Algoritmos de *bulk-loading* para o método de acesso métrico Onion-*tree*

Arthur Emanuel de O. Carosia
Profa. Dra. Cristina Dutra de Aguiar Ciferri

ICMC USP SÃO CARLOS
Grupo de Bases de Dados e Imagens USP - São Carlos
FAPESP
CNPq Conselho Nacional de Desenvolvimento Científico e Tecnológico
CAPES

Visão Geral

Onion-tree

- **Índice** para dados complexos.
- Memória **primária.**
- Inserção dos dados **um-a-um.**

Bulk-loading

- Construção da estrutura considerando **todos os elementos** do conjunto de dados de uma **única vez.**
- Analisar os dados **antecipadamente.**

Proposta

Elaborar algoritmos de bulk-loading para a Onion-tree.

Proposta de Bulk-Loading

Onion-tree

- Oferece suporte apenas à **inserção** de dados **um-a-um**.
- Índice **dependente** da ordem na qual os dados são inseridos.

Bulk-loading

- Criação do índice a partir de **grande conjunto de dados**.
- **Reconstrução** do índice.
- Analisar os dados **antecipadamente** para garantir melhor particionamento do espaço métrico.

Proposta de Bulk-Loading

Algoritmos desenvolvidos:

- GreedyBL → F-Onion-tree
- SampleBL
- HeightBL

SampleBL e HeightBL → F-Onion-tree e V-Onion-tree

Proposta de Bulk-Loading

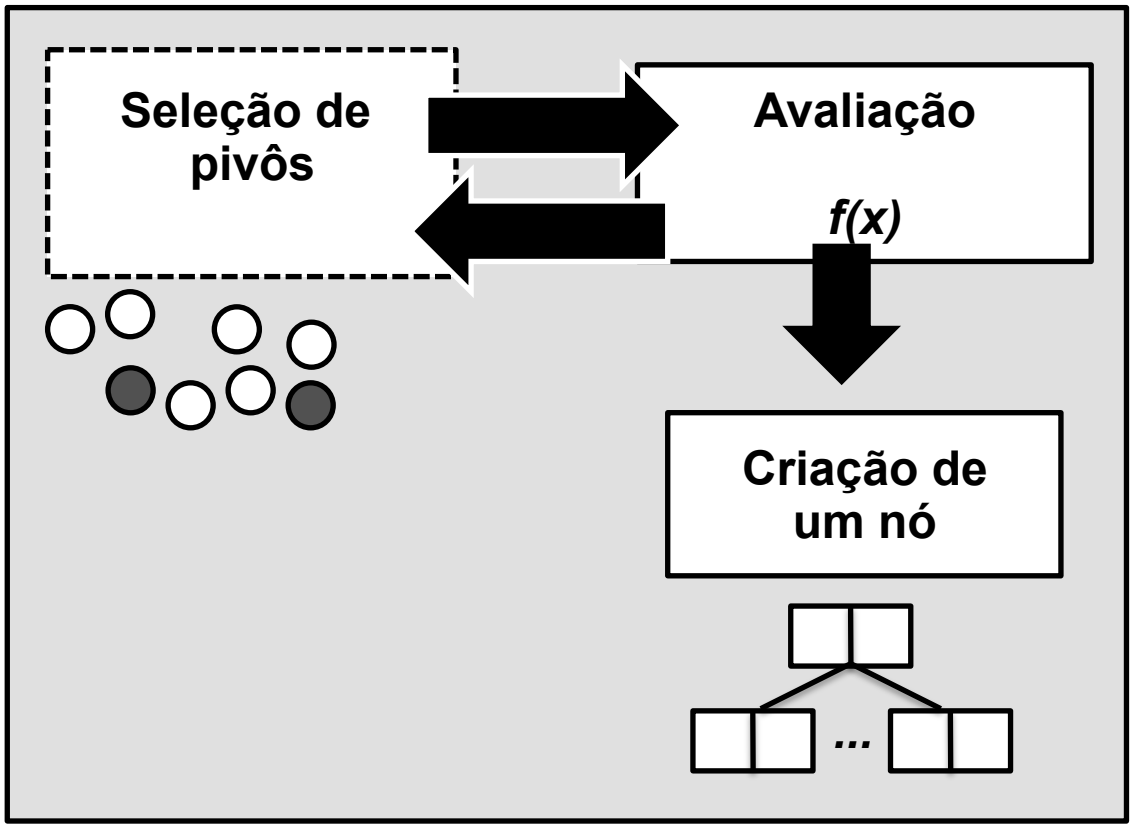
Algoritmos desenvolvidos:

- GreedyBL
- SampleBL
- HeightBL

Algoritmo GreedyBL

- Características
  - Top-Down.
  - Abordagem gulosa.
  - Obtém um índice balanceado.
  - Base para outros algoritmos.

Algoritmo GreedyBL

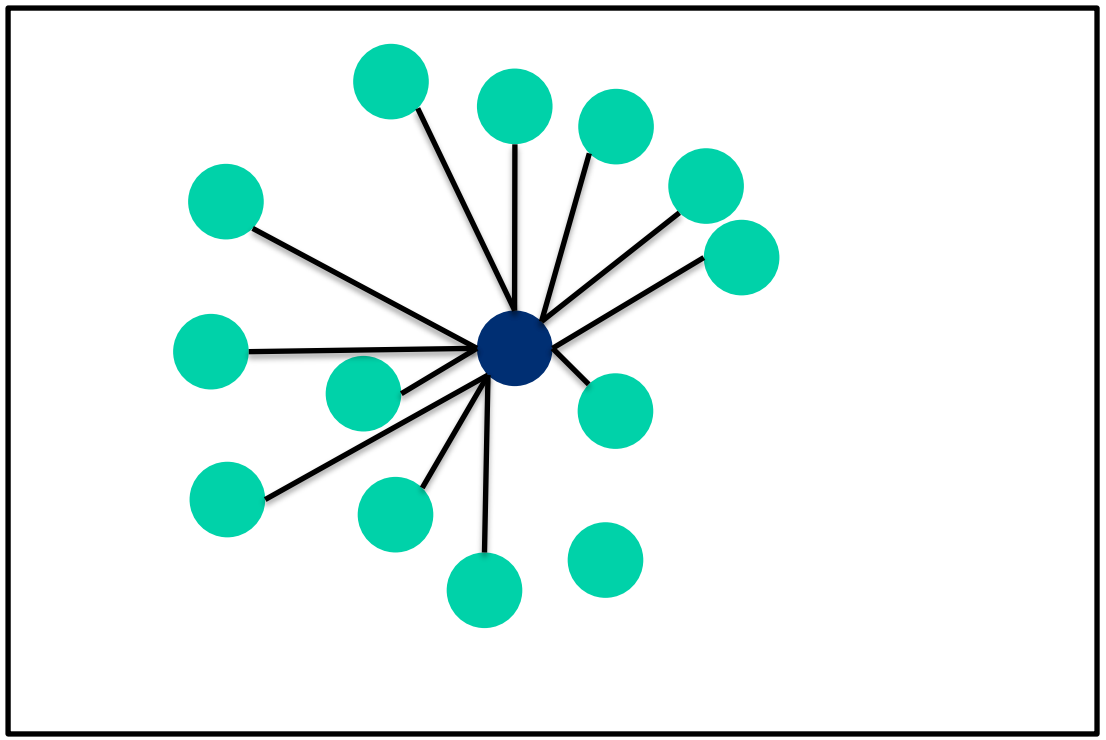


Algoritmo GreedyBL

Problema 1

Alta complexidade e demora no tempo de execução para grandes conjuntos de dados

Muitos pares de pivôs analisados

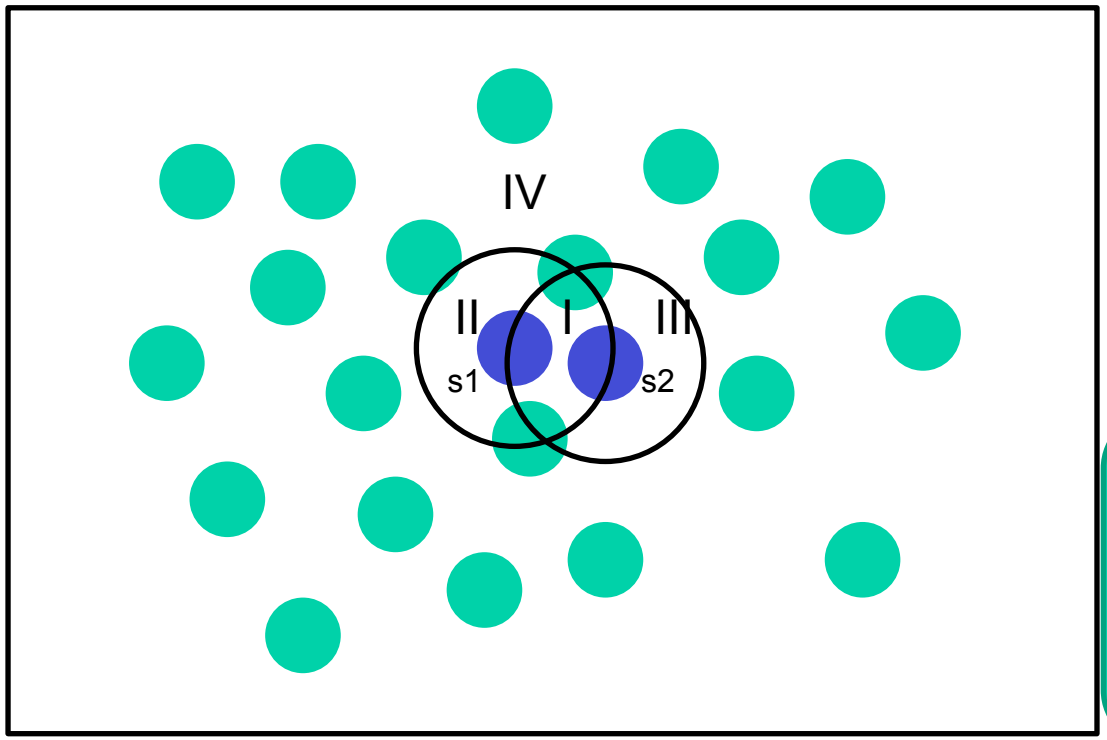


Algoritmo GreedyBL

Problema 2

Pares de pivôs muito próximos

Grande parte dos elementos na região externa da Onion-tree.



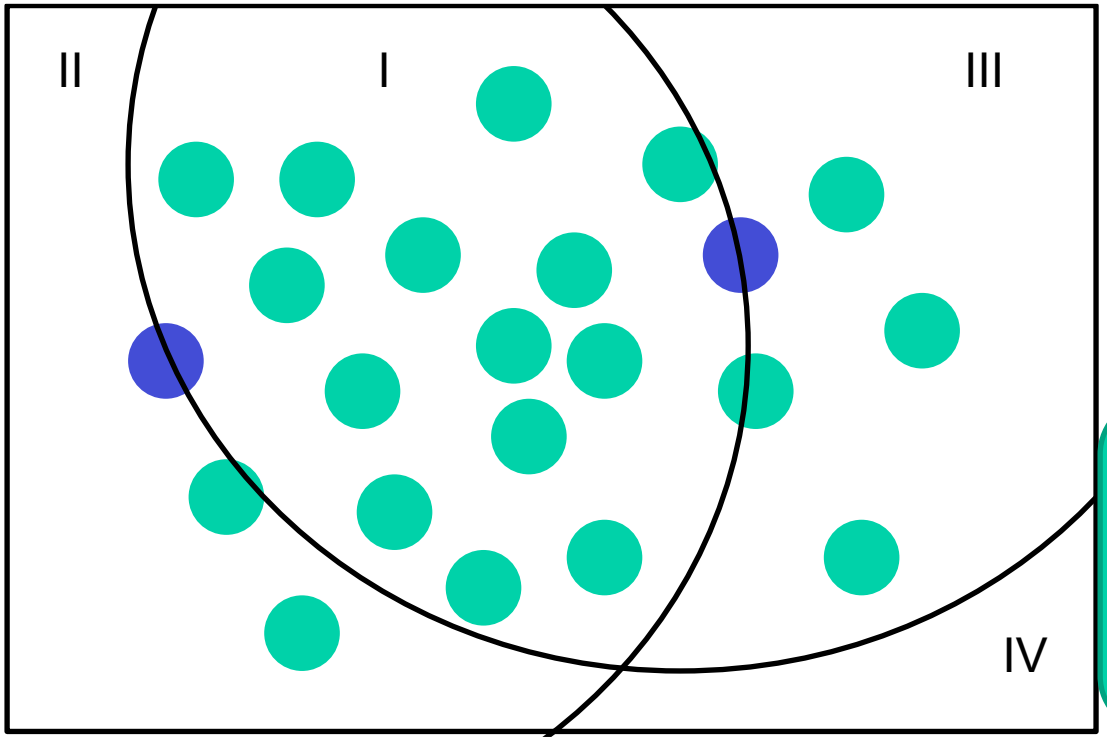
Pivôs: s1, s2
Distribuição
I – 2
II- 0
III- 0
IV- 20

Algoritmo GreedyBL

Problema 3

Pares de pivôs muito distantes

Grande parte dos elementos na região I da Onion-tree.



Pivôs: s2, s12
Distribuição
I – 16
II- 2
III- 4
IV- 0

Proposta de Bulk-Loading

Algoritmos desenvolvidos:

- GreedyBL
- SampleBL
- HeightBL

Algoritmo SampleBL

- Características
  - Top-Down.
  - Introduz a etapa de amostragem.
  - Obtém um índice balanceado.

Algoritmo SampleBL

- Características
  - Top-Down.
  - Introduz a etapa de amostragem.
  - Obtém um índice balanceado.

Objetivo

Construir uma estrutura balanceada

Algoritmo SampleBL

- Características
  - Top-Down.
  - Introduz a etapa de amostragem.
  - Obtém um índice balanceado.

Objetivo

Construir uma estrutura balanceada → Escolher **pivôs** que distribuam os elementos **uniformemente** entre as regiões da Onion-tree

$$ValorEsperado = \lceil (QtdElNivel)/(QtdReg) \rceil$$

$$Objetivo = \Sigma_{r=0}^{r=n} \sqrt{(QtdEl[r] - ValorEsperado)^2}/QtdReg$$

Algoritmo SampleBL

- Características
  - Top-Down.
  - Introduz a etapa de amostragem.
  - Obtém um índice balanceado.

