

Tópicos de Análise de Algoritmos

Algoritmos de clustering

Sec 4.5, 4.6 e 4.7 do KT

Clustering

U : conjunto de n itens $\{p_1, \dots, p_n\}$.

$d(p_i, p_j)$: distância entre p_i e p_j .

- ▶ $d(p_i, p_i) = 0$ para $i = 1, \dots, n$
- ▶ $d(p_i, p_j) = d(p_j, p_i) > 0$ para $i \neq j$

Clustering

U : conjunto de n itens $\{p_1, \dots, p_n\}$.

$d(p_i, p_j)$: distância entre p_i e p_j .

- ▶ $d(p_i, p_i) = 0$ para $i = 1, \dots, n$
- ▶ $d(p_i, p_j) = d(p_j, p_i) > 0$ para $i \neq j$

k : número de grupos.

k -clustering: partição de U em k partes.

Clustering

U : conjunto de n itens $\{p_1, \dots, p_n\}$.

$d(p_i, p_j)$: distância entre p_i e p_j .

- ▶ $d(p_i, p_i) = 0$ para $i = 1, \dots, n$
- ▶ $d(p_i, p_j) = d(p_j, p_i) > 0$ para $i \neq j$

k : número de grupos.

k -clustering: partição de U em k partes.

espaçamento de k -clustering: distância mínima entre dois elementos de partes distintas do clustering.

Clustering

U : conjunto de n itens $\{p_1, \dots, p_n\}$.

$d(p_i, p_j)$: distância entre p_i e p_j .

- ▶ $d(p_i, p_i) = 0$ para $i = 1, \dots, n$
- ▶ $d(p_i, p_j) = d(p_j, p_i) > 0$ para $i \neq j$

k : número de grupos.

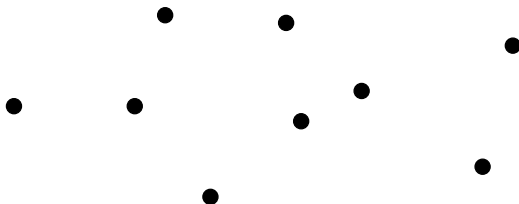
k -clustering: partição de U em k partes.

espaçamento de k -clustering: distância mínima entre dois elementos de partes distintas do clustering.

Problema: Dados U , d e k ,
encontrar um k -clustering com espaçamento máximo.

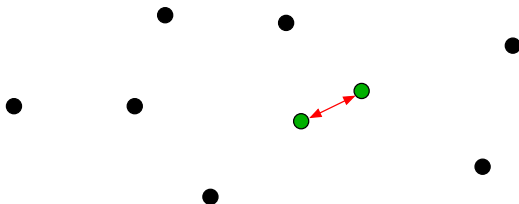
Clustering

Problema: Dados U , d e k ,
encontrar um k -clustering com espaçamento máximo.



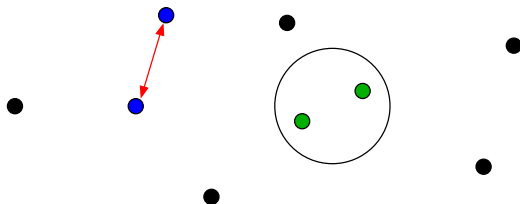
Clustering

Problema: Dados U , d e k ,
encontrar um k -clustering com espaçamento máximo.



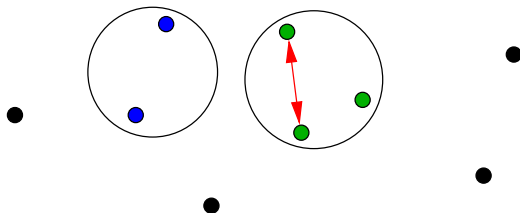
Clustering

Problema: Dados U , d e k ,
encontrar um k -clustering com espaçamento máximo.



Clustering

Problema: Dados U , d e k ,
encontrar um k -clustering com espaçamento máximo.



Algoritmo inspirado no Kruskal

Lembram do algoritmo de Kruskal?

Problema: Dado um grafo $G = (V, E)$ conexo, e um custo $c_e > 0$ para cada aresta e em E .

Árvore geradora mínima

Algoritmo inspirado no Kruskal

Lembram do algoritmo de Kruskal?

Problema: Dado um grafo $G = (V, E)$ conexo, e um custo $c_e > 0$ para cada aresta e em E .

