

1. INTRODUÇÃO

Trata este artigo do estudo realizado em torno do problema de localização de equipes de campos da ?ESCELSA?(ESCELSA).

O objetivo estabelecido para o projeto consiste em propor sistema computacional, doravante denominado Sistema de Escalonamento de Equipes de Campo da ESCELSA, ou simplesmente *SEECE*, para definir a localização de equipes de atendimento emergencial. A metodologia de pesquisa adotada teve como objetivos:

- Estabelecer um conjunto de algoritmos para o problema em questão.
- Definir um modelo de validação dos algoritmos propostos.
- Projetar e implementar um sistema computacional para cálculo de localização de bases de equipes de atendimento.
- Validar sistema computacional a partir de dados históricos da empresa.

Este artigo apresenta os algoritmos usados na localização de bases de equipes de atendimento emergencial.

2. REQUISITOS DO SISTEMA COMPUTACIONAL

O objetivo fundamental do projeto consistiu em estabelecer ambiente computacional que permitisse realizar simulações para fornecer subsídios para a alocação e despacho de equipes para atendimentos de emergência. Os dados fundamentais dessas simulações são:

- a localização geográfica das chaves onde podem ocorrer as emergências;
- a distribuição geográfica das bases onde as equipes de atendimento se encontram estacionadas;
- o histórico de ocorrências, incluindo:
 - data e hora de registro da ocorrência;
 - tempo e distância de deslocamento da equipe;
 - tempo de espera pela chegada da equipe ao local do serviço;
 - tempo de reparo;
 - definição de conjuntos de consumidores afetados.
- a definição dos conjuntos de consumidores do Estado;
- os dados históricos de DEC e TMM.

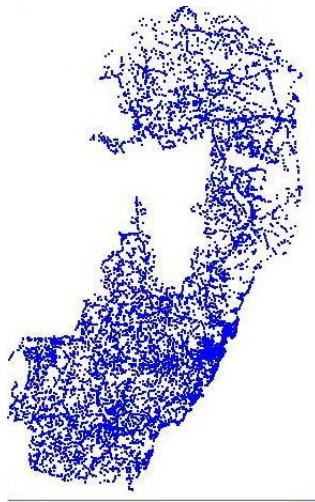


FIGURA 1. Mapa de chaves

A partir desses dados, o sistema computacional proposto deveria permitir ao usuário:

- realizar simulações de despacho de equipes, gerando relatórios de DEC e TMM e usando dados de:
 - distribuições de equipes de atendimento, obtidas automaticamente ou não;
 - dados históricos de ocorrências;
- propor distribuição de equipes de campo, incluindo localização geográfica e definição do número de equipes.

A figura 1 mostra mapa com todas as posições de todas as chaves da empresa no Estado do Espírito Santo. São mais de 2^{16} chaves. Cada chave representa potencialmente um ponto onde pode ocorrer a qualquer momento uma falha, demandando um atendimento emergencial. Reduzir os tempos de atendimento demanda reduzir essencialmente o tempo de espera entre a detecção da falha, possivelmente via registro de ocorrência procedente de reclamação de usuário, e a chegada da equipe de atendimento à chave correspondente. Isso é em essência um problema de despacho e roteamento de equipes de manutenção, ver [6], que se tratado por algoritmos exaustivos poderia demandar, em casos extremos, milênios de processamento de dados, como veremos a seguir.

3. O PROBLEMA DE DESPACHO DE EQUIPES

Otimizar despacho de equipes de atendimento de emergência é no final o problema fundamental do objeto do nosso estudo. Uma redução nos tempos de atendimento de emergência seria o indicador final de sucesso de um sistema de localização de bases das referidas equipes. O problema de despacho das mesmas, por sua vez, pode ser reduzido ao *problema do caixeiro viajante*, de agora em diante referenciado como TSP, ver [4], que fornece um modelo matemático clássico que permite antever a complexidade computacional do problema de despacho de equipes de campo. Neste modelo, assume-se a existência de:

- uma equipe de atendimento, na terminologia clássica, o caixeiro viajante (*salesman*), estacionada em alguma chave;
- um conjunto de chaves com requisições de atendimento associadas;
- uma medida de distância para as possíveis conexões entre as chaves.

Um programa de computador deve neste caso calcular o caminho mais curto que permita à equipe de atendimento sair da chave inicial, visitar todas as chaves com requisições associadas e voltar à chave inicial. Esse caminho mais curto serviria, então, de roteiro de atendimento.