Objetivo

Construir uma estrutura balanceada

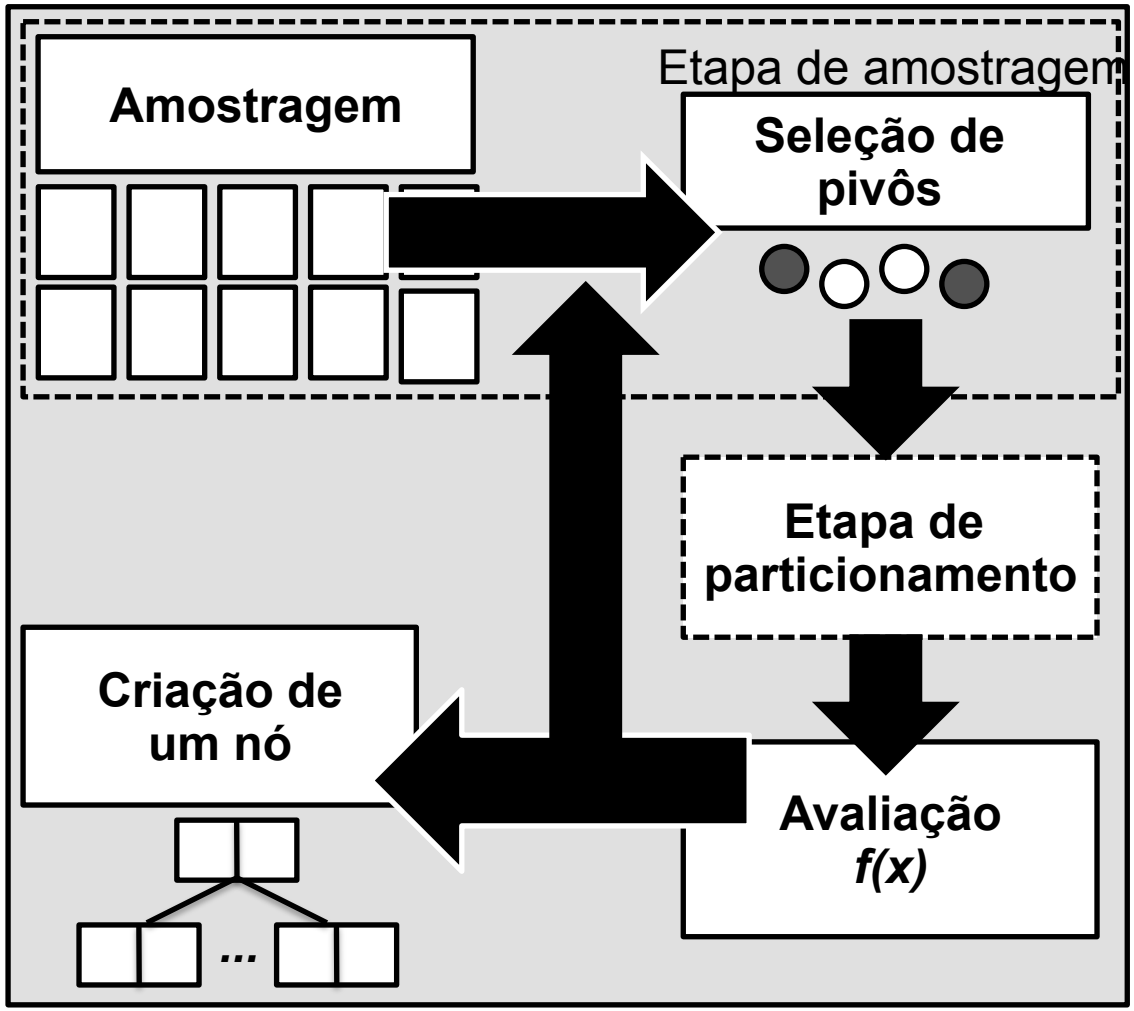
Amostragem →

Escolher **pivôs** que distribuam os elementos **uniformemente** entre as regiões da Onion-tree

$$ValorEsperado = \lceil (QtdElNivel)/(QtdReg) \rceil$$

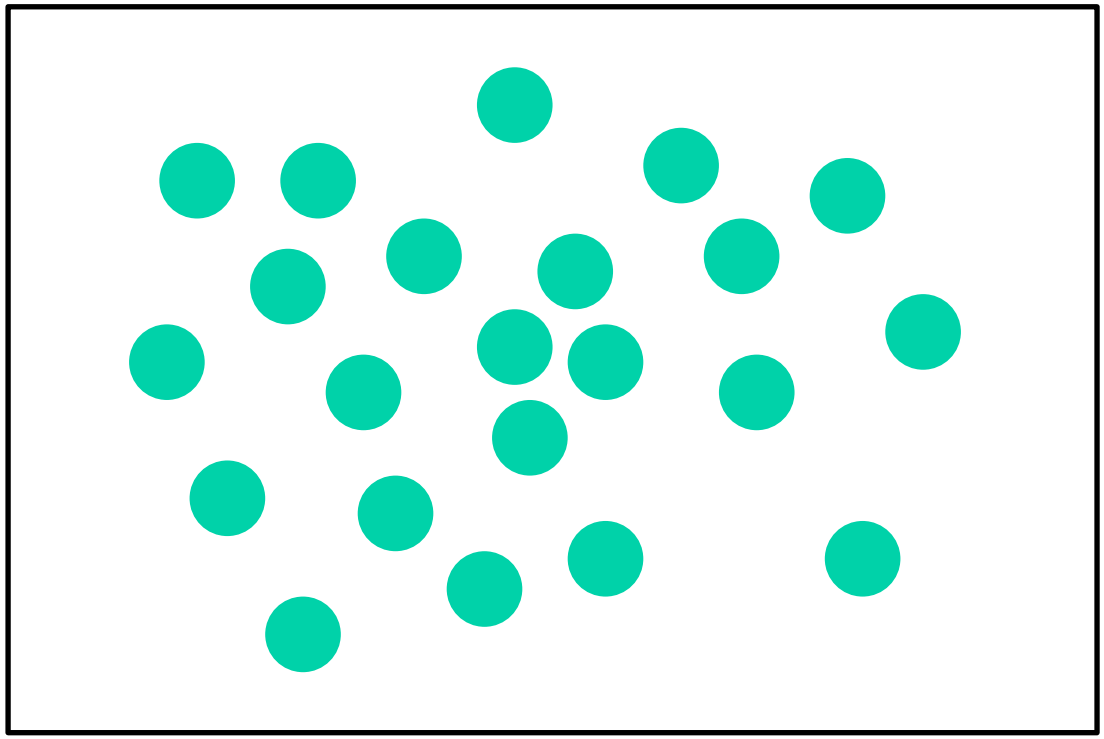
$$Objetivo = \Sigma_{r=0}^{r=n} \sqrt{(QtdEl[r] - ValorEsperado)^2}/QtdReg$$

Algoritmo SampleBL



Algoritmo SampleBL

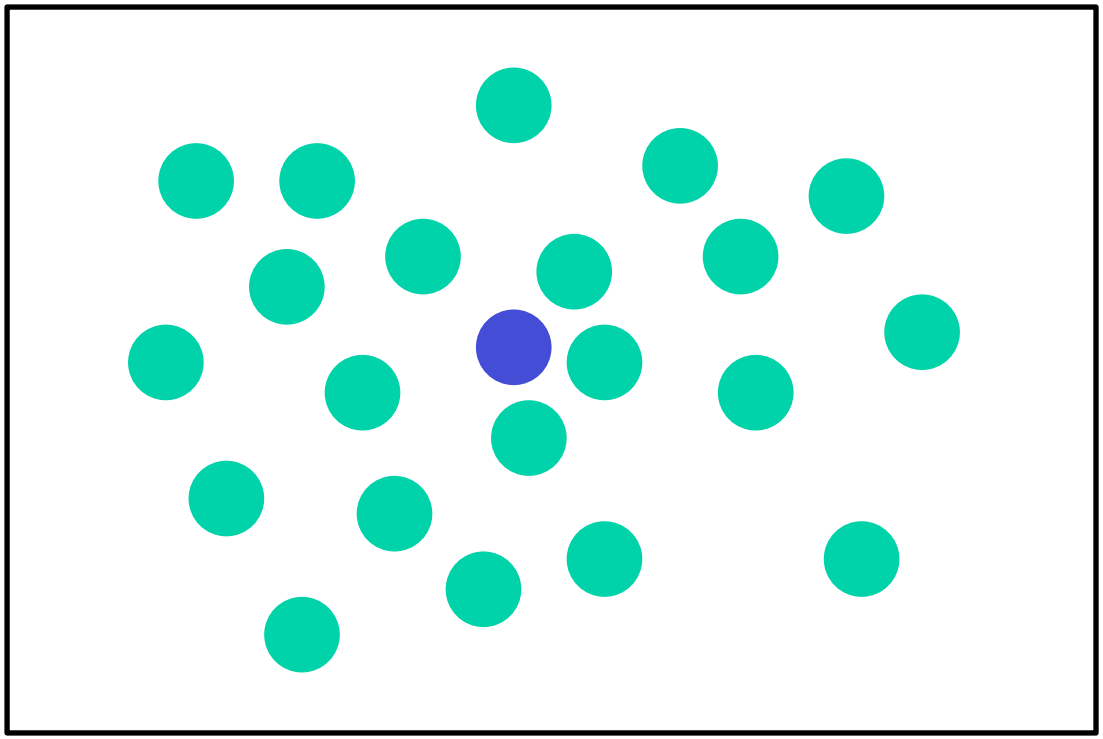
Conjunto de dados inicial



Algoritmo SampleBL

Etapa de Amostragem

1. Determinar o elemento medóide



Algoritmo SampleBL

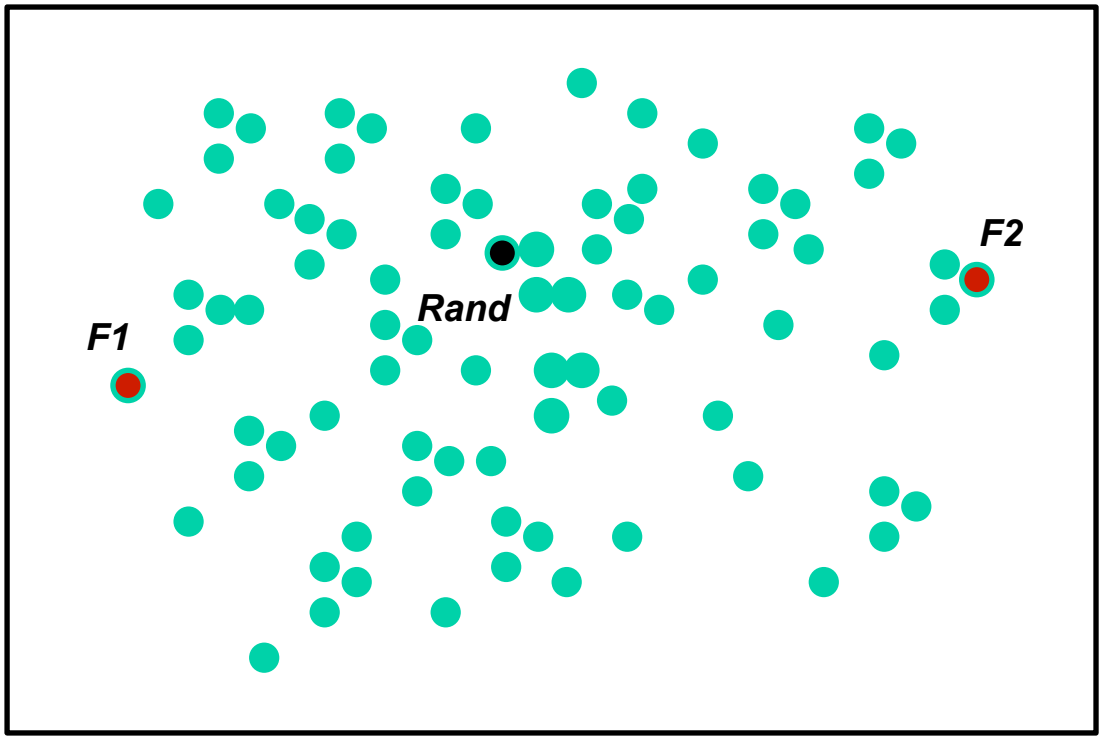
Medóide aproximado

- Técnica para determinar **medóide** de um conjunto de dados é **quadrática**.
- Desenvolvido um método para determinar o medóide **aproximadamente**, com complexidade *n*log(n)*.

Algoritmo SampleBL

Medóide Aproximado

1. *Rand* é escolhido aleatoriamente.
2. F1 é o elemento mais distante de R.
3. F2 é o elemento mais distante de F1.

