Heurísticas e Metaheurísticas para o *MSSC* Euclidiano

Projeto Final da Disciplina Heurísticas e Metaheurísticas, DCC-UFMG

Matheus Henrique do Nascimento Nunes
Universidade Federal de Minas Gerais
Belo Horizonte, Minas Gerais
mhnnunes@dcc.ufmg.br

Tarsila Bessa Nogueira Assunção
Universidade Federal de Minas Gerais
Belo Horizonte, Minas Gerais
tarsila.bessa@dcc.ufmg.br

ABSTRACT

O MSSC (*Minimum Sum-of-Squares Clustering* - Agrupamento Mínimo por Soma de Quadrados) Euclidiano é um problema de otimização combinatória famoso na literatura de aprendizado de máquina e teoria de computação. Conhecido popularmente como "*k-means*", devido ao nome da primeira heurística proposta que o resolve, recebe como entrada um conjunto de n pontos distribuídos em $\mathbb{R}^m$ e um número k de grupos. O objetivo é agrupar os m pontos em k grupos, de forma a minimizar as distâncias (utiliza-se a distância Euclidiana) entre os pontos e os centros de seus respectivos grupos. Existem implementações eficientes de algoritmos para este problema, e em seu caso médio, os algoritmos são relativamente rápidos. Contudo, em seu pior caso é NP-Hard até para pontos no plano. O objetivo deste trabalho é apresentar um panorama da literatura e propor novas heurísticas e metaheurísticas para o problema, avaliando-as e comparando-as com técnicas já propostas.

ACM Reference Format:
Matheus Henrique do Nascimento Nunes and Tarsila Bessa Nogueira Assunção. 2018. Heurísticas e Metaheurísticas para o *MSSC* Euclidiano: Projeto Final da Disciplina Heurísticas e Metaheurísticas, DCC-UFMG. In *Proceedings of Projeto Final da Disciplina Heurísticas e Metaheurísticas (HeM).* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUÇÃO

O MSSC (*Minimum Sum-of-Squares Clustering* - Agrupamento Mínimo por Soma de Quadrados) Euclidiano é um problema de otimização combinatória famoso na literatura de aprendizado de máquina, teoria da computação e geometria computacional [4]. Enunciado inicialmente em 1967 por *MacQueen, J.B* [14], se popularizou com o nome de "*k-means*" (do inglês *k*-médias), pois este foi o nome dado pelo autor ao método proposto no artigo. O problema recebe como entrada um conjunto de n pontos distribuídos em $\mathbb{R}^m$ e um número k de centros. Seu objetivo é agrupar os n pontos em k grupos, de forma a minimizar as distâncias (utiliza-se a distância Euclidiana) entre os pontos e seus respectivos centros. Existem implementações eficientes de algoritmos para este problema [10] [6] e em seu caso médio os algoritmos são relativamente rápidos. Contudo, em seu pior caso, o problema é NP-Hard até para pontos no plano [15].

A primeira aplicação do *k-means* foi registrada em 1982 por *Stuart Lloyd* [13], com o objetivo de agrupar amostras de transmissão de um sinal elétrico para reproduzir este sinal em um receptor da melhor maneira possível, ou seja, com o mínimo de potência do ruído. O ruído foi definido como a diferença entre o sinal recebido e o sinal original, e a potência do ruído é a média do ruído ao quadrado. O *k-means* se aplica ao se constatar que as amostras de sinal podem ser vistas como pontos em um espaço euclidiano, e seu agrupamento neste espaço permite a redução no número de informação transmitida, pois após este processo é necessário transmitir apenas os grupos em que cada amostra se encaixa para o receptor, e a reprodução do sinal original será possível a partir desta informação.

A análise de *clusters* é uma técnica que tem como objetivo a separação de dados em grupos de forma a maximizar a similaridade de elementos em um mesmo grupo e minimizar a similaridade entre dados de grupos distintos. O MSSC é um problema muito utilizado para modelar problemas de clusterização, sendo que o algoritmo *k-means* de Lloyd é uma das heurísticas mais utilizadas para a análise de *clusters*, tal que cada *cluster* é representado pela média de todos os objetos contidos no grupo [11]. Neste trabalho, é apresentado um estudo da literatura de heurísticas e metaheurísticas para o MSSC, além da proposta e análise de novas abordagens para resolver o problema.

Uma definição formal do problema se encontra na seção 2. Nesta seção também provamos que o problema é *NP-Hard* a partir da redução do problema de *Densest Cut* (Corte Mais Denso em um Grafo). Em seguida apresentamos um panorama da literatura de heurísticas e metaheurísticas para o problema na seção 3. São apresentadas na seção 4 novas heurísticas e metaheurísticas para o problema, e os resultados dos experimentos computacionais realizados para demonstrar o desempenho das mesmas são apresentados na seção 5. Conclui-se que a configuração inicial é um dos aspectos mais importantes do algoritmo, e que uma maior diversificação de soluções causa uma melhoria na qualidade geral das soluções encontradas.

2 FORMALIZAÇÃO

O *MSSC* pode ser formalizado da seguinte maneira: Dado um conjunto $P = \{p_1, ..., p_n\}$ de n pontos em $\mathbb{R}^m$, deseja-se encontrar um conjunto $B = \{b_1, ..., b_k\}$ de k pontos, tal que:

$$\sum_{i=1}^{n} [d(p_i, b(p_i)]^2$$

seja mínimo. Este valor mínimo é denominado $Opt(S, k)$. O valor $b(p_i)$ representa o ponto em B mais próximo ao ponto p_i, e $d(p_i, b(p_i))$ representa a distância Euclidiana entre p_i e $b(p_i)$.

Apesar da forte relação entre a heurística *k-means* [14] e o *MSSC*, existem exemplos de situações em que a solução encontrada pela heurística é bem distante da ótima. Algoritmos aproximativos já foram propostos para o problema, e à extensão do conhecimento dos autores, nenhum algoritmo obteve fator de aproximação melhor do que $(9 + \epsilon)$ de maneira eficiente. Os algoritmos que aproximam

HeM, Novembro 2018, DCC-UFMG, Belo Horizonte, MG
2018. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

o *k-means* possuem dependência linear no número de pontos e no número de dimensões, mas exponencial no número de grupos k [4].

Portanto, o objetivo desta seção é apresentar uma prova de que o *MSSC* é *NP-Hard* a partir da redução do problema de Corte Mais Denso em um Grafo (*Densest Cut*). Esta prova foi enunciada inicialmente por *Aloise, D. et al.* [2], e será apresentada a seguir.

O problema de Corte Mais Denso pode ser formalizado da seguinte maneira: Dado um grafo $G = (V, E)$ não-direcionado, escolha dois subconjuntos $P, Q \subseteq V$ como uma bipartição dos vértices de G, de forma que a razão $\frac{|E(P,Q)\|}{|P||Q|}$ seja máxima, onde E(P,Q) representa o numero de arestas do corte. Este problema é análogo ao *Sparset Cut* (Corte mais Esparso em um Grafo) no grafo complementar. A prova de que o *Sparsest Cut* é *NP-Hard* foi apresentada por *Matula and Shahrokhi* [16].

Teorema 1. *O MSSC é NP-Hard em geral, para* $k = 2$

Prova: Dado um grafo $G = (V, E)$ sem arestas paralelas. Define-se uma matriz M de dimensão $|V|\cdot|E|$ da seguinte maneira: uma entrada (v, e) de M é:

- igual a 0 se $e \in E$ não é incidente ao vértice $v \in V$
- igual a +1 ou −1 se $e \in E$ é incidente a $v \in V$. Não importa qual ponta da aresta é marcada como +1, desde que a outra ponta seja −1

Portanto, cada coluna de M possui exatamente uma entrada igual a −1 e uma entrada igual a +1.

