

ESCALONAMENTO DE EQUIPES DE CAMPO DA ESCELSA RELATÓRIO PARCIAL

RAUL H.C. LOPES

1. INTRODUÇÃO

O objetivo deste relatório consiste em apresentar resultados referentes à terceira fase da pesquisa do projeto *Sistema de Escalonamento de Equipes de Campo da ESCELSA(SEECE)*.

Nesta fase, foram selecionados algoritmos para o problema fundamental do projeto: distribuição das equipes de atendimento de emergência. Além disso, foi definida uma metodologia para avaliação da qualidade dos algoritmos definidos.

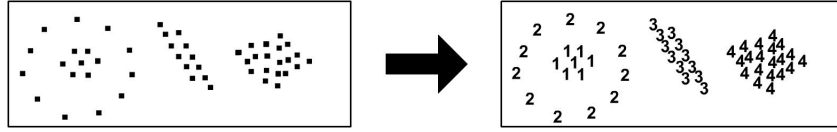
2. O PROCESSO DE VALIDAÇÃO

O objetivo final do *SEECE* consiste em produzir um sistema que permite fornecer subsídios para a distribuição de equipes de manutenção pelo estado. A validação das distribuições eventualmente propostas deverá ser realizada via simulação computacional de despachos de equipes e de atendimentos baseados nos dados históricos da ESCELSA. Conforme planejamento apresentado em relatórios anteriores, o *SEECE* será, então, composto de duas funcionalidades básicas:

- subsistema de distribuição de equipes de manutenção pelo estado;
- subsistema de simulação de despacho de equipes que, atuando sobre dados históricos da empresa, permite avaliar a qualidade das soluções propostas pela primeira funcionalidade.

Durante a etapa atual do projeto foram realizadas as seguintes atividades:

- seleção e implementação de algoritmos de agrupamento (*clustering*) para tratamento da funcionalidade de distribuição de equipes pelo estado.
- seleção e implementação de algoritmos para simulação de despacho e atendimento.

FIGURA 1. Um exemplo de *clustering*

É importante observar que o objetivo final do projeto consiste em produzir um sistema que proponha distribuição de equipes que permita realizar atendimentos que levem à redução dos índices de **TMM** e **DEC**.

3. ALGORITMOS PARA DISTRIBUIÇÃO DE EQUIPES

O problema de distribuição de equipes pelo estado vem sendo identificado neste projeto como o problema de se agrupar os possíveis focos de problema e associar uma ou mais equipes de atendimento a cada grupo reduzindo assim o tempo de deslocamento de cada equipe.

Uma das técnicas clássicas de agrupamento é identificada na literatura [9, 5] como *clustering*: a classificação não-supervisionada de padrões (observações, itens de dados ou vetores de características) em grupos (*clusters*)” [10].

4. O PROBLEMA DE AGRUPAMENTO

Um exemplo clássico da literatura de computação geométrica e de *clustering* consiste em escolher onde localizar k hospitais numa cidade de modo que nenhum habitante more mais longe que o necessário do hospital mais próximo. Na literatura de geometria combinatorial, ver por exemplo [3, 4], este problema é usualmente conhecido como *Post Office Problem* e resolvido usando diagramas de Voronoi. Este problema pode, no entanto, não ser facilmente resolvido, dependendo da definição de distância entre os pontos. *Alpert* cita-o como exemplo de um problema de *clustering* em [1]. Outro exemplo é o de agrupar os pontos mais similares entre si da primeira parte da figura 1. Na segunda parte da figura, cada ponto foi substituído pelo número do *cluster* a que ele pertence, indentificado pelo processo de *clustering*.

Bastante estudado na literatura, *clustering* trata-se basicamente de agrupar determinados objetos em grupos de acordo com características comuns a eles, e separá-los de acordo com características que os diferenciem. Segundo Guha *et al.* [6], o problema de *clustering* pode ser definido como a seguir: dados n objetos num espaço métrico de d dimensões, particiona-se estes objetos em k grupos (*clusters*), tais que

os objetos em um grupo são mais similares entre si do que em relação a objetos de outros grupos.