Podem ser consideradas como medidas de distâncias entre pontos:

- distância euclideana, calculada a partir das coordenadas geográficas dos mesmos;
- distância percorrida pelas equipes da empresa e registrada nos dados históricos de ocorrências.

Como salientado antes, é um caso clássico de TSP. Embora, a simplificação do problema para a modelagem via TSP fuja da realidade da empresa em vários aspectos, como será visto a seguir, ainda assim ele serve para fornecer base para definir expectativas de desempenho de um provável sistema computacional que reduza os tempos de viagem das equipes e, conseqüentemente, de espera do usuário.

Dado que a empresa possui hoje $\Omega(2^{16})$ chaves e que as soluções conhecidas de TSP demandam $\Omega(n!)$ transições para obtenção de melhor solução para n pontos, pode-se esperar

que no extremo um dia movimentado demande para cálculo do melhor roteiro para uma equipe de atendimento

$$\Omega(2^{20}) \text{ anos.}$$

Esse cálculo assume a existência de um computador capaz de executar com um clock de 64Ghz , podendo gerar 2^{26} soluções possíveis em um ano. Além disso, esse computador deverá apresentar um memória com mais de 10^{3000} bytes.

A inspeção visual da figura 2, no entanto, mostra as distribuição de chaves (pontos em cinza) e de ocorrência (pontos em vermelho) no Estado, no mês de janeiro de 2004. Essa inspeção visual já é suficiente para mostrar que um grande número de chaves não apresentou ocorrência alguma. No entanto, o número de ocorrências mensal ainda está na casa de 10^4 , trazendo o número de ocorrências diárias para $\mathcal{O}(10^3)$. Mesmo que se considere apenas uma região como Vitória, o número de ocorrências ainda pode chegar a algumas centenas em um curto período do dia, o que, baseado nos cálculos apresentados acima, ainda representa um peso exagerado para um algoritmo de busca exaustiva de melhor solução para o TSP.

Por um outro lado, a possibilidade de se particionar o Estado em regiões e de se agrupar as ocorrências de uma região permite reduzir significativamente o número máximo de ocorrências que uma equipe deve atender em um turno possibilitando o tratamento computacional em tempos aceitáveis.

4. ALGORITMOS PARA DISTRIBUIÇÃO DE EQUIPES

O problema de distribuição de equipes pelo estado foi 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 [5, 3] como *clustering*: a classificação não-supervisionada de padrões (observações, itens de dados ou vetores de características) em grupos (*clusters*)” [?].

4.1. 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

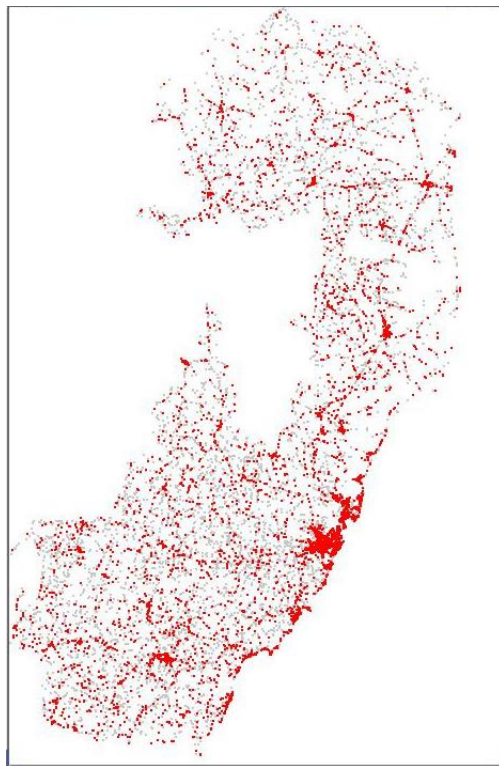


FIGURA 2. Mapa de ocorrências

hospital mais próximo. Na literatura de geometria combinatorial, ver por exemplo [1, 2], 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 [?]. Outro exemplo é o de agrupar os pontos mais similares entre si da primeira parte da figura 3. 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.* [?], 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

