MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

DATA CUBE ALGORITHM FOR HIGH SEQUENTIALITY SATELLITE TELEMETRY DATA ANALYSIS

Yuri Matheus Dias Pereira

Orientadores:

Dr. Mauricio Gonçalves Vieira Ferreira

Dr. Rodrigo Rocha Silva

Problema

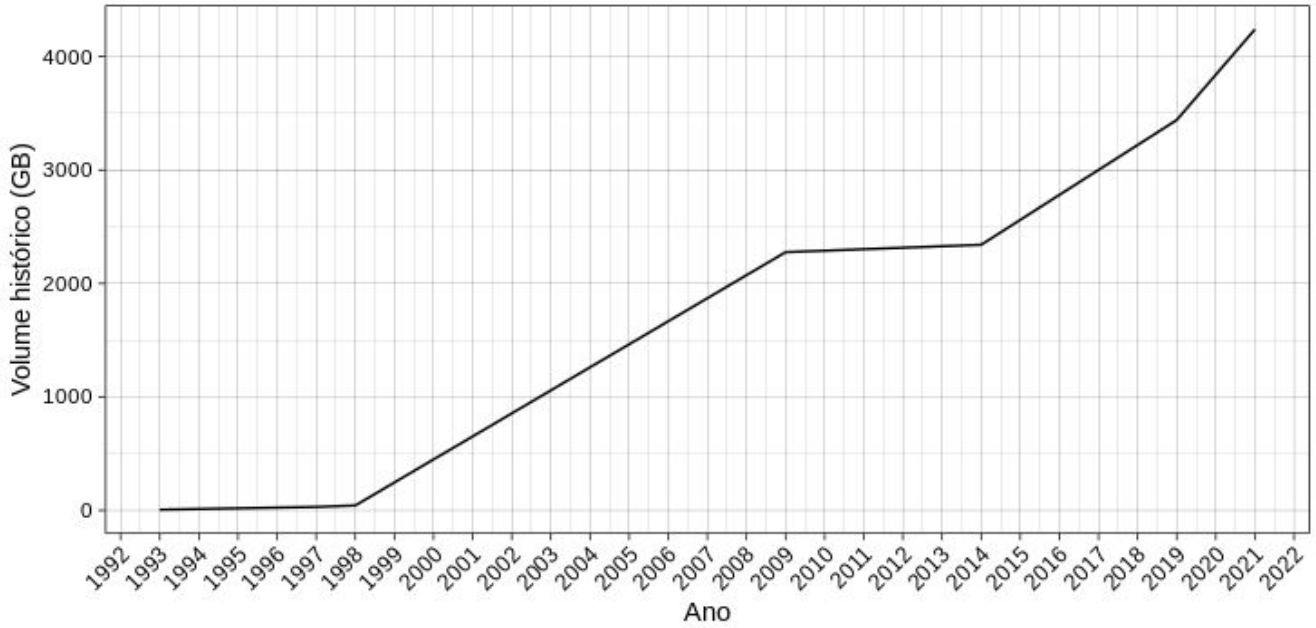
- Operadores de Satélite precisam de conhecimento especializado do sistema e como os seus subsistemas estão relacionadas para operar o satélite
- Fazem uso utilizando de dados de telemetria que são gerados continuamente pelo satélite, e a sua análise pode envolver anos de dados ao mesmo
- Esta análise não é trivial, e é necessário saber quais perguntas devem ser feitas e o como lidar com o volume e complexidade dos dados

Solução

- Executar consultas complexas não é trivial para os operadores, e as executar em anos de dados é uma operação lenta em banco de dados comuns
- Um Armazém de Dados (Data Warehouse) é uma solução eficiente para facilitar as análises, trabalhando em cima de dados prontos para a análise e em uma interface que permita a execução de consultas arbitrárias

Volume histórico

- Atualmente: ~3TB com base na geração dos CBERS e SCDs



Objetivos

- **Criar uma arquitetura baseada em cubo de dados para representar dados de telemetria de satélite de uma missão, utilizando da distribuição de valores para facilitar a análise e consultas no estado do satélite pelos engenheiros de satélite.**
  - Testar uma abordagem que usa da Alta Dimensionalidade dos dados versus uma baseada em particionar a entrada por dados de Baixa Dimensionalidade
  - Testar um algoritmo de compressão de lista invertida baseada em intervalos para o algoritmo Frag-Cubing

Cubo de dados

- “operador relacional que gera todas as combinações possíveis de seus atributos de acordo com uma medida” (GRAY et al., 1996)
- Dimensões: entidades que compõem os dados
- Medidas: atributos calculado pelo relacionamento entre dimensões
- Conceito *ALL*: agregação de todas as combinações de um conjunto de atributos
- Objetivo principal é modelar e visualizar dados em múltiplas dimensões

Cubo de dados

- *n*-dimensional: não necessariamente um “cubo”
- Células são níveis de agregação das tuplas de entrada
- 2 dimensões, 1 medida ->