Os objetos analisados precisam ter características que possam vir a ser usadas para agrupá-los ou separá-los. Eles costumam ter, portanto, um vetor de d dimensões com essas características. Para analisá-las, é necessário uma função de similaridade que leve em conta as características mais relevantes para a classificação. Por exemplo, pontos num espaço bidimensional poderiam ser agrupados de acordo com suas coordenadas cartesianas, e a função de similaridade poderia usar apenas a Distância Euclidiana como fator de classificação. É importante salientar que, quando a função de similaridade é dada pela distância Euclideana entre os pontos, freqüentemente soluções da geometria computacional, como diagramas de Voronoi e triangulação revelam-se mais precisos e eficientes.

Jain *et al.* [10] citam 5 passos típicos envolvidos num processo de agrupamento:

- (1) Representação dos Padrões : Leva em consideração o número de padrões, número de classes e número, tipo e escala das características. Nesta etapa, opcionalmente, pode haver a *Seleção das Características* mais apropriadas a serem analisados no agrupamento. Pode haver também a *Extração das Características*, que é o ato de transformar as características dos objetos em algo mensurável.
- (2) Definição da medida de proximidade dos padrões apropriada ao domínio de dados (normalmente, uma função de distância entre pares de objetos).
- (3) Agrupamento (*clustering*).
- (4) Abstração dos dados: Trata-se de simplificar e compactar a representação de um conjunto de dados, para uma análise automática posterior por alguma máquina ou para ser compreendida por humanos.
- (5) Geração de uma saída (*output*).

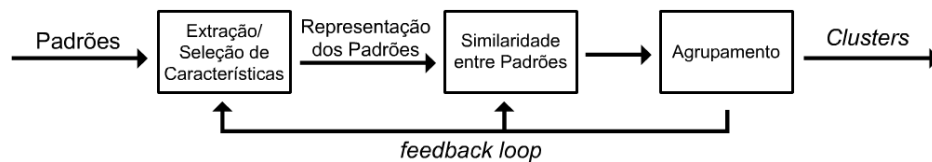


FIGURA 2. Etapas do agrupamento

Além destas etapas, pode haver ainda uma última de validação dos grupos gerados, onde será feita uma análise do que foi obtido com o algoritmo.

A figura 2 mostra uma sequência típica das três primeiras etapas, incluindo ainda um *feedback loop*, no qual a saída do processo de agrupamento pode afetar a próxima extração de características ou as computações de similaridade.

5. AGRUPAMENTO E DISTRIBUIÇÃO DE EQUIPES

No problema estudado os objetos a serem agrupados serão as chaves da empresa, que estão distribuídas geograficamente pelo estado. As características destas chaves a serem analisadas no processo de *clustering* são, além de suas posições geográficas, suas densidades de ocorrências de atendimentos retiradas do histórico e a quantidade de consumidores atingidos por elas.

Para sugerir os novos posicionamentos das equipes, diversos algoritmos de *clustering* foram implementados e comparados com o objetivo de determinar os que melhor se adaptam ao problema. A validação das distribuições dos algoritmos será feita por simulações de despachos de equipes com dados históricos da empresa.

6. DEFINIÇÃO FORMAL

Uma definição formal do problema de *clustering* é encontrada em [1, 2] e descrita a seguir:

- $X = \{X_1, X_2, \dots, X_n\}$ é o conjunto de todos os n objetos da base de dados a serem classificados. Cada $X_i \in \mathbb{R}^d$ é um vetor de dimensão d (as d características do objeto).
- $C = \{C_1, C_2, \dots, C_k\}$ são os k grupos de objetos (*clusters*) formados pelo algoritmo. As seguintes propriedades devem ser respeitadas no processo de formação dos *clusters*.
 - (1) $C_1 \cup C_2 \cup \dots \cup C_k = X$;
 - (2) $C_i \neq \emptyset, \forall i, 1 \leq i \leq k$;
 - (3) $C_i \cap C_j = \emptyset, \forall i \neq j, 1 \leq i \leq k, 1 \leq j \leq k$

Em suma, as propriedades acima significam que cada grupo precisa ter ao menos um objeto e que cada objeto deve estar contido em exatamente um grupo.

