). A principal diferença entre *Threads* e processos está na impossibilidade de um processo acessar diretamente dados de outro processo. Já as *Threads* de um mesmo processo compartilham espaço de memória entre si, o que permite comunicação direta entre elas mas demanda a implantação de mecanismos de sincronização para garantir a integridade dos dados.

A Figura 6.1 ilustra o fluxo de execução da etapa de pré-processamento, destacando a paralelização da etapa de cálculos das distâncias e preenchimento da estrutura. Na estratégia adotada neste trabalho, o processo de criação do digrafo foi dividido igualmente entre as *threads* criadas. Em um primeiro momento, o número de *threads* corresponde ao número de processadores disponíveis, com o intuito de que as *threads* executem a menor quantidade de nós possível. Neste trabalho foram criadas 4 *threads*, sendo que cada uma destas ficou responsável pelos cálculos relacionados a aproximadamente 25% do total de nós. Ao final desta etapa o Digrafo está pronto para ser manipulado na Fase de Processamento.

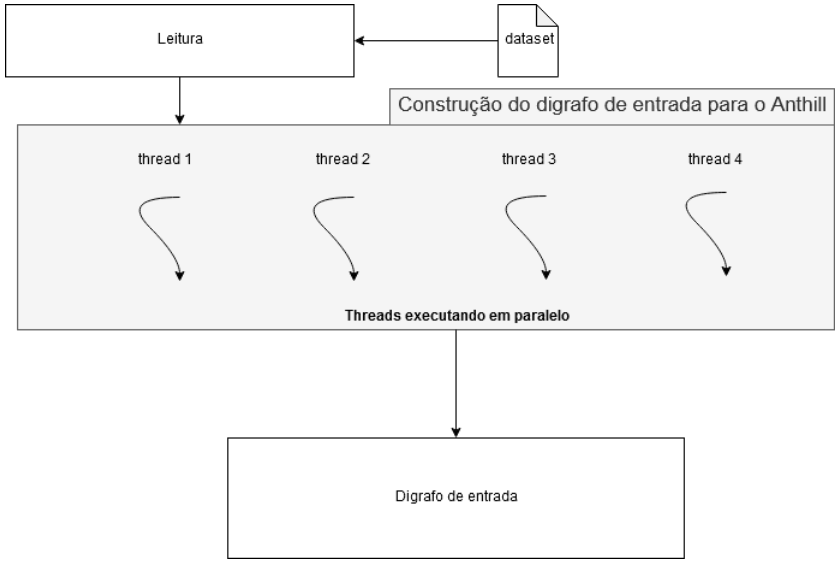


Figura 6.1: Pré-processamento usando *threads*

6.2 CAMINHAMENTO DAS FORMIGAS

Como descrito anteriormente, no algoritmo *Anthill*, cada uma das formigas caminha sobre o digrafo reforçando os arcos depositando feromônio. Neste sentido, em termos computacionais, as formigas estão executando tarefas independentes cujos resultados são consolidados para a construção da solução. Então, para paralelizar o processo de caminhamento das formigas no digrafo a definição de *threads* para cada uma delas torna-se viável. Entretanto, antes de definir o processo de paralelizar o *Anthill*, algumas propriedades devem ser consideradas.

Se duas *threads* executam o mesmo procedimento, cada uma terá sua própria cópia das variáveis locais, compartilhando os dados globais do programa. O desenvolvimento de aplicações em paralelo com uso de *threads* pode ser chamado de programação paralela de memória compartilhada. Já a programação em paralelo através da criação de processos recebe o nome de programação paralela por troca de mensagens (LIN et al., 2008).

A parte do código onde há acesso a memória compartilhada é chamada região crítica (TANENBAUM; FILHO, 1995), onde o acesso simultâneo a variáveis compartilhadas pode gerar instabilidade na aplicação. Diferentes abordagens se propõem a resolver problemas em regiões críticas. Dentre elas, os mecanismos de exclusão mútua são muito utilizados, pois permitem que uma *thread*, ao acessar a região crítica, bloqueie a entrada de outras *threads* até que o processamento nessa região seja finalizado. Embora tais abordagens aumentem a segurança e a confiabilidade na manipulação de dados compartilhados, o

excesso de "travas" e sincronizações pode provocar *overhead*, que ocorre quando o tempo gasto coordenando o paralelismo é maior que o tempo de execução do processamento da tarefa em si.

Sincronizações bloqueantes diminuem o desempenho de programas multitarefas. Adotar estruturas capazes de trabalhar verdadeiramente de forma simultânea proporciona melhorias no desempenho e redução dos riscos inerentes ao manuseio de múltiplas *threads*. Estruturas que trabalham de forma sincronizada mantêm um bloqueio durante a execução de cada operação, restringindo o acesso a uma única *thread* de cada vez.

No algoritmo *Anthill* a comunicação entre as formigas consiste em sinalizar o trajeto percorrido. Sendo assim, cada formiga k faz seu caminhamento pelo digrafo marcando os arcos utilizados. A formiga pode realizar escolhas aleatórias ou seguir pelo melhor caminho, neste caso escolhendo o arco com maior valor na combinação de feromônio com a informação heurística.

Na versão sequencial do *Anthill* todo o processamento é feito pela *thread* principal, o que faz com que o código fique parado, aguardando cada formiga k visitar os vértices alcançáveis a partir do seu ponto de origem. Nestas condições, ocorre um esquema de fila onde uma formiga executa todas as suas tarefas enquanto todas as outras ficam aguardando, quando ela finaliza sua execução o processamento passa para a próxima formiga e assim sucessivamente. Assim, no desenvolvimento do algoritmo, as formigas compartilham uma estrutura de mapa que armazena todos os arcos. Mapas são estruturas que utilizam o par $< chave, valor >$. A *chave* é composta pelo identificador do objeto de dado de origem O_i e o objeto de destino O_j. O mapa de arcos é atualizado ao final de cada iteração.

No *Anthill* existem duas regiões críticas conforme pode ser observado na Figura 6.2. A parte superior da Figura ilustra a etapa 1 de Pré-processamento, onde cada *thread* acessa o mapa de objetos de dados e constrói o mapa de arcos de forma concomitante. Nesta etapa, são realizados os cálculos das distâncias e preenchimento da estrutura, que será a entrada do algoritmo.

Já na etapa 2a de Caminhamento as formigas percorrem o digrafo marcando os arcos utilizados de forma concorrente e não bloqueante. Neste sentido, como as formigas manipulam o digrafo ao mesmo tempo, é importante adotar estratégias que suportem a manipulação dos arcos por várias *threads*, mas sem bloquear todas as operações. A solu-

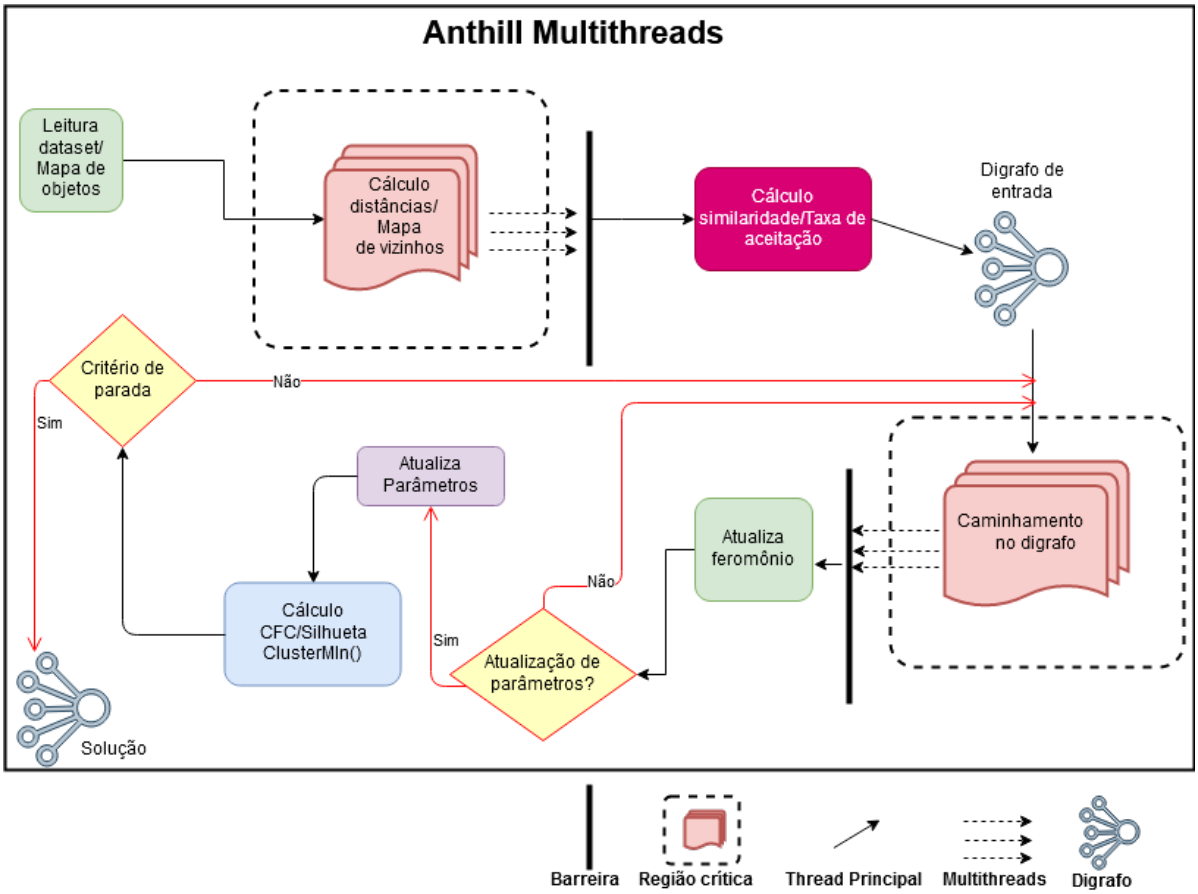


Figura 6.2: Anthill Multithreads

ção adotada neste trabalho foi o Framework *Java Collections* (*ConcurrentHashMap*), que é um mapa baseado em *hash* que permite um maior grau de acesso compartilhado. Possibilitando muitos acessos simultâneos tanto para leitura quanto para escrita bem como múltiplas gravações concorrentes (GOETZ et al., 2006).

Na versão paralela do *Anthill*, as *threads* são associadas às formigas, que trabalham de forma simultânea, caminhando pelo digrafo guiadas pela Função de probabilidade definida na Equação 4.6. A cada escolha as formigas incrementam o contador de utilização do arco no mapa. Essa é a informação que é utilizada para atualizar o feromônio e remover arcos abaixo do limiar de corte conforme a Equação 4.8. Vale ressaltar que a utilização de *threads* neste caso permite a execução de trechos de códigos de forma paralela, com o intuito de reduzir o tempo computacional necessário, mas não altera a forma como as soluções são construídas.

Após paralelizar o algoritmo foi necessário adotar métricas para avaliar sua eficiência, que estão descritas na próxima seção.

6.3 EFICIÊNCIA DO PARALELISMO

Para mensurar se houve melhora no desempenho ao modificar o algoritmo se faz necessário o uso de métricas próprias de avaliação de aplicações paralelas. Dentre as várias existentes, duas foram utilizadas nesse trabalho sendo elas o *Speedup* e a Eficiência.

Em termos simples, o ***Speedup*** é a razão entre o tempo de execução serial e o tempo de execução paralela (BRESHEARS, 2009). Tal medida avalia o grau de desempenho da aplicação, mensura o quão mais rápido é a versão paralela em comparação ao programa sequencial, conforme pode ser visto na Equação 6.1, onde $T(1)$ é o tempo de processamento sequencial e $T(p)$ é o tempo de execução em paralelo utilizando p processadores.

$$S = \frac{T(1)}{T(p)} \tag{6.1}$$

Idealmente, o ganho de *speedup* deveria tender a p, que seria o seu valor ideal. Alguns autores citam casos de possíveis ganhos maiores a p (*SuperSpeedup*), ou seja, $S \geq p$ (BRESHEARS, 2009). Alguns fatores que contribuem para essa condição são:

- Custos de comunicação/sincronização/iniciação praticamente inexistentes;
- Tolerância à latência da comunicação;
- Aumento da capacidade de memória (o problema passa a caber todo em memória);
- Subdivisão do problema (tarefas menores geram menos cache misses);
- Aleatoriedade da computação em problemas de optimização ou com múltiplas soluções.

A **Eficiência** mede o grau de aproveitamento dos recursos computacionais, relacionando o *Speedup* ao número de processadores. Também diz o quanto da capacidade dos processadores está sendo utilizado. A Eficiência é calculada segundo a Equação 6.2. Quanto maior for a parcela sequencial, menor é a Eficiência da arquitetura na execução do programa. Desta forma, a Eficiência é uma medida que indica a proporção do tempo que os processadores estão ocupados. No caso ideal ($S = p$), a Eficiência seria máxima e teria valor 1 (100%).

$$Eficiencia = \frac{S}{p} \tag{6.2}$$

7 EXPERIMENTOS COMPUTACIONAIS DO ANTHILL-PARALELO

Nesse capítulo são conduzidos os experimentos para avaliação da nova proposta paralelizada do algoritmo *Anthill* denominada *Anthill-Paralelo*. Essa avaliação é exposta e conduzida sob a perspectiva de algoritmos paralelos, onde são contabilizados o número de processadores utilizados bem como a melhora de desempenho obtida. As métricas de qualidade utilizadas são as descritas no capítulo anterior: *Speedup* e *Eficiência*.

Nesses experimentos foram utilizados 4 processadores e, com isso, os melhores valores de *Speedup* são os mais próximos de 4. Os resultados relacionados a tal experimento podem ser vistos nas Tabelas 7.1, 7.2 e 7.3. O número de *threads* criadas é igual ao número de processadores 4. Os experimentos realizados foram executados no mesmo ambiente dos experimentos computacionais apresentados no Capítulo 5.

Os gráficos 7.1, 7.2 e 7.3 trazem um comparativo dos tempos de execução entre o *Anthill* e o *Anthill-Paralelo*, para os *datasets* dos conjuntos DS1, DS2 e DS3 respectivamente. Já no Gráfico 7.4, um comparativo entre *Anthill*, *Anthill-Paralelo*, HH e o AECBL1 é apresentado de forma que o eixo X representa os *datasets* ordenados e de forma crescente por quantidade de elementos, e o eixo Y se refere aos tempos de execução.