Árvore geradora mínima

MST-KRUSKAL (V, E, c)

- 1 $e_1, \dots, e_m \leftarrow \text{ORDENE}(E, c)$ $\triangleright$ pelo valor de c_e
- 2 $T \leftarrow \emptyset$ $i \leftarrow 0$
- 3 enquanto $G' = (V, T)$ não é conexo faça
- 4 $i \leftarrow i + 1$
- 5 se $T \cup e_i$ não tem circuito
- 6 então $T \leftarrow T \cup e_i$
- 7 devolva T

Algoritmo inspirado no Kruskal

Como adaptar **MST-KRUSKAL** para resolver o k -clustering?

Algoritmo inspirado no Kruskal

Como adaptar **MST-KRUSKAL** para resolver o k -clustering?

Acrescentamos arestas até termos k componentes.

Algoritmo inspirado no Kruskal

Como adaptar **MST-KRUSKAL** para resolver o k -clustering?

Acrescentamos arestas até termos k componentes.

CLUSTERING-KRUSKAL (V, E, c, k)

- 1 $e_1, \dots, e_m \leftarrow \text{ORDENE}(E, c)$ $\triangleright$ pelo valor de c_e
- 2 $F \leftarrow \emptyset \quad i \leftarrow 0$
- 3 **enquanto** $G' = (V, F)$ tem $> k$ componentes **faça**
- 4 $i \leftarrow i + 1$
- 5 **se** $F \cup e_i$ não tem circuito
- 6 **então** $F \leftarrow F \cup e_i$
- 7 **devolva** F

Algoritmo inspirado no Kruskal

Como adaptar **MST-KRUSKAL** para resolver o k -clustering?

Acrescentamos arestas até termos k componentes.

CLUSTERING-KRUSKAL (V, E, c, k)

```
1   $e_1, \dots, e_m \leftarrow \text{ORDENE}(E, c)$   $\triangleright$  pelo valor de  $c_e$ 
2   $F \leftarrow \emptyset$        $i \leftarrow 0$ 
3  enquanto  $G' = (V, F)$  tem  $> k$  componentes faça
4       $i \leftarrow i + 1$ 
5      se  $F \cup e_i$  não tem circuito
6          então  $F \leftarrow F \cup e_i$ 
7  devolva  $F$ 
```

Este algoritmo resolve o problema?

Análise da correção

CLUSTERING-KRUSKAL (V, E, c, k)

```
1   $e_1, \dots, e_m \leftarrow \text{ORDENE}(E, c)$   $\triangleright$  pelo valor de  $c_e$ 
2   $F \leftarrow \emptyset$   $i \leftarrow 0$ 
3  enquanto  $G' = (V, F)$  tem  $\geq k$  componentes faça
4       $i \leftarrow i + 1$ 
5      se  $F \cup e_i$  não tem circuito
6          então  $F \leftarrow F \cup e_i$ 
7  devolva as componentes de  $F$ 
```

Qual é o espaçamento do clustering devolvido?

Análise da correção

CLUSTERING-KRUSKAL (V, E, c, k)

```
1   $e_1, \dots, e_m \leftarrow \text{ORDENE}(E, c)$   $\triangleright$  pelo valor de  $c_e$ 
2   $F \leftarrow \emptyset$   $i \leftarrow 0$ 
3  enquanto  $G' = (V, F)$  tem  $\geq k$  componentes faça
4       $i \leftarrow i + 1$ 
5      se  $F \cup e_i$  não tem circuito
6          então  $F \leftarrow F \cup e_i$ 
7  devolva as componentes de  $F$ 
```

Qual é o espaçamento do clustering devolvido?

É o custo da próxima aresta que seria incluída em F .

Análise da correção

CLUSTERING-KRUSKAL (V, E, c, k)

```
1   $e_1, \dots, e_m \leftarrow \text{ORDENE}(E, c)$   $\triangleright$  pelo valor de  $c_e$ 
2   $F \leftarrow \emptyset$   $i \leftarrow 0$ 
3  enquanto  $G' = (V, F)$  tem  $\geq k$  componentes faça
4       $i \leftarrow i + 1$ 
5      se  $F \cup e_i$  não tem circuito
6          então  $F \leftarrow F \cup e_i$ 
7  devolva as componentes de  $F$ 
```

Qual é o espaçamento do clustering devolvido?

É o custo da próxima aresta que seria incluída em F .

A próxima aresta que não forma circuito com F .

Análise da correção

Seja e a próxima aresta que seria incluída em F .

Análise da correção

Seja e a próxima aresta que seria incluída em F .

Seja $\mathcal{C} = \{C_1, \dots, C_k\}$ o k -clustering devolvido,
e $\mathcal{C}^* = \{C_1^*, \dots, C_k^*\}$ um k -clustering ótimo.

Análise da correção

Seja e a próxima aresta que seria incluída em F .