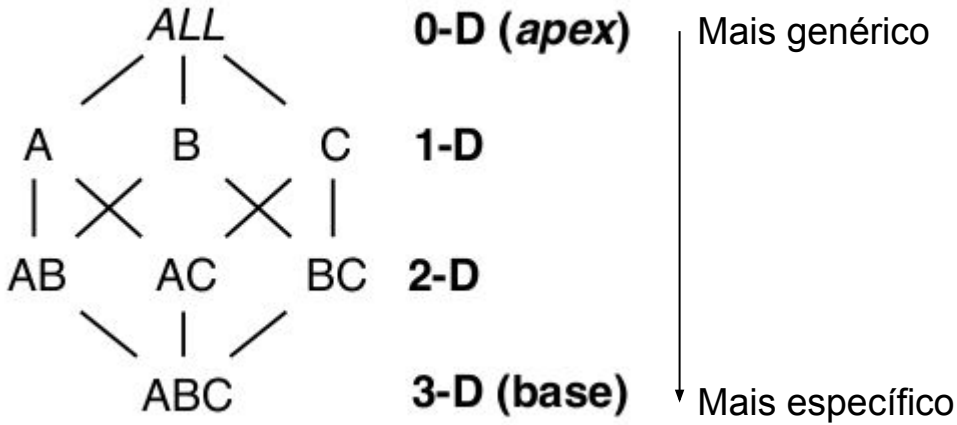
Satélite	Telemetria	Valores
sat1	TM1	23
sat1	TM2	20

Cubo

Satélite	Telemetria	Valores
sat1	TM1	23
sat1	TM2	20
sat1	ALL	43
ALL	TM1	23
ALL	TM2	20
ALL	ALL	43

Células do cubo

- Cubo de dados é composto de **subcubos**
- Células Agregadas: utilizam "*ALL*" para demonstrar que agregam todos os valores daquela dimensão
- Células base: nível mais baixo de agregação

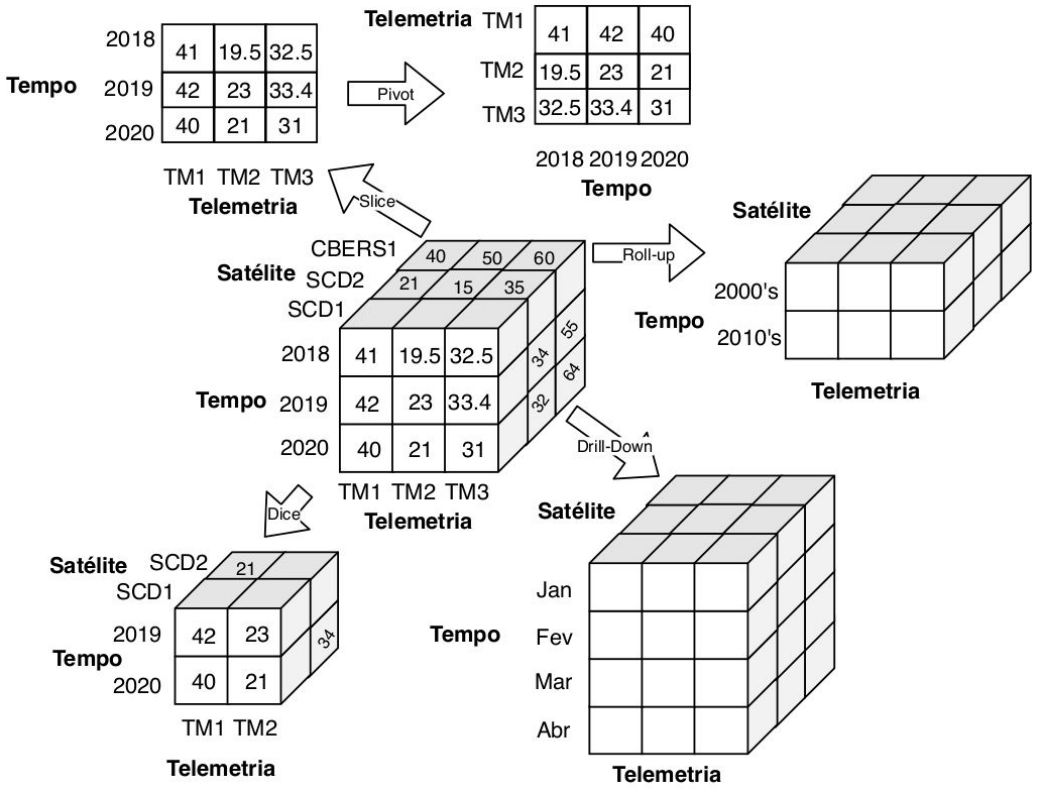


Medidas

- Calculada para cada célula do cubo agregando os dados correspondentes a combinação de dimensões e valores -> $\langle (d_1, d_2, \ldots, d_n), medidas \rangle$ com cada d_n um valor de dimensão
- Distributiva: cálculo pode ser particionado -> soma
- Algébrica: duas ou mais medidas distributivas -> média = soma / contagem
- Holística: não pode ser particionada, precisa de executar sobre todos os dados para ter respostas exatas -> moda

Cubo de Dados

- Composto de conjuntos de dimensões e medidas
- Dimensões são os atributos constituintes dos dados
- Medidas são relacionamentos calculados entre as dimensões



Computação do cubo de dados

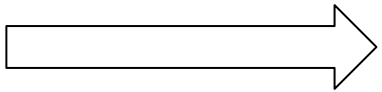
- Materialização completa geralmente não é possível
- **Maldição da Dimensionalidade:** requisitos de espaço aumentam de forma exponencial, 2^n com *n* o número de dimensões
- Materialização parcial
  - *Iceberg*: critério de mínimo para as células
  - *Shell Fragment*: subcubos de 3 a 5 dimensões
  - *Closed Shell* ou *quotient cube:* medidas idênticas na mesma abstração