7. ALGORITMOS TRADICIONAIS DE AGRUPAMENTO

Os algoritmos conhecidos de *clustering* atendem a diferentes tipos de requisitos, tais como [2]:

- encontrar ou não um número adequado de *clusters* (alguns algoritmos precisam de um valor previamente determinado de *clusters*).
- ser capazes de desprezar ou não os ruídos.
- descobrir *clusters* com formas arbitrárias.
- identificar *clusters* de tamanhos variados.
- trabalhar com objetos de diversos números de atributos.
- fornecer resultados interpretáveis e utilizáveis.
- apresentar o resultado num tempo satisfatório.

Como não há um método que atenda a todas estas características da melhor forma, foram criados algoritmos que se adaptam melhor a alguns conjuntos dessas características. Ao utilizar um certo método, deve-se analisar quais dos aspectos citados acima ele satisfaz e escolher aquele que se adaptar melhor às suas necessidades. De acordo com essas características, os algoritmos de *clustering* foram divididos em categorias [2]:

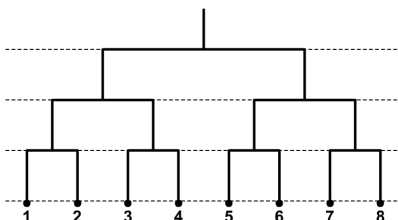
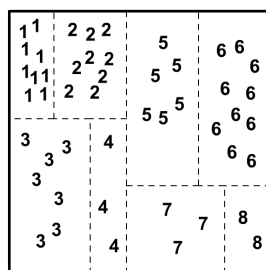
- Métodos por particionamento;
- Métodos hierárquicos;
- Métodos baseados em densidade;
- Métodos baseados em grade;
- Métodos baseados em modelos;

Os algoritmos mais tradicionais de *clustering* existentes são os métodos por particionamento e hierárquicos, descritos a seguir.

7.1. Métodos por Particionamento. A idéia principal deste tipo de algoritmo é que ele divide o conjunto de objetos em k grupos, sendo que k é dado como entrada pelo usuário. Na maioria dos problemas este número não é conhecido, tornando o algoritmo inutilizável em alguns casos.

Primeiramente, o algoritmo escolhe k objetos da base de dados que serão os centros iniciais de k *clusters*. Cada objeto é então atribuído ao *cluster* cujo centro lhe é mais similar, de acordo com a função de similaridade adotada. Quando todo objeto já estiver em um grupo, o centro de cada *cluster* é recalculado e os objetos são reatribuídos aos seus centros mais próximos. O algoritmo termina quando um critério de parada é atingido. Esse critério pode ser, por exemplo, um número máximo de iterações ou quando não há mais mudança nos centros dos grupos.

Os algoritmos de *clustering* por particionamento se distinguem pela forma como os k centros iniciais são escolhidos e como tais centros são recalculados em cada iteração.

FIGURA 3. Exemplo de um *dendograma*FIGURA 4. *Clusters* formados por um método hierárquico

7.2. Métodos Hierárquicos. Ao contrário dos métodos por partição, os hierárquicos não requerem um número k de grupos fixado *a priori*. Assim como os de partição, são também bastante populares, apesar de serem geralmente mais custosos.

Num método hierárquico, é formada uma árvore na qual cada nível representa uma subdivisão nos conjuntos de objetos, conhecida como *dendograma* (figura 3). O *dendograma* irá, então, representar os *clusters* gerados pelo algoritmo. No fim da árvore temos cada objeto em apenas um grupo (figura 4).

Se a árvore for sendo formada de cima para baixo, o *clustering* é chamado de aglomerativo. Dessa forma, começamos com cada objeto num *cluster* e, a cada passo, juntamos um par de grupos até que haja apenas um grande grupo que contenha todos os objetos (raiz da árvore).

Caso contrário, o *clustering* é chamado de divisivo. Começa-se com todos os objetos em um só grupo, a partir do qual se formam subgrupos até que se isole cada objeto num só *cluster*.