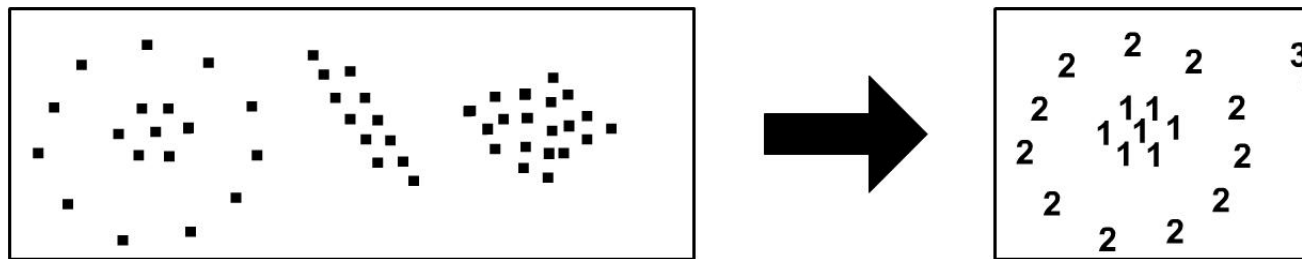


FIGURA 3. Um exemplo de *clustering*

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 Euclidiana entre os pontos, frequentemente soluções da geometria computacional, como diagramas de Voronoi e triangulação revelam-se mais precisos e eficientes.

Jain *et al.* [?] 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*).

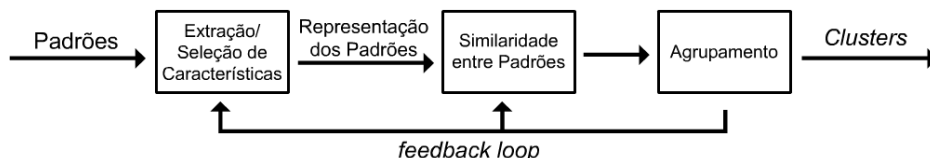


FIGURA 4. Etapas do agrupamento

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 4 mostra uma seqüência 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.

5.1. Definição Formal. Uma definição formal do problema de *clustering* é encontrada em [?, ?] 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.

5.2. Algoritmos Tradicionais de agrupamento. Os algoritmos conhecidos de *clustering* atendem a diferentes tipos de requisitos, tais como [?]:

- 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 [?]:

- 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.

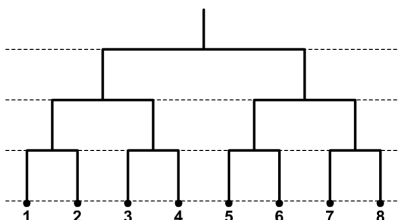


FIGURA 5. Exemplo de um *dendograma*

5.3. 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.

5.4. 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 5). O *dendograma* irá, então, representar os *clusters* gerados pelo algoritmo. No fim da árvore temos cada objeto em apenas um grupo (figura 6).

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

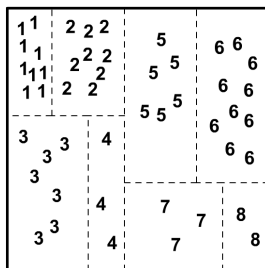


FIGURA 6. *Clusters* formados por um método hierárquico

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*.

5.5. Algoritmo *k-means*. O algoritmo *k-means* [5] 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 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

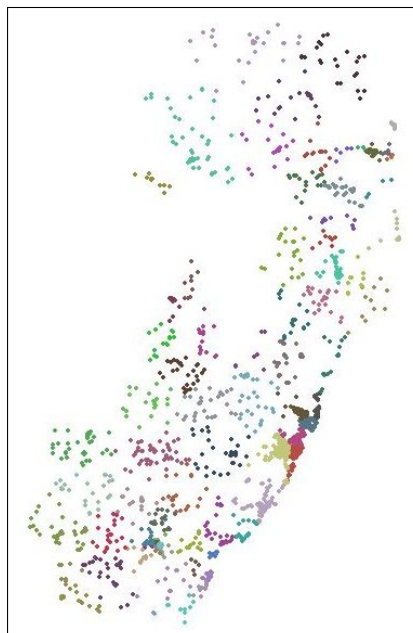


FIGURA 7. Um resultado do *kmeans*

total de objetos da base de dados, k é o número de grupos formados e t é o número de iterações. Segundo *Carlantonio* [?], 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 7, 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.

5.6. Algoritmo PAM. *PAM* também é um método por partição. *Carlantonio* [?] 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*.

5.7. Algoritmo DBSCAN. O algoritmo *DBSCAN* é um exemplo de algoritmo por densidade. Esses métodos são adequados para descobrirmos agrupamentos de formas arbitrárias. *Carlantonio* [?] 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 ruídos;
                # 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 objeto como sendo um ruído. A função *setID(N,id)* atribui a todos os objetos da lista *N* um certo *id*. A figura 8 mostra uma comparação entre o resultado de um *clustering* por particionamento e um por densidade, para uma mesma base de dados.