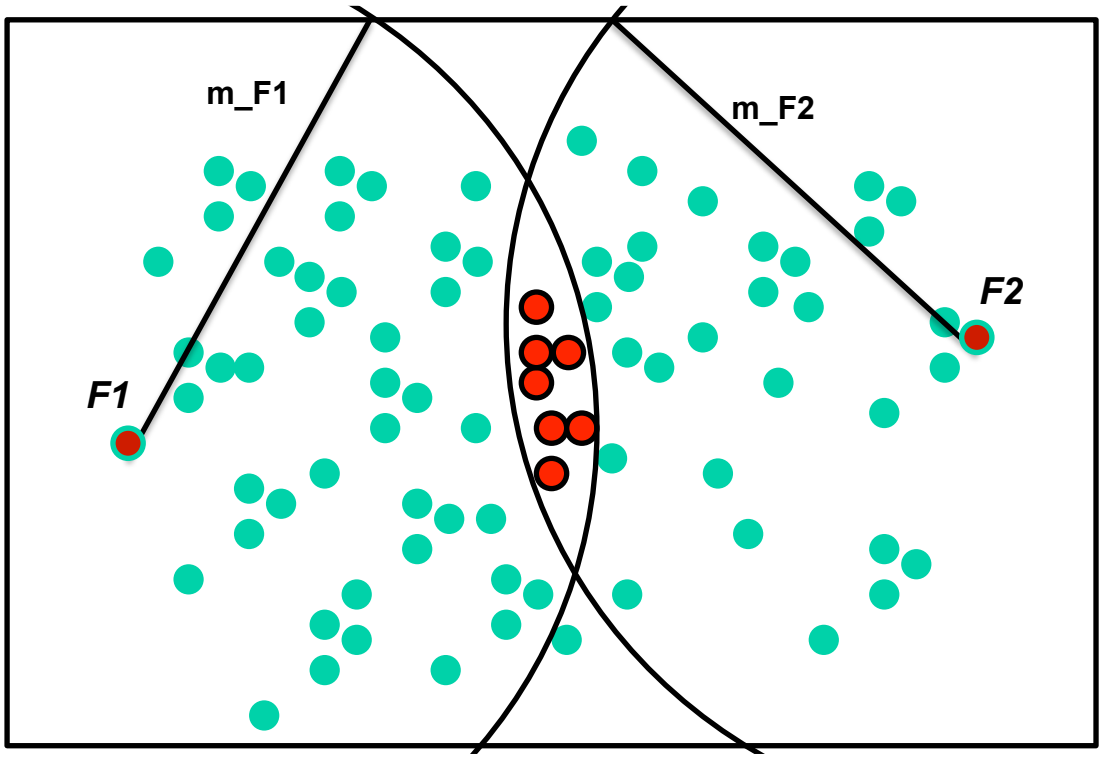


Algoritmo SampleBL

Medóide Aproximado

1. **m_f1** é determinada a mediana das distâncias de **F1** aos demais elementos
2. **m_f2** é determinada a mediana das distâncias de **F2** aos demais elementos

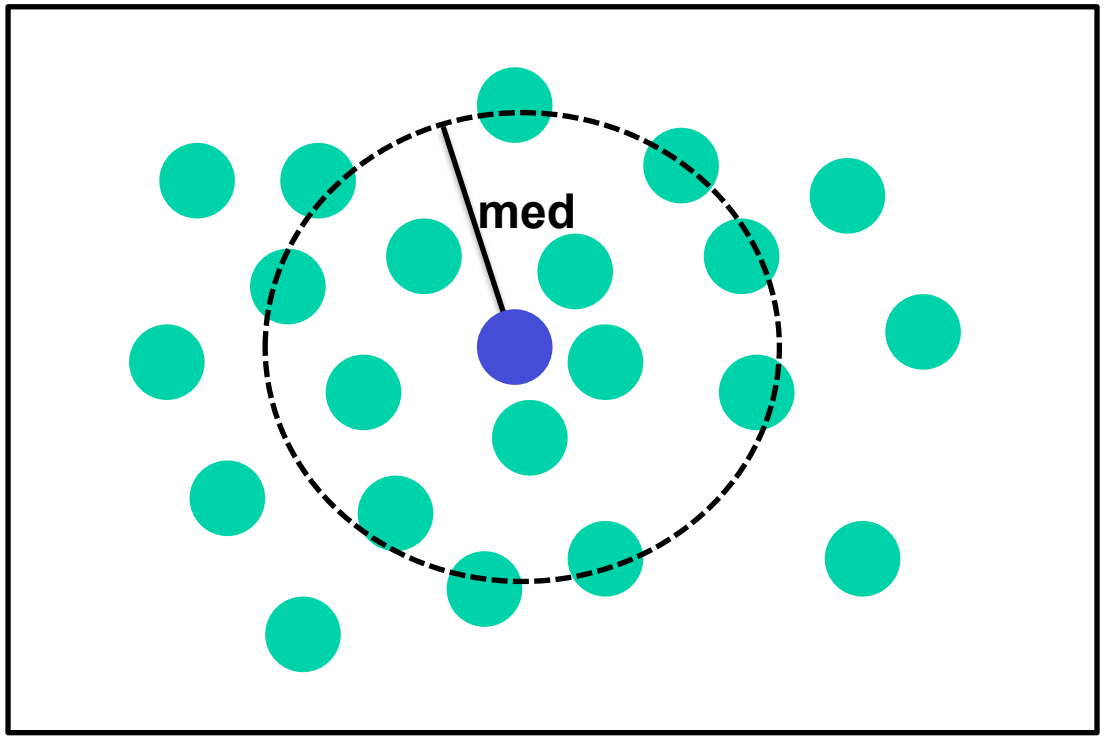
m_f1 e **m_f2** são incrementadas até que elementos candidatos



Algoritmo SampleBL

Etapa de Amostragem

2. Calcular a distância mediana entre o medóide e os demais elementos

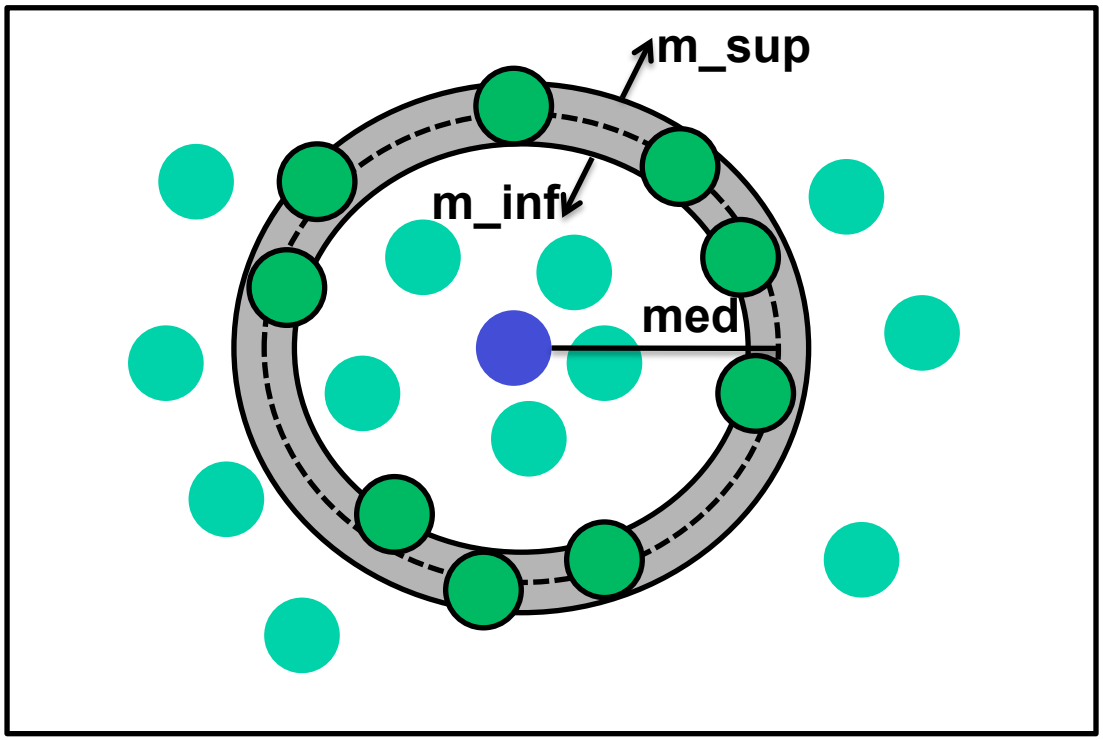


r = mediana

Algoritmo SampleBL

Etapa de Amostragem

3. Definir um ‘anel’ ao redor das amostras que serão coletadas



Quantidade de amostras:
2 < N < *dimensionalidade*

$$Incremento = QtdElementos/(Dimensionalidade + QtdElementos)$$

Algoritmo SampleBL

Etapa de Filtragem

Filtrando amostras

Elementos similares

- Introduzem aproximadamente a **mesma contribuição** como pivôs quando testados.
- Podem ser reduzidos a **somente um** membro do conjunto.
- Retornados da **Etapa de Amostragem**.
- Possuem entre si **distância** menor do que δ

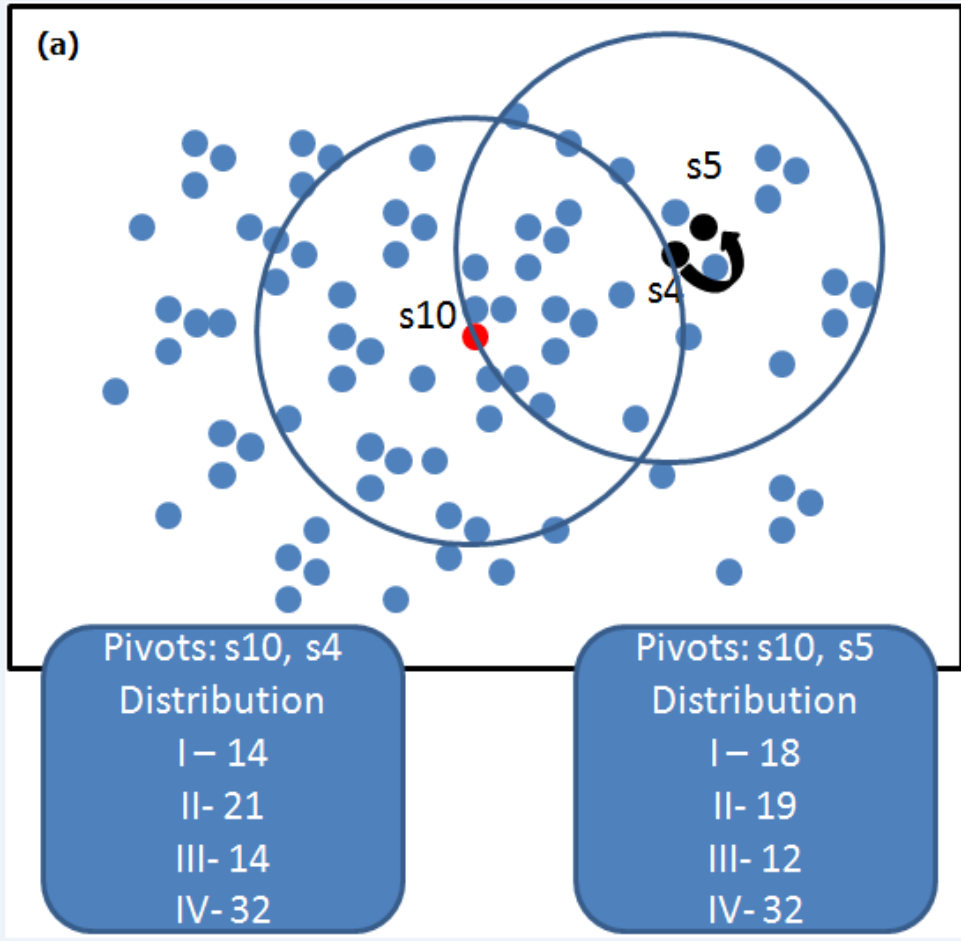
$$\delta = QtdElementos/(Dimensionalidade + QtdElementos)$$

Algoritmo SampleBL

Etapa de Filtragem

A troca entre elementos similares não causa grande mudança na estrutura resultante.

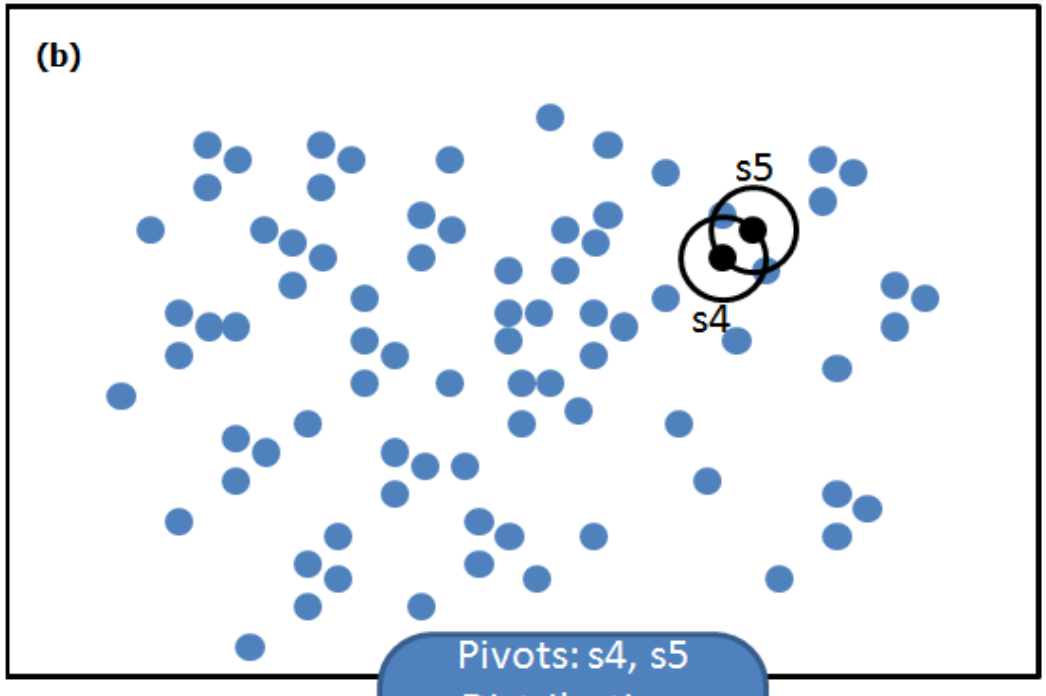
A divisão dos elementos pelas regiões definidas no espaço métrico pouco se altera.