Seja $\mathcal{C} = \{C_1, \dots, C_k\}$ o k -clustering devolvido,
e $\mathcal{C}^* = \{C_1^*, \dots, C_k^*\}$ um k -clustering ótimo.

O espaçamento de $\mathcal{C}$ é c_e .

Basta mostrar que o espaçamento de $\mathcal{C}^*$ é no máximo c_e .

Análise da correção

Seja e a próxima aresta que seria incluída em F .

Seja $\mathcal{C} = \{C_1, \dots, C_k\}$ o k -clustering devolvido,
e $\mathcal{C}^* = \{C_1^*, \dots, C_k^*\}$ um k -clustering ótimo.

O espaçamento de $\mathcal{C}$ é c_e .

Basta mostrar que o espaçamento de $\mathcal{C}^*$ é no máximo c_e .

Algum C_i não está contido em nenhum C_j^* .

Análise da correção

Seja e a próxima aresta que seria incluída em F .

Seja $\mathcal{C} = \{C_1, \dots, C_k\}$ o k -clustering devolvido,
e $\mathcal{C}^* = \{C_1^*, \dots, C_k^*\}$ um k -clustering ótimo.

O espaçamento de $\mathcal{C}$ é c_e .

Basta mostrar que o espaçamento de $\mathcal{C}^*$ é no máximo c_e .

Algum C_i não está contido em nenhum C_j^* .

Seja j tal que $C_i \cap C_j^* \neq \emptyset$.

Análise da correção

Seja e a próxima aresta que seria incluída em F .

Seja $\mathcal{C} = \{C_1, \dots, C_k\}$ o k -clustering devolvido,
e $\mathcal{C}^* = \{C_1^*, \dots, C_k^*\}$ um k -clustering ótimo.

O espaçamento de $\mathcal{C}$ é c_e .

Basta mostrar que o espaçamento de $\mathcal{C}^*$ é no máximo c_e .

Alguns C_i não estão contidos em nenhum C_j^* .

Seja j tal que $C_i \cap C_j^* \neq \emptyset$.

Seja f aresta de F em C_i com um único extremo em C_j^* .

Análise da correção

Seja e a próxima aresta que seria incluída em F .

Seja $\mathcal{C} = \{C_1, \dots, C_k\}$ o k -clustering devolvido,
e $\mathcal{C}^* = \{C_1^*, \dots, C_k^*\}$ um k -clustering ótimo.

O espaçamento de $\mathcal{C}$ é c_e .

Basta mostrar que o espaçamento de $\mathcal{C}^*$ é no máximo c_e .

Algun C_i não está contido em nenhum C_j^* .

Seja j tal que $C_i \cap C_j^* \neq \emptyset$.

Seja f aresta de F em C_i com um único extremo em C_j^* .

Claro que $c_f \leq c_e$ e espaçamento de $\mathcal{C}^*$ é no máximo c_f .

Implementação

Como se implementa este algoritmo?

Implementação

Como se implementa este algoritmo?

Como se implementa o algoritmo de Kruskal?

Implementação

Como se implementa este algoritmo?

Como se implementa o algoritmo de Kruskal?

Estrutura de dados para conjuntos disjuntos...

Implementação

Como se implementa este algoritmo?

Como se implementa o algoritmo de Kruskal?

Estrutura de dados para conjuntos disjuntos...

`makeset( $x$ )`

1. $\text{pai}[x] \leftarrow x$
2. $\text{rank}[x] \leftarrow 0$

`findset( $x$ )`

1. se $x \neq \text{pai}[x]$
2. então $\text{pai}[x] \leftarrow \text{findset}(\text{pai}[x])$
3. devolva $\text{pai}[x]$

Union-Find

makeset(x)

1. $\text{pai}[x] \leftarrow x$
2. $\text{rank}[x] \leftarrow 0$

union(x, y)

1. $x' \leftarrow \text{findset}(x)$
2. $y' \leftarrow \text{findset}(y)$
3. **se** $x' \neq y'$
4. **então** link(x', y')

findset(x)

1. **se** $x \neq \text{pai}[x]$
2. **então** $\text{pai}[x] \leftarrow \text{findset}(\text{pai}[x])$
3. **devolva** $\text{pai}[x]$

link(x, y)

1. **se** $\text{rank}[x] > \text{rank}[y]$
2. **então** $\text{pai}[y] \leftarrow x$
3. **senão** $\text{pai}[x] \leftarrow y$
4. **se** $\text{rank}[x] = \text{rank}[y]$
5. **então** $\text{rank}[y] \leftarrow \text{rank}[y] + 1$

Union-Find

makeset(x)