Agora, supondo que as linhas de M são pontos em $\mathbb{R}^{|E|}$, utilizando o *MSSC*, computamos o valor para a bipartição ($k = 2$) entre os clusters P e Q, sendo $|P| = p$, $|Q| = q$ e $p + q = n$. O centróide do cluster P possui em sua coordenada e o valor $e = \frac{+1}{p}$ ou $e = \frac{-1}{p}$ se $e \in E(P, Q)$, ou $e = 0$ caso contrário. O mesmo acontece para o centróide de Q. Então calculamos:

$$\sum_{e \in E} c_P + c_Q$$

onde c_P e c_Q representam o custo de P e de Q devido à coordenada e, respectivamente, da seguinte maneira:

$$\sum_{e \in E} c_P + c_Q$$

$$= \sum_{e \in E(P,Q)} (p-1)\frac{1}{p^2} + \left(1 - \frac{1}{p}\right)^2 + (q-1)\frac{1}{q^2} + \left(1 - \frac{1}{q}\right)^2 + \sum_{e \notin E(P,Q)} 2$$

$$= \left(2 - \frac{1}{p} - \frac{1}{q}\right)|E(P,Q)| + 2|E(P,P)| + 2|E(Q,Q)|$$

$$= 2|E| - \frac{n}{p \cdot q}|E(P,Q)|$$

O custo total da bipartição ótimo, que minimiza o valor acima, é se $p + q = n$, e este custo é obtido utilizando o *k-means* com $k = 2$. Logo, este valor maximiza $\frac{|E(P,Q)|}{|P|\cdot|Q|}$, e encontra o corte mais denso no grafo G.

Portanto, ao se aplicar uma redução polinomial de uma instância do problema de *Densest Cut*, resolvendo-se esta instância reduzido utilizando o *MSSC*, obtemos a resposta ótima para o problema original. Logo, provamos que o *MSSC* é *NP-Hard* para $k = 2$.

3 REFERENCIAL TEÓRICO

Majahan, M. et al. provaram que o problema MSSC é NP-Hard para pontos no plano, utilizando uma redução do 3SAT planar [15]. Muitas heurísticas foram propostas para resolver esse problema. Elas podem ser divididas em dois grupos: heurísticas locais e globais, sendo que essas últimas costumam ser utilizadas para inicializar as heurísticas locais [19]. Nesta seção, algumas das heurísticas e metaheurísticas mais conhecidas para resolver o problema são apresentadas.

3.1 Heurísticas

Em 1967, *MacQueen* apresentou uma heurística para o problema MSSC [14]. O algoritmo começa com a escolha de k pontos da entrada como pontos centrais de cada grupo. Os outros $n - k$ pontos são adicionados aos conjuntos de pontos mais próximo e, após cada adição, a média do grupo é calculado novamente. Essa primeira parte do algoritmo pode ser considerada uma heurística construtiva e, em seguida, é realizada uma busca local na qual itera-se por todos os pontos, verificando se eles podem ser trocados de grupo. Isso é feito de maneira iterativa até convergir, ou seja, até o momento em que os pontos param de trocar suas posições entre os diferentes *clusters*. Esta heurística é simples de ser implementada e converge rápido.

Hartigan também apresentou uma heurística para o problema em 1975 [9][8]. O algoritmo criado por ele se difere do anterior porque os pontos não são necessariamente adicionados ao *cluster* cuja média é mais próxima a eles. Neste algoritmo, os pontos são analisados um de cada vez e para cada um, move-se o ponto para um novo *cluster* se a função objetivo, a soma de quadrados, melhora. Todos os pontos são percorridos até que não ocorram mais mudanças entre grupos.

Em 1982, foi apresentada a heurística de Lloyd, que é até hoje, uma das heurísticas mais populares para o MSSC [13]. O algortimo é mais semelhante ao de MacQueen, começando com a separação dos n pontos recebidos como entrada em k conjuntos. Em seguida, calcula-se o ponto médio, ou centroide, de cada conjunto. Então, o conjunto de pontos é novamente dividido sendo que cada ponto é associado ao centroide mais próximo a ele. Assim como na heurística de MacQueen, a primeira parte do algoritmo de Lloyd é considerada uma heurística construtiva. Porém, em seguida também é realizada uma busca local, repetindo a troca de pontos entre *clusters* até o algoritmo convergir. O algoritmo de *Lloyd* é o mais utilizado para resolver o problema porque converge rapidamente, na prática [22]. Porém, em 2010, foi mostrado que o algoritmo de Hartigan possui melhor desempenho que o de Lloyd [21].

A qualidade da solução encontrada pelos algoritmos anteriores depende da divisão dos pontos recebidos como entrada em conjuntos iniciais. Essas heurísticas são denominadas heurísticas locais. Um outro conjunto de heurísticas, as heurísticas globais, têm como principal objetivo encontrar a melhor divisão inicial dos pontos de entrada [18]. Elas costumam ser utilizadas em conjunto com outros algoritmos locais.

O algoritmo "*k-means++*" [3] é uma das heurísticas globais mais conhecidas para o MSSC. Proposto em 2007, o algoritmo apresenta uma maneira específica de se escolher os *clusters* iniciais a partir dos dados de entrada e então executa o k-means original com essa

configuração inicial. O k-means++ melhora o desempenho e acurácia da heurística original de Lloyd. Este algoritmo será explicado em mais detalhes na seção 4, uma vez que foi implementado neste trabalho.

Em 2013, *Likas et al* apresentaram um algoritmo conhecido como "*k-means* global" [12]. Essa heurística utiliza o *k-means* para fazer uma busca local, mas não depende da configuração inicial como foi proposto por *Lloyd*. Para resolver o problema, o algoritmo começa com apenas um *cluster* ($k = 1$), sendo que seu centro corresponde ao centro de todos os pontos dados como entrada. A heurística segue de maneira incremental, ou seja, resolve-se então o problema para dois *clusters* ($k = 1$): são feitas N execuções do *k-means* sendo que o centro do primeiro *cluster* é colocado na posição ótima para o problema com $k = 1$. O segundo centro é colocado na posição do dado de entrada x_n em cada uma das N execuções, tal que $n = 1, ..., N$. A melhor solução encontrada entre essas N execuções é escolhida como solução para o problema $k = 2$. Esse processo é repetido de acordo com o valor escolhido para k.

3.2 Metaheurísticas

Metaheurísticas também podem ser utilizadas para revolver o problema do MSSC. Porém, esses algoritmos não são tão eficazes quando aplicados a conjuntos de dados muito grandes [20]. Sendo assim, metaheurísticas para este problema não são tão populares quanto heurísticas construtivas ou buscas locais. Ainda assim, é possível encontrar, na literatura, algumas variações de metaheurísticas clássicas para resolver o MSSC.

Em 1995, *Al-Sultan* apresentou um algoritmo de busca tabu para o MSSC [1]. O algoritmo executa por várias iterações e cria possíveis soluções para o problema. Essas soluções são ordenadas da melhor para a pior e é verificado se a melhor nova solução encontrada é melhor que a solução atual. Em caso positivo, atualiza-se a melhor solução e a coloca na lista tabu. Para criar possíveis soluções para o problema, o autor atualiza de maneira aleatória, um dos *clusters* da solução. Essa escolha segue uma distribuição de probabilidade que deve ser definida pelo programador. Além dessa probabilidade, o algoritmo depende da definição de outros três parâmetros: tamanho da lista tabu, número de novas possíveis soluções e número máximo de iterações.