Tabela 7.1: Comparação dos algoritmos *Anthill-Paralelo* e *Anthill* (DS1)

	Tempo (s)			
Dataset	**Anthill-Paralelo**	**Anthill**	***Speedup***	**Eficiência**
200DATA	**6.092**	13.199	2.167	0.542
gauss9	**265.846**	875.935	3.295	0.824
iris	**2.527**	5.304	2.099	0.525
maronna	**5.358**	14.045	2.621	0.655
ruspini	**0.135**	0.375	2.778	0.694
vowel2	**27.698**	66.706	2.408	0.602
spherical_4d3c	**31.057**	82.776	2.665	0.666
yeast	**709.889**	2294.758	3.233	0.808
wine	**3.565**	11.053	3.100	0.775

Tabela 7.2: Comparação dos algoritmos *Anthill-Paralelo* e *Anthill* (DS2)

	Tempo (s)			
Dataset	**Anthill-Paralelo**	**Anthill**	***Speedup***	**Eficiência**
100p2c1	**0.649**	1.534	2.364	0.591
100p3c	**0.509**	1.297	2.548	0.637
100p3c1	**0.472**	1.376	2.915	0.729
100p5c1	**0.533**	1.879	3.525	0.881
100p7c	**0.319**	1.047	3.282	0.821
100p7c1	**0.669**	1.956	2.924	0.731
100p8c1	**0.504**	1.274	2.528	0.632
100p10c	**0.496**	1.142	2.302	0.576
200p2c1	**4.574**	12.447	2.721	0.680
200p3c1	**6.802**	18.856	2.772	0.693
200p4c	**4.998**	11.834	2.368	0.592
200p4c1	**6.484**	17.859	2.754	0.689
200p7c1	**6.768**	19.263	2.846	0.712
200p8c1	**7.058**	23.776	3.369	0.842
200p12c1	**6.509**	18.074	2.777	0.694
300p2c1	**12.387**	37.419	3.021	0.755
300p3c	**11.781**	35.867	3.044	0.761
300p3c1	**12.997**	38.471	2.960	0.740
300p4c1	**11.119**	36.544	3.287	0.822
300p6c1	**12.801**	37.724	2.947	0.737
300p10c1	**14.632**	39.114	2.673	0.668
300p13c1	**7.312**	22.425	3.067	0.767
400p3c	**24.875**	81.475	3.275	0.819
400p4c1	**23.856**	82.956	3.477	0.869
400p17c1	**23.134**	80.057	3.461	0.865
500p3c	**51.458**	150.547	2.926	0.731
500p4c1	**52.902**	158.763	3.001	0.750
500p6c1	**54.173**	153.178	2.828	0.707
500p19c1	**53.633**	155.696	2.903	0.726
600p3c1	**90.417**	271.534	3.003	0.751
600p15c	**92.404**	277.687	3.005	0.751
700p4c	**165.970**	442.662	2.667	0.667
700p15c1	**166.540**	451.071	2.708	0.677
800p4c1	**202.687**	621.547	3.067	0.767
800p10c1	**207.427**	622.071	2.999	0.750

Continua na próxima página

Tabela 7.2 – *Continuação da tabela*

Dataset	**Anthill-Paralelo**	**Anthill**	***Speedup***	**Eficiência**
800p18c1	**210.954**	626.954	2.972	0.743
800p23c	**209.663**	620.546	2.960	0.740
900p5c	**280.781**	867.943	3.091	0.773
900p12c	**288.046**	871.876	3.027	0.757
1000p6c	**370.513**	1067.187	2.880	0.720
1000p5c1	**380.006**	1056.553	2.780	0.695
1000p14c	**324.523**	897.647	2.766	0.692
1000p27c1	**357.938**	1060.045	2.962	0.740
1100p6c1	**390.631**	1261.735	3.230	0.807
1300p17c	**525.512**	1860.587	3.541	0.885
1500p6c1	**656.978**	2302.271	3.504	0.876
1800p22c	**989.193**	3045.171	3.078	0.770
1900p24c	**1098.483**	3327.563	3.029	0.757
2000p9c1	**1101.133**	3547.482	3.222	0.805
2000p11c	**1098.532**	3539.953	3.222	0.806
2000p26c	**1104.118**	3552.764	3.218	0.804

A modificação do *Anthill* com a adaptação de trechos que demandam alto processamento à abordagens paralelas gerou uma redução no tempo de execução em média de 65.03%. Conforme o Gráfico 7.5 o ***Speedup*** máximo alcançado foi de 3.541 que é próximo do número de processadores utilizados (4) e O *Speedup* mínimo obtido foi de 2.020, o médio 2.909 com desvio padrão de 0.363. No pior caso a paralelização gerou um *Speedup* de 2.020 o que significa reduzir pela metade o tempo de execução.

A **Eficiência** também apresentou bons resultados, indicando que os recursos computacionais disponíveis foram bem utilizados. De acordo com o Gráfico 7.6 a Eficiência máxima alcançada foi de 0.885. Em média essa esse valor foi de 0.727 com desvio parão de 0.091 e no pior caso a eficiência obtida foi de 0.505.

Conforme apresentado, os resultados indicam que as modificações realizadas no *Anthill* melhoraram o desempenho em relação ao tempo. Além disso, a abordagem adotada é flexível, permitindo executar o algoritmo em computadores com hardware simples com recursos limitados. Neste capítulo não foram analisados os índices de silhuetas, pois não houve alteração em relação aos resultados já apresentados no Capítulo 5, ou seja, o Anthill sequencial e paralelizado obtiveram como resultado os mesmos valores de silhueta.