8. ALGORITMOS ESTUDADOS

8.1. *k-means*. O algoritmo *k-means* [9] é o mais conhecido método por particionamento. Nesse algoritmo, os k centros iniciais são escolhidos aleatoriamente. No final de cada iteração, esses centros passam a

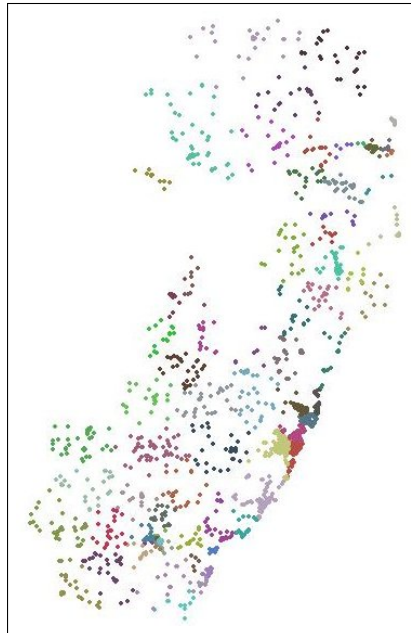


FIGURA 5. Um resultado do *kmeans*

ser o centro de gravidade de cada *cluster* (média dos pontos do grupo). Abaixo, apresenta-se um pseudo-código do algoritmo.

```
def kmeans(k, objects):
    means = sample(objects, k)
    changed = True
    while changed:
        clusters = group(means, points)
        newMeans = reCalcMeans(points)
        changed = (means == newMeans)
        means = newMeans
    return clusters
```

A função *sample()* retorna uma lista de k objetos escolhidos aleatoriamente. A função *group()* inclui cada objeto no *cluster* cujo centro lhe é mais similar, retornando uma lista com os k *clusters* e seus respectivos objetos. Por fim, *reCalcMeans()* servirá para calcular os novos centros de gravidade de cada grupo. O algoritmo termina quando os centros não se modificam mais.

O *kmeans* é bastante popular por ser fácil de implementar e por ter um bom desempenho em base de dados grandes, já que sua complexidade computacional é $O(n \cdot k \cdot t)$, onde n é o total de objetos da base de

dados, k é o número de grupos formados e t é o número de iterações. Segundo *Carlantonio* [2], normalmente $n \gg k$ e $n \gg t$.

O fato de o algoritmo *kmeans* exigir que definamos inicialmente quantos grupos serão formados não foi prejudicial no nosso problema. Isso porque utilizamos k = número de equipes de atendimento que a empresa possui no Estado. Em contrapartida, o algoritmo não permitiu que levássemos em conta o número de consumidores afetados por uma interrupção de energia na formação dos *clusters*.

Uma variação deste método utilizada foi: ao recalcularmos os novos centros a cada iteração, foi utilizado em cada grupo o objeto mais próximo do seu centro de gravidade, ao invés do próprio centro de gravidade. Essa alteração não representou muita variação no formato dos *clusters* obtidos em relação ao algoritmo original.

Na figura 5, vemos um resultado do algoritmo *kmeans* com a adaptação descrita acima. As chaves de mesma cor foram agrupadas em um mesmo *cluster*. Foram gerados 75 grupos, que é uma média da quantidade de equipes da empresa no Estado.

Vale lembrar que o *kmeans* é aleatório e, por isso, a qualidade de seus resultados depende muito dos k centros iniciais escolhidos. O ideal é rodarmos o algoritmo várias vezes em busca de um melhor solução.

8.2. PAM. *PAM* também é um método por partição. *Carlantonio* [2] explica o algoritmo: depois de uma seleção aleatória inicial de k medoids, o algoritmo repetidamente tenta fazer a melhor escolha de medoids. Todos os pares possíveis de objetos são analisados, onde um objeto em cada par é considerado um medoid e o outro um não-medoid. A qualidade do agrupamento resultante é calculada para cada uma de tais combinações. Um objeto, O_j , é substituído pelo objeto que causa a maior redução no erro-quadrado. O conjunto dos melhores objetos para cada cluster em uma iteração forma os medoids para a próxima iteração. Para valores muito grandes de n e k , tal computação torna-se muito custosa. Segue um pseudo-código do PAM:

```
def pam(objects, k):
    medoids = sample(objects, k)
    changed = True
    while changed:
        clusters = group(medoids, points)
        newMedoids = reCalcMedoids(points)
        changed = (medoids != newMedoids)
        medoids = newMedoids
    return clusters
```


A diferença entre ele e o *kmeans* é a forma como os novos centros são escolhidos. Em *reCalcMedoids()*, é calculado o novo centros de cada *cluster* como sendo o primeiro que tiver custo total menor que o custo do centro atual. Custo total de um ponto é a soma de todas as distâncias entre ele e os demais pontos do *cluster*.