6. VALIDAÇÃO VIA DESPACHO

A validação dos algoritmos de distribuição de equipes é feita através de algoritmos heurísticos que se enquadram duas categorias:

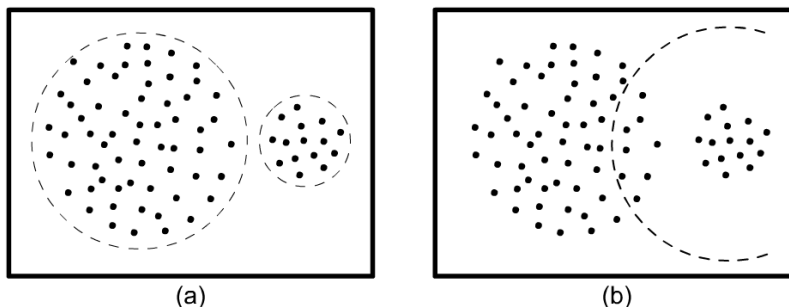


FIGURA 8. Agrupamento por Densidade (a) X Agrupamento por Partição (b)

- algoritmos do paradigma guloso que buscam simular a atuação de um operador;
- algoritmos de aproximação que buscam obter um escalonamento de equipes de despacho com desempenho sempre aceitável.

A figura 9 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 implementados apresenta uma economia de até quarenta por cento no deslocamento total das equipes da empresa.

7. CONCLUSÃO E TRABALHOS FUTUROS

REFERÊNCIAS

1. Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf, *Computational geometry*, Kluwer Academic Publishers, 1998.
2. Herbert EdelsBrunner, *Algorithms in combinatorial geometry*, Springer-Verlag, 1987.

| Mês | Médias | | Real |
|---------|------------|--------|---------|
| | TimeWindow | Guloso | |
| 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 |

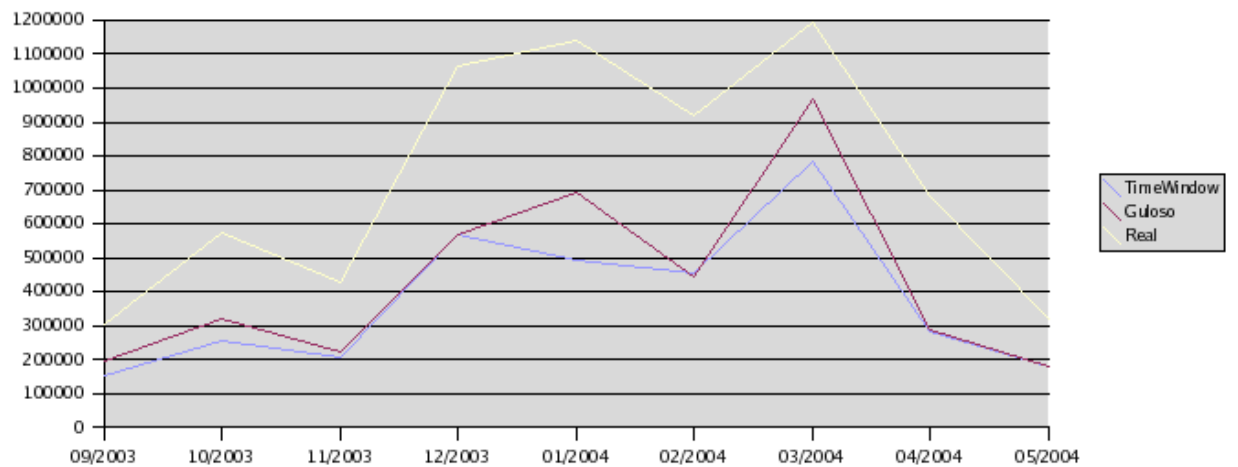


FIGURA 9. Comparação de TMM mês a mês

3. Daniel Fasulo, *Ana analysis of recent work on clustering algorithms*, Tech. Report 01-03-02, Department of Computer Science, University of Washington, 1999.
4. Gregory Gutin and Abraham P. Punen, *Traveling salesman problem and its variations*, Kluwer Academic Publishers, 2002.
5. John Hartigan, *Clustering algorithms*, JOHN WILEY & SONS, 1975.
6. Jennifer L. Rich, *A computational study of vehicle routing applications*, Ph.D. thesis, Rice University, 1999.