J. R. Cano et al apresentaram, em 2012, um algoritmo de GRASP para resolver o problema [5]. O algoritmo utiliza a inicialização de Kaufman como base para a fase de construção da solução gulosa. Nesta fase, primeiramente, escolhe-se o ponto mais central do conjunto de dados. Então, calcula-se a distância entre todos os pontos não escolhidos e a menor distância entre os pontos não escolhidos e os já escolhidos. Esses dois valores são utilizados para calcular a possível melhora da solução ao selecionar um determinado ponto. Os l melhores pontos são selecionados e colocados em uma lista. Em seguida, é escolhe-se, de maneira aleatória, um ponto desta lista. Terminada a fase de construção do algoritmo, é executada uma busca local. Neste caso, o algoritmo de Lloyd é utilizado.

Em 2003, *Peter Mertz* apresentou um algoritmo de *iterated local search* para resolver o MSSC [17]. O algoritmo é bem simples: é executada uma busca local em uma solução gerada de maneira aleatória e então, o ótimo local sofre uma mutação e é feita uma nova busca local para encontrar outro ótimo local. Se o novo ótimo é melhor que o anterior, então a melhor solução é atualizada. Assim como o algoritmo de GRASP descrito anteriormente, essa metaheurística utiliza o algoritmo de Lloyd para fazer a busca local. Para a mutação, o algoritmo altera, de maneira aleatória, um dos pontos considerados como centro de um *cluster* da solução.

```
1 LloydInitialHeur (n_i, k)
    entradas : Um conjunto de n pontos no plano, um valor
               inteiro k de centros
    saída    : Vetor com a divisão dos n pontos em k clusters,
               um valor inteiro ssq com a soma de quadrados
2   calcular distância entre todos os pontos ;
3   centros ← conjunto de k pontos aleatórios de n ;
4   for i ← 0; i < n; i ← i + 1 do
5       calcular distância de i para todos os centros ;
6       colocar i no cluster cujo centro é mais próximo dele ;
7   end
8   calcular soma de quadrados ;
```

Algorithm 1: Pseudocódigo da heurística construtiva de Lloyd.

Gyamfi et al apresentaram, em 2017, um novo algoritmo de lista tabu para o problema [7]. Nesta implementação, os *clusters* da solução atual são colocados em uma lista tabu. Então, percorre-se os pontos que não estão na lista e calcula-se quanto cada ponto poderia melhorar a solução. Então, os k pontos que melhoram mais a solução são escolhidos como novos *clusters*. A principal vantagem deste algoritmo em relação ao apresentado por Al-Sultan é que este não possui tantos parâmetros que precisam ser definidos.

4 METODOLOGIA

O objetivo desta seção é propôr duas novas heurísticas e duas novas metaheurísticas para resolver o MSSC. Serão apresentadas também duas heurísticas e duas metaheurísticas da literatura, que serão utilizadas como *baseline* para comparação com os algoritmos propostos. As heurísticas e metaheurísticas apresentadas foram escolhidas por sua simplicidade de implementação e popularidade. As heurísticas são apresentadas na seção 4.1 e as metaheurísticas na seção 4.2.

4.1 Heurísticas construtivas

A heurística construtiva de Lloyd é apresentada pelo algoritmo 1. Foi considerada como heurística construtiva a primeira parte do algoritmo de Lloyd encontrado na literatura. Essa parte é executada antes da parte de busca local do algoritmo proposto. Como entrada, recebe-se um conjunto de n pontos e o valor k de centros. A saída é um vetor de n posições, no qual a posição $[i]$ guarda o índice do centro do grupo ao qual o ponto de índice i faz parte. A heurística se inicia com o cálculo de distância entre todos os pontos, na linha dois. Na linha três acontece a escolha, de maneira aleatória, de k pontos como centro. Na linha quatro, é iniciado um *loop* usado para percorrer todos os pontos recebidos como entrada. Dentro deste *loop*, na linha cinco, é feito o cálculo da distância de todos os pontos para os centros escolhidos anteriormente e, na linha seis, coloca-se o ponto no *cluster* cujo centro está mais próximo a ele. O algoritmo termina com o cálculo da soma de quadrados na linha oito.

O algoritmo 2 apresenta a heurística *k-means++*. Este algoritmo possui a mesma entrada e a mesma saída que o algoritmo de Lloyd

```
1  KMeansPP (n_i, k)
       entradas : Um conjunto de n pontos no plano, um valor
                  inteiro k de centros
       saída    : Vetor com a divisão dos n pontos em k clusters,
                  um valor inteiro ssq com a soma de quadrados
2      calcular distância entre todos os pontos ;
3      centro[0] ← um ponto aleatório de n ;
4      for i ← 1; i < k; i ← i + 1 do
5          calcular D² entre centro[0] e todos os pontos ;
6          d2 ← D²/ soma das D² de todos os pontos ;
7          centro[i] ← ponto aleatório de n tal que a
             probabilidade da escolha de um ponto é igual a d2 ;
8      end
9      for i ← 0; i < n; i ← i + 1 do
10         calcular distância de i para todos os centros ;
11         colocar i no cluster cujo centro é mais próximo dele ;
12     end
13     calcular soma de quadrados ;
```

Algorithm 2: Pseudocódigo da heurística construtiva k-means++.

```
1  KFurthestHeur (n_i, k)
       entradas : Um conjunto de n pontos no plano, um valor
                  inteiro k de centros
       saída    : Vetor com a divisão dos n pontos em k clusters,
                  um valor inteiro ssq com a soma de quadrados
2      calcular distância entre todos os pontos ;
3      centro[0] ← um ponto aleatório de n ;
4      calcular distância entre o primeiro centro escolhido e
         todos os pontos de n ;
5      for i ← 1; i < k; i ← i + 1 do
6          centro[i] ← ponto mais distante do centro[0] que
             ainda não foi adicionado como centro;
7      end
8      for i ← 0; i < n; i ← i + 1 do
9          calcular distância de i para todos os centros ;
10         colocar i no cluster cujo centro é mais próximo dele ;
11     end
12     calcular soma de quadrados ;
```

Algorithm 3: Pseudocódigo do algoritmo K-Furthest.

descrito anteriormente. A heurística se difere da anterior na escolha dos k pontos centrais. O algoritmo se inicia, na linha dois, como o cálculo da distância entre todos os pontos. Na linha três, acontece a escolha de um ponto aleatório da entrada como primeiro centro. Então, na linha quatro, é iniciado um *loop* que é usado para encontrar os outros $k-1$ centros. Para cada ponto da entrada, na linha cinco, é calculada a D^2, que é a menor distância do ponto ao centro mais próximo que já foi escolhido elevada ao quadrado. Na linha seis, é feita uma normalização deste valor, dividindo-o pela soma das D^2 de todos os pontos. O resultado dessa normalização é utilizado como probabilidade para escolha, de maneira aleatória, de um novo centro, na linha sete. Após a escolha dos k centros, o algoritmo segue da mesma maneira que o algoritmo de Lloyd, com um *loop* que percorre todos os pontos e dentro dele, o cálculo da distância do ponto atual para todos os centros, na linha dez, e a escolha do *cluster* cujo centro está mais próxima do ponto, na linha onze. O algoritmo também termina com o cálculo da soma de quadrados, que é realizado na linha treze.

As heurísticas propostas neste trabalho são inspiradas no algoritmo de Lloyd, mas diferem na maneira como são escolhidos os k centros iniciais. Elas foram denominadas *K-Furthest* e *K-Popular* e serão descritas a seguir.

O algoritmo 3 apresenta a heurística *K-Furthest*. Sua entrada e saída são iguais às das heurísticas anteriores. O algoritmo começa com o cálculo da distância entre todos os pontos, na linha dois. Na linha três, ocorre, de maneira aleatória, a escolha do primeiro centro. Na linha quatro, calcula-se a distância deste centro inicial para todos os pontos da entrada. As linhas cinco a sete mostram como são escolhidos os outros centros: os $k-1$ pontos com maior distância do primeiro centro. O algoritmo, então, continua da mesma maneira que o algoritmo de Lloyd, como é visto das linhas oito a doze.