8.3. DBSCAN. O algoritmo *DBSCAN* é um exemplo de algoritmo por densidade. Esses métodos são adequados para descobrirmos agrupamentos de formas arbitrárias. *Carlantonio* [2] esclarece que a idéia chave dos métodos baseados em densidade é que para cada objeto de um cluster, sua vizinhança, para algum dado raio (*Eps*), tem que conter ao menos um número mínimo de objetos (*MinPts*). *Eps* e *MinPts* são parâmetros de entrada destes métodos. Abaixo, um código resumido do algoritmo (*implementado em Python*).

```
def dbscan(objects, Eps, MinPts):
    id = 1
    for o in objects:
        if o.unclassified():
            o.expandCluster(objects, Eps, MinPts, id)
            id += 1
```

O algoritmo supõe que, inicialmente, todos os objetos de *objects* estão não-classificados. Abaixo, segue a função *expandCluster()* utilizada em *dbscan()*.

```
def expandCluster(self, objects, Eps, MinPts, id):
    N1 = self.epsNeighbourhood(Eps)
    if len(N1) < MinPts:
        self.isNoise()
        return
    else:
        setID(N1,id)
        N1.remove(self)
        for o in N1:
            N2 = o.epsNeighbourhood(Eps)
            if len(N2) ≥ MinPts:
                # Seleccionamos todos os objetos ainda
                # não classificados ou ruidos;
                # adicionamos os não classificados em N1;
                # marcamos todos com o cluster-id atual
        return
```

A função *epsNeighbourhood(Eps)* retorna uma lista com todos os objetos da vizinhança *Eps* de um objeto. A função *isNoise()* marca um

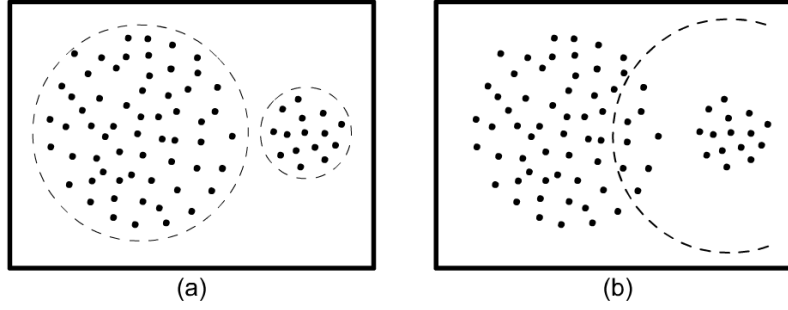


FIGURA 6. Agrupamento por Densidade (a) X Agrupamento por Partição (b)

objeto como sendo um ruído. A função $setID(N, id)$ atribui a todos o objetos da lista N um certo id . A figura 6 mostra uma comparação entre o resultado de um *clustering* por particionamento e um por densidade, para uma mesma base de dados.

9. VALIDAÇÃO VIA DESPACHO

A validação dos algoritmos de distribuição de equipes no presente estágio é feita através de um algoritmo guloso para construção de um caminho mais curto que percorra todos os pontos com emergências demandando atendimento. Esse algoritmo objetiva simular o trabalho de um operador que ao receber uma notificação de uma ocorrência despacha para o atendimento a equipe que se encontrar mais próxima do local da ocorrência.

A figura 7 mostra os dados de deslocamentos obtidos usando o histórico de ocorrências da Escelsa de setembro de 2003 a maio de 2004. A tabela mostra três colunas:

- **TimeWindow:** deslocamento total das equipes de atendimento usando distribuição de equipes proposta por algoritmo de agrupamento *kmeans* e algoritmo de despacho por aproximação de **TSP**.
- **Guloso:** deslocamento total das equipes de atendimento usando distribuição de equipes proposta por algoritmo de agrupamento *kmeans* e algoritmo de despacho, que simula o trabalho de um operador, despachando a equipe mais próxima da ocorrência.
- **Real:** dados reais de deslocamento, obtidos do histórico da empresa.

Os dados da tabela mostram que o pior desempenho dos algoritmos até agora implementados apresenta uma economia de até quarenta por cento no deslocamento total das equipes da empresa.