Computação do cubo de dados

- Frag-Cubing, por Li et al (2004) é um dos algoritmos mais tradicionais, utilizando de conceitos de fragmentos de camadas (shell fragment), de um índice invertido para computar as respostas a consultas e de uma operação de iceberg para remover células fora do critério mínimo
- Várias formas diferentes e melhorias foram propostas ao longo dos anos, porém a base do algoritmo ainda são essas duas operações
  - Memória externa: HFrag, HIC (Silva et al., 2015; 2016)
  - Cubos fechados: C-Cubing (Dong Xin et al (2006)), Closed Frag Shells (Zhao et al (2018))

FragCubing - Índice Invertido

TID	A	B	C
tid1	a1	b1	c1
tid2	a2	b2	c2
tid3	a1	b3	c2
tid4	a1	b3	c1
tid5	a1	b1	c3
tid6	a2	b1	c3
tid7	a1	b1	c3



Operação do Cubo

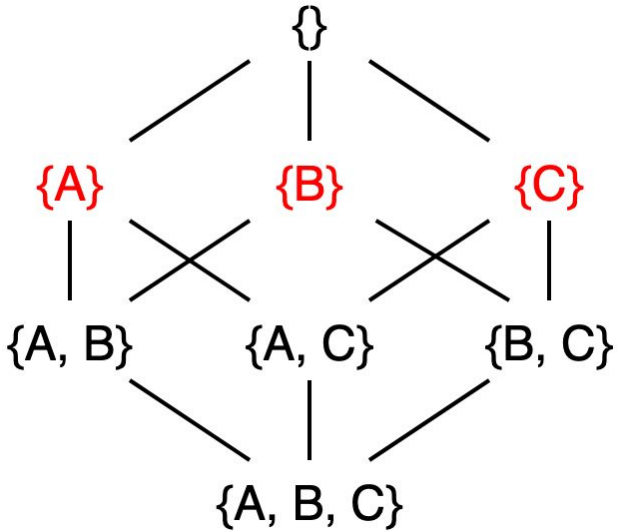
Tamanho, neste caso, é a medida computada

Valor	Lista de TID	Tamanho
a1	1, 3, 4, 5, 7	5
a2	2, 6	2
b1	1, 5, 6, 7	4
b2	2	1
b3	3, 4	2
c1	1, 4	2
c2	1, 2	2
c3	5, 6, 7	3

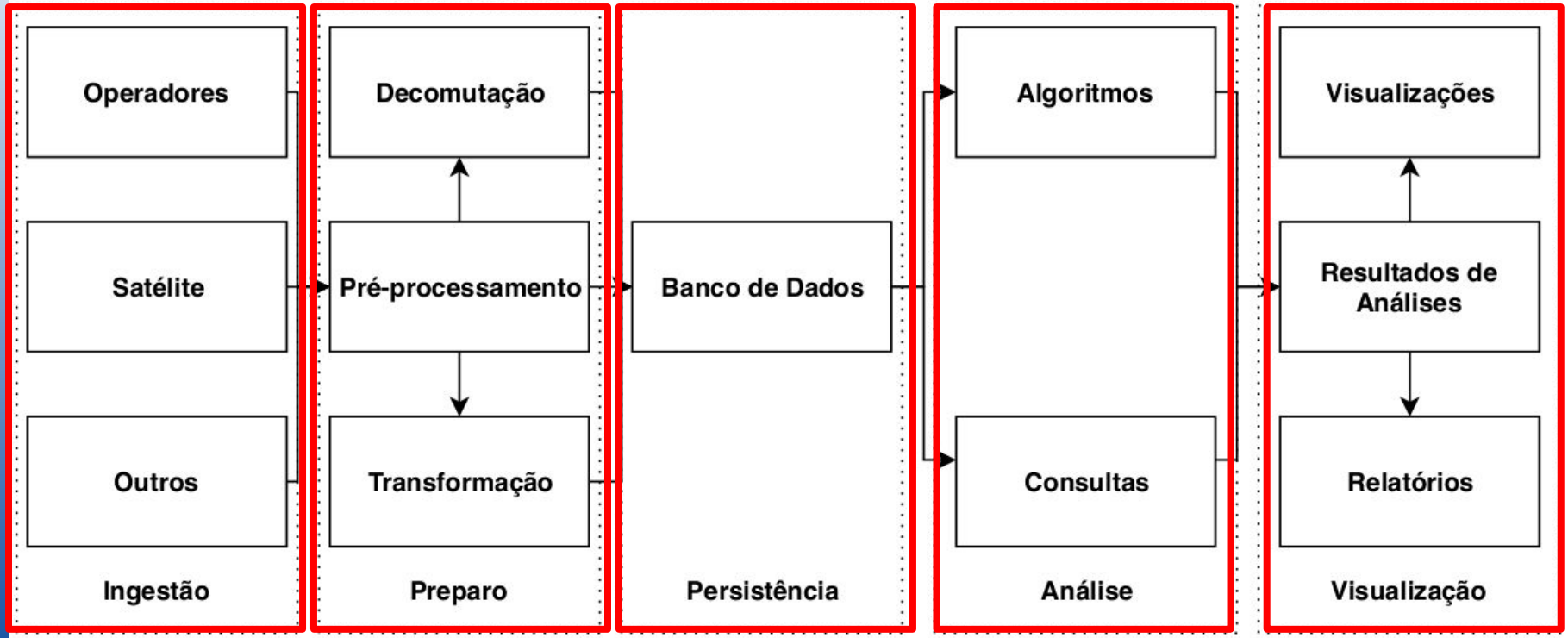
Consultas são respondidas fazendo intersecção nas listas de TID com as dimensões relacionadas