A heurística *K-Popular* é apresentada pelo algoritmo 4. Sua entrada e saída também são iguais às heurísticas de Lloyd, *k-means++* e *K-Furthest*. O algoritmo também se inicia com o cálculo de distância entre todos os pontos da entrada, na linha dois. Porém, a escolha dos centros difere dos algoritmos anteriores. Primeiramente, na linha três, é calculada a média das distâncias entre todos os pontos. Na linha quatro, inicia-se um *loop* que itera por todos os pontos do problema. Dentro deste *loop*, na linha cinco, para cada ponto i do problema, conta-se quantos dos seus vizinhos possui distância maior que a média calculada na linha três. Na linha sete, é feita a ordenação do vetor que guarda os valores calculados no *loop* anterior. Então, o *loop* que se inicia na linha oito é usado para escolher os centros: os k centros que possuem mais vizinhos cuja distância é maior que a média das distâncias entre todos os pontos. A partir deste momento, o algoritmo também funciona igual ao algoritmo de Lloyd, como é mostrado entre as linhas treze e dezesseis.

Todas as heurísticas apresentadas nesta seção possuem complexidade $O(n^2)$, na qual n é o número de pontos recebidos na entrada. Em todos os casos, é feito o cálculo da distância entre todos os pontos, que é $O(n^2)$ e o cálculo da soma de quadrados, que é $O(n)$. Portanto, temos $O(n^2) + O(n) = O(n^2)$.

4.2 Metaheurísticas

A primeira metaheurística da literatura implementada é uma busca tabu [7], apresentada no algoritmo 6. Assim como as heurísticas implementadas anteriormente, este algoritmo recebe como entrada o conjunto n de pontos no espaço e o valor inteiro k que representa o número de *clusters* do problema. A saída também é um vetor com a divisão final dos pontos entre os *clusters* e o valor da soma de quadrados. Este algoritmo se inicia, na linha dois, com o uso da heurística construtiva de Lloyd, apresentada pelo algoritmo 1, para criação de uma solução inicial para o problema. Então, por um determinado número de iterações, representado pelo *loop* na linha quatro, cria-se uma nova solução a partir do método TabuNeighborhoodSearch apresentado no algoritmo 5. Na linha seis, essa nova solução é adicionada à lista tabu, de forma a evitar que essa solução seja visitada novamente. Então, na linha sete, verifica-se a nova solução é melhor que a melhor solução. Em caso positivo, a melhor solução passa a ser a nova solução. A execução do *loop* é controlada

```
1  KPopularHeur (n_i, k)
       entradas : Um conjunto de n pontos no plano, um valor
                  inteiro k de centros
       saída    : Vetor com a divisão dos n pontos em k clusters,
                  um valor inteiro ssq com a soma de quadrados
2      calcular distância entre todos os n pontos;
3      media ← média das distâncias entre todos os pontos;
4      for i ← 0; i < n; i ← i + 1 do
5          neighbors[i] ← número de vizinhos de i cuja distância
             para ele é maior que media ;
6      end
7      ordenação do vetor neighbors ;
8      for i ← 0; i < k; i ← i + 1 do
9          j ← n − 1 ;
10         centros[i] ← neighbors[j] ;
11         j ← j − 1 ;
12     end
13     for i ← 0; i < n; i ← i + 1 do
14         colocar i no cluster cujo centro é mais próximo dele ;
15     end
16     calcular soma de quadrados ;
```

Algorithm 4: Pseudocódigo do algoritmo K-Popular.

pela variável *sem_mudanca*. Essa variável conta quantas iterações se passaram sem a melhor solução mudar, portanto a variável recebe valores na linha dez, treze e dezessete do algoritmo. O valor da variável é atualizado com zero se houve mudança na solução ou é somado mais um ao seu valor se não aconteceu alguma mudança.

O método TabuNeighborhoodSearch, que é utilizado na metaheurística descrita acima, é apresentado no algoritmo 5. Ele é iniciado na linha três, em um *loop* que percorre todos os centros da solução inicial recebida como parâmetro. Na linha quatro, calcula-se o *delta_muk*: a distância entre o centro atual para todos os outros pontos. Em seguida, na linha cinco, percorre-se todos os pontos da entrada e na linha seis, é calculado o valor da função objetivo se o ponto atual for colocado na solução. Na linha oito, ordena-se o vetor com o cálculo dos valores da função objetivo. Este vetor começa ser percorrido na linha nove e é verificado, na linha dez, se o ponto referente a cada valor em *deltaJ* está na lista tabu. Em caso negativo, o ponto é adicionado à nova solução na linha onze.

Três das metaheurísticas implementadas neste trabalho utilizam busca local para melhorar suas soluções. Sendo assim, foi implementada a parte de busca local do algoritmo de Lloyd encontrado na literatura. Ele é utilizado como o método de busca local dessas três heurísticas. Esse procedimento é apresentado no algoritmo 7. O método recebe como entrada um vetor com uma solução inicial para o problema e um valor inteiro para controlar o número de iterações do algoritmo. Ele é iniciado na linha dois com um *loop* que itera enquanto o algoritmo não converge. Na linha três, temos um *loop* que percorre todos os pontos recebidos como entrada e os colocam no *cluster* cujo centro é mais próximo a eles. Em seguida, percorre-se os *k* centros, calculando a média de cada *cluster* na linha sete, escolhendo um novo centróide (ponto mais próximo a média do *cluster* na linha oito, e finalmente, na linha onze, adicionando o novo centróide em um vetor com todos os novos centros. Na

```
1  TabuNeighborhoodSearch (n_i, k)
       entradas : A melhor solução atual para o problema, que
                  consiste da divisão dos n pontos em k centros e
                  uma lista tabu
       saída    : Uma nova solução para o problema
2      for i ← 0; i < k; i ← i + 1 do
3          delta_muk ← distância entre centro i para todos os
             pontos ;
4          for j ← 0; j < n; j ← j + 1 do
5              deltaJ[j] = −2∗ distância entre i e j
                 ∗delta_muk[j] + delta_muk[j]^2 ;
6          end
7          ordenar deltaJ ;
8          for j ← 0; j < deltaJ.size; j ← j + 1 do
9              if j.point não está na lista tabu then
10                 adicionar ponto em nova solução ;
11             end
12         end
13     end
```

Algorithm 5: Pseudocódigo do método *TabuNeighborhoodSearch* usado no algoritmo de busca tabu.

```
1  TabuSearch (n_i, k)
       entradas : Um conjunto de n pontos no plano, um valor
                  inteiro k de centros
       saída    : Vetor com a divisão dos n pontos em k clusters,
                  um valor inteiro ssq com a soma de quadrados
2      calcular distância entre todos os pontos ;
3      melhor_solucao ← LloydInitHeur(n_i, k) ;
4      sem_mudanca ← 0 ;
5      while sem_mudanca ≤ max_iteracoes do
6          nova_solucao ←
             TabuNeighborhoodSearch(melhor_solucao, lista_tabu)
             ;
7          adicionar nova_solucao à lista_tabu ;
8          if nova_solucao.ssq ≤ melhor_solucao.ssq then
9              melhor_solucao ← nova_solucao ;
10             if diferença entre nova_solucao e melhor_solucao
                 não é significativa then
11                 sem_mudanca ← sem_mudanca + 1 ;
12             end
13             else
14                 sem_mudanca ← 0 ;
15             end
16         end
17         else
18             sem_mudanca ← sem_mudanca + 1 ;
19         end
20     end
```

Algorithm 6: Pseudocódigo da metaheurística de busca tabu.