Algoritmo SampleBL

Etapa de Filtragem

Quando testados como pares de pivôs ente si, resultam em má divisão do espaço métrico

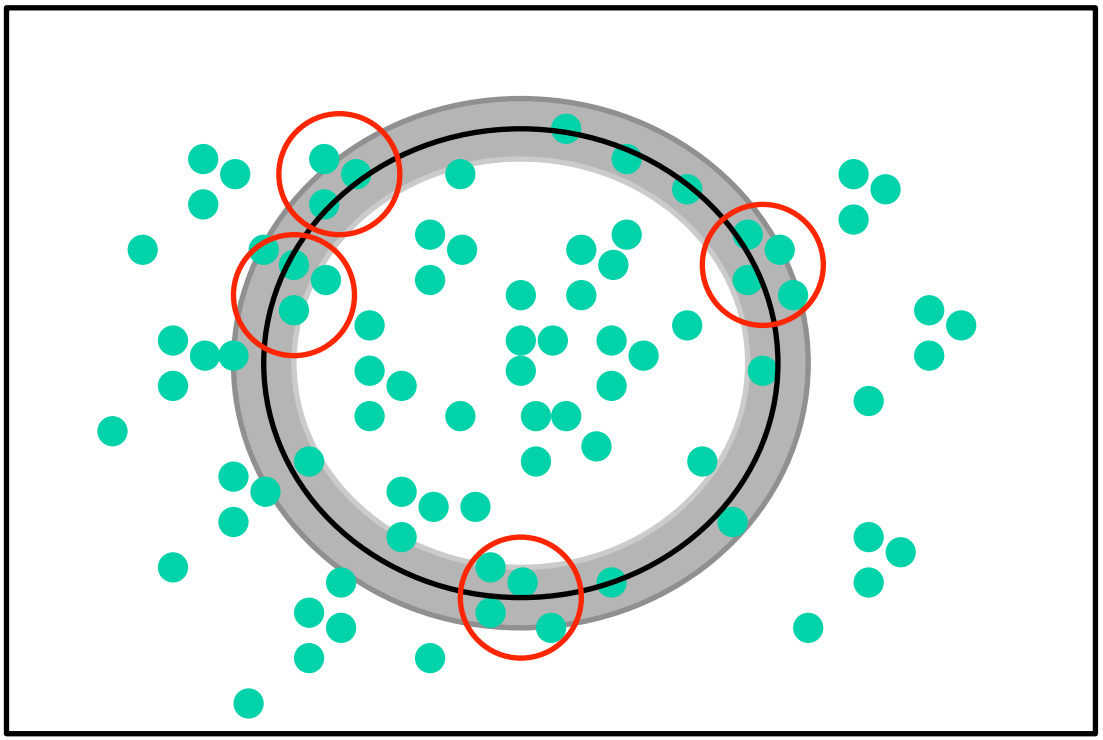


Algoritmo SampleBL

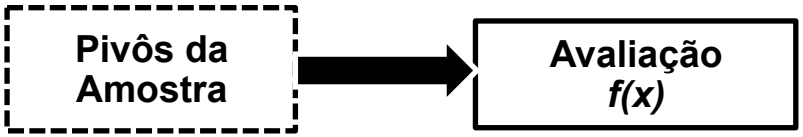
Etapa de Filtragem

Remoção dos n-1 elementos de cada conjunto de elementos similares.

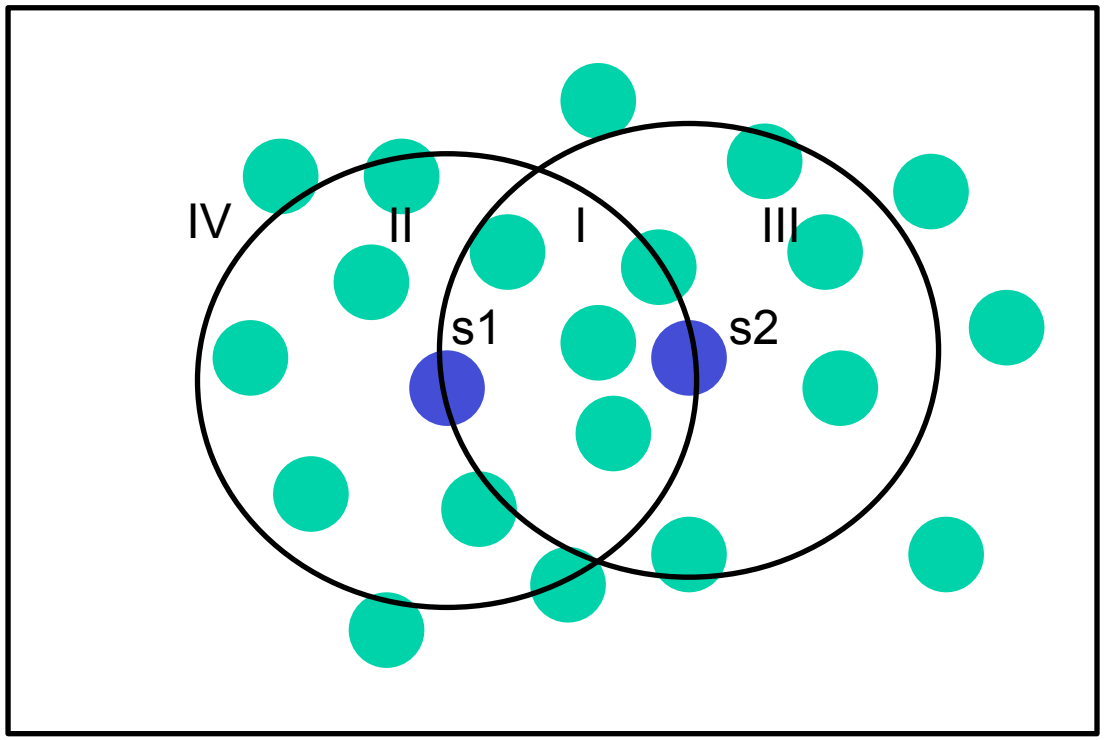
Redução do número de amostras que irão ser checadas pelo algoritmo SampleBL.



Algoritmo SampleBL



4. Para cada par de pivôs da amostra, analizar a distribuição dos demais elementos entre as regiões determinadas e avaliar com uma função objetivo

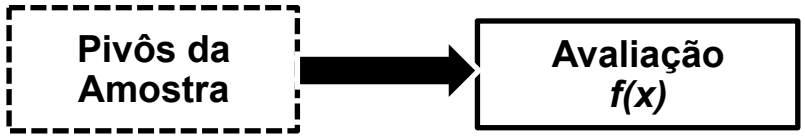


Pivôs: s1, s2
Distribuição
I – 4
II- 5
III- 4
IV- 7

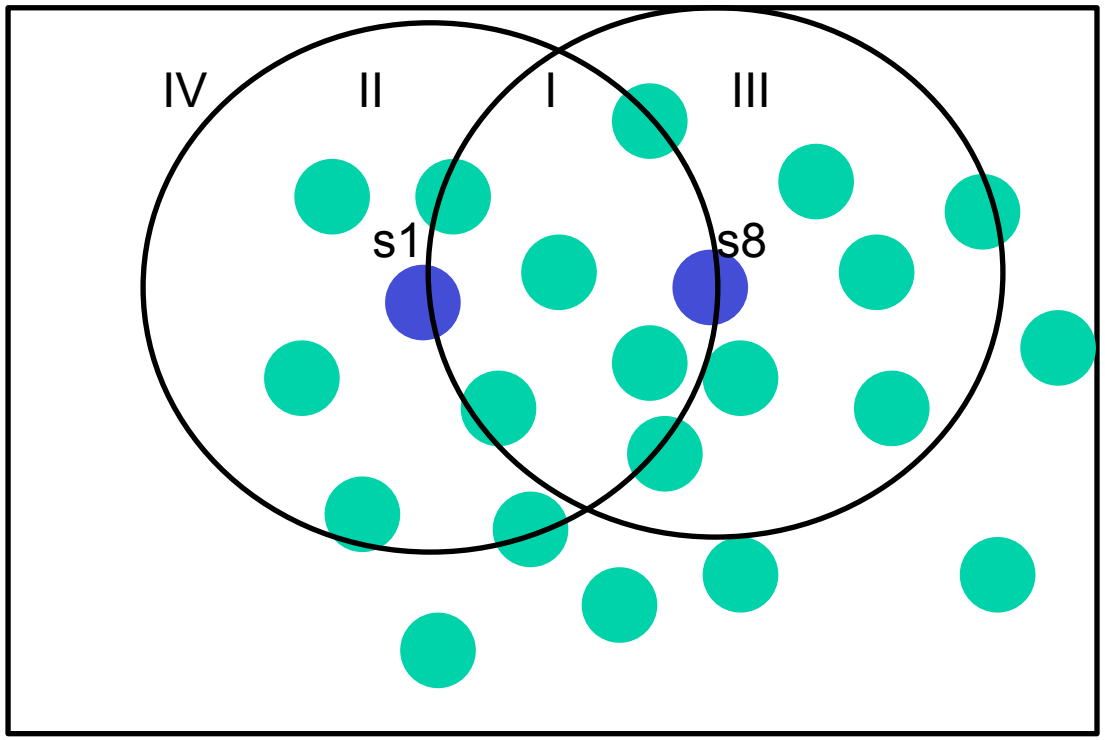
Particionamento

Avaliação

Algoritmo SampleBL



4. Para cada par de pivôs da amostra, analisar a distribuição dos demais elementos entre as regiões determinadas e avaliar com uma função objetivo



Pivôs: s1, s8
Distribuição
I – 3
II- 4
III- 7
IV- 6

Particionamento

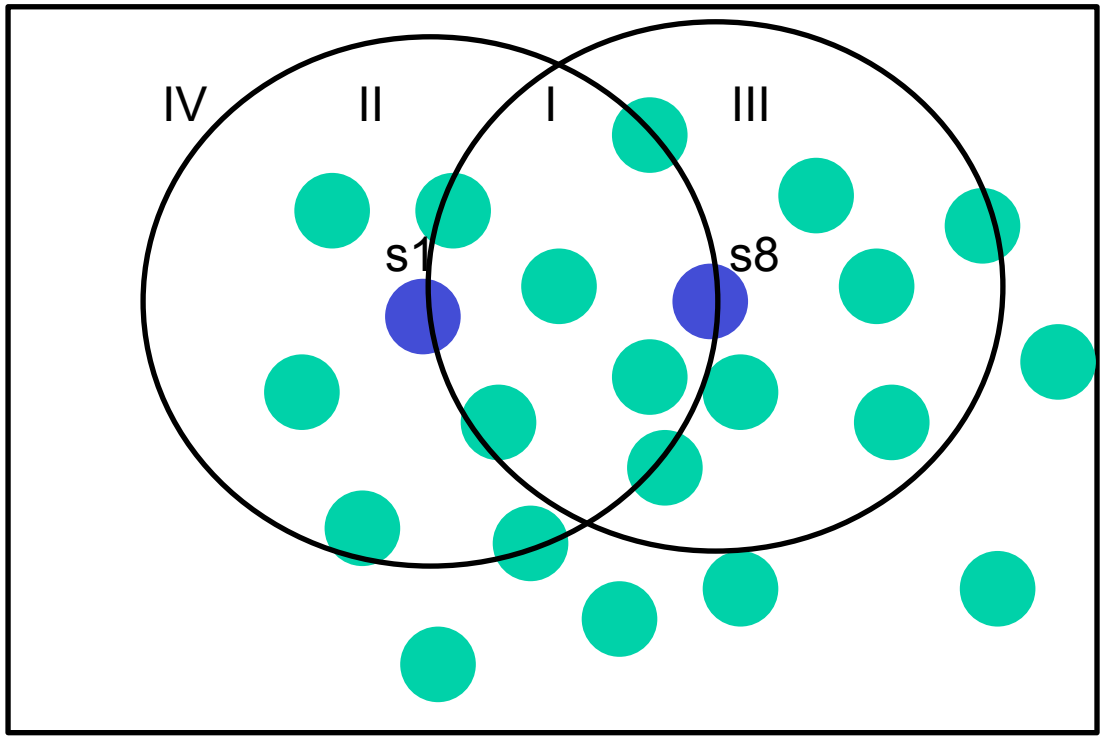
↓

Avaliação

Algoritmo SampleBL

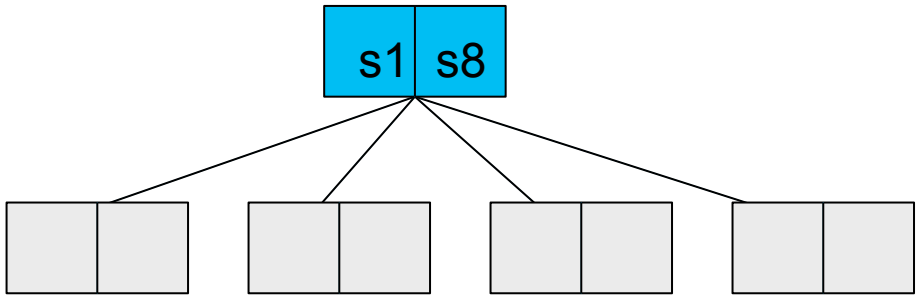
Criação de um nó

5. Escolher o par de pivôs que minimiza a função objetivo



Pivôs: s1, s8
Distribuição
I – 3
II- 4
III- 7
IV- 6

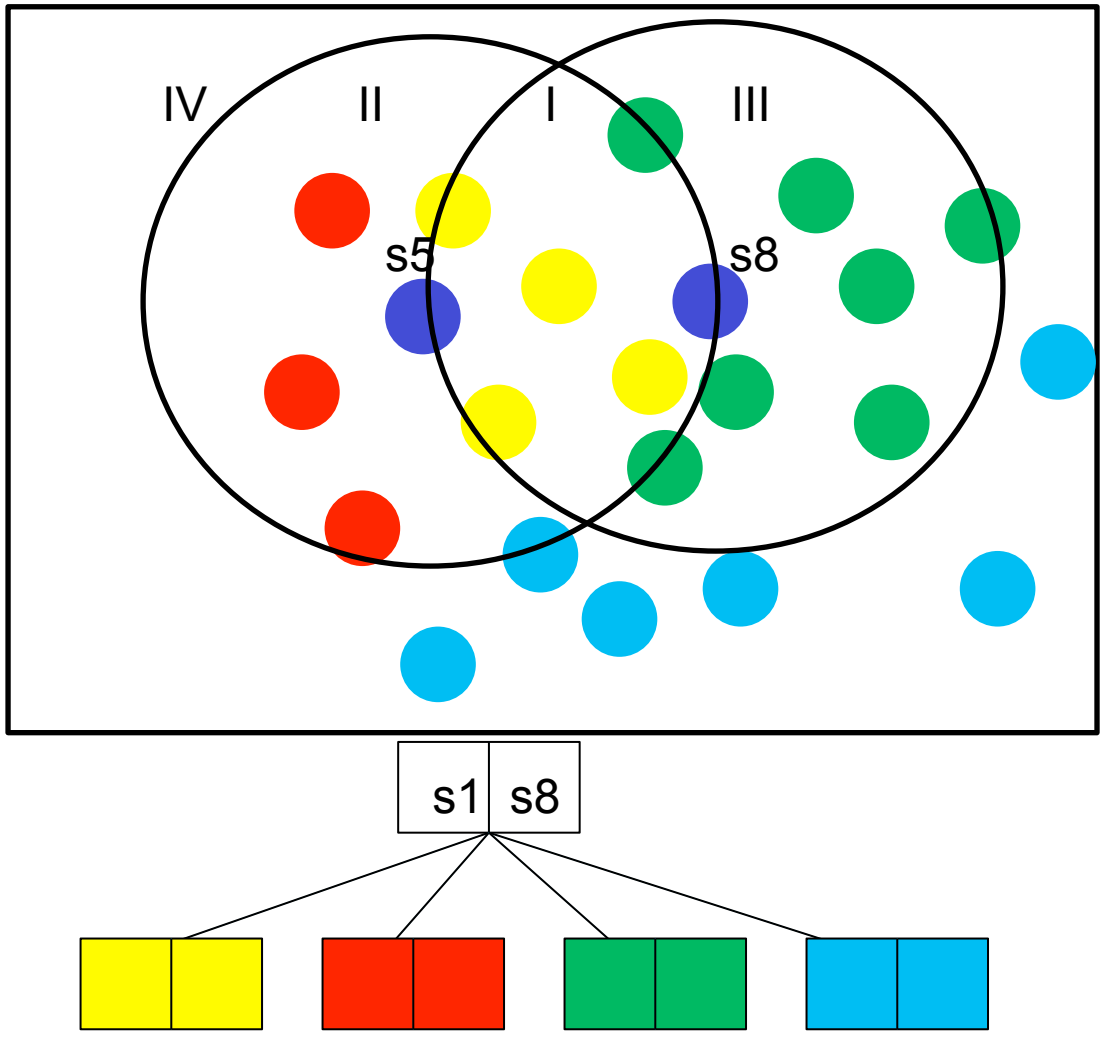
Particionamento



Avaliação

Algoritmo SampleBL

6. Realizar chamadas recursivas para cada região, usando como dados de entrada os elementos associados a cada uma delas.



Recursão

Enquanto Região possuir mais que dois elementos

Algoritmo SampleBL

V-Onion-tree

Particionamento da V-Onion-tree:

- Número de partições de um nó = (raio do nó pai)/(raio do nó filho)
- Quantidade de expansões pode ser elevada

Considere

- Nó pai cujo raio seja 1.
- Nó filho cujo raio tenha valor 0,001.
- Resulta em 1000 expansões

Algoritmo SampleBL

V-Onion-tree

Dois novos métodos de particionamento:

- **Expansão baseada em quantidade**
  - Número de expansões não pode exceder a quantidade *n* de elementos a ser inserida em um dado nível do índice

$$Numero\ de\ Regiões = n/2$$

$$ExpMax = (n/2 - 4)/3$$

- **Expansão baseada em média**
  - Assume que os *n* elementos a serem inseridos em um nível irão ser armazenados nos níveis inferiores do índice
  - *M1: distância média do pivô s1 aos demais elementos*
  - *M2: distância média do pivô s2 aos demais elementos*
  - *Número de expansões deve apresentar maior raio aproximadamente (m1+m2)/2*

Proposta de Bulk-Loading

Algoritmos desenvolvidos:

- GreedyBL
- SampleBL
- HeightBL

Algoritmo HeightBL

- Características
  - Top-Down.
  - Baseado em estimativa da altura do índice.
  - Obter a Onion-tree tal que não ultrapasse a altura estimada.

Algoritmo HeightBL

- Características
  - Top-Down.
  - Baseado em estimativa da altura do índice.
  - Obter a Onion-tree tal que não ultrapasse a altura estimada.

Objetivo

Construir uma estrutura preenchida

Algoritmo HeightBL

- Características
  - Top-Down.
  - Baseado em estimativa da altura do índice.
  - Obter a Onion-tree tal que não ultrapasse a altura estimada.

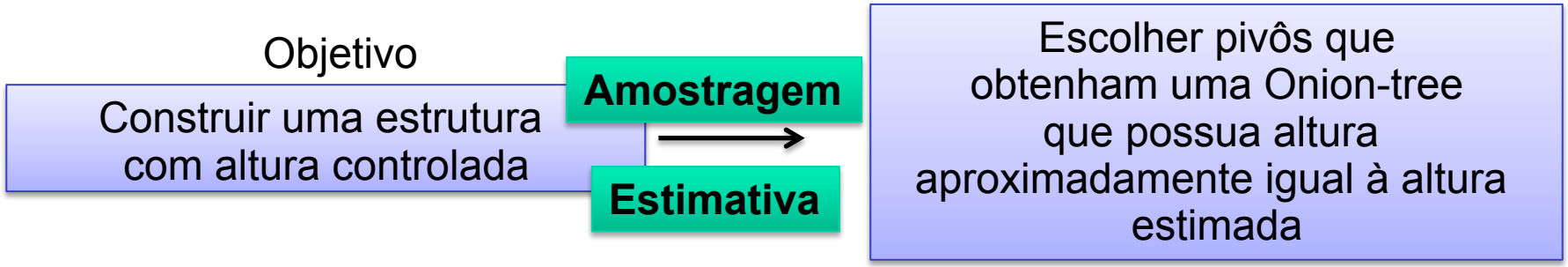
Objetivo

Construir uma estrutura preenchida → Escolher pivôs que obtenham uma Onion-tree que possua altura aproximadamente igual à altura estimada

$$AlturaEstimada = \lceil \log_R^{(\frac{n*(R-1)}{2}+1)} - 1 \rceil$$

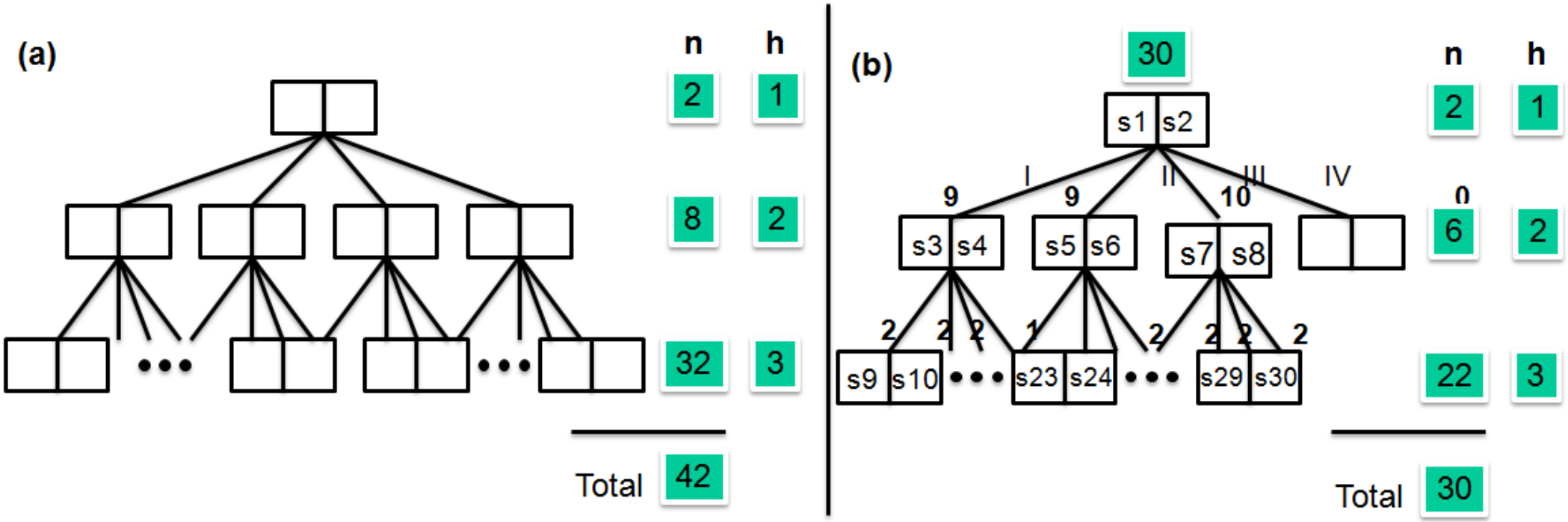
Algoritmo HeightBL

- Características
  - Top-Down.
  - Baseado em estimativa da altura do índice.
  - Obter a Onion-tree tal que não ultrapasse a altura estimada.



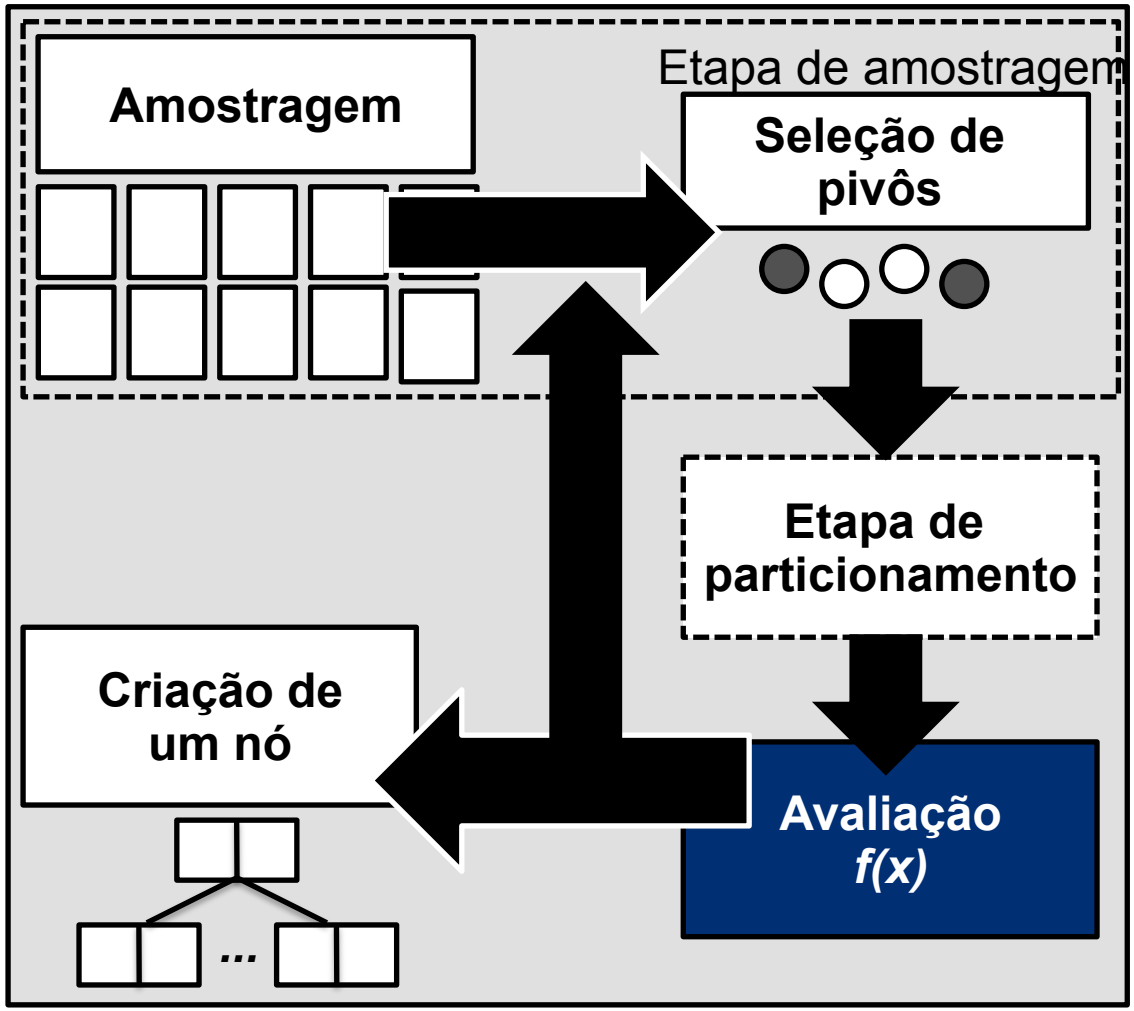
$$AlturaEstimada = \lceil \log_R^{(\frac{n*(R-1)}{2}+1)} - 1 \rceil$$