Mês	Médias		
	TimeWindow	Guloso	Real
09/2003	151965	196589	304598
10/2003	257025	318542	572934
11/2003	207189	224749	427747
12/2003	569689	567475	1063008
01/2004	494560	690378	1137527
02/2004	456469	446799	917201
03/2004	781769	967730	1192993
04/2004	281364	287498	682415
05/2004	182790	181683	318638

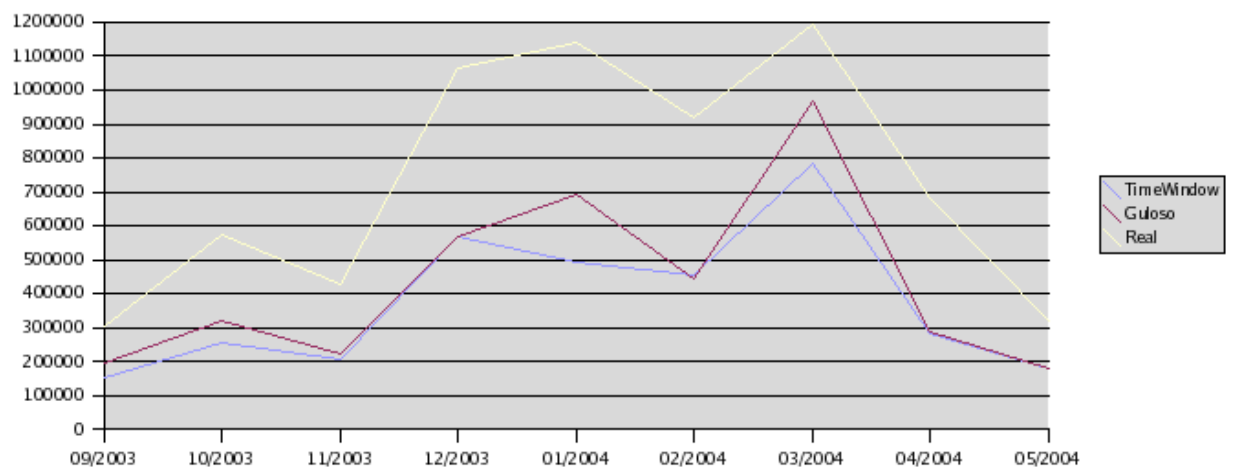


FIGURA 7. Comparação de TMM mês a mês

10. PRÓXIMOS PASSOS

Os próximos passos do projeto incluem:

- Implementação de um protótipo que permita ligar as duas funcionalidades básicas do sistema.
Essa implementação já está concluída, devendo ser objeto de relato do próximo relatório parcial.
- Implementação de dois novos algoritmos para simulação de despacho: um algoritmo baseado em aproximação do problema de **TSP**[8] e um algoritmo baseado no problema de casamentos estável em grafos [7].
- Incorporação de algoritmo de minimização de **DEC**, às duas funcionalidades básicas do sistema.

- Realização de testes exaustivos com todos os dados históricos de 2003 e 2004, que já está parcialmente completado e deverá ser objeto do próximo relatório técnico.

REFERÊNCIAS

1. Charles Jay Alpert, *Multi-way graph and hypergraph partitioning*, Ph.D. thesis, University of California, 1996.
2. Lando Mendonça di Carlantonio, *Novas metodologias para clusterização de dados*, Master's thesis, COPPE/UFRJ, 2001.
3. Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf, *Computational geometry*, Kluwer Academic Publishers, 1998.
4. Herbert EdelsBrunner, *Algorithms in combinatorial geometry*, Springer-Verlag, 1987.
5. Daniel Fasulo, *Ana analysis of recent work on clustering algorithms*, Tech. Report 01-03-02, Department of Computer Science, University of Washington, 1999.
6. Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim, *CURE: an efficient clustering algorithm for large databases*, In Proceedings of ACM SIGMOD International Conference on Management of Data, ACM, 1998, pp. 73–84.
7. Dan Gusfield and Robert W. Irving, *The stable marriage problem: structure and algorithms*, The MIT Press, 1989.
8. Gregory Gutin and Abraham P. Punen, *Traveling salesman problem and its variations*, Kluwer Academic Publishers, 2002.
9. John Hartigan, *Clustering algorithms*, JOHN WILEY & SONS, 1975.
10. A. K. Jain, M. N. Murty, and P. J. Flynn, *Data clustering: a review*, ACM Computing Surveys **31** (1999), no. 3, 264–323.