linha onze, é feita a substituição dos antigos centros pelos novos e o algoritmo termina com o cálculo da soma de quadrados na linha treze. O controle do número de iterações do algoritmo é feito a

```
1  LloydLocalSearch (solucao_inicial, threshold)
       entradas : Um vetor com uma solução inicial para o
                  problema e um valor inteiro para controlar o
                  número de iterações
       saída    : Vetor com a divisão dos n pontos em k clusters,
                  um valor inteiro ssq com a soma de quadrados
2      while não convergir do
3          for i ← 0; i < n; i ← i + 1 do
4              colocar i no cluster cujo centro é mais próximo
                dele ;
5          end
6          for i ← 0; i < centros; i ← i + 1 do
7              centroide ← média do cluster de centro[i] ;
8              novocentro ← ponto mais próximo de centroide ;
9              novoscentros ← adição de novocentro ;
10         end
11         centros ← novoscentros ;
12     end
13     calcular soma de quadrados ;
```

Algorithm 7: Pseudocódigo da busca local do algoritmo de Lloyd.

partir da variável de *threshold*, que evita que o algoritmo continue a iterar quando os pontos param de trocar de posição entre *clusters*. A execução do *loop* é controlada pela variável $sem_mudanca$. Essa variável conta quantas iterações se passaram sem a melhor solução mudar, portanto a variável recebe valores na linha quatro, dez e treze do algoritmo. O valor da variável é atualizado com zero se houve mudança na solução ou é somado mais um ao seu valor se não aconteceu alguma mudança.

A outra metaheurística da literatura implementada foi a *Iterated Local Search* [17], que é apresentada no algoritmo 8. Ela possui a mesma entrada e saída que a metaheurística de busca tabu. O algoritmo se inicia, na linha dois, com o cálculo da distância entre todos os pontos. Na linha três, a heurística construtiva de Lloyd, apresentada no algoritmo 1, é usada para geração de uma solução inicial. Na linha quatro é usada a busca local de Lloyd, descrita anteriormente no algoritmo 7, para encontrar a melhor solução atual. Então, na linha seis, inicia-se um ciclo que executa por um certo número de iterações. Dentro dele, na linha sete, é feita uma perturbação na melhor solução atual. No caso, troca-se de maneira aleatória um dos pontos considerados como centros. Então, é feita uma busca local a partir desta solução alterada, como visto na linha oito. Depois disso, na linha nove, é verificado se a nova solução encontrada na busca local é melhor que a melhor solução. Em caso positivo, a melhor solução passa a ser a nova solução.

As metaheurísticas propostas neste trabalho são um algoritmo de GRASP e um algoritmo genético. O algoritmo genético é usado apenas para gerar uma configuração de *clusters* iniciais e utiliza o algoritmo de Lloyd em seguida. O algoritmo de GRASP é inspirado na heurística K-Furthest proposta neste trabalho e utiliza a busca local de Lloyd.

O algoritmo 9 apresenta o método de construção de solução GRASP, chamado de GRASPConstruct. Ele é iniciado com o cálculo da distância entre todos os pontos. Na linha três, ocorre a escolha de um ponto aleatório como centro inicial. Na linha quatro, calcula-se

```
1  IteratedLocalSearch (n_i, k)
       entradas : Um conjunto de n pontos no plano, um valor
                  inteiro k de centros
       saída    : Vetor com a divisão dos n pontos em k clusters,
                  um valor inteiro ssq com a soma de quadrados
2      calcular distância entre todos os pontos ;
3      solucao_inicial ← LloydInitHeur(n_i, k) ;
4      melhor_solucao ←
        LloydLocalSearch(solucao_inicial, threshold) ;
5      sem_mudanca ← 0 ;
6      while sem_mudanca ≤ max_iteracoes do
7          solucao_alterada ← melhor_solucao com a troca de
            um dos centros, de maneira aleatória ;
8          nova_solucao ←
            LloydLocalSearch(solucao_inicial, threshold) ;
9          if nova_solucao.ssq ≤ melhor_solucao.ssq then
10             melhor_solucao ← nova_solucao ;
11             sem_mudanca ← 0 ;
12         end
13         else
14             sem_mudanca ← sem_mudanca + 1 ;
15         end
16     end
```

Algorithm 8: Pseudocódigo da metaheurística de *iterated local search*.

a distância entre o centro escolhido e todos os pontos recebidos como entrada. Na linha cinco, é iniciado um *loop* responsável por escolher os outros $k - 1$ centros. Dentro deste ciclo, na linha seis, atualiza-se a variável *max_dist* para o maior valor de distância encontrado entre o centro inicial e os outros pontos. Em seguida, utiliza-se um *loop* na linha sete para percorrer todos os pontos de entrada, verificando, na linha oito, se a distância entre o ponto e o centro é maior ou igual a $(max\_dist - alpha * max\_dist)$. Em caso positivo, esse ponto é adicionado a uma linha de candidatos possíveis na linha nove. Finalmente, na linha doze, escolhe-se, de maneira aleatória, um ponto da lista de candidatos possíveis para ser o novo *cluster*.

No algoritmo 10 é apresentada a metaheurística GRASP. Ela é iniciada com a construção de uma solução através do método GRASPConstruct descrito anteriormente, na linha dois. Em seguida, por um número de iterações determinado por um parâmetro de entrada, chama-se o método GRASPConstruct novamente para criar uma solução inicial, na linha quatro. Na linha cinco, é feita a busca local do algoritmo de Lloyd a partir da solução inicial gerada no passo anterior. Na linha seis, é verificado se a nova solução é melhor que a melhor solução. Em caso positivo, na linha sete, a melhor solução é atualizada com a nova solução. O algoritmo termina com o cálculo da soma de quadrados na linha dez.

O algoritmo 11 apresenta o algoritmo genético proposto. A metaheurística é iniciada com a criação da população inicial na linha dois. Um indivíduo é um conjunto de k centros iniciais para o problema. Na linha três, acontece o cálculo da *fitness* da população inicial. A *fitness* representa a qualidade de um indivíduo. Como nosso algoritmo foi desenvolvido com o objetivo de apenas gerar

```
1  GRASPConstruct (n_i, k)
       entradas : Um conjunto de n pontos no plano, um valor
                  inteiro k de centros, um valor inteiro α
       saída    : Vetor com a divisão dos n pontos em k clusters,
                  um valor inteiro ssq com a soma de quadrados
2      calcular distância entre todos os pontos ;
3      centro[0] ← um ponto aleatório de n ;
4      calcular distância entre o primeiro centro escolhido e
         todos os pontos de n ;
5      for i ← 1; i < k; i ← i + 1 do
6          max_dist ← maior distância encontrada entre
             centro[0] e todos os pontos de n ;
7          for j ← 0; j < n; j ← j + 1 do
8              if distância entre j e
                 centro[0] ≥ (max_dist − alpha ∗ max_dist) then
9                  adicionar j à lista de candidatos possíveis ;
10             end
11         end
12         escolher, de maneira aleatória, um ponto da lista de
             candidatos possíveis para ser o novo cluster ;
13     end
```

Algorithm 9: Pseudocódigo da método GRASPConstruct.

```
1  GRASPMetaHeur (n_i, k)
       entradas : Um conjunto de n pontos no plano, um valor
                  inteiro k de centros, um valor inteiro α, um
                  valor inteiro n_iter para o número de iterações
                  do algoritmo
       saída    : Vetor com a divisão dos n pontos em k clusters,
                  um valor inteiro ssq com a soma de quadrados
2      melhor_solucao ← GraspConstruct(α) ;
3      for i ← 0; i < n_iter; i ← i + 1 do
4          solucao_inicial ← GraspConstruct(α) ;
5          nova_solucao ←
             LloydLocalSearch(solucao_inicial, n_iter) ;
6          if nova_solucao.ssq ≤ melhor_solucao.ssq then
7              melhor_solucao ← nova_solucao ;
8          end
9      end
10     calcular soma de quadrados ;
```