FragCubing - Fragmentação

- O número da camada de fragmentação é definido de antemão (F-number)
- Para F = 1



Fluxo dos dados



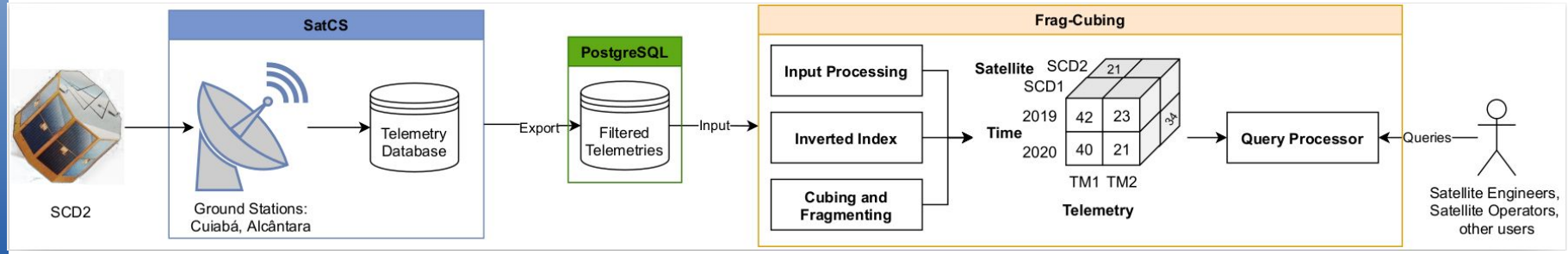
Método

Estudo de Caso: SCD2

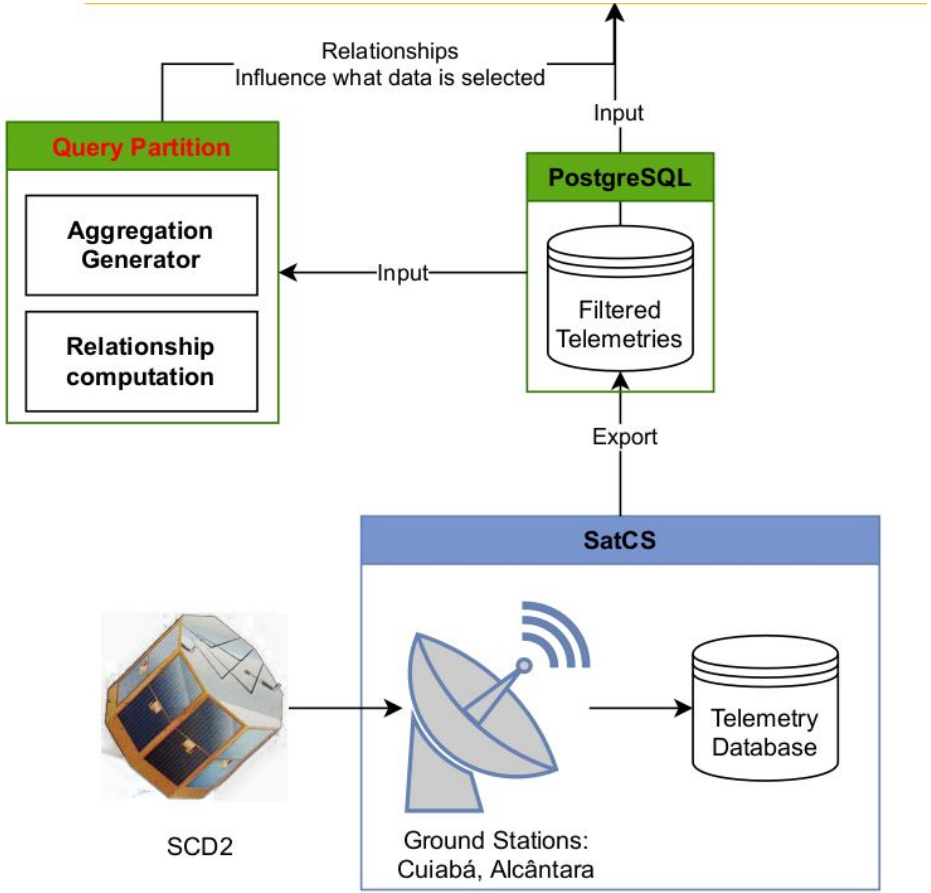
- Quatro anos de telemetrias do SCD2 fornecidas pelo CCS, entre 2014 e 2018
- 135 telemetrias, resultando em um arquivo de 23GB em formato CSV exportado do SatCS utilizado como base