Algoritmo HeightBL



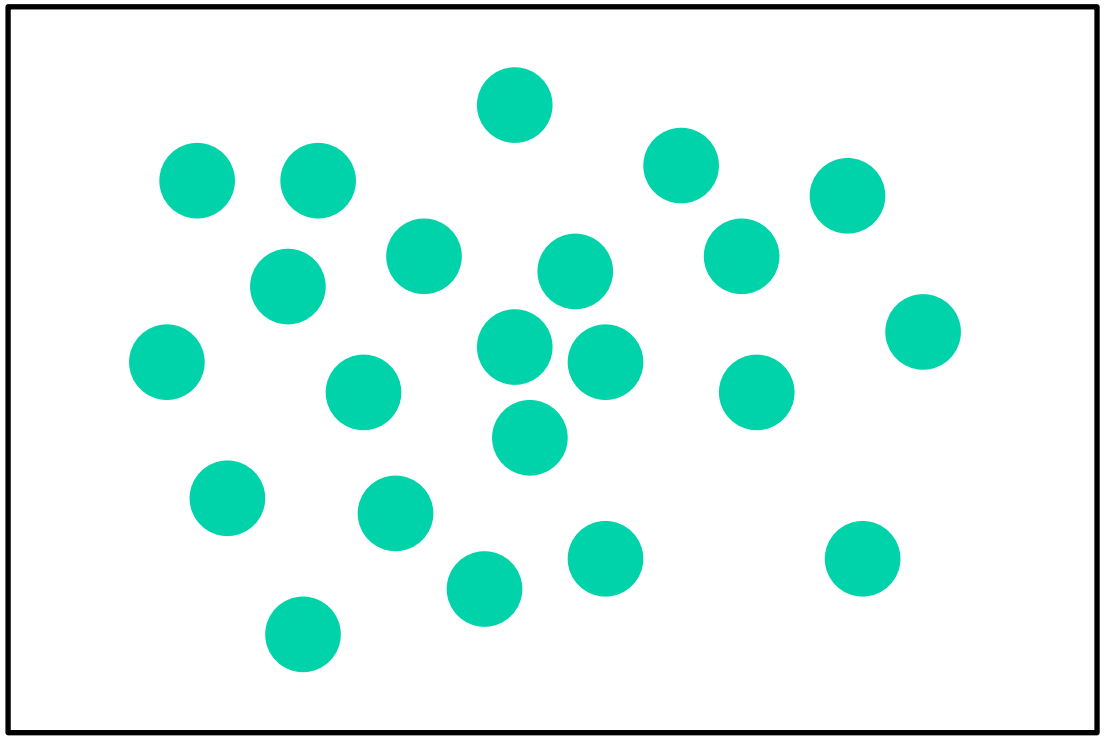
n – número de elementos
h- altura

Algoritmo HeightBL



Algoritmo HeightBL

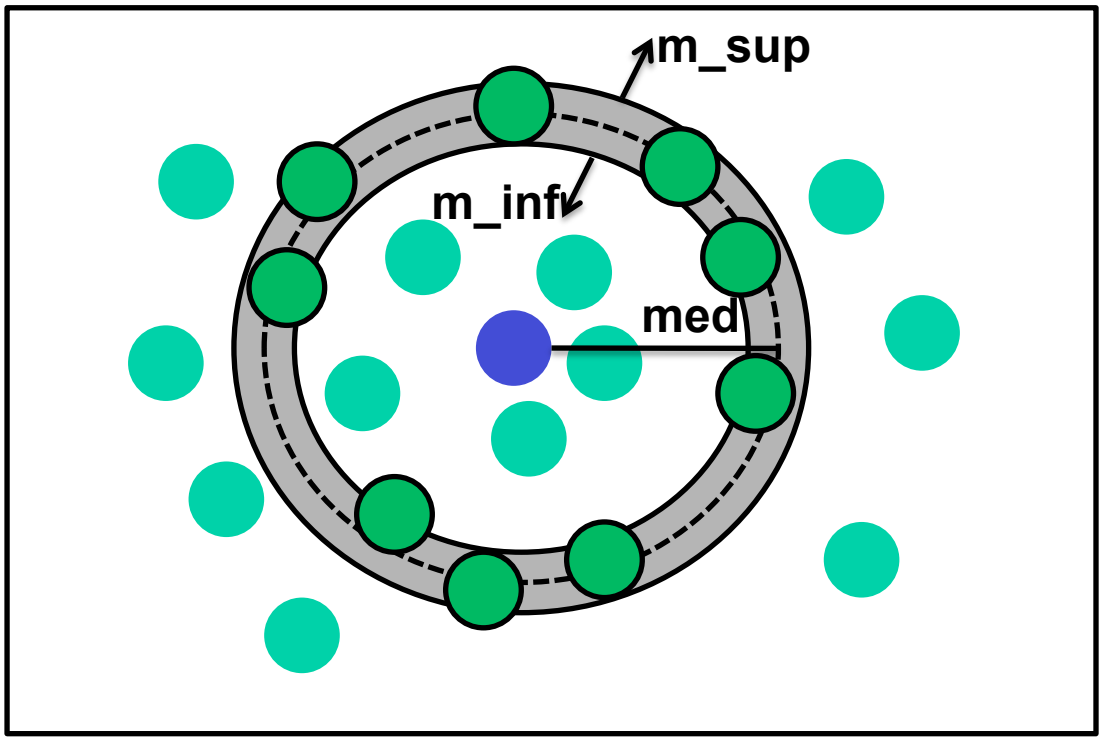
Conjunto de dados inicial



N = 22
Expansões: 0
Altura estimada: 3

Algoritmo HeightBL

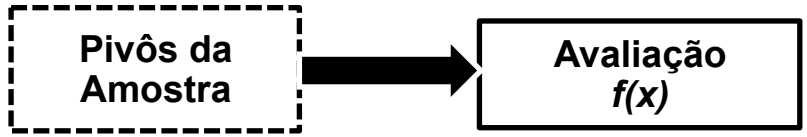
Etapa de Amostragem

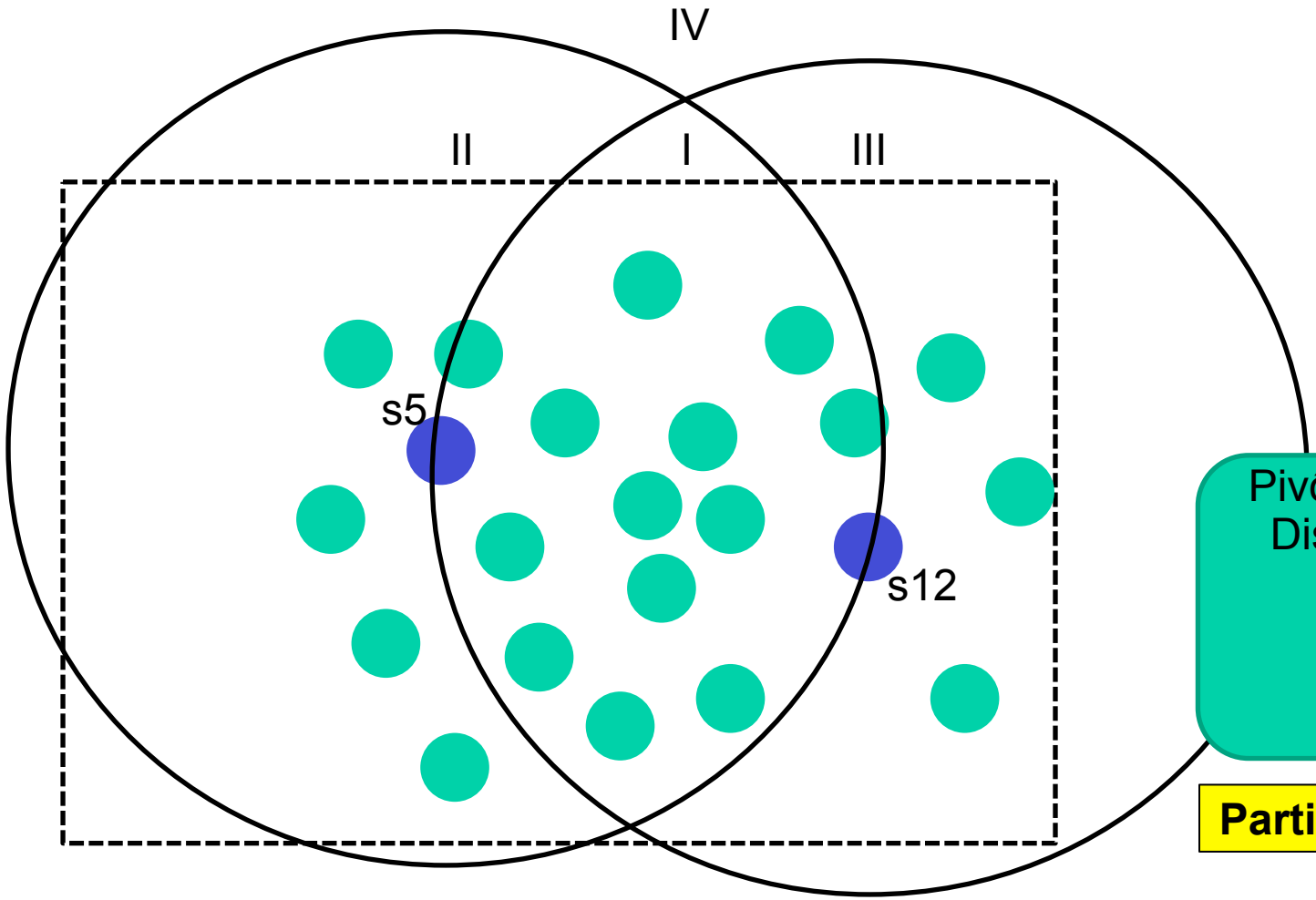


Quantidade de amostras:
2 < N < *dimensionalidade*

$$Incremento = QtdElementos/(Dimensionalidade + QtdElementos)$$

Algoritmo HeightBL





Pivôs: s5, s12
Distribuição
I – 13
II- 4
III- 3
IV- 0

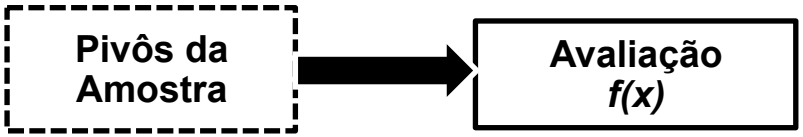
Particionamento

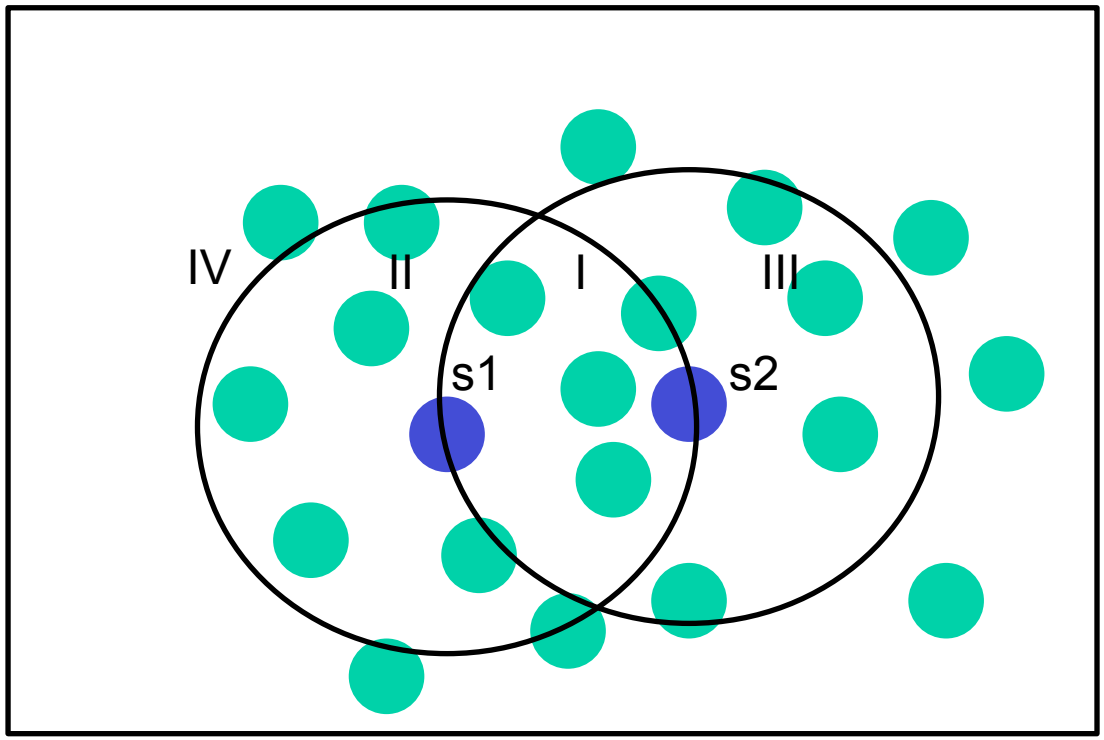
Avaliação

N = 22
Expansões: 0
Altura estimada: 3

Altura Estimada I: 3
Altura Estimada II: 2
Altura Estimada III: 2
Altura Estimada IV: 0

Algoritmo HeightBL





Pivôs: s1, s2
Distribuição
I – 4
II- 5
III- 4
IV- 7

Particionamento

Avaliação

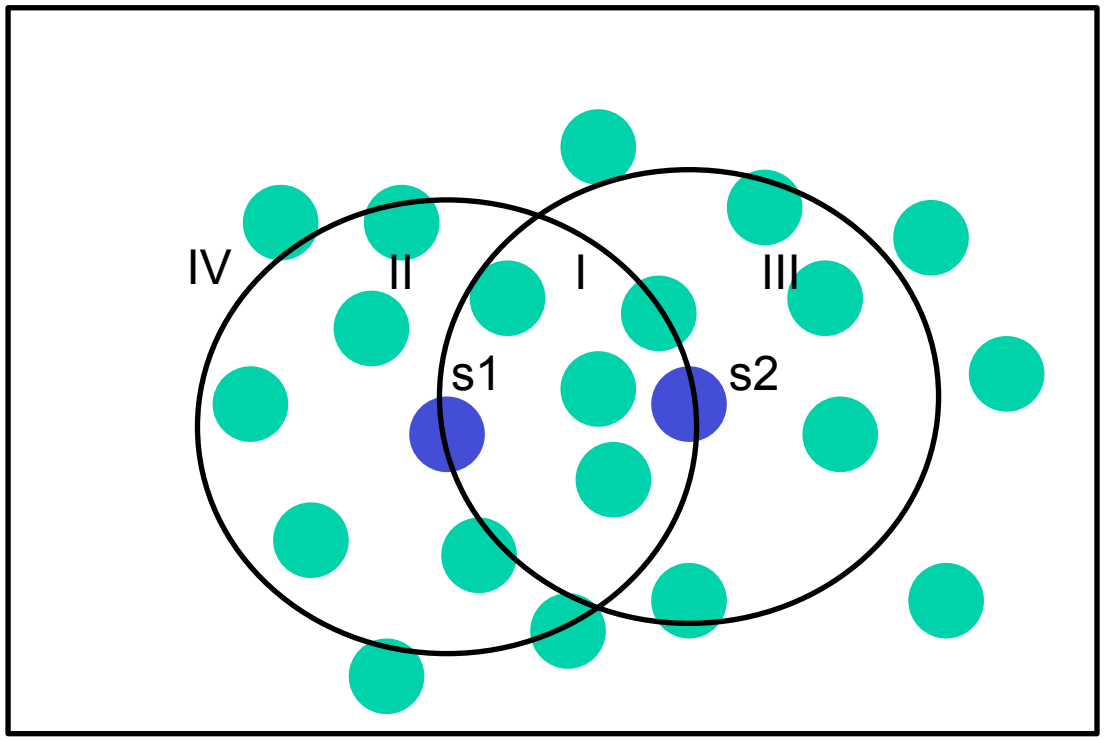
N = 22
Expansões: 0
Altura estimada: 3

Altura Estimada I: 2
Altura Estimada II: 2
Altura Estimada III: 2
Altura Estimada IV: 2

Algoritmo HeightBL

Criação de um nó

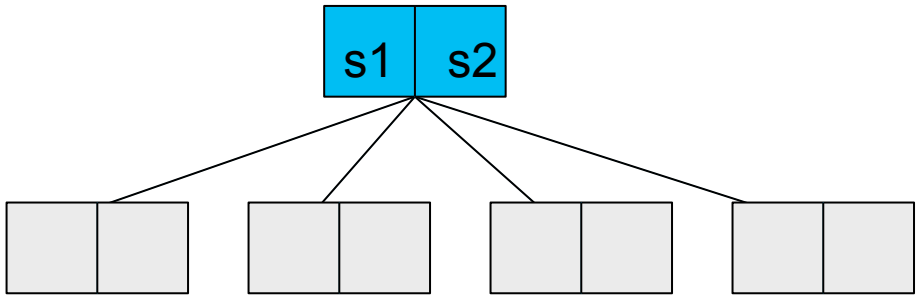
5. Escolher o par de pivôs que minimiza a função objetivo