Algorithm 10: Pseudocódigo da metaheurística de GRASP proposta.

uma configuração de centros iniciais, foi escolhido como função de *fitness* a soma das distâncias entre os centros escolhidos. Quanto maior o valor da *fitness*, melhor é o indivíduo. Dessa forma, o cálculo da *fitness* não depende da divisão dos outros $n-k$ pontos entre os *clusters* formados por esses k centros. Na linha quatro, é iniciado um ciclo que executa em quanto não alcançar a condição de parada. Neste caso, o algoritmo executa por um número determinado de gerações, que é definido por um parâmetro de entrada. Na linha cinco é feita a seleção dos melhores indivíduos e são aplicado os operadores de cruzamento e mutação nesses indivíduos na linha seis. Para a mutação, escolhe-se um ponto aleatório da solução e

```
1  GAMetaHeur (n_i, k)
       entradas : Um conjunto de n pontos no plano, um valor
                  inteiro k de centros, um valor inteiro α, um
                  valor inteiro n_iter para o número de iterações
                  do algoritmo
       saída    : Vetor com a divisão dos n pontos em k
2      criar população inicial ;
3      calcular fitness da população inicial ;
4      while não alcançou condição de parada do
5          selecionar os melhores indivíduos ;
6          fazer mutação e cruzamento ;
7          calcular fitness dos novos indivíduos ;
8          atualizar população ;
9      end
```

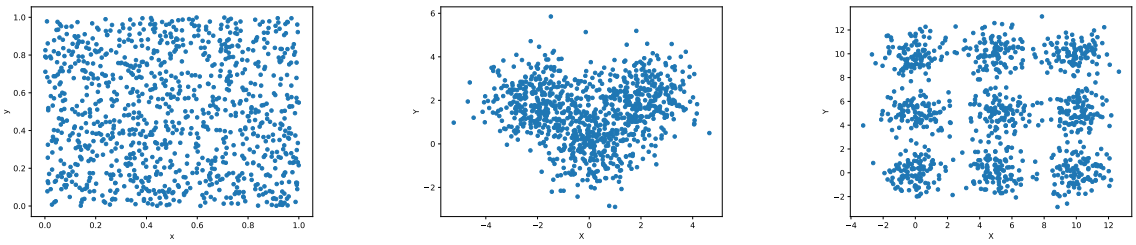
Algorithm 11: Pseudocódigo da metaheurística de algoritmo genético proposta.

este ponto é alterado por algum outro ponto aleatório que está no conjunto $n-k$. Para o cruzamento, é feito o cruzamento de um ponto. Escolhe-se um ponto aleatório da solução, o que está antes desse ponto no indivíduo A é anexado ao que está depois deste ponto no indivíduo B. Na linha sete, calcula-se a *fitness* para os novos indivíduos criados e finalmente, na linha oito, a população é atualizada com os novos indivíduos que foram gerados. Esse algoritmo genético é usado para gerar uma configuração inicial de centros. Sendo assim, essa configuração inicial é, então, usada como solução inicial para a busca local do algoritmo de Lloyd apresentado em 7.

Três das metaheurísticas apresentadas aqui possuem complexidade $O(n^2)$, na qual n é o número de pontos da entrada do problema. O algoritmo de GRASP, *iterated local search* e lista tabu executam o cálculo de distância entre todos os pontos, que é $O(n^2)$. A complexidade do algoritmo genético é um pouco diferente. Apesar de também calcular a distância entre todos os pontos, a complexidade desta metaheurística não depende apenas da entrada, mas também dos parâmetros definidos no algoritmo, como número de gerações e tamanho da população. A heurística de algoritmo genético é dividida em duas partes: geração da configuração inicial por evolução e uso da busca local de Lloyd. A complexidade da busca local é $O(n^2)$. Já o algoritmo evolutivo executa por g gerações com m indivíduos. Considerando que os operadores de mutação e cruzamento têm complexidade $O(k)$, sendo k o número de centros do problema, temos complexidade $O(gmk)$. Portanto, a complexidade desta metaheurística é $O(n^2) + O(gmk)$.

5 EXPERIMENTOS COMPUTACIONAIS

O objetivo desta seção é apresentar dados que demonstram o desempenho das heurísticas da literatura e das novas heurísticas propostas e implementadas. Uma série de experimentos foram realizados e serão descritos a seguir. Os experimentos foram realizados em uma máquina com processador AMD® Ryzen™ 5 2400G, de 8 núcleos a 3.60GHz. A máquina opera a 32 ou 64 bits, e possui 16GB de memória RAM DIMM DDR4. O sistema operacional utilizado foi o Ubuntu 18.04.1 LTS *Bionic Beaver*. As heurísticas foram implementadas na



(a) *Dataset* uniforme **(b) *Dataset* triangular** **(c) *Dataset* Grid**

Figura 1: Instâncias *ad-hoc* gerados para testes das heurísticas

linguagem *Python*, utilizando as bibliotecas *numpy* e *pandas* para realização de operações vetoriais.

As instâncias de testes foram escolhidas com o objetivo de demonstrar características importantes das heurísticas: **complexidade algorítmica** e **robustez**. Três instâncias *ad-hoc* foram criadas, e de cada uma foram gerados *datasets* contendo $\{100, 500, 1000, 5000, 10000\}$ pontos, a fim de se estudar o comportamento das heurísticas nos dois aspectos citados acima. A primeira instância, apresentada na figura 1a, é uma sequência de pontos distribuídos em uma região do espaço bi-dimensional de maneira uniforme. Assim, qualquer agrupamento encontrado pelo MSSC será arbitrariamente ruim. A segunda instância, apresentada na figura 1b, é constituída por uma sequência de pontos seguindo uma distribuição gaussiana com três pontos centrais: $(-2, 2), (0, 0), (2, 2)$. Os pontos destas gaussianas se confundem, o que pode levar a uma classificação errônea pelo MSSC. Por fim, a terceira instâncias gerada, apresentada na figura 1c, é constituída por pontos distribuídos seguindo seis gaussianas, centradas nos pontos $(0, 0), (0, 5), (0, 10), (5, 0), (5, 5), (5, 10), (10, 0), (10, 5), (10, 10)$. Desta forma os pontos estão identicamente distribuídos, e é fácil para o MSSC separá-los corretamente. Instâncias reais também foram utilizadas para demonstrar a robustez dos algoritmos. Foram utilizadas as bases *Breast Cancer*[1], *Iris*[2], e *Wine*[3], do repositório de Aprendizado de Máquina da *University of California, Irvine*[4]. Estas bases foram escolhidas por serem as mais utilizadas na literatura de aprendizadode máquina, em problemas de classificação ou de clustering. As bases foram utilizadas também nos *papers* utilizados como referência para as metaheurísticas da literatura [7] [17].

Os testes foram realizados da seguinte forma: para cada um dos 18 *datasets* (15 *datasets* sintéticos, de três tipos, e cinco tamanhos variados, e três *datasets* reais), foi variado o valor de k (número de centros) de 1 a 10, e a semente do gerador de números aleatórios, de 1 a 30. O desconhecimento do valor ótimo do *MSSC* para cada instância impossibilitou o cálculo da diferença entre os resultados das heurísticas implementadas e o ótimo. Portanto, os resultados serão comparados apenas entre si.