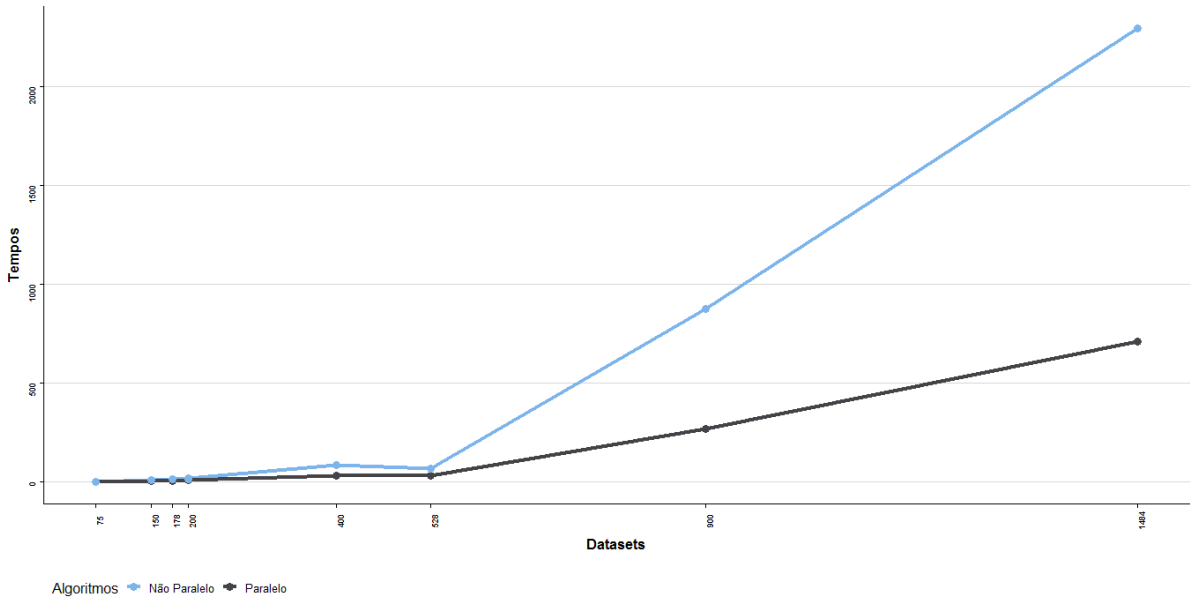


Figura 7.1: Comparativo Tempo de Execução DS1

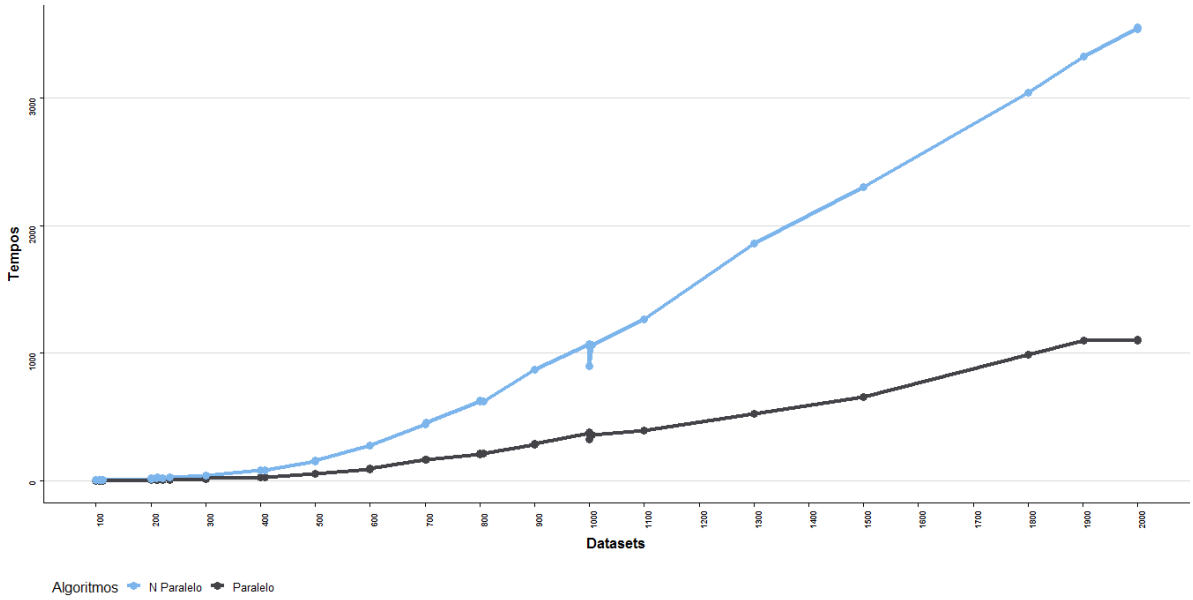


Figura 7.2: Comparativo Tempo de Execução DS2

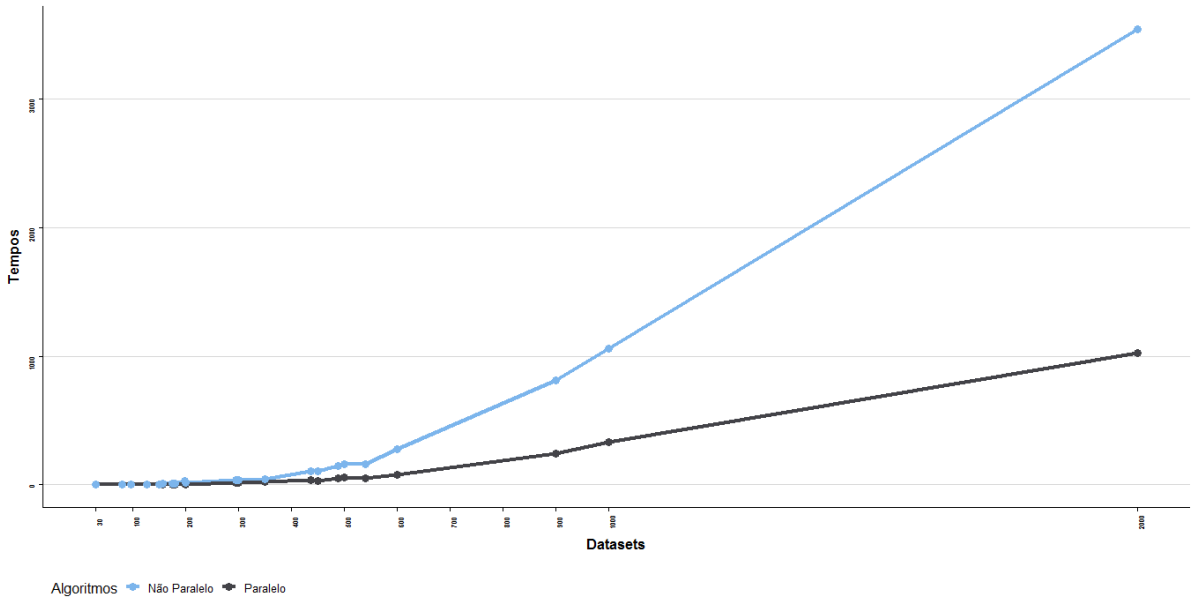


Figura 7.3: Comparativo Tempo de Execução DS3

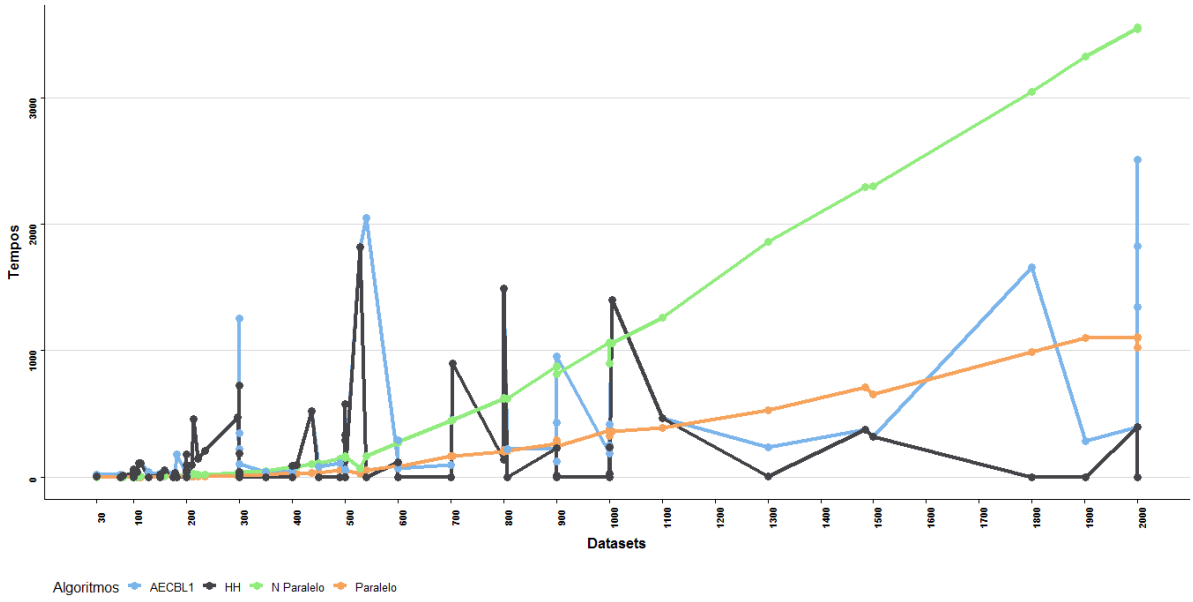


Figura 7.4: Comparativo Tempo de Execução Total

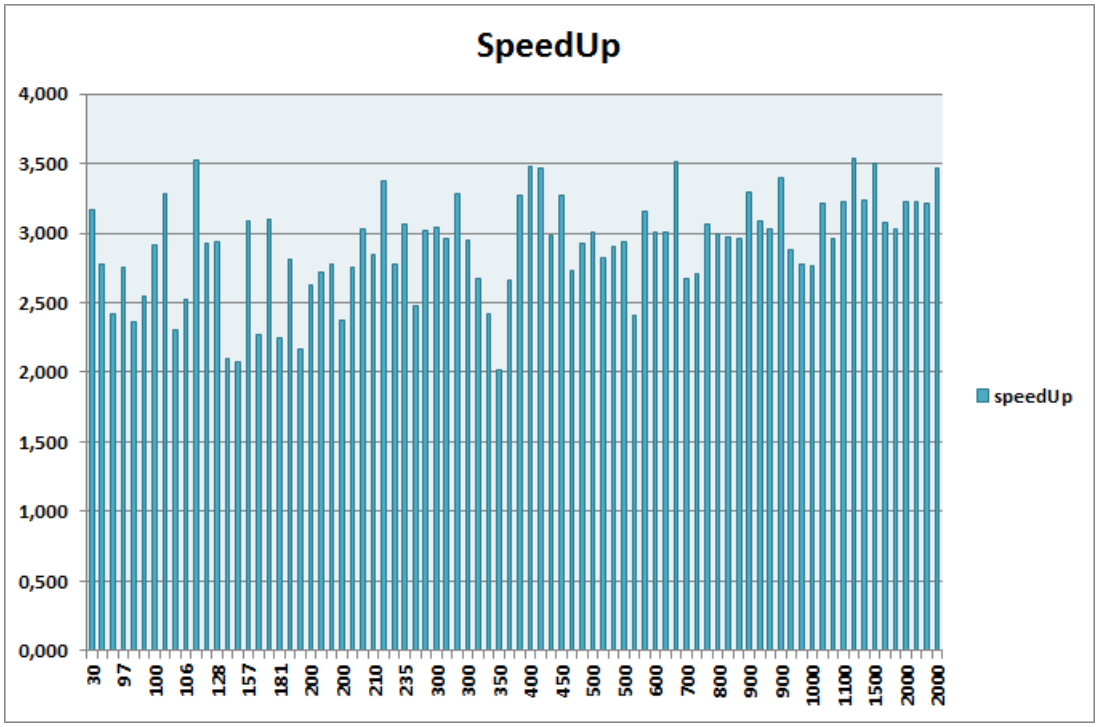


Figura 7.5: Speedup Total

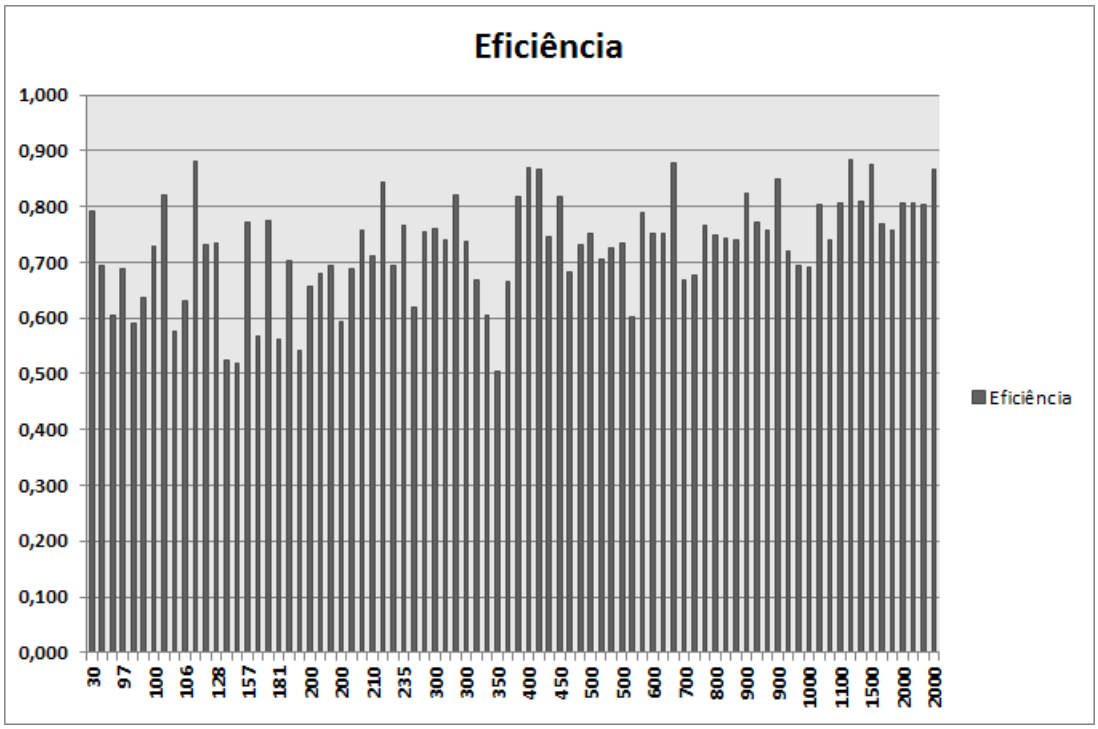


Figura 7.6: Eficiência

Tabela 7.3: Comparação dos algoritmos *Anthill-Paralelo* e *Anthill* (DS3)

	Tempo (s)			
Dataset	**Anthill-Paralelo**	**Anthill**	***Speedup***	**Eficiência**
2face	**5.412**	16.421	3.034	0.759
3dens	**0.331**	0.971	2.934	0.733
30p	**0.131**	0.415	3.168	0.792
97p	**0.487**	1.342	2.756	0.689
157p	**2.003**	6.193	3.092	0.773
181p	**3.171**	7.139	2.251	0.563
300p4c	**15.685**	37.901	2.416	0.604
350p5c	**21.889**	44.218	2.020	0.505
450p4c	**32.692**	106.851	3.268	0.817
500p3c	**55.489**	163.168	2.941	0.735
600p3c	**78.584**	275.802	3.510	0.877
900p5c	**239.578**	813.703	3.396	0.849
1000p6c	**328.954**	1056.768	3.213	0.803
2000p11c	**1023.523**	3545.063	3.464	0.866
convexo	**9.894**	27.828	2.813	0.703
convdensity	**3.241**	7.364	2.272	0.568
Face	**14.563**	36.071	2.477	0.619
moreshapes	**52.744**	144.081	2.732	0.683
numbers	**34.563**	102.982	2.980	0.745
numbers2	**51.795**	163.172	3.150	0.788
Outliers	**2.854**	5.916	2.073	0.518
outliers_ags	**0.457**	1.106	2.420	0.605

8 CONSIDERAÇÕES FINAIS

Considerando a grande quantidade de dados gerada diariamente e a dificuldade em transformar estes dados em informações relevantes, o presente trabalho buscou apoiar este processo através do desenvolvimento de uma abordagem heurística baseada no comportamento de colônia de formigas. Para isso, primeiramente foram apresentados os principais conceitos que envolvem a análise de *cluster*, a avaliação da qualidade do agrupamento e a Otimização por Colônia de Formigas (*ACO - Ant Colony optimization*). Adicionalmente, foram apresentados os principais trabalhos existentes na literatura, os quais foram utilizados para a comparação e avaliação dos resultados obtidos.

Visto isso, foi proposto um algoritmo de inteligência coletiva baseado no comportamento das formigas, denominado *Anthill*, para a resolução do problema de agrupamento automático de dados. O algoritmo *Anthill* foi submetido a testes de desempenho e foi avaliado utilizando três critérios distintos: validação interna (silhueta), inspeção visual e tempo de execução. O algoritmo obteve bom desempenho quando comparado a outros algoritmos da literatura, sob a perspectiva de valores de silhueta.

O algoritmo mostrou ser capaz de obter particionamentos significativos que representam bem a estrutura real dos dados, vide os valores de silhuetas encontrados, que são indicativos de que os elementos estão bem alocados aos grupos aos quais foram atribuídos. Apesar de ter alcançado bons resultados em termos de tempo de execução, inclusive sendo o mais rápido em média para alguns dos experimentos, o *Anthill* apresentou problemas com *datasets*s maiores demandando um tempo de execução elevado quando comparado aos demais algoritmos da literatura.

Uma vez constatada a necessidade de redução do tempo computacional, necessário para realizar o agrupamento de grandes volumes de dados, uma abordagem paralelizada do *Anthill* foi apresentada. Nessa versão, algumas etapas do algoritmo, possíveis de serem paralelizadas, foram identificadas e seu fluxo foi alterado de sequencial para paralelo através do uso de múltiplas *threads*. Implementando um modelo de paralelismo de memória compartilhada, através do uso de estruturas de dados que suportam múltiplos acessos simultâneos tanto para leitura quanto para escrita. Essas alterações geraram uma redução no tempo de execução médio de 65.03% com 4 núcleos de processamento em uso.

Com valores de *Speedup* próximos ao desejado que é igual ao número de processadores disponíveis.

Como contribuição da solução proposta neste trabalho, destacam-se: (i) aumento do número de soluções geradas em relação ao trabalho de Chen et al. (2005); (ii) definição de um critério de parada guiado pela qualidade das soluções; (iii) alteração da maneira como a formiga explora o digrafo, inserindo o conceito de densidade dos grupos em um processo que anteriormente era totalmente estocástico; e (iv) redução da característica gulosa na escolha das arcos, aumentando a probabilidade de encontrar novas soluções, bem como a fuga de ótimos locais. Além disso, (v) a adaptação desta proposta a uma abordagem paralela aumentou a escalabilidade do algoritmo, o que torna possível aplicar o *Anthill* a *datasets*s de maior dimensionalidade, sendo outra contribuição deste trabalho.

Como trabalhos futuros pretendemos explorar, com mais detalhes, os resultados obtidos neste trabalho. O objetivo é analisar soluções que obtiveram melhores resultados com números de grupos diferentes. Outra consideração é identificar características dos problemas que foram bem solucionados pelo *Anthill* e as que dificultam o uso da nossa abordagem.

Por outro lado, pretendemos usar o *Anthill* na resolução de problemas reais de grande porte. O objetivo é aplicar o algoritmo em bases de dados da saúde, para auxiliar os médicos no diagnóstico de doenças graves. Essas bases são desafiadoras, pois possuem grande volume de dados e alta dimensionalidade.

8.1 PUBLICAÇÕES

PACHECO, T. M.; BRUGIOLO, L.; STRÖELE, V; SOARES, S. S. R. F. Metaheurística inspirada no comportamento das formigas aplicada ao problema de agrupamento. *In*: **Congresso Brasileiro de Inteligência Computacional - (CBIC 2017)**, 2017, Niterói. XIII Congresso Brasileiro de Inteligência Computacional, 2017.

PACHECO, T. M.; BRUGIOLO, L.; STRÖELE, V.; SOARES, S. S. R. F. An Ant Colony Optimization for Automatic Data Clustering Problem. *In*: **2018 IEEE Congress on Evolutionary Computation (CEC)**, 2018, Rio de Janeiro. 2018 IEEE Congress on Evolutionary Computation (CEC), 2018. p. 1-8.
http://dx.doi.org/10.1109/CEC.2018.8477806

REFERÊNCIAS

ANKERST, M.; BREUNIG, M. M.; KRIEGEL, H.-P.; SANDER, J. Optics: ordering points to identify the clustering structure. In: ACM SIGMOD RECORD. **ACM**, 1999. v. 28, n. 2, p. 49–60.

ARABIE, P.; HUBERT, L. J.; SOETE, G. D. **Clustering and classification**, 1996.

BALL, T.; LARUS, J. R. Optimally profiling and tracing programs. **ACM Transactions on Programming Languages and Systems (TOPLAS)**, ACM, v. 16, n. 4, p. 1319–1360, 1994.

BANDYOPADHYAY, S.; MAULIK, U. Nonparametric genetic clustering: comparison of validity indices. **IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)**, IEEE, v. 31, n. 1, p. 120–125, 2001.

BANDYOPADHYAY, S.; MAULIK, U.; PAKHIRA, M. K. Clustering using simulated annealing with probabilistic redistribution. **International Journal of Pattern Recognition and Artificial Intelligence**, World Scientific, v. 15, n. 02, p. 269–285, 2001.

BASKAN, O.; OZAN, C. An ant colony optimization algorithm for area traffic control. In: , 2013.

BERKHIN, P. **Survey Of Clustering Data Mining Techniques**, 2002.

BLUM, C.; PUCHINGER, J.; RAIDL, G. R.; ROLI, A. Hybrid metaheuristics in combinatorial optimization: A survey. **Applied Soft Computing**, Elsevier, v. 11, n. 6, p. 4135–4151, 2011.

BORYCZKA, U. Finding groups in data: Cluster analysis with ants. **Applied Soft Computing**, v. 9, p. 61–70, 2009.

BRESHEARS, C. **The art of concurrency: A thread monkey's guide to writing parallel applications**, 2009.

CHEN, L.; TU, L.; CHEN, H.-J. Data clustering by ant colony on a digraph. In: IEEE. **Machine Learning and Cybernetics, 2005. Proceedings of 2005 International Conference on**, 2005. v. 3, p. 1686–1692.

COWGILL, M. C.; HARVEY, R. J.; WATSON, L. T. A genetic algorithm approach to cluster analysis. **Computers & Mathematics with Applications**, Elsevier, v. 37, n. 7, p. 99–108, 1999.