Arquitetura - Estática

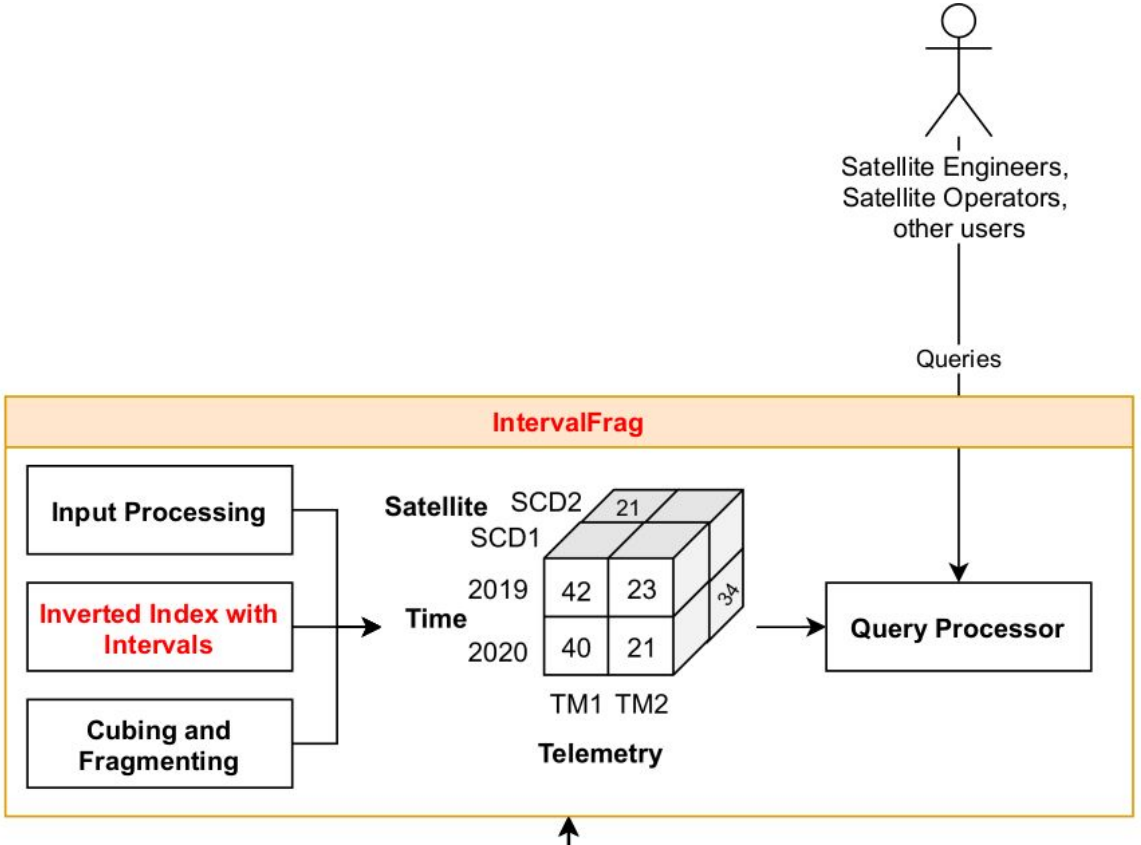
- Adaptar os dados existentes em uma arquitetura baseada em cubo de dados
- Objetivo principal é responder consultas de operadores: como melhorar essa base?



Arquitetura - Proposta



Arquitetura - Proposta



Particionamento por Consulta

Particionamento por consulta

- Como utilizar de características dos dados para criar cubos de dados com performance melhor?
- Processar os dados em duas abordagens:
  - Alta Dimensionalidade, com todas as dimensões e todos os dados
  - Baixa Dimensionalidade, apenas as dimensões que são relacionadas com a consulta

Gerador de agregações

- Algoritmo que gera as os conjuntos de telemetrias que estão relacionadas entre si
  - Utiliza da agregação de cada dimensão, calculando estatísticas descritivas sobre o relacionamento entre as dimensões
  - Força do relacionamento é calculada com base na cardinalidade das telemetrias e alguns parâmetros (mediana, desvio padrão)
- Cutoff do algoritmo arbitrário, dependendo da característica do conjunto de telemetrias

Consultas

ID	Telemetries	Product of cardinalities
Q1	TM072, TM081, TM077, TM078, TM082, TM083	983.920
Q2	TM003, TM004, TM005	15.813.251
Q3	TM003, TM006	63.001
Q4	TM001, TM002	26.075
Q5	TM130, TM131, TM075	14.397.360

- Q1: As baterias estão sendo carregadas?
- Q2: Qual a orientação atual do satélite?
- Q3: Existe alguma discrepância nas leituras dos magnetômetros do satélite?
- Q4: As antenas da payload estão funcionando?
- Q5: Existe alguma discrepância entre nas correntes dos painéis solares e a sua temperatura?