Os resultados das execuções das heurísticas construtivas, descritas na seção 4.1, nos *datasets* são apresentados nas tabelas 1 e 2. Pode-se observar primeiramente que os valores do *MSSC* para os *datasets* reais são ordens de grandeza maior que os valores para os sintéticos. Esta discrepância pode ser explicada pelo fato de que os pontos pertencentes aos conjuntos reais estão em um espaço de maior dimensionalidade, e portanto são mais difíceis de serem agrupados. Pode-se observar que tanto nos conjuntos reais quanto

Dataset	Método	MSSC médio	Desvio	Mediana
breast	lloyd	1.442×10^8	2.843×10^5	7.531×10^7
	kmeans++	1.002×10^8	2.331×10^5	3.500×10^7
	k-furthest	2.601×10^8	2.345×10^5	2.075×10^8
	k-popular	3.468×10^9	4.086×10^5	3.725×10^9
iris	lloyd	3.809×10^2	4.950×10^2	1.230×10^2
	kmeans++	2.478×10^2	4.534×10^2	8.752×10^1
	k-furthest	4.539×10^2	4.001×10^2	3.226×10^2
	k-popular	1.262×10^3	9.519×10^2	6.669×10^2
wine	lloyd	9.507×10^6	1.126×10^6	4.083×10^6
	kmeans++	5.607×10^6	1.175×10^6	1.370×10^6
	k-furthest	1.334×10^7	1.249×10^6	1.031×10^7
	k-popular	1.067×10^8	3.114×10^6	1.093×10^8

Tabela 1: Resultado dos testes das heurísticas construtivas para os *datasets* reais

Dataset	Método	MSSC Médio	Desvio	Mediana
grid6	lloyd	36176.06	51828.21	14120.20
	kmeans++	27860.67	44191.51	9723.82
	k-furthest	74272.72	85328.87	34835.48
	k-popular	95628.63	124572.17	24851.52
triangle	lloyd	4685.97	7639.52	1545.49
	kmeans++	3791.01	6370.24	1218.33
	k-furthest	10548.29	14350.10	3935.71
	k-popular	26600.63	39555.36	8057.80
uniform	lloyd	6750.07	9881.96	2092.00
	kmeans++	5580.98	8531.75	1592.97
	k-furthest	16518.46	20055.36	4574.51
	k-popular	24674.24	28304.67	7157.48

Tabela 2: Resultados das execuções das heurísticas construtivas nos *datasets* sintéticos

[1] http://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29
[2] http://archive.ics.uci.edu/ml/datasets/Iris
[3] http://archive.ics.uci.edu/ml/datasets/Wine
[4] http://archive.ics.uci.edu/ml/index.php

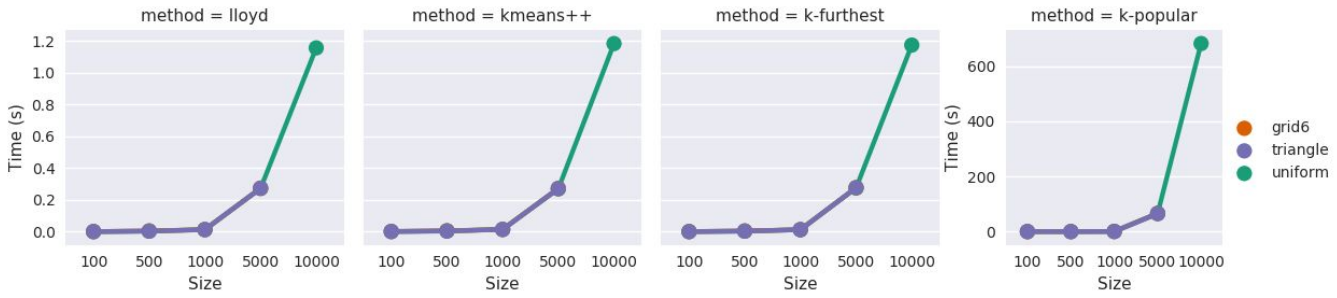


Figura 2: Gráfico de tempo de execução para as heurísticas construtivas implementadas

nos sintéticos, a heurística construtiva de melhor performance é a **k-means++** (ressaltada em negrito nas tabelas). Isto comprova a importância da escolha inicial dos k centros. A qualidade da solução encontrada por qualquer algoritmo de busca local aplicado posteriormente (*e.g.* Algoritmo de Lloyd) depende muito desta escolha e o *k-means++* foi desenvolvido exatamente com o objetivo de aprimorar esta configuração inicial.

Os gráficos da figura 2 apresentam os resultados de tempo de execução de cada uma das heurísticas construtivas nos *datasets* **sintéticos**. Pode-se observar que as heurísticas possuem comportamento aproximadamente quadrático, como esperado após as análises de complexidade realizadas na seção 4.1. É importante observar que a heurística *K-Popular* (gráfico mais a direita da figura) se destaca por possuir um tempo consideravelmente maior do que as outras. Esta diferença se dá por ordem de uma constante, pois o formato do gráfico ainda é similar ao das outras heurísticas.

Os resultados das execuções das metaheurísticas descritas na seção 4.2 nos *datasets* são apresentados nas tabelas 3 e 4. Os seguintes parâmetros foram utilizados para os algoritmos: $threshold = 10$ (para todas as metaheurísticas, significando número de iterações toleradas onde nenhuma melhora no valor da função objetivo é observada), $alpha = 0.5$ para GRASP, e para o GA foi utilizada a configuração:

- $tamanho\_da\_populacao = 25$,
- $numero\_de\_geracoes = 50$,
- $prob\_de\_cruzamento = 0.6$,
- $prob\_de\_mutacao = 0.4$

Pode-se observar que, para os *datasets* reais, a metaheurística que alcançou a melhor performance foi a *ILS* (ressaltada em negrito na tabela 3). Os melhores valores para o *MSSC* médio e mediano foram atingidos utilizando este algoritmo. Isto mostra que uma pequena pertubação em um ótimo local (a troca de um único centro em uma solução considerada como boa na iteração corrente) já é suficiente para permitir uma exploração maior do espaço de soluções, o que leva o algoritmo a encontrar soluções cada vez melhores. Os resultados nos *datasets* sintéticos já são um pouco diferentes. A metaheurística que obteve os melhores valores de *MSSC* médio foi a *GRASP* (ressaltada em negrito na tabela 4). Os valores da mediana do *MSSC* obtidos pela *ILS* neste caso, foram os melhores. Por isto ressaltamos esta metaheurística em itálico na mesma tabela.

Dataset	Método	MSSC médio	Desvio Padrão	Mediana
breast	Tabu	4.105×10^7	2.161×10^7	3.464×10^7
	ILS	2.714×10^7	2.165×10^7	1.736×10^7
	GRASP	2.940×10^7	2.067×10^7	1.838×10^7
	GA	3.359×10^7	3.688×10^7	1.767×10^7
iris	Tabu	91.36	75.71	67.37
	ILS	60.85	38.23	44.00
	GRASP	62.49	36.46	46.56
	GA	102.37	136.58	54.41
wine	Tabu	1.843×10^6	1.388×10^6	1.330×10^6
	ILS	1.278×10^6	1.339×10^6	6.571×10^5
	GRASP	1.329×10^6	1.296×10^6	6.571×10^5
	GA	2.165×10^6	3.663×10^6	7.446×10^5

Tabela 3: Resultados das execuções das metaheurísticas nos *datasets* reais

Dataset	Método	MSSC Médio	Desvio	Mediana
grid6	Tabu	34213.30	52212.94	11307.28
	ILS	25705.49	42952.88	7140.06
	GRASP	25667.95	42900.91	7164.68
	GA	26881.01	44431.86	7473.62
triangle	Tabu	5471.11	7342.13	1605.86
	ILS	4262.55	5945.40	1178.40
	GRASP	4250.10	5940.86	1185.70
	GA	4325.55	5962.18	1217.02
uniform	Tabu	4335.85	6158.84	1291.59
	ILS	3319.63	5040.80	867.47
	GRASP	3306.44	5033.94	868.09
	GA	3375.01	5077.71	910.94