CRUZ, M. D. **O PROBLEMA DE CLUSTERIZAC AO AUTOMATICA**. Tese (Doutorado) — Universidade Federal do Rio de Janeiro, 2010.

DAS, S.; ABRAHAM, A.; KONAR, A. Automatic clustering using an improved differential evolution algorithm. **IEEE Transactions on systems, man, and cybernetics-Part A: Systems and Humans**, IEEE, v. 38, n. 1, p. 218–237, 2008.

DEBORAH, L. J.; BASKARAN, R.; KANNAN, A. A survey on internal validity measure for cluster validation. **International Journal of Computer Science & Engineering Survey**, v. 1, n. 2, p. 85–102, 2010.

DENEUBOURG, J.; GOSS S., F. N.; SENDOVA-FRANKS, A.; DETRAIN, C.; CHRéTIEN, L. The dynamics of collective sorting: Robot-like ants and ant-like robots. In: **Proceedings of From Animals to Animats, 1st International Conference on the Simulation of Adaptive Behaviour**, 1990. p. 356—-363.

DORIGO, M. **Optimization, Learning and Natural Algorithms**. Tese (Doutorado) — Politecnico di Milano, Italy, 1992.

DORIGO, M.; MANIEZZO, V.; COLORNI, A. Ant system: optimization by a colony of cooperating agents. **IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)**, IEEE, v. 26, n. 1, p. 29–41, 1996.

DRABOWSKI, M.; WANTUCH, E. Scheduling in manufacturing systems – ant colony approach. In: , 2013.

ELVEZIA, C. Ant colony system : A cooperative learning approach to the traveling salesman problem. In: , 1996.

ESTER, M.; KRIEGEL, H.-P.; SANDER, J.; XU, X. et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In: **Kdd**, 1996. v. 96, n. 34, p. 226–231.

FISHER, D. H. Improving inference through conceptual clustering. In: **AAAI**, 1987. v. 87, p. 461–465.

GOETZ, B.; PEIERLS, T.; LEA, D.; BLOCH, J.; BOWBEER, J.; HOLMES, D. **Java concurrency in practice**, 2006.

GUHA, S.; RASTOGI, R.; SHIM, K. Cure: an efficient clustering algorithm for large databases. In: ACM. **ACM Sigmod Record**, 1998. v. 27, n. 2, p. 73–84.

HANDL, J.; MEYER, B. Ant-based and swarm-based clustering. **Swarm Intelligence**, Springer, v. 1, n. 2, p. 95–113, 2007.

HRUSCHKA, E. R.; CAMPELLO, R. J.; FREITAS, A. A. et al. A survey of evolutionary algorithms for clustering. **IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)**, IEEE, v. 39, n. 2, p. 133–155, 2009.

IZAKIAN, H.; ABRAHAM, A. Fuzzy c-means and fuzzy swarm for fuzzy clustering problem. **Expert Systems with Applications**, Elsevier, v. 38, n. 3, p. 1835–1838, 2011.

JAIN, A.; MURTY, M.; FLYNN, P. Data clustering: A review. **ACM Computing Surveys**, v. 31, n. 3, 1999.

JOO, H.; LIM, Y. Ant colony optimized routing strategy for electric vehicles. In: , 2018.

JOSÉ-GARCÍA, A.; GÓMEZ-FLORES, W. Automatic clustering using nature-inspired metaheuristics: A survey. **Applied Soft Computing**, Elsevier, v. 41, p. 192–213, 2016.

KARABOGA, D. **An idea based on honey bee swarm for numerical optimization**, 2005.

KAUFMAN, L.; ROUSSEEUW, P. J. **Finding groups in data: an introduction to cluster analysis**, 2009.

KENNEDY, J. Stereotyping: Improving particle swarm performance with cluster analysis. In: IEEE. **Evolutionary Computation, 2000. Proceedings of the 2000 Congress on**, 2000. v. 2, p. 1507–1512.

KUO, R.; WANG, H.; HU, T.; CHOU, S. Application of ant k-means on clustering analysis. **Computers & Mathematics with Application**, v. 50, p. 1709–1724, 2005.

KURIHARA, S. Traffic-congestion forecasting algorithm based on pheromone communication model. In: , 2013.

LIN, C. et al. **Principles of parallel programming**, 2008.

LINGRAS, P.; WEST, C. Interval set clustering of web users with rough k-means. **Journal of Intelligent Information Systems**, Springer, v. 23, n. 1, p. 5–16, 2004.

LÓPEZ-IBÁNEZ, M.; DUBOIS-LACOSTE, J.; STÜTZLE, T.; BIRATTARI, M. The irace package: Iterated racing for automatic algorithm configuration. **IRIDIA, Universite Libre de Bruxelles, Brussels, Belgium, Tech. Rep. TR/IRIDIA/2011-004**, 2011.

LOURENÇO, H. R.; MARTIN, O. C.; STÜTZLE, T. Iterated local search: Framework and applications. In: **Handbook of metaheuristics**, 2010. p. 363–397.

LUMER, E.; FAIETA, B. Diversity and adaptation in populations of clustering ants. In: **Proceedings of From Animals to Animats, 3rd International Conference on the Simulation of Adaptive Behaviour**, 1994. p. 501—-508.

MACKIEWICZ, A.; RATAJCZAK, W. Principal components analysis (pca). **Computers and Geosciences**, v. 19, p. 303–342, 1993.

MACQUEEN, J. et al. Some methods for classification and analysis of multivariate observations. In: OAKLAND, CA, USA. **Proceedings of the fifth Berkeley symposium on mathematical statistics and probability**, 1967. v. 1, n. 14, p. 281–297.

MONMARCHÉ, N.; SLIMANE, M.; VENTURINI, G. On improving clustering in numerical databases with artificial ants. In: FLOREANO, D.; NICOUD, J.-D.; MONDADA, F. (Ed.). **Advances in Artificial Life**, 1999. p. 626–635. ISBN 978-3-540-48304-5.

NICHOLSON, M. **Genetic Algorithms and grouping problems**, 1998.

PACHECO, T. M.; GONCALVES, L. B.; STRÖELE, V.; SOARES, S. S. R. F. An ant colony optimization for automatic data clustering problem. In: **2018 IEEE**

Congress on Evolutionary Computation, CEC 2018, Rio de Janeiro, Brazil, July 8-13, 2018, 2018. p. 1–8. ISBN 978-1-5090-6017-7. Disponível em: <https://doi.org/10.1109/CEC.2018.8477806>.

PAN, S.-M.; CHENG, K.-S. Evolution-based tabu search approach to automatic clustering. **IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)**, IEEE, v. 37, n. 5, p. 827–838, 2007.

PANG, W.; WANG, K.-p.; ZHOU, C.-g.; DONG. Fuzzy discrete particle swarm optimization for solving traveling salesman problem. In: IEEE. **Computer and Information Technology, 2004. CIT'04. The Fourth International Conference on**, 2004. p. 796–800.

PARPINELLI, R.; LOPES, H.; FREITAS, A. Tdata mining with an ant colony optimization algorithm. In: **Proceedings IEEE Transactions on Evolutionary Computation**, 202. v. 6, p. 321—-332.

RENDÓN, E.; ABUNDEZ, I. M.; GUTIERREZ, C.; ZAGAL, S. D.; ARIZMENDI, A.; QUIROZ, E. M.; ARZATE, H. E. A comparison of internal and external cluster validation indexes. In: **Proceedings of the 2011 American Conference, San Francisco, CA, USA**, 2011. v. 29.

ROUSSEEUW, P. J. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. **Journal of computational and applied mathematics**, Elsevier, v. 20, p. 53–65, 1987.

SCHIKUTA, E.; ERHART, M. The bang-clustering system: Grid-based data analysis. In: SPRINGER. **International Symposium on Intelligent Data Analysis**, 1997. p. 513–524.

SEMAAN, G. S. **Algoritmos para o Problema de Agrupamento Automático**. Tese (Doutorado) — Tese de Doutorado, Instituto de Computação, Universidade Federal Fluminense, 2013.

SETYOHADI, D. B.; BAKAR, A. A.; OTHMAN, Z. A. Optimization overlap clustering based on the hybrid rough discernibility concept and rough k-means. **Intelligent Data Analysis**, IOS Press, v. 19, n. 4, p. 795–823, 2015.

SHELOKAR, P.; JAYARAMAN, V. K.; KULKARNI, B. D. An ant colony approach for clustering. **Analytica Chimica Acta**, Elsevier, v. 509, n. 2, p. 187–195, 2004.

SIBSON, R. Slink: an optimally efficient algorithm for the single-link cluster method. **The computer journal**, Oxford University Press, v. 16, n. 1, p. 30–34, 1973.

SOARES, S.; OCHI, L. S. Um algoritmo evolutivo com reconexão de caminhos para o problema de clusterização automática. In: **Proc. of XII CLAIO**, 2004. p. 7–13.

SOARES, S. S. **Metaheurísticas para o problema de clusterização Automática**. Dissertação (Mestrado) — Universidade Federal Fluminense, Niterói, 2004.

TALBI, E.-G. **Metaheuristics: from design to implementation**, 2009.

TANENBAUM, A. S.; FILHO, N. M. **Sistemas operacionais modernos**, 1995.

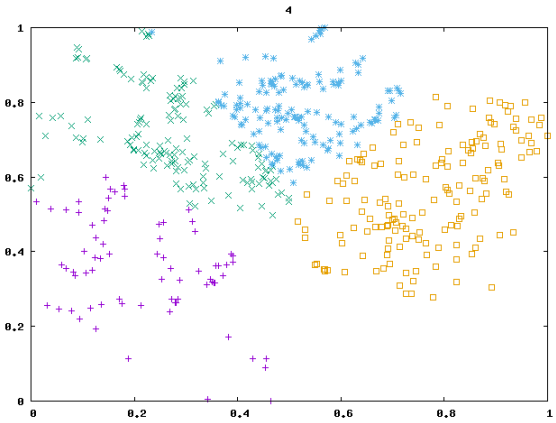
TIWARI, R.; HUSAIN, M.; GUPTA, S.; SRIVASTAVA, A. Improving ant colony optimization algorithm for data clustering. In: **International Conference and Workshop on Emerging Trends in Technology**, 2010. p. 529–534.

WANG, W.; YANG, J.; MUNTZ, R. et al. Sting: A statistical information grid approach to spatial data mining. In: **VLDB**, 1997. v. 97, p. 186–195.

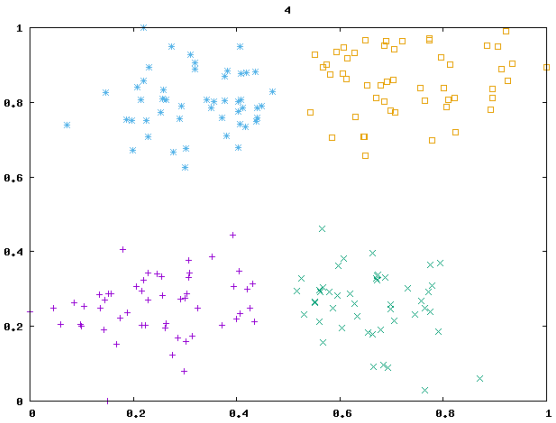
WILCOXON, F. Individual comparisons by ranking methods. **Biometrics bulletin**, JSTOR, v. 1, n. 6, p. 80–83, 1945.

XU, R.; WUNSCH, D. Survey of clustering algorithms. **IEEE Transactions on neural networks**, Ieee, v. 16, n. 3, p. 645–678, 2005.

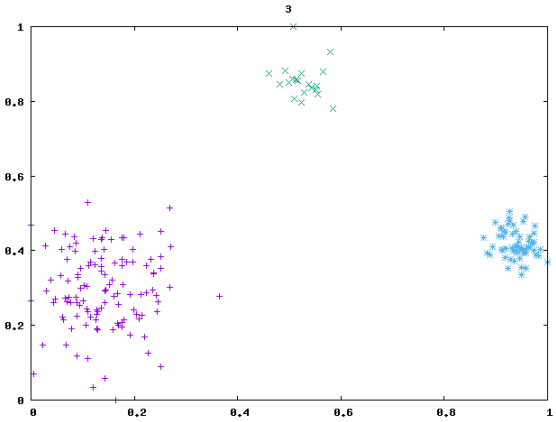
Apêndice A - APRESENTAÇÃO VISUAL DOS AGRUPAMENTOS