Pivôs: s1, s2
Distribuição
I – 4
II- 5
III- 4
IV- 7

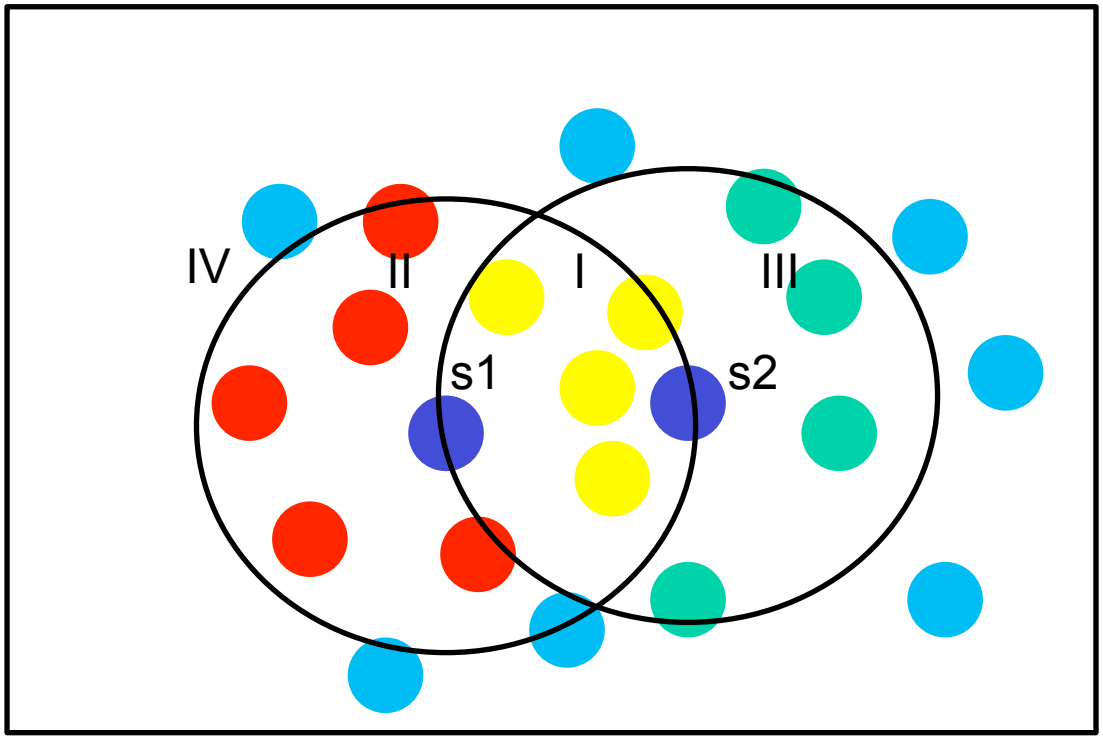
Particionamento

Avaliação



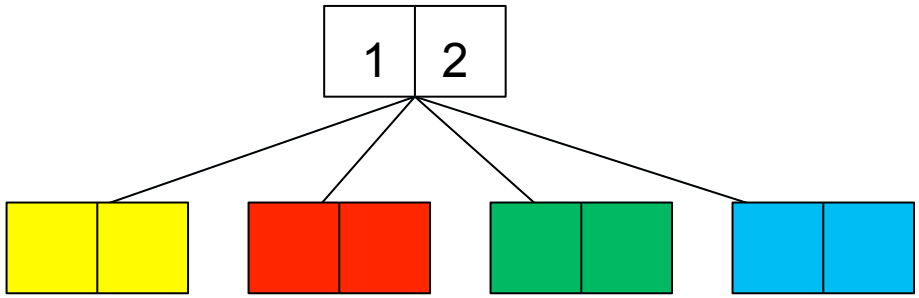
Algoritmo HeightBL

Recursão



Recursão

Enquanto Região possuir mais que dois elementos



Roteiro

Fundamentação Teórica

Trabalhos Correlatos

Proposta

Validação

Conclusão

Validação

Conjunto de Dados	Número de Elementos	Dimensionalidade	Métrica e Expansões F-Onion
Color Histograms	68.025	32	L_2 7 Expansões
Ozone	2.536	73	*Dynamic time warping* 7 Expansões
KDD Cup 2008	102.240	117	L_2 11 Expansões

Comparação

Algoritmo de bulk-loading para F-Onion-tree e V-Onion-tree

Inserção um-a-um da F-Onion-tree e da V-Onion-tree

Validação

Medidas

- Tempo de construção
- Número de cálculos de distância para construção

- Tamanho do índice

- Tempo de consulta
- Número de cálculos de distância na consulta

- 500 Consultas
  - 500 elementos como centro de consultas

- Resultados destacados:
  - SampleBL
  - HeightBL

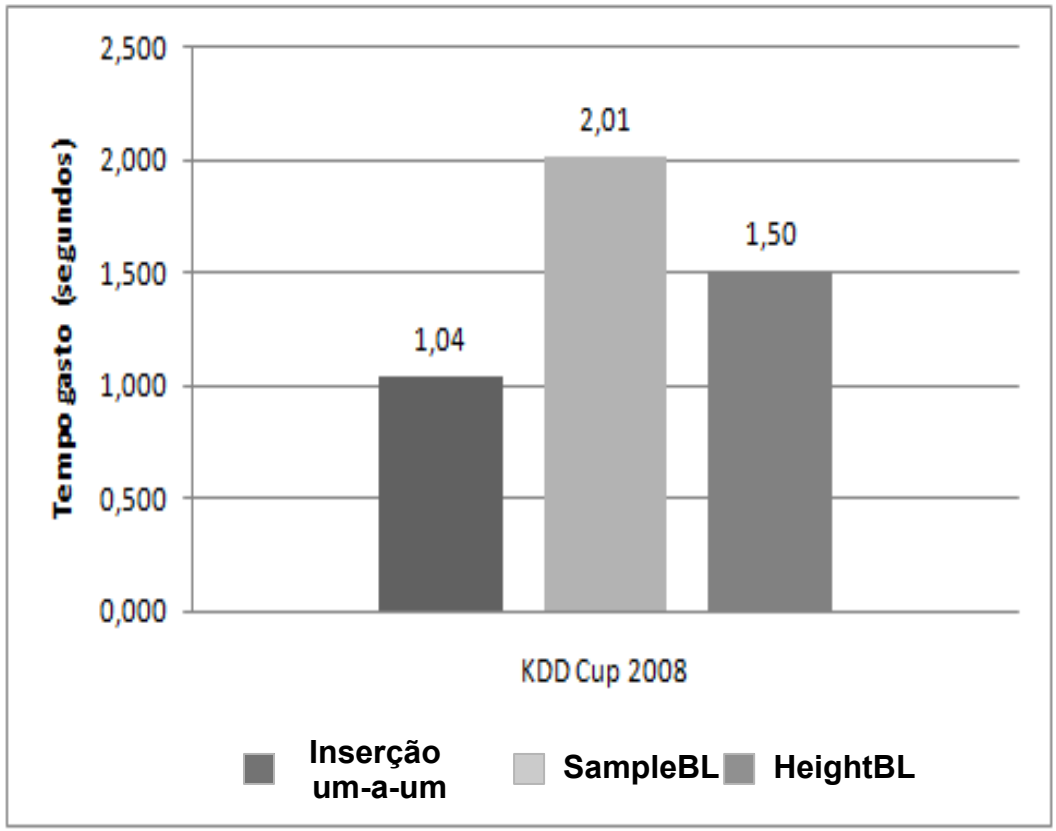
Range	**Configuração**	**Resultados**
1% a 10%		**Gerais**
K	Intel Core i7 2.67GHz	**F-Onion-tree**
2 a 20	Memória RAM 12 Gb	**V-Onion-tree**
	Disco Rígido de 1 TB	

Validação

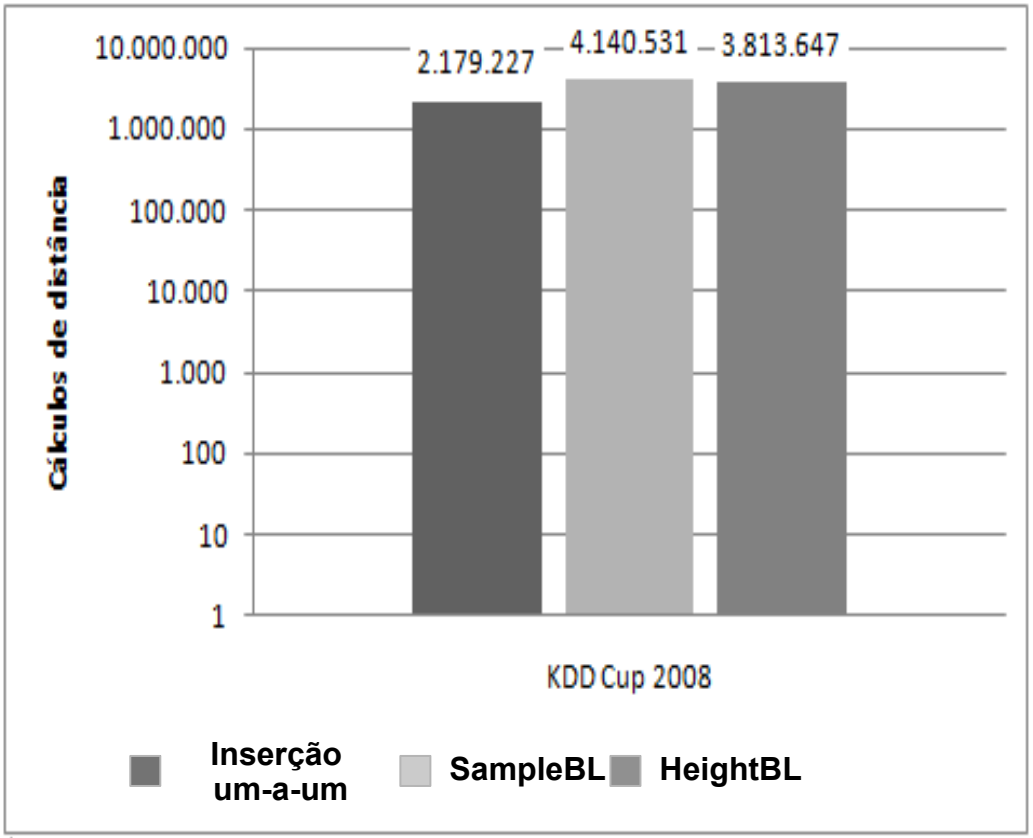
Resultados Gerais

Construção do índice

- Inserção um-a-um é de 5% a 87% mais rápida
- Inserção um-a-um realiza 29% a 55% menos cálculos de distância



Tempo gasto de construção do índice



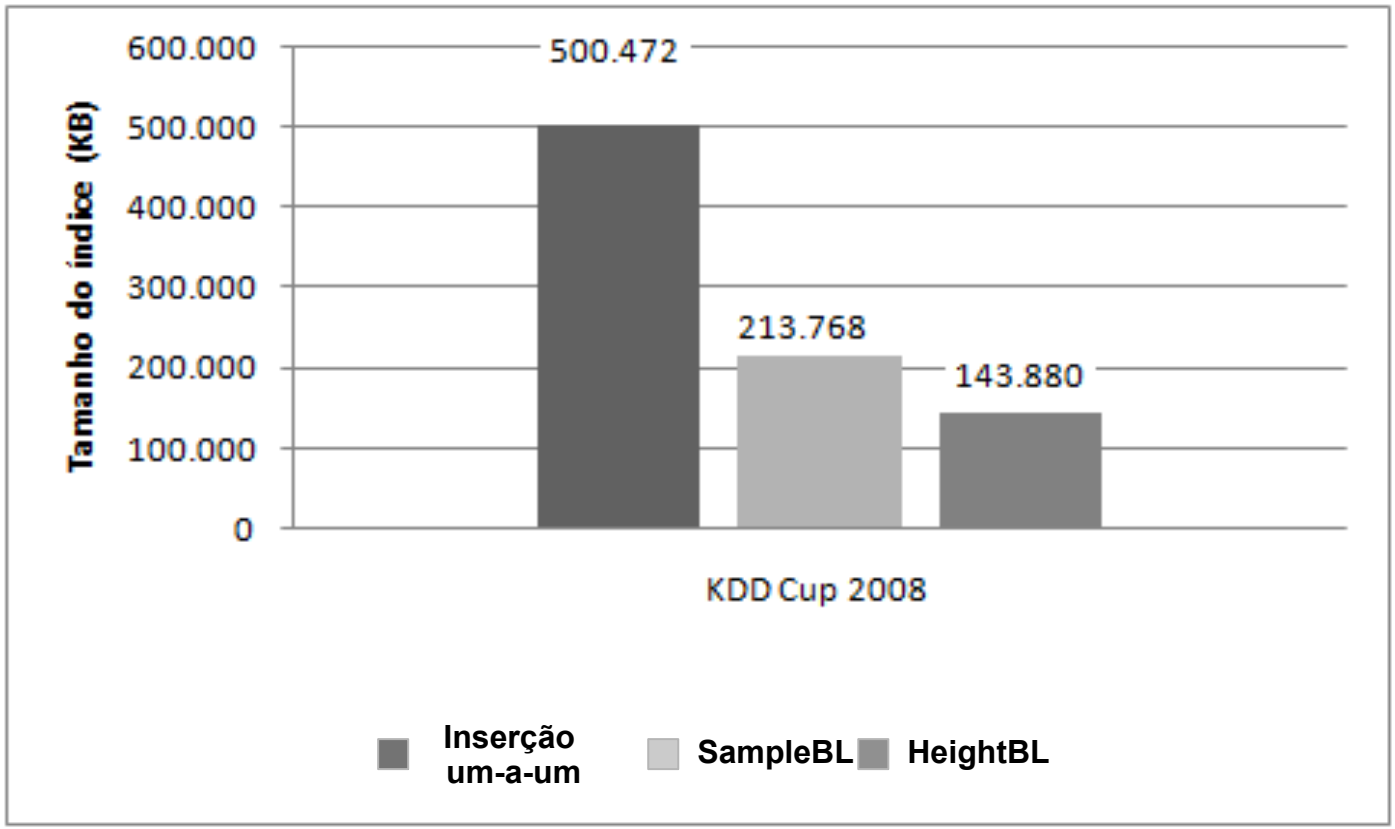
Número de cálculos de distância para construção do índice