Tabela 4: Resultados das execuções das metaheurísticas nos *datasets* sintéticos

Os gráficos da figura 3 apresentam os resultados de tempo de execução para cada uma das metaheurísticas apresentadas na seção 4.2. Pode-se observar que todas as metaheurísticas possuem um

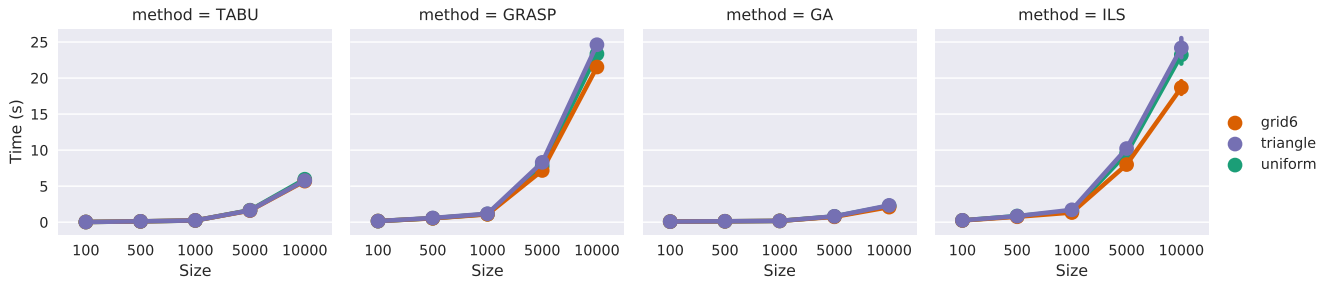


Figura 3: Gráfico de tempo de execução para as metaheurísticas implementadas, nos *datasets* sintéticos

comportamento aproximadamente quadrático, assim como observado nas análises de complexidade realizadas na seção 4.2. É visível que as metaheurísticas GRASP e ILS possuem algum fator constante que aumenta sua inclinação em relação às outras metaheurísticas (suas curvas são mais acentuadas). Isto pode ser explicado pelo fato de que estas duas heurísticas executam o algoritmo de Lloyd para intensificação em um loop, até atingir a convergência, e isto causa um aumento em sua complexidade algoritmica.

6 CONCLUSÃO

O MSSC (*Minimum Sum-of-Squares Clustering* - Agrupamento Mínimo por Soma de Quadrados) Euclidiano é um problema de otimização combinatória famoso na literatura de aprendizado de máquina, teoria da computação e geometria computacional [4]. Apesar de existirem implementações eficientes de algoritmos para o problema [10] [6] e em seu caso médio os algoritmos serem relativamente rápidos, em seu pior caso o problema é NP-Hard até para pontos no plano [15].

O objetivo deste trabalho é, propor duas novas heurísticas construtivas e duas novas metaheurísticas que sejam competitivas com o estado da arte, e realizar as devidas análises comparativas. As heurísticas construtivas da literatura implementadas foram a inicialização da heurística de Lloyd (*LloydInitial*) e a *k-means++*, e as propostas pelo trabalho são a *K-furthest* e *K-popular*. As metaheurísticas da literatura implementadas são a *Tabu-Search* e a *ILS*, e as propostas são a *GRASP* e o Algoritmo Genético para inicialização (*GA*).

Experimentos realizados com *datasets* reais e sintéticos comprovaram que a inicialização de uma solução é crucial na convergência da mesma. Entre as heurísticas construtivas, a *k-means++* obteve o melhor desempenho observado para todos os *datasets*, com vantagem da ordem de 10^3 para algumas instâncias. Já entre as metaheurísticas, o *ILS* obtém melhores resultados para os *datasets* reais, e o *GRASP* se destaca entre os sintéticos, o que mostra que intensificação e diversificação são igualmente importantes no processo de busca da solução ótima.

REFERÊNCIAS

[1] Khaled S Al-Sultan. 1995. A tabu search approach to the clustering problem. *Pattern recognition* 28, 9 (1995), 1443–1451.
[2] Daniel Aloise, Amit Deshpande, Pierre Hansen, and Preyas Popat. 2009. NP-hardness of Euclidean sum-of-squares clustering. *Machine Learning* 75, 2 (01 May 2009), 245–248. https://doi.org/10.1007/s10994-009-5103-0
[3] David Arthur and Sergei Vassilvitskii. 2007. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 1027–1035.
[4] Pranjal Awasthi, Moses Charikar, Ravishankar Krishnaswamy, and Ali Kemal Sinop. 2015. The hardness of approximation of euclidean k-means. *arXiv preprint arXiv:1502.03316* (2015).
[5] José Ramón Cano, Oscar Cordón, Francisco Herrera, and Luciano Sánchez. 2002. A GRASP algorithm for clustering. In *Ibero-American Conference on Artificial Intelligence*. Springer, 214–223.
[6] Rui Máximo Esteves, Thomas Hacker, and Chunming Rong. 2013. Competitive k-means, a new accurate and distributed k-means algorithm for large datasets. In *2013 IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 17–24.
[7] Kojo Sarfo Gyamfi, James Brusey, and Andrew Hunt. 2017. K-Means Clustering using Tabu Search with Quantized Means. *arXiv preprint arXiv:1703.08440* (2017).
[8] John A Hartigan. 1975. Clustering algorithms. (1975).
[9] John A Hartigan and Manchek A Wong. 1979. Algorithm AS 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 28, 1 (1979), 100–108.
[10] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu. 2002. An Efficient k-Means Clustering Algorithm: Analysis and Implementation. *IEEE Transactions on Pattern Analysis & Machine Intelligence* 24 (07 2002), 881–892. https://doi.org/10.1109/TPAMI.2002.1017616
[11] T Warren Liao. 2005. Clustering of time series data—a survey. *Pattern recognition* 38, 11 (2005), 1857–1874.
[12] Aristidis Likas, Nikos Vlassis, and Jakob J Verbeek. 2003. The global k-means clustering algorithm. *Pattern recognition* 36, 2 (2003), 451–461.
[13] Stuart Lloyd. 1982. Least squares quantization in PCM. *IEEE transactions on information theory* 28, 2 (1982), 129–137.
[14] James MacQueen et al. 1967. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, Vol. 1. Oakland, CA, USA, 281–297.
[15] Meena Mahajan, Prajakta Nimbhorkar, and Kasturi Varadarajan. 2009. The planar k-means problem is NP-hard. In *International Workshop on Algorithms and Computation*. Springer, 274–285.
[16] David W Matula and Farhad Shahrokhi. 1990. Sparsest cuts and bottlenecks in graphs. *Discrete Applied Mathematics* 27, 1-2 (1990), 113–123.
[17] Peter Merz. 2003. An iterated local search approach for minimum sum-of-squares clustering. In *International Symposium on Intelligent Data Analysis*. Springer, 286–296.
[18] Glenn W Milligan. 1980. An examination of the effect of six types of error perturbation on fifteen clustering algorithms. *Psychometrika* 45, 3 (1980), 325–342.
[19] Frank Nielsen and Richard Nock. 2014. Further heuristics for k-means: The merge-and-split heuristic and the (k, l)-means. *arXiv preprint arXiv:1406.6314* (2014).
[20] Gabriela Trazzi Perim, Estefhan Dazzi Wandekokem, and Flávio Miguel Varejão. 2008. K-means initialization methods for improving clustering by simulated annealing. In *Ibero-American Conference on Artificial Intelligence*. Springer, 133–142.
[21] Matus Telgarsky and Andrea Vattani. 2010. Hartigan's Method: k-means Clustering without Voronoi. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. 820–827.
[22] Chen Zhang and Shixiong Xia. 2009. K-means clustering algorithm with improved initial center. In *Knowledge Discovery and Data Mining, 2009. WKDD 2009. Second International Workshop on*. IEEE, 790–792.