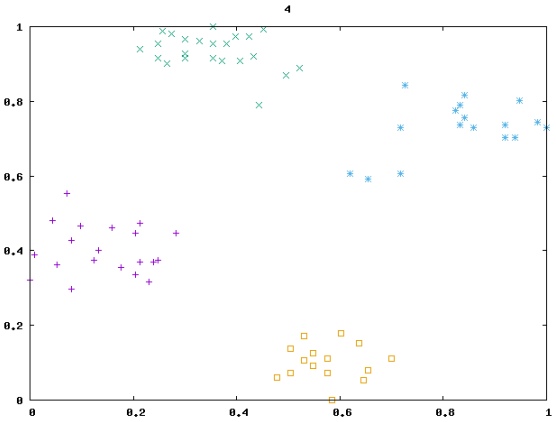
(a) vowel



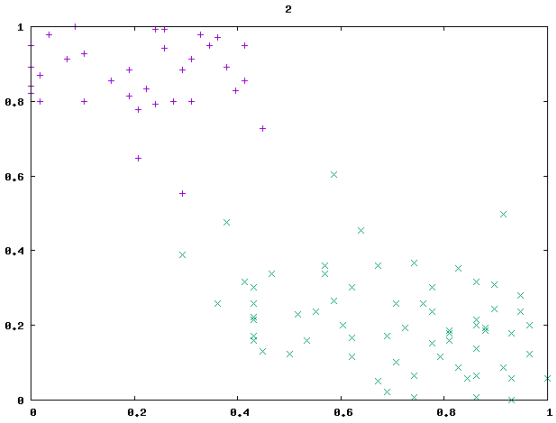
(b) maronna



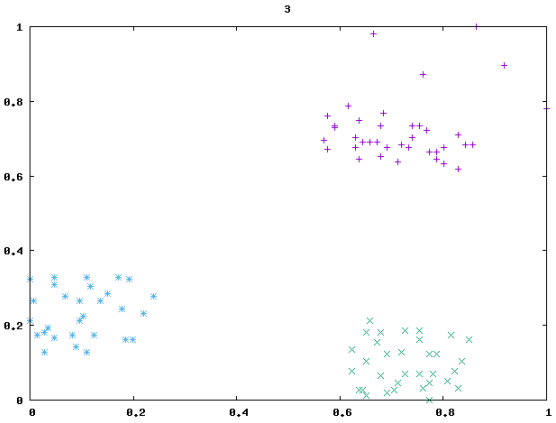
(c) 200DATA



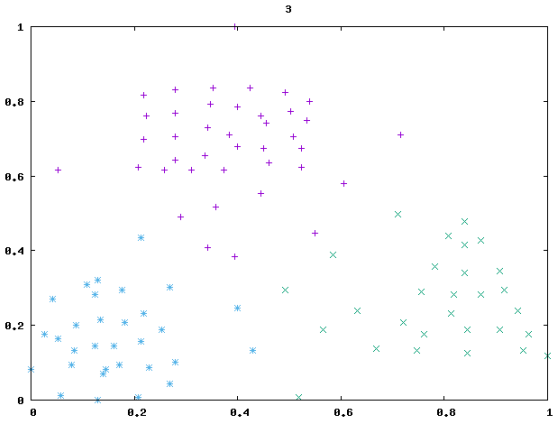
(d) ruspini



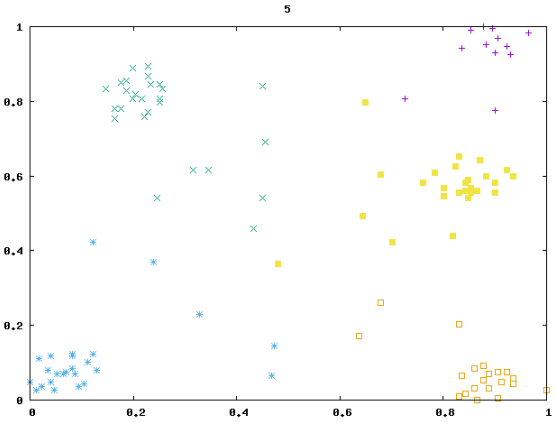
(e) 100p2c1



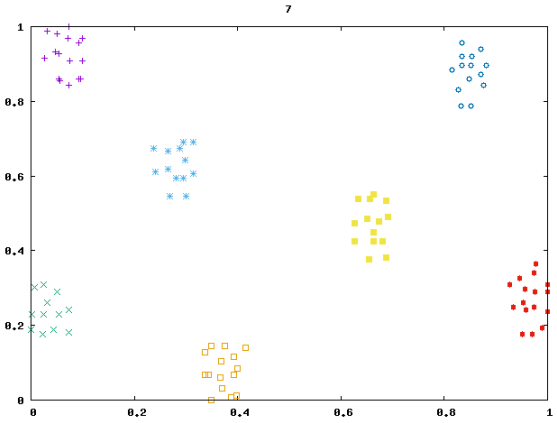
(f) 100p3c



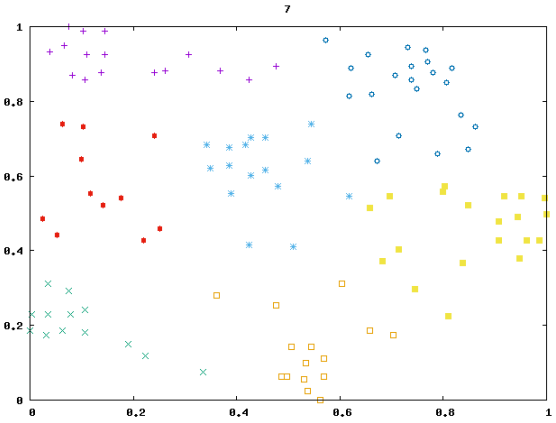
(g) 100p3c1



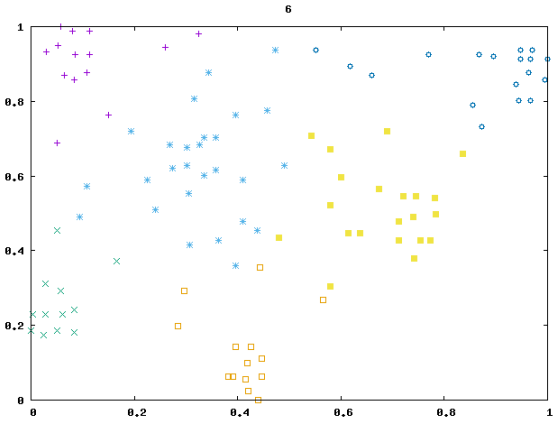
(h) 100p5c1



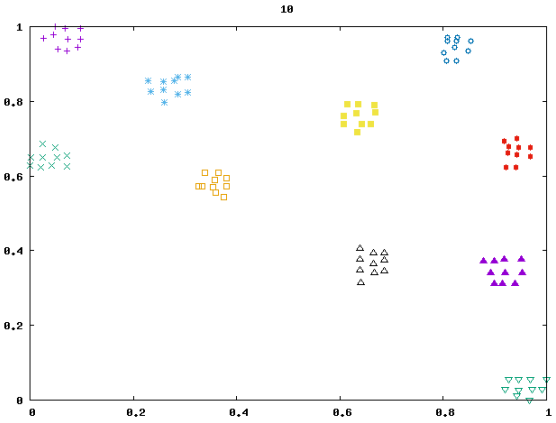
(i) 100p7c



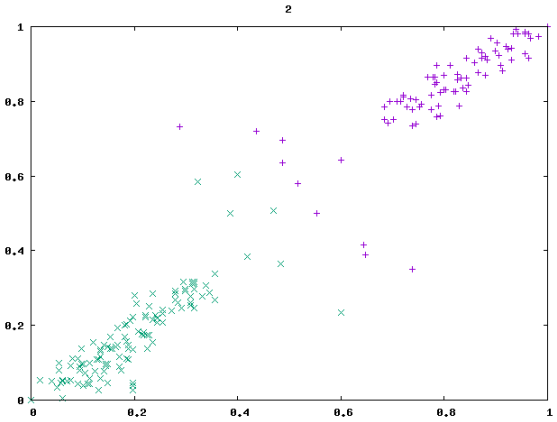
(j) 100p7c1



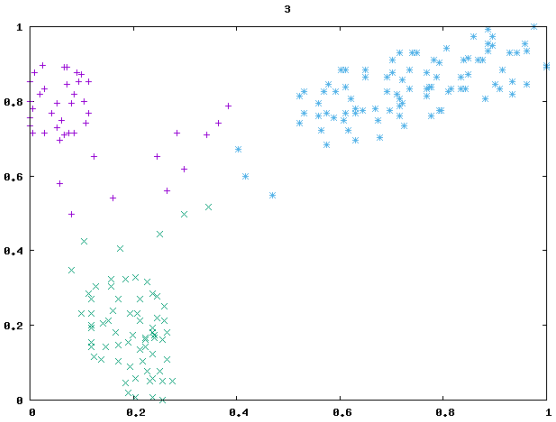
(k) 100p8c1



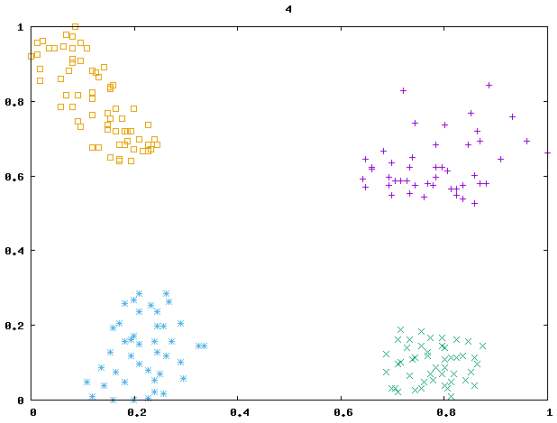
(l) 100p10c



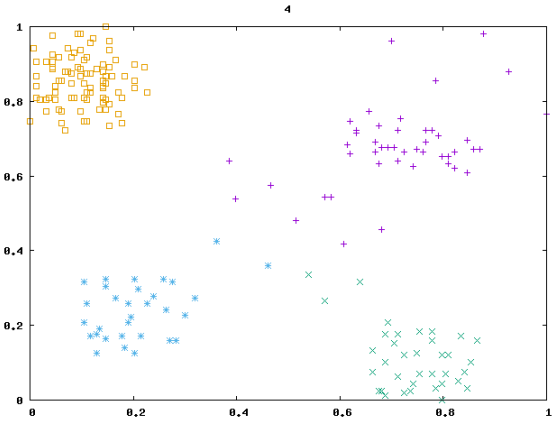
(m) 200p2c1



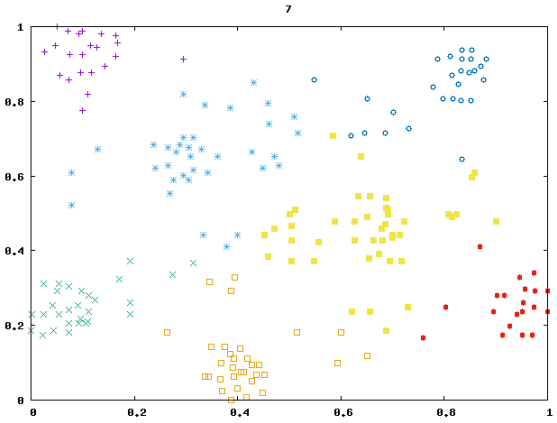
(n) 200p3c1



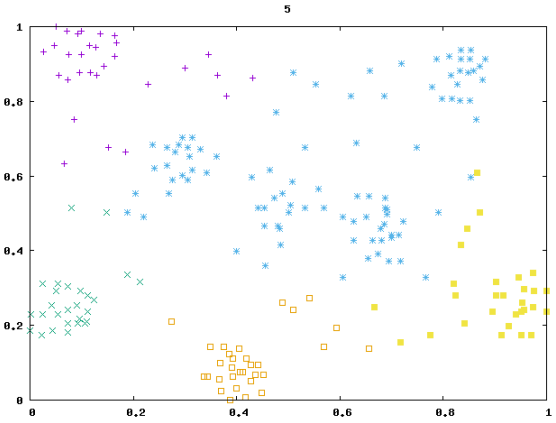
(o) 200p4c



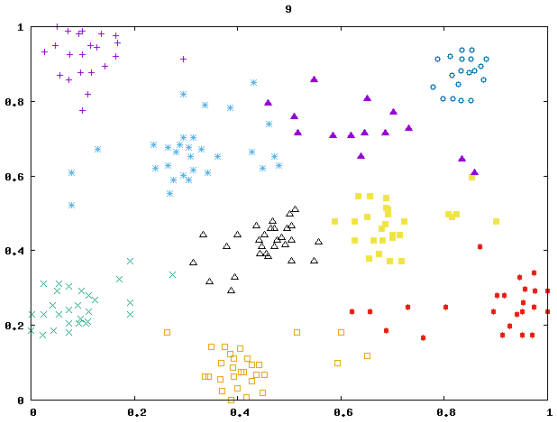
(p) 200p4c1



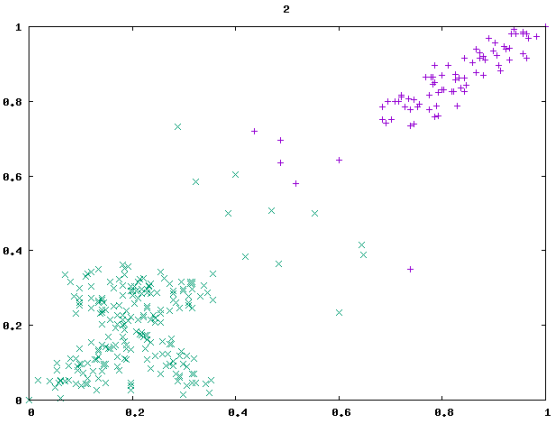
(q) 200p7c1



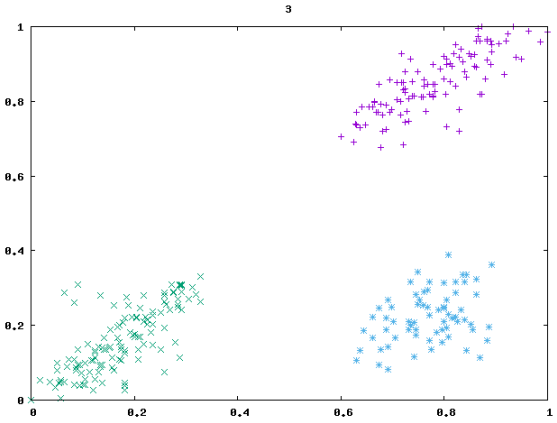
(r) 200p8c1



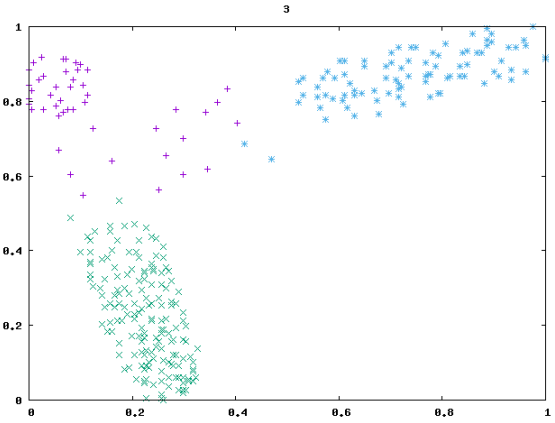
(s) 200p12c1



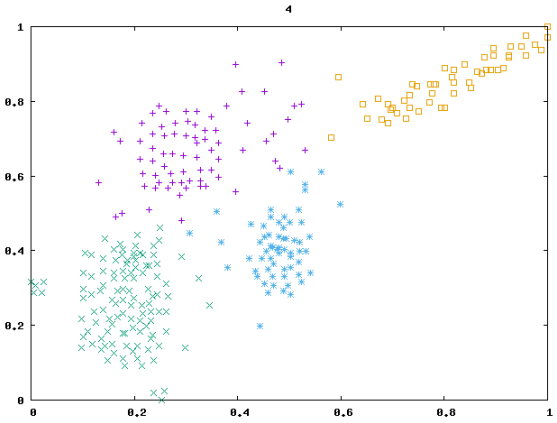
(t) 300p2c1



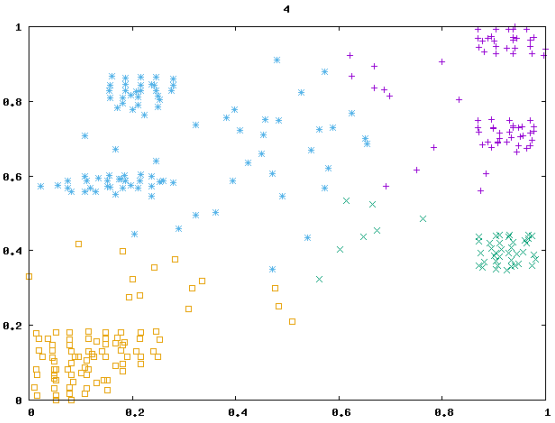
(u) 300p3c



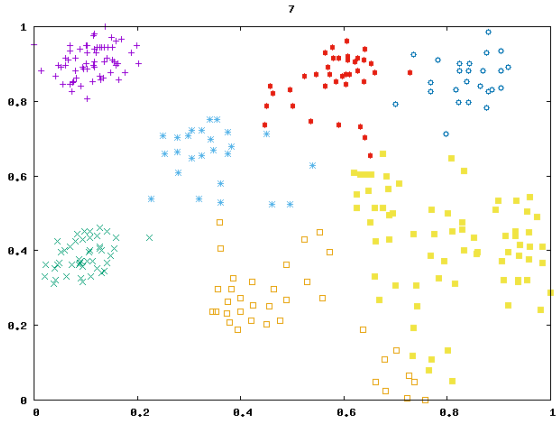
(v) 300p3c1



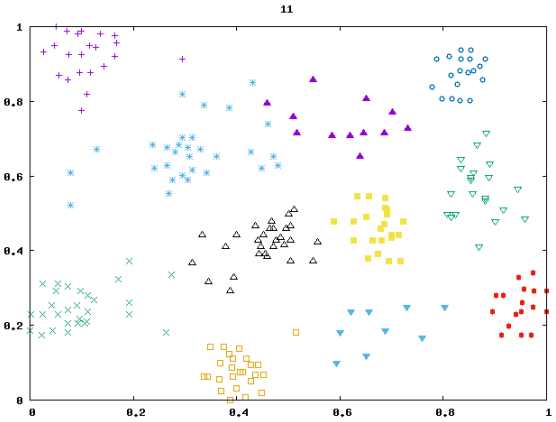
(w) 300p4c1



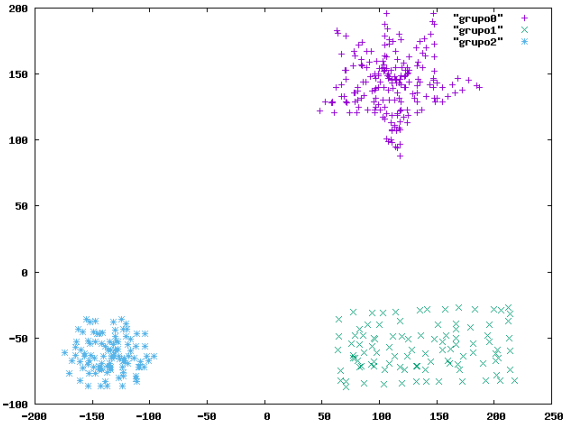
(x) 300p6c1



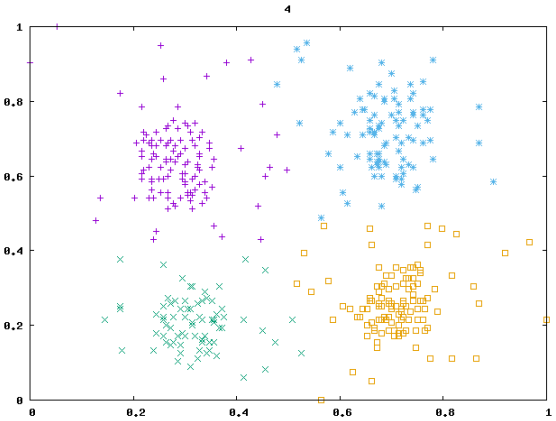
(y) 300p10c1



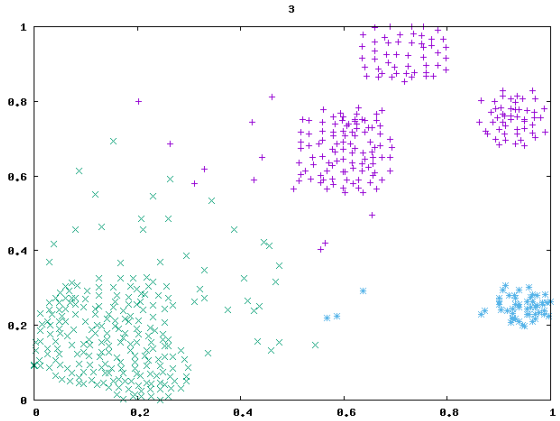
(z) 300p13c1