Validação

Resultados Gerais

Tamanho do índice

- Algoritmos de *bulk-loading* garantem de 9% a 88% de economia de espaço de armazenamento do tamanho do índice



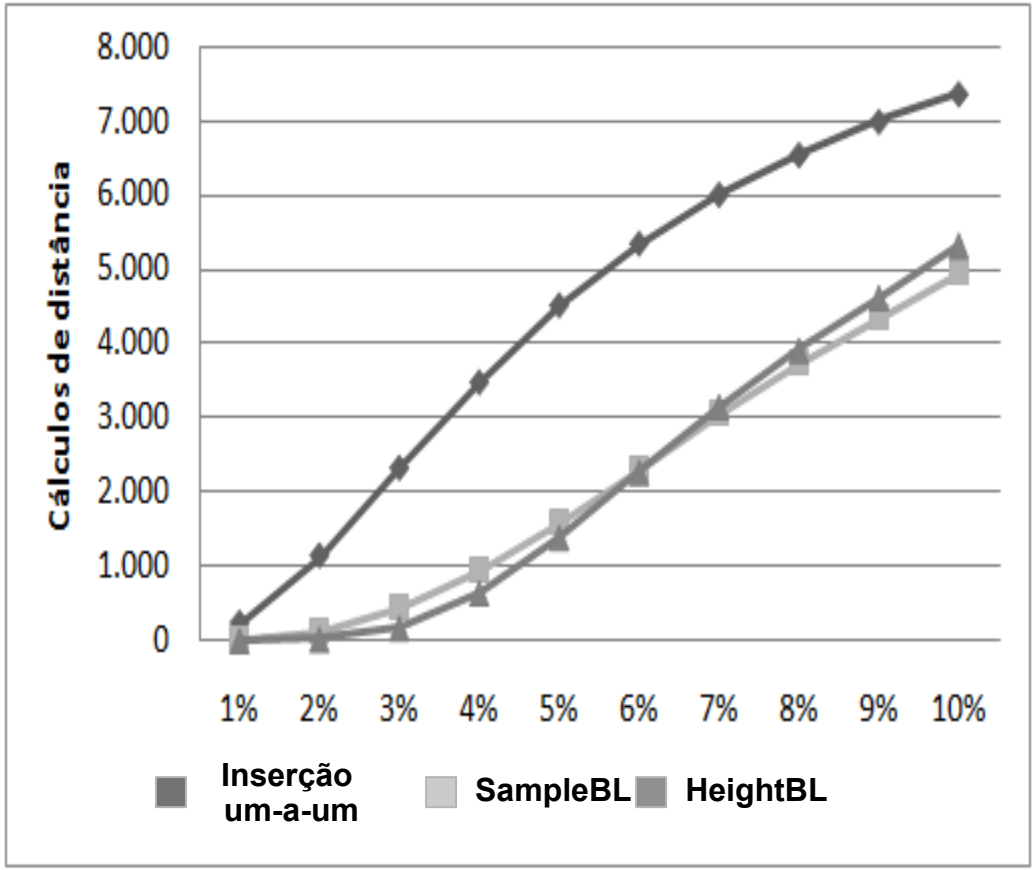
Tamanho do índice

Validação

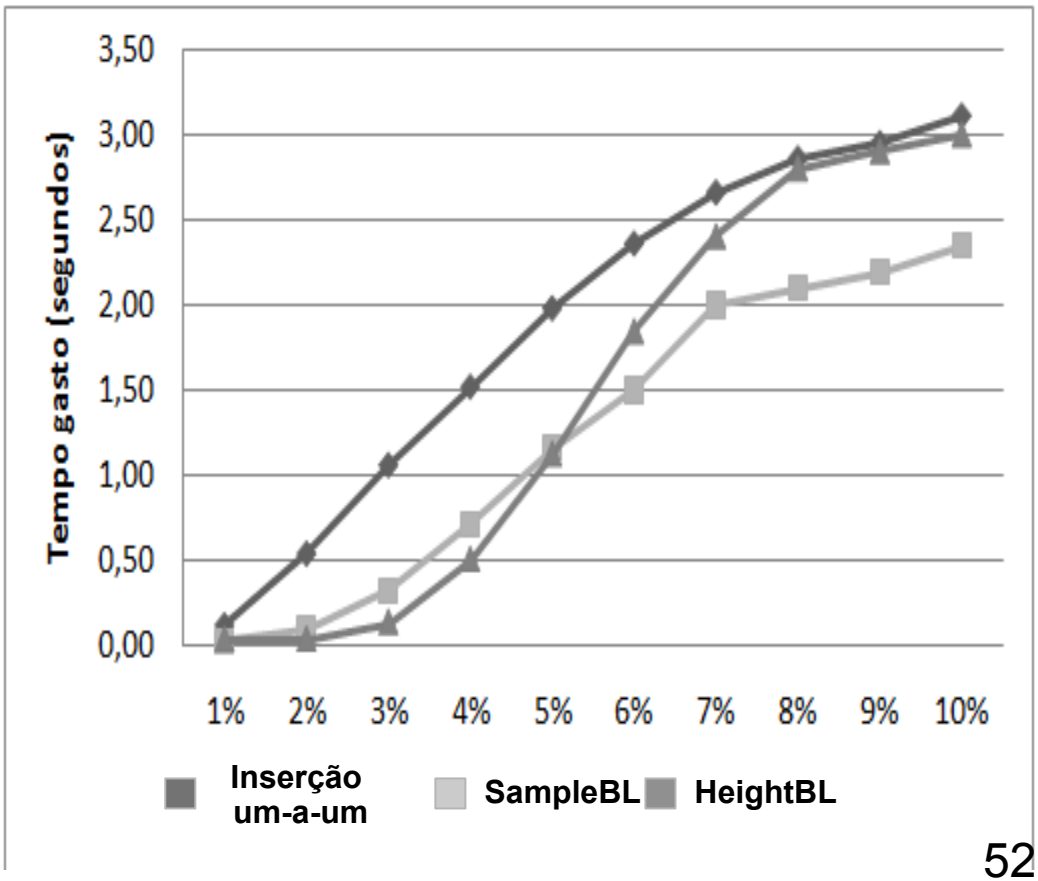
Resultados Gerais

Consultas por abrangência

- Algoritmos de *bulk-loading* gastaram de 9% a 99% menos tempo
- Algoritmos de *bulk-loading* requereram de 16% a 99% menos cálculos de distância



Tempo gasto para executar consultas por abrangência



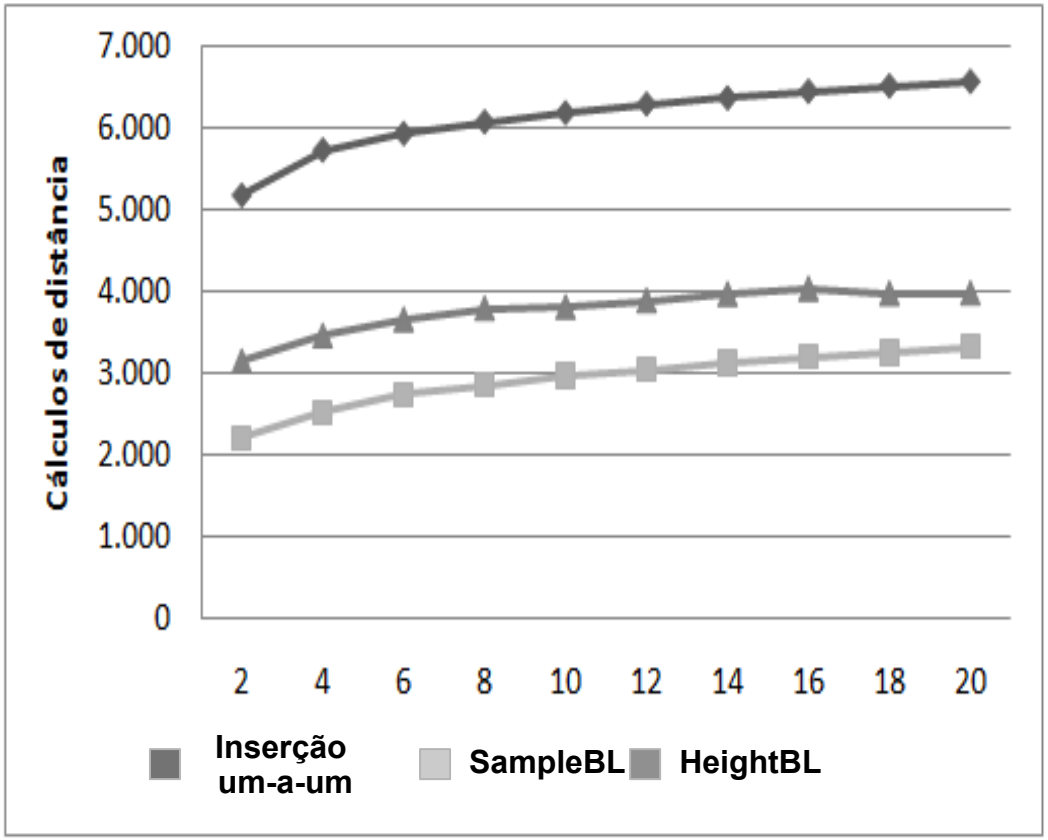
Número de cálculos de distância em consultas por abrangência

Validação

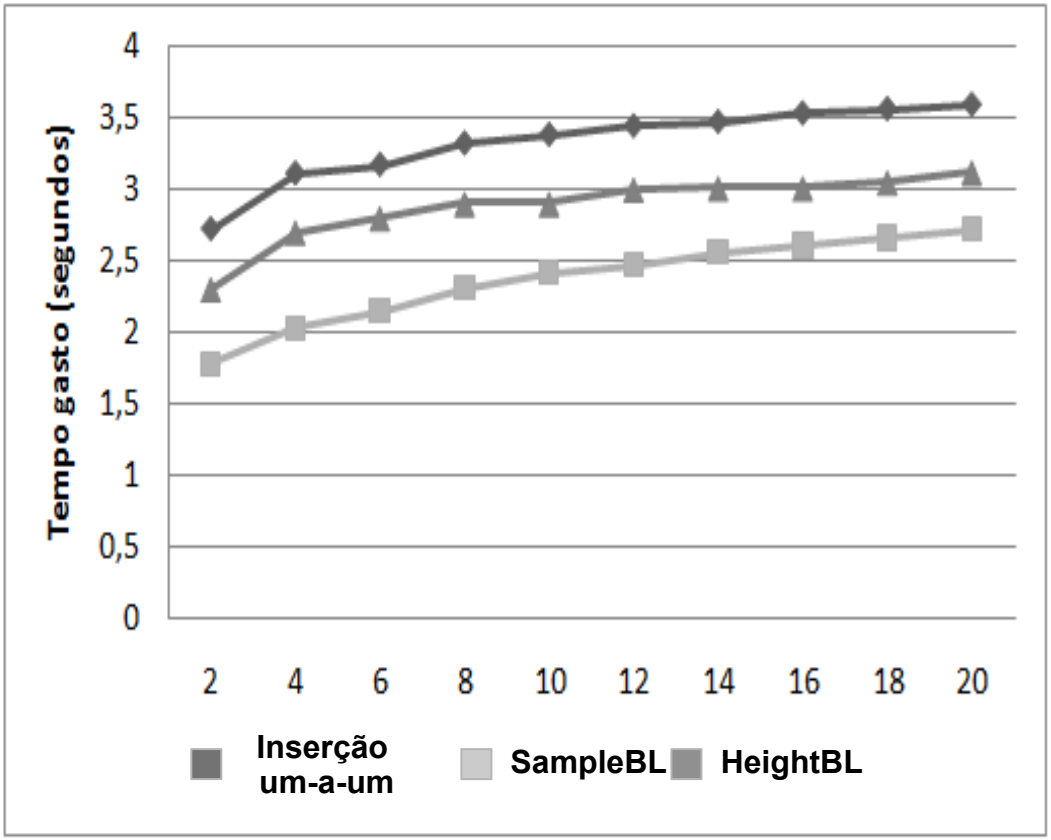
Resultados Gerais

Consultas aos *k*-vizinhos mais próximos

- Algoritmos de *bulk-loading* gastaram de 9% a 63% menos tempo
- Algoritmos de *bulk-loading* requereram de 13% a 86% menos cálculos de distância



Tempo gasto para executar consultas aos *k*-vizinhos mais próximos



Número de cálculos de distância em consultas aos *k*-vizinhos mais próximos

Validação

Resultados Específicos

SampleBL x HeightBL

- HeightBL
  - De 2% a 58% menos memória.
  - De 6% a 43% menos cálculos de distância para construção do índice
  - De 3% a 75% menos tempo para construção do índice
- SampleBL
  - De -8% a 64% menos cálculos de distância em consultas.
  - De -15% a 67% menos tempo em consultas.

Bulk-loading para F-Onion-tree x Bulk-loading para V-Onion-tree

- Ganhos aproximadamente iguais.
- Método de particionamento dos algoritmos de *bulk-loading* para a V-Onion-tree foram o diferencial.

Roteiro

Fundamentação Teórica

Trabalhos Correlatos

Proposta

Validação

Conclusão

Conclusão

- **GreedyBL**
  - Abordagem **gulosa**.
  - **Base** para a criação dos demais algoritmos propostos.
- **SampleBL**
  - Etapa de **amostragem**.
  - Novo método de **particionamento** da **V-Onion-tree**.
- **HeightBL**
  - **Estimativa** da **altura** final da Onion-tree.
- **Resultados positivos em relação à inserção um-a-um**
  - **Tamanho** do índice gerado.
  - **Tempo** de processamento de consultas.
  - Número de **cálculos de distâncias**.

Obrigado

Algoritmos de *bulk-loading* para o método de acesso métrico Onion-*tree*