Validação Experimental

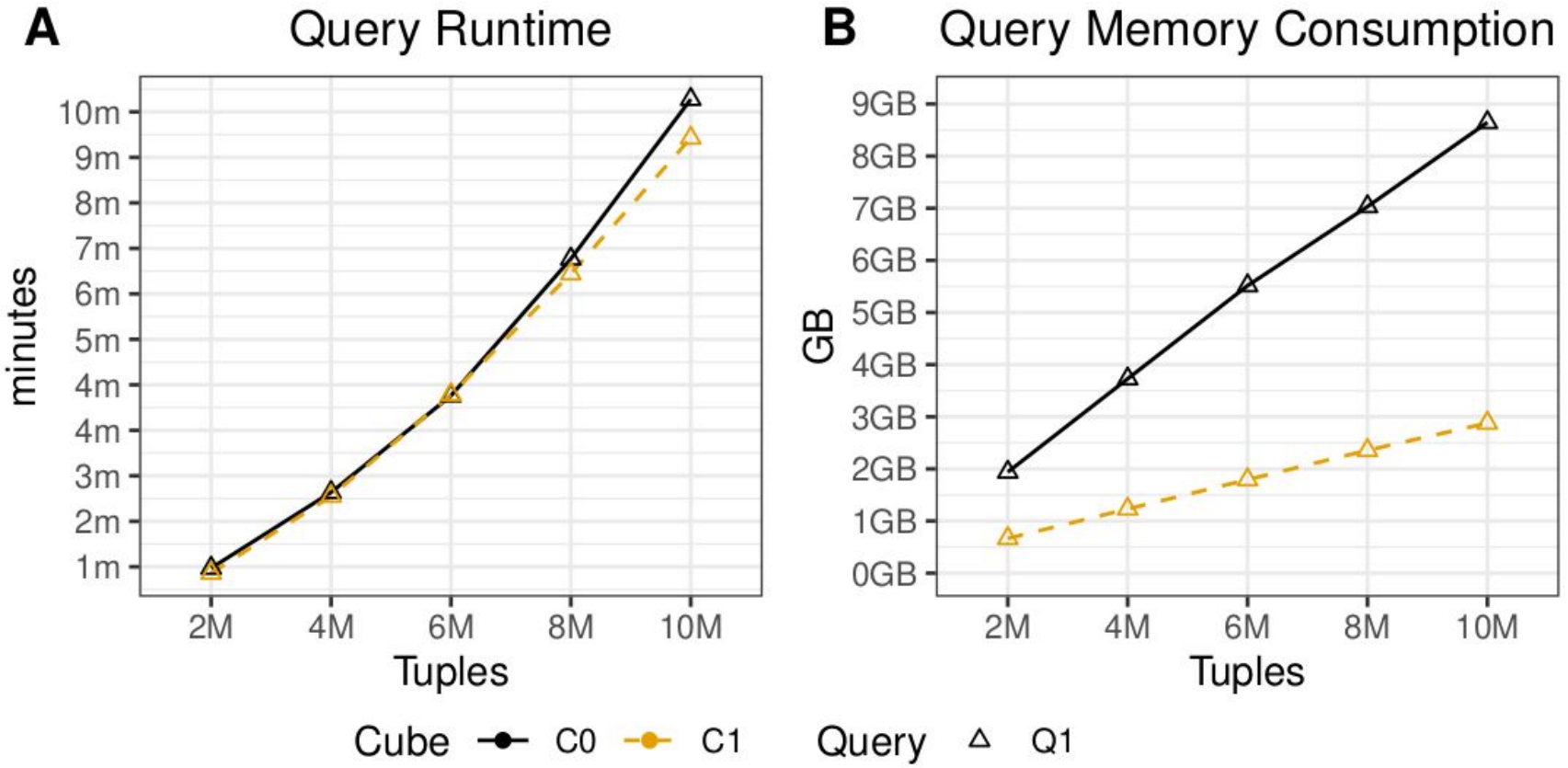
- Dividir os dados disponíveis em Alta Dimensionalidade e Baixa Dimensionalidade
- Construir cubo de dados baseado no Frag-Cubing, e medir:
  - Consumo de memória da computação do cubo;
  - Tempo para construção do cubo;
  - Tempo de resposta da consulta;
  - Consumo de memória da consulta
- Cada teste é feito 5 vezes, utilizando a média da medida

Cubos

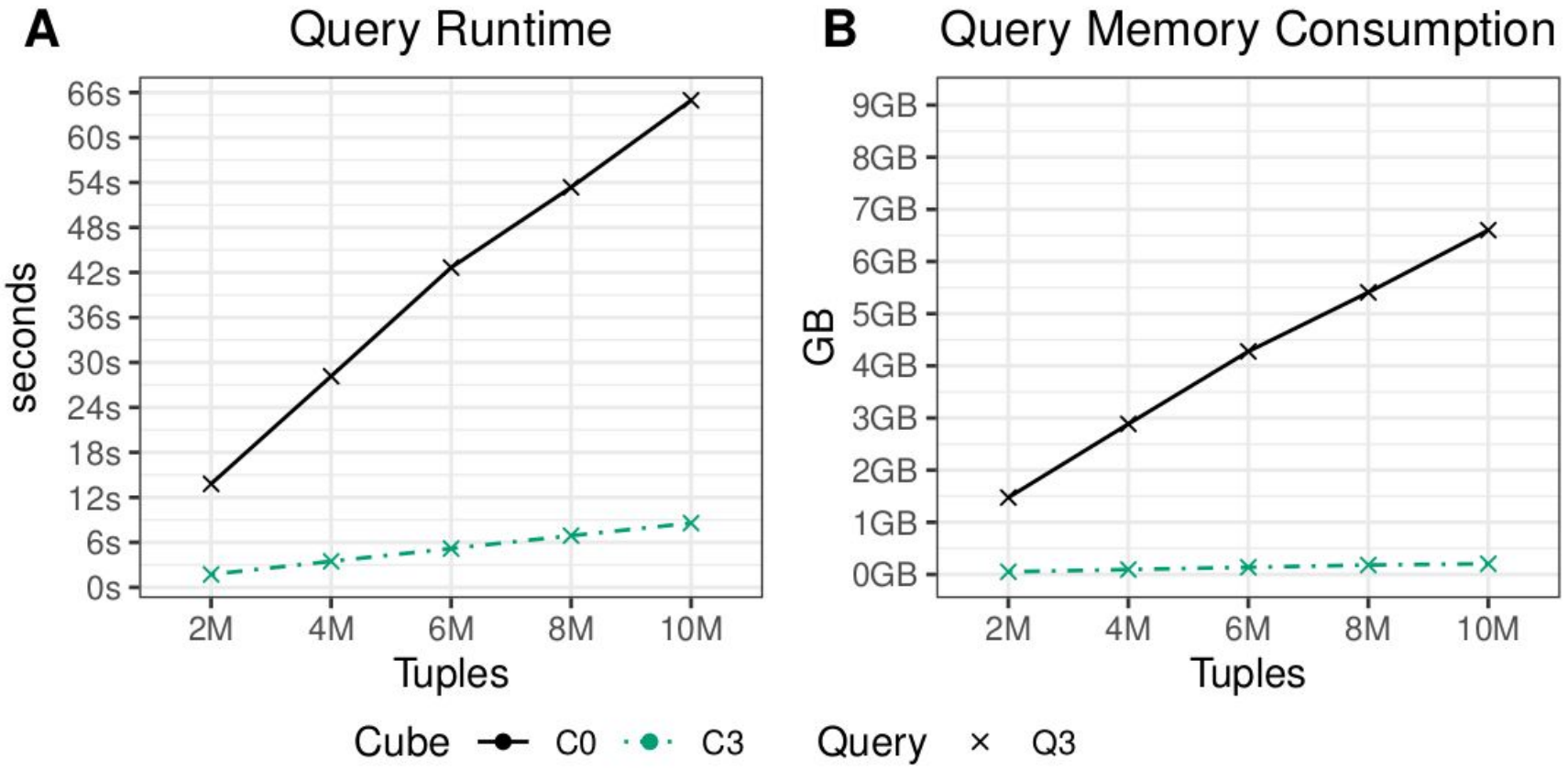
- Cada cubo é um conjunto diferente de telemetrias
- C0 é o de Alta Dimensionalidade, com todas as dimensões
- 5 sequências de tuplas para cada cubo: 2M, 4M, 6M, 8M e 10M