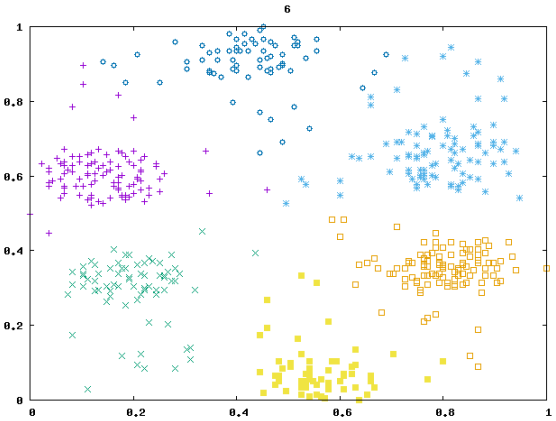
(aa) 400p3c



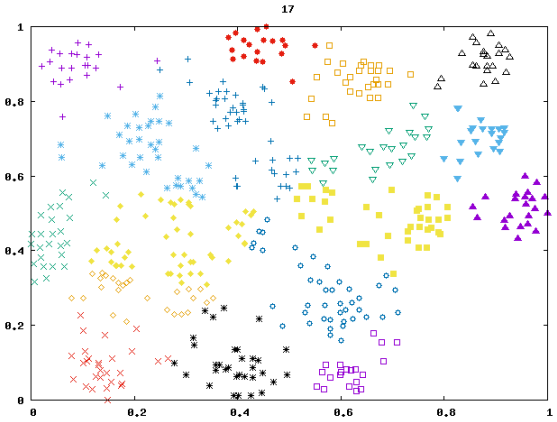
(ab) 400p4c1



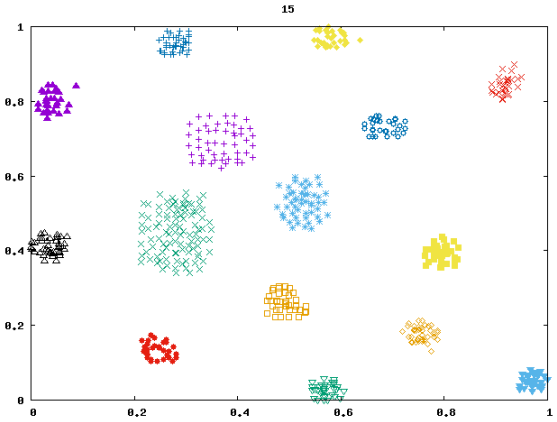
(ac) 500p4c1



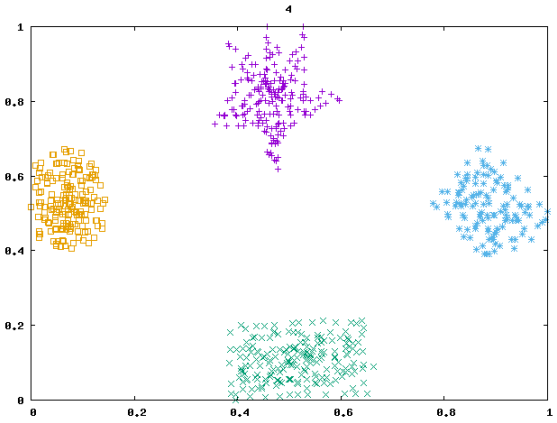
(ad) 500p6c1



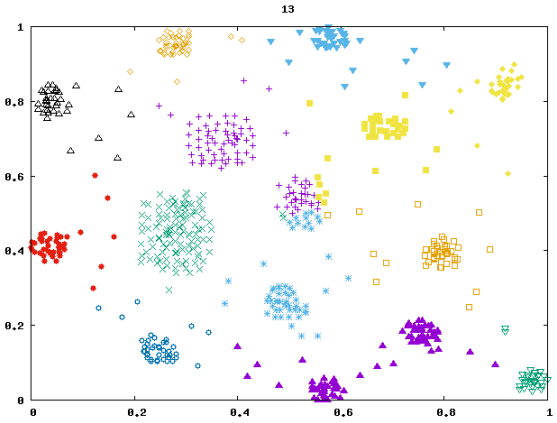
(ae) 500p19c1



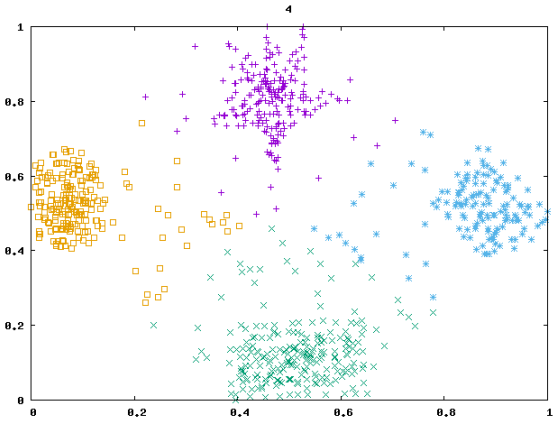
(af) 600p15c



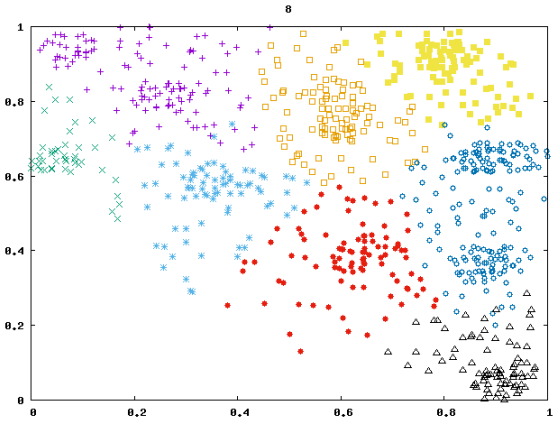
(ag) 700p4c



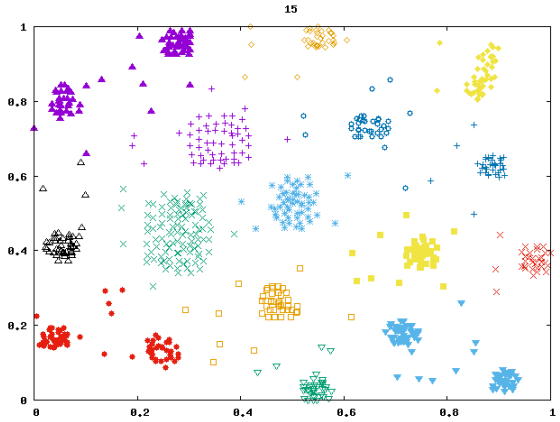
(ah) 700p15c1



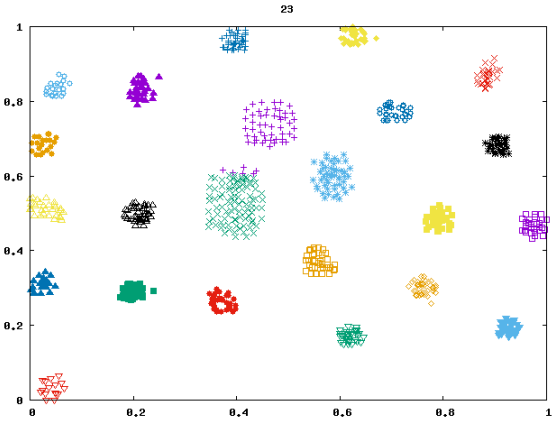
(ai) 800p4c1



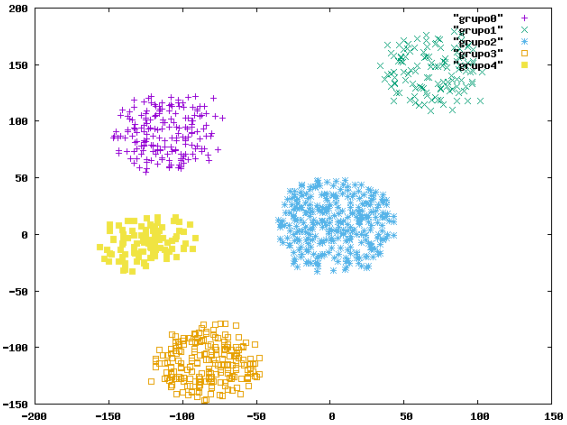
(aj) 800p10c1



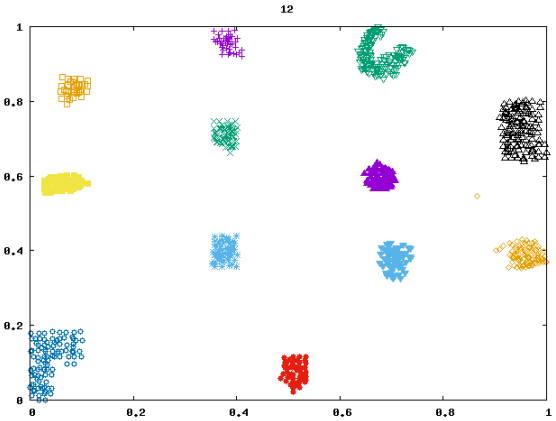
(ak) 800p18c1



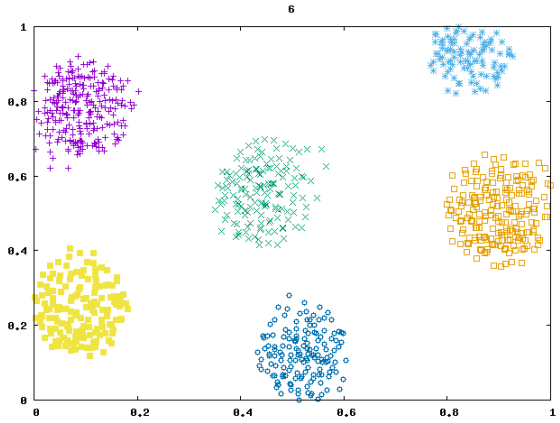
(al) 800p23c



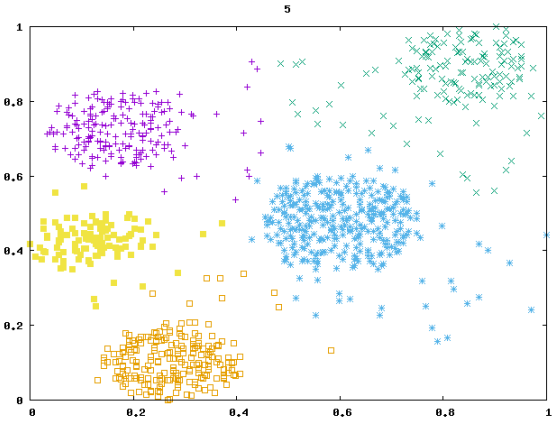
(am) 900p5c



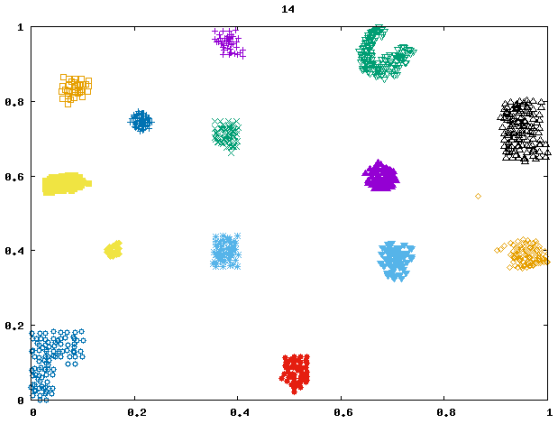
(an) 900p12c



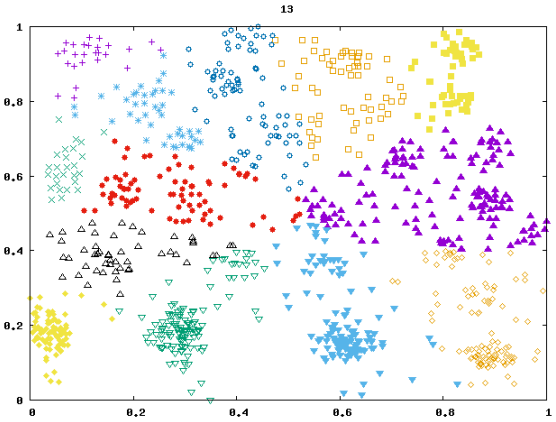
(ao) 1000p6c



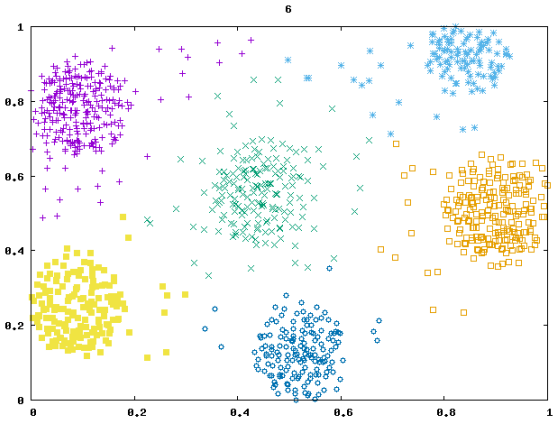
(ap) 1000p5c1



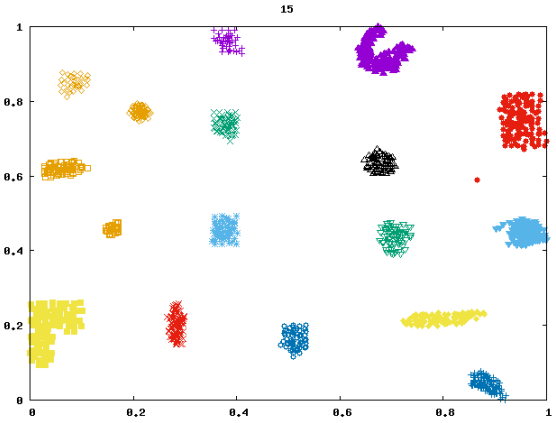
(aq) 1000p14c



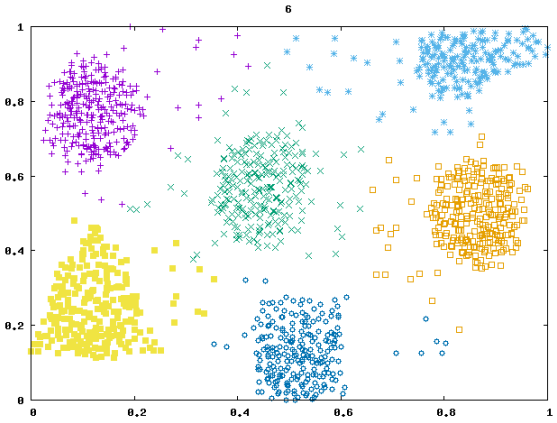
(ar) 1000p27c1



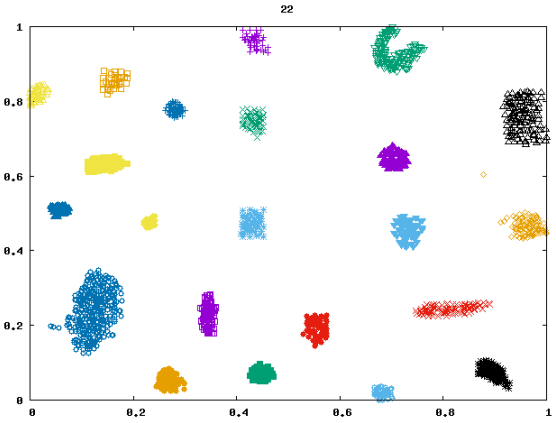
(as) 1100p6c1



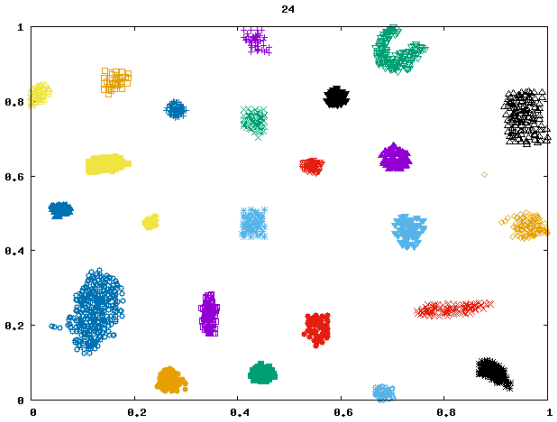
(at) 1300p17c



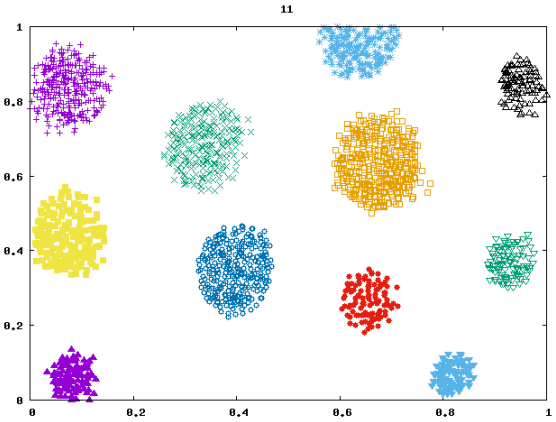
(au) 1500p6c1



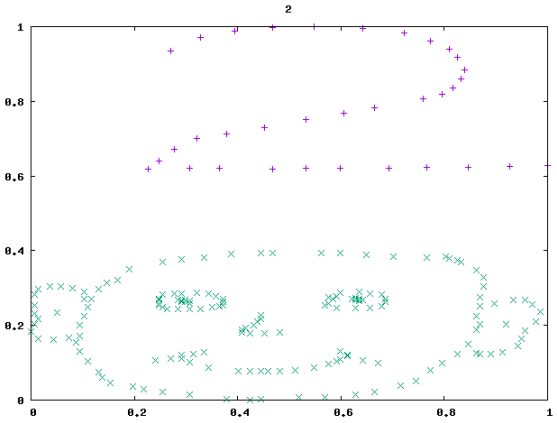
(av) 1800p22c



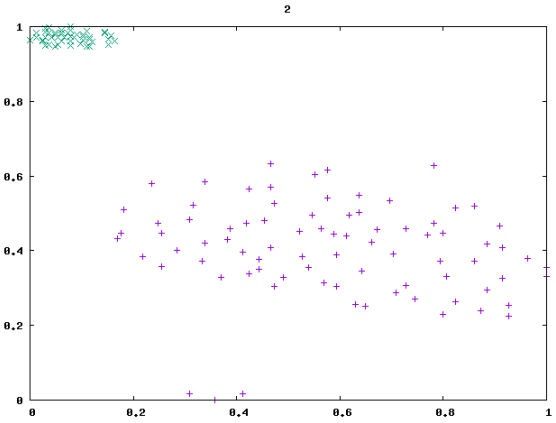
(aw) 1900p24c



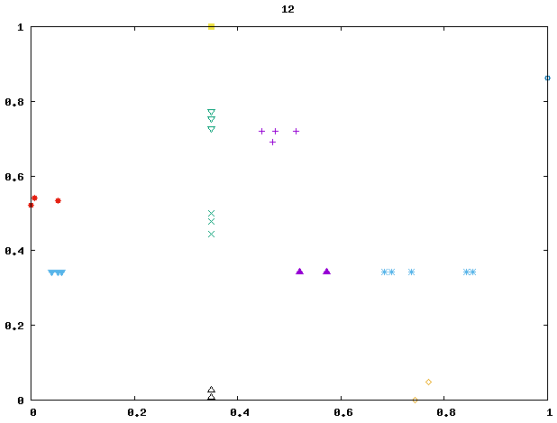
(ax) 2000p11c



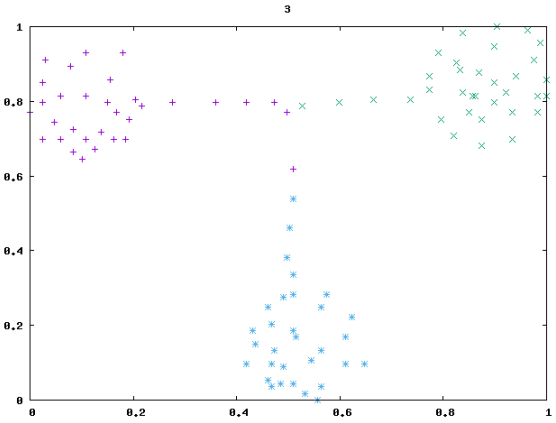
(ay) 2face