1. $\text{pai}[x] \leftarrow x$
2. $\text{rank}[x] \leftarrow 0$

findset(x)

1. **se** $x \neq \text{pai}[x]$
2. **então** $\text{pai}[x] \leftarrow \text{findset}(\text{pai}[x])$
3. **devolva** $\text{pai}[x]$

union(x, y)

1. $x' \leftarrow \text{findset}(x)$
2. $y' \leftarrow \text{findset}(y)$
3. **se** $x' \neq y'$
4. **então** $\text{link}(x', y')$

link(x, y)

1. **se** $\text{rank}[x] > \text{rank}[y]$
2. **então** $\text{pai}[y] \leftarrow x$
3. **senão** $\text{pai}[x] \leftarrow y$
4. **se** $\text{rank}[x] = \text{rank}[y]$
5. **então** $\text{rank}[y] \leftarrow \text{rank}[y] + 1$

Altura da árvore enraizada em x : comprimento de um caminho mais longo de x até um descendente de x na árvore.

Note que a altura da árvore enraizada em x é no máximo $\text{rank}[x]$.

Exercício: Mostre que se x é uma raiz e $\text{rank}[x] = r$, então a árvore enraizada em x tem pelo menos 2^r elementos.

Union-Find: análise de pior caso

makeset(x)

1. $\text{pai}[x] \leftarrow x$
2. $\text{rank}[x] \leftarrow 0$

findset(x)

1. **se** $x \neq \text{pai}[x]$
2. **então** $\text{pai}[x] \leftarrow \text{findset}(\text{pai}[x])$
3. **devolva** $\text{pai}[x]$

union(x, y)

1. $x' \leftarrow \text{findset}(x)$
2. $y' \leftarrow \text{findset}(y)$
3. **se** $x' \neq y'$
4. **então** $\text{link}(x', y')$

link(x, y)

1. **se** $\text{rank}[x] > \text{rank}[y]$
2. **então** $\text{pai}[y] \leftarrow x$
3. **senão** $\text{pai}[x] \leftarrow y$
4. **se** $\text{rank}[x] = \text{rank}[y]$
5. **então** $\text{rank}[y] \leftarrow \text{rank}[y] + 1$

Exercício: Mostre que se x é uma raiz e $\text{rank}[x] = r$,
então a árvore enraizada em x tem pelo menos 2^r elementos.

Conclua que, se n é o número de elementos nas florestas,
então cada operação custa no pior caso $O(\lg n)$.

Union-Find: análise amortizada

makeset(x)

1. $\text{pai}[x] \leftarrow x$
2. $\text{rank}[x] \leftarrow 0$

findset(x)

1. **se** $x \neq \text{pai}[x]$
2. **então** $\text{pai}[x] \leftarrow \text{findset}(\text{pai}[x])$
3. **devolva** $\text{pai}[x]$

union(x, y)

1. $x' \leftarrow \text{findset}(x)$
2. $y' \leftarrow \text{findset}(y)$
3. **se** $x' \neq y'$
4. **então** $\text{link}(x', y')$

link(x, y)

1. **se** $\text{rank}[x] > \text{rank}[y]$
2. **então** $\text{pai}[y] \leftarrow x$
3. **senão** $\text{pai}[x] \leftarrow y$
4. **se** $\text{rank}[x] = \text{rank}[y]$
5. **então** $\text{rank}[y] \leftarrow \text{rank}[y] + 1$

Próximas aulas: análise amortizada.

Union-Find: análise amortizada

makeset(x)

1. $\text{pai}[x] \leftarrow x$
2. $\text{rank}[x] \leftarrow 0$

findset(x)

1. **se** $x \neq \text{pai}[x]$
2. **então** $\text{pai}[x] \leftarrow \text{findset}(\text{pai}[x])$
3. **devolva** $\text{pai}[x]$

union(x, y)

1. $x' \leftarrow \text{findset}(x)$
2. $y' \leftarrow \text{findset}(y)$
3. **se** $x' \neq y'$
4. **então** $\text{link}(x', y')$

link(x, y)

1. **se** $\text{rank}[x] > \text{rank}[y]$
2. **então** $\text{pai}[y] \leftarrow x$
3. **senão** $\text{pai}[x] \leftarrow y$
4. **se** $\text{rank}[x] = \text{rank}[y]$
5. **então** $\text{rank}[y] \leftarrow \text{rank}[y] + 1$

Próximas aulas: análise amortizada.

Vamos mostrar que

cada operação acima tem custo amortizado de $O(\lg^* n)$,
onde $\lg^* n$ é o número de vezes que temos que aplicar
o $\lg$ até atingir um número menor ou igual a 1.