ID	Query	Dimensions	Total Size
C0	-	135	11,29 GB
C1	Q1	6	0,44 GB
C2	Q2	3	0,34 GB
C3	Q3	2	0,22 GB
C4	Q4	2	0,20 GB
C5	Q5	3	0,34 GB

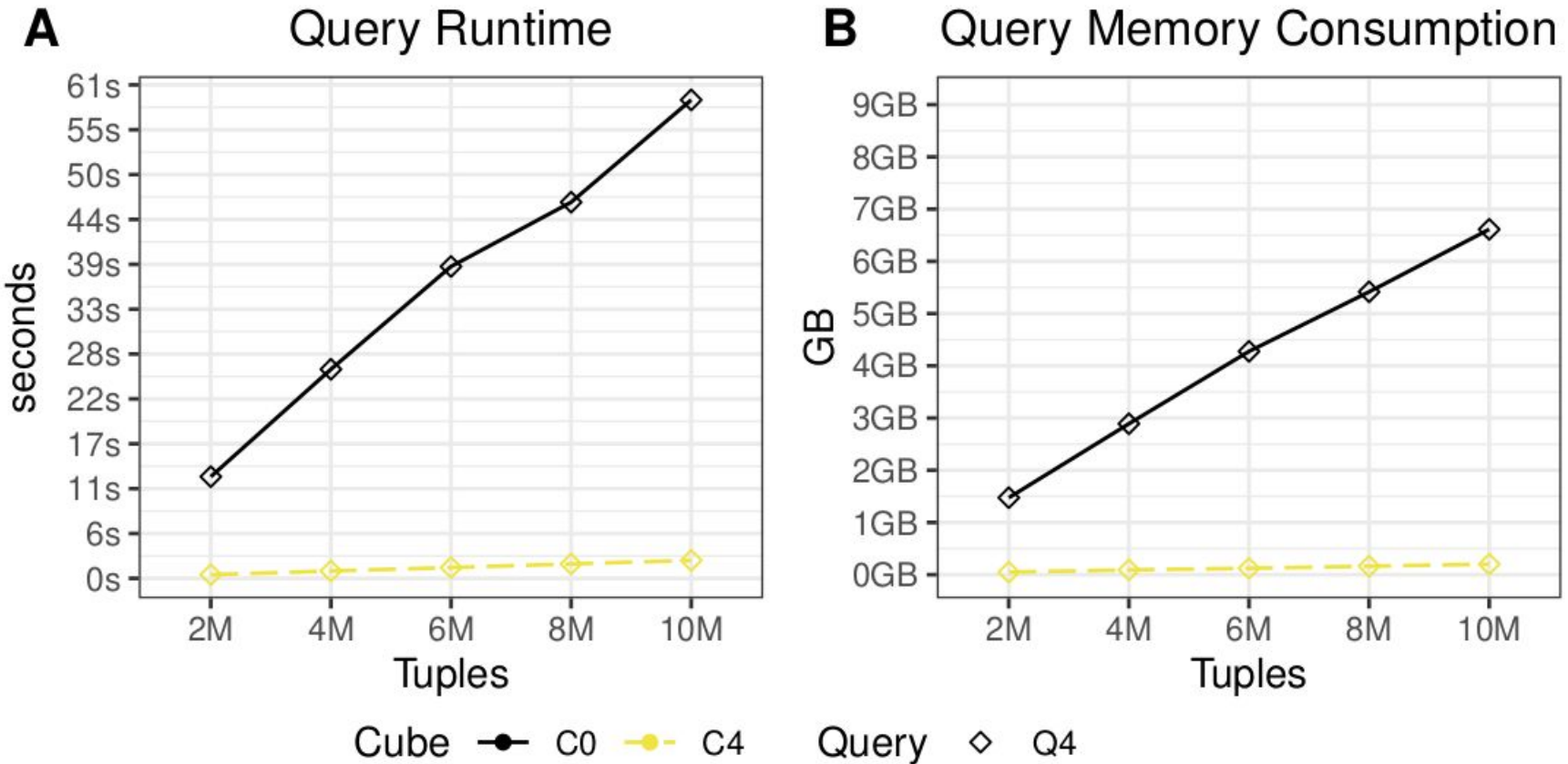
Resultados - Q1



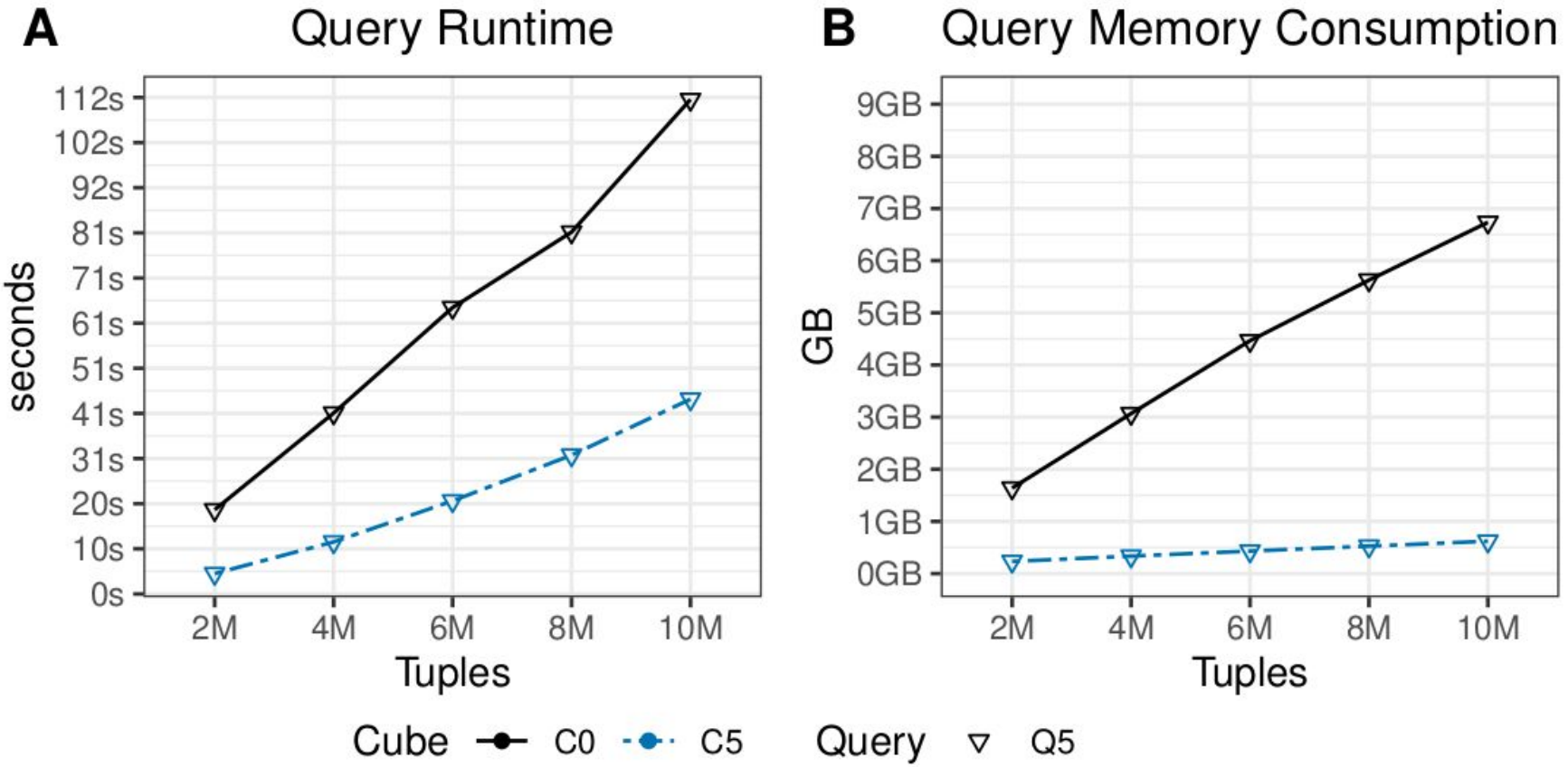
Resultados - Q3



Resultados - Q4



Resultados - Q5



Sumário

- Busca exaustiva dos conjuntos gera resultados muito esparsos, que são complicados de serem avaliados até por especialistas. Porém, atividades de operação geralmente realizam consultas com baixa dimensionalidade;
- Particionar a relação de entrada do cubo pelas consultas que serão executadas pode utilizar entre 1% e 33% da memória para responder a mesma consulta.
- Diminuição no tempo de resposta indica que seria mais rápido construir um cubo com baixa dimensionalidade e consultar ele, do que consultar um cubo de dados de alta dimensionalidade já construído.