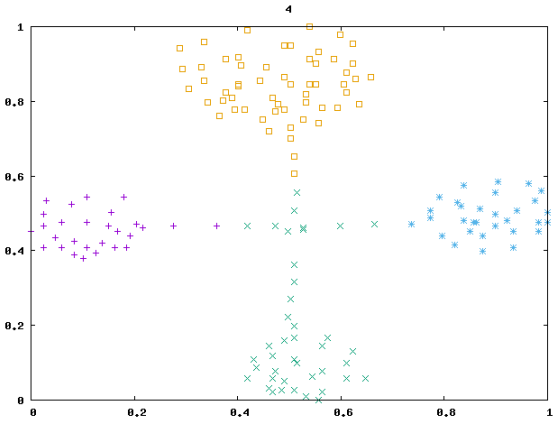
(az) 3dens



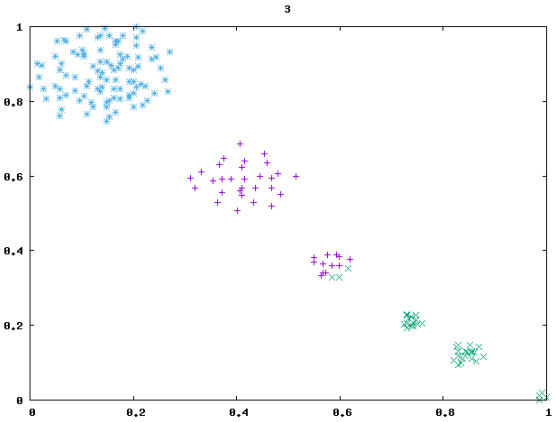
(ba) 30p



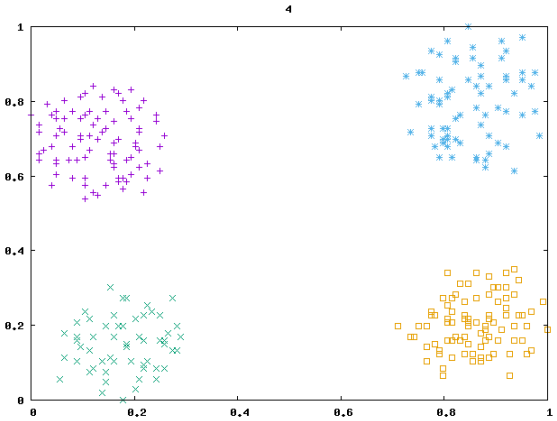
(bb) 97p



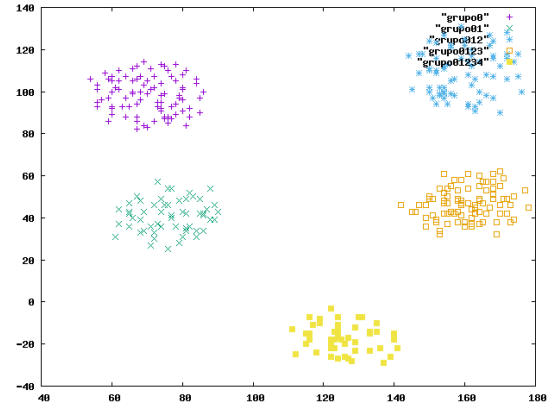
(bc) 157p



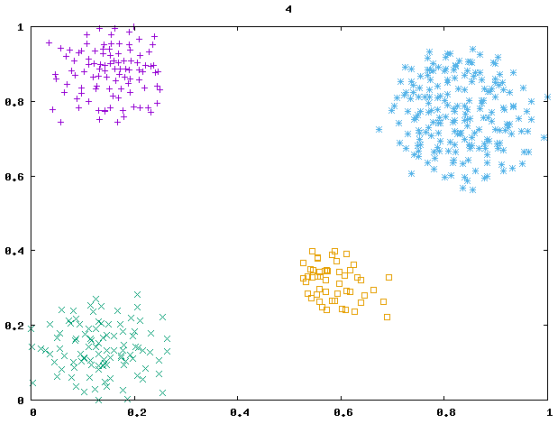
(bd) 181p



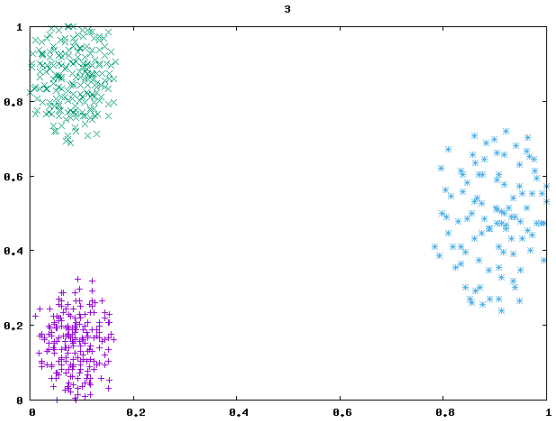
(be) 300p4c



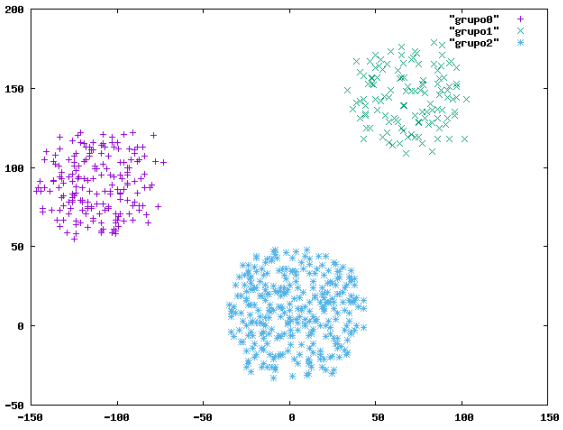
(bf) 350p5c



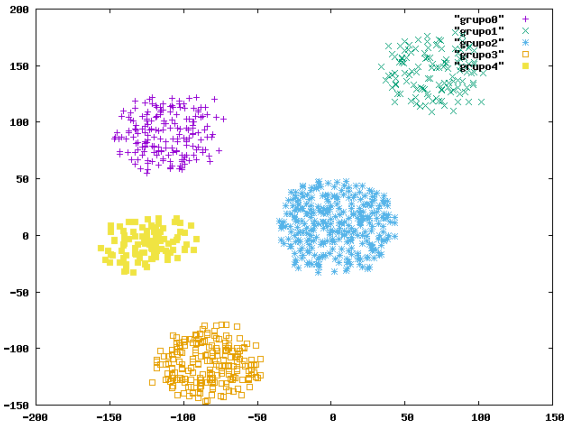
(bg) 450p4c



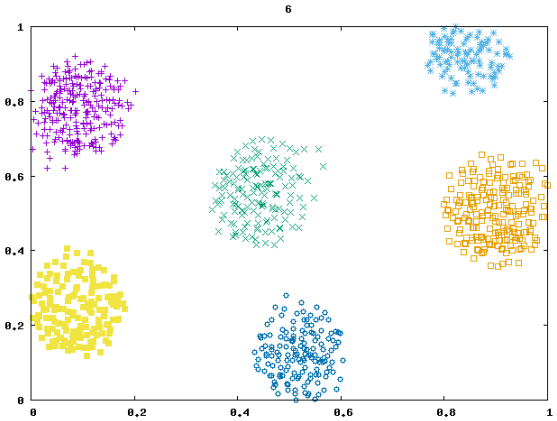
(bh) 500p3c



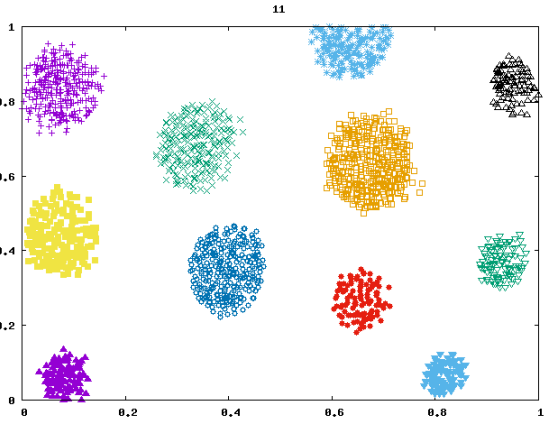
(bi) 600p3c



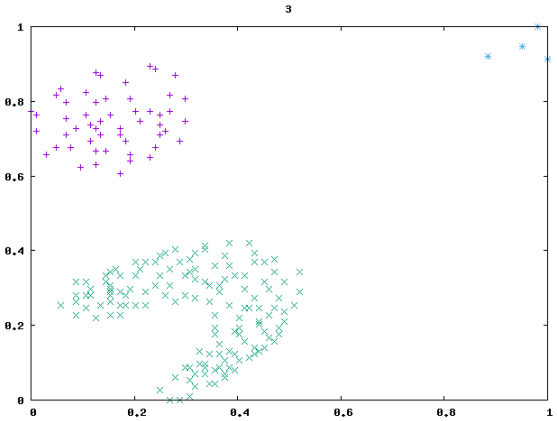
(bj) 900p5c



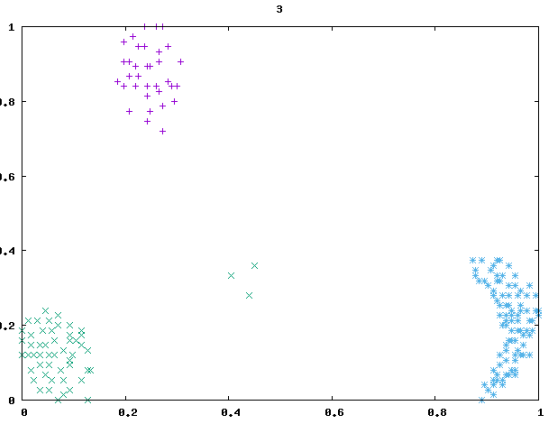
(bk) 1000p6c



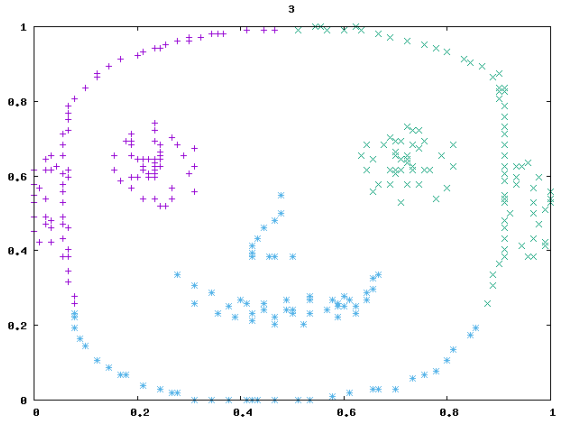
(bl) 2000p11c



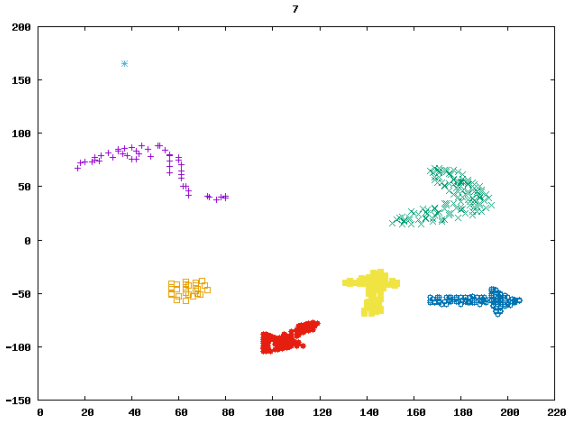
(bm) convexo



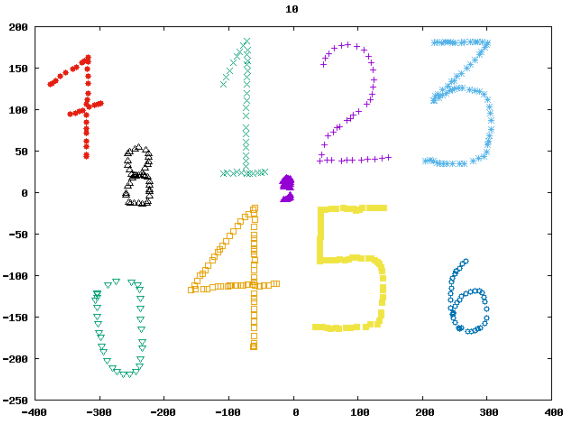
(bn) convdensity



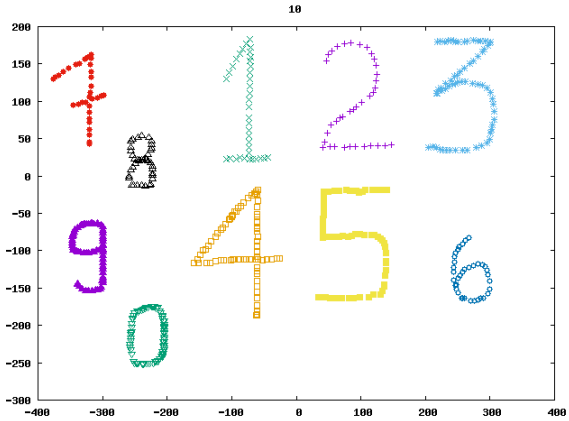
(bo) face



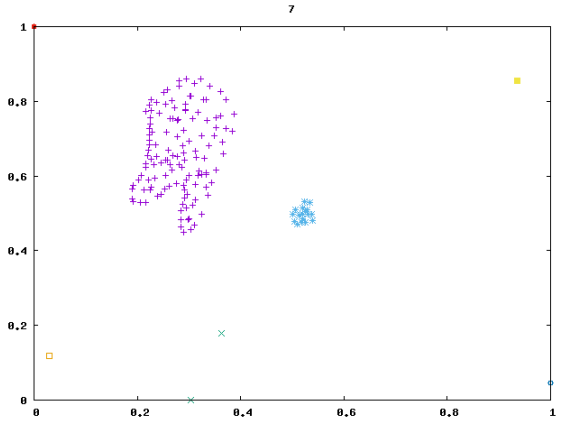
(bp) moreshapes



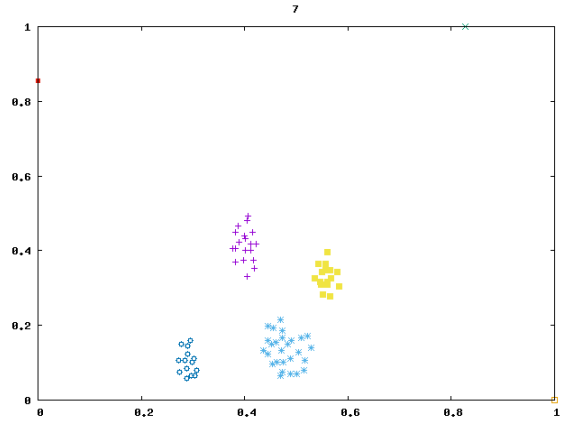
(bq) numbers



(br) numbers2



(bs) outliers



(bt) outliers_ags

Apêndice B - *DATASETS* UTILIZADOS NOS EXPERIMENTOS

Os *datasets* utilizados nos experimentos são apresentados nas tabelas B.1, B.2 e B.3. Onde a coluna *Dataset* é referente ao nome do *dataset*, a coluna Nº dimensões apresenta a quantidade de atributos e a coluna Nº de Objetos indica a quantidade de objetos pertencentes a cada *dataset*.

Tabela B.1: Conjunto de *datasets* (DS1)

Dataset	**Nº Dimensões**	**Nº Objetos**
200Data	R2	200
gauss9	R2	900
iris	R4	150
maronna	R2	200
ruspini	R2	75
spherical_4d3c	R3	400
vowel2	R2	528
wine	R13	178
yeast	R7	1484

Tabela B.2: Conjunto de *datasets* (DS2)

Dataset	**Nº Dimensões**	**Nº Objetos**	**Dataset**	**Nº Dimensões**	**Nº Objetos**
100p10c	R2	100	500p19c1	R2	500
100p2c1	R2	100	500p3c	R2	500
100p3c	R2	100	500p4c1	R2	500
100p3c1	R2	100	500p6c1	R2	500
100p7c	R2	100	600p15c	R2	600
100p8c1	R2	106	600p13c1	R2	600
100p5c1	R2	110	700p4c	R2	700
100p7c1	R2	112	700p15c1	R2	703
200p2c1	R2	200	800p10c1	R2	800
200p3c1	R2	200	800p18c1	R2	800
200p4c	R2	200	800p4c1	R2	800
200p4c1	R2	200	800p23c	R2	806
200p7c1	R2	210	900p12c	R2	900
200p8c1	R2	212	900p5c	R2	900
200p12c1	R2	222	1000p14c	R2	1000
300p13c1	R2	235	1000p5c1	R2	1000
300p10c1	R2	300	1000p6c	R2	1000
300p2c1	R2	300	1000p27c1	R2	1005
300p3c	R2	300	1100p6c1	R2	1100
300p3c1	R2	300	1300p17c	R2	1300
300p4c1	R2	300	1500p6c1	R2	1500
300p6c1	R2	300	1800p22c	R2	1800
400p3c	R2	400	1900p24c	R2	1901
400p4c1	R2	400	2000p11c	R2	2000
400p17c1	R2	408	2000p26c	R2	2000

Tabela B.3: Conjunto de *datasets* (DS3)

Dataset	**Nº de Atributos**	**Nº Objetos**
30p	R2	30
outliers_args	R2	80
97p	R2	97
3dens	R2	128
Outliers	R2	150
157p	R2	157
convdensity	R2	175
181p	R2	181
convexo	R2	199
2face	R2	200
300p4c	R2	300
350p5c	R2	350
numbers	R2	437
450p4c	R2	450
moreshapes	R2	489
500p3c	R2	500
numbers2	R2	540
600p3c	R2	600
900p5c	R2	900
1000p6c	R2	1000
2000p11c	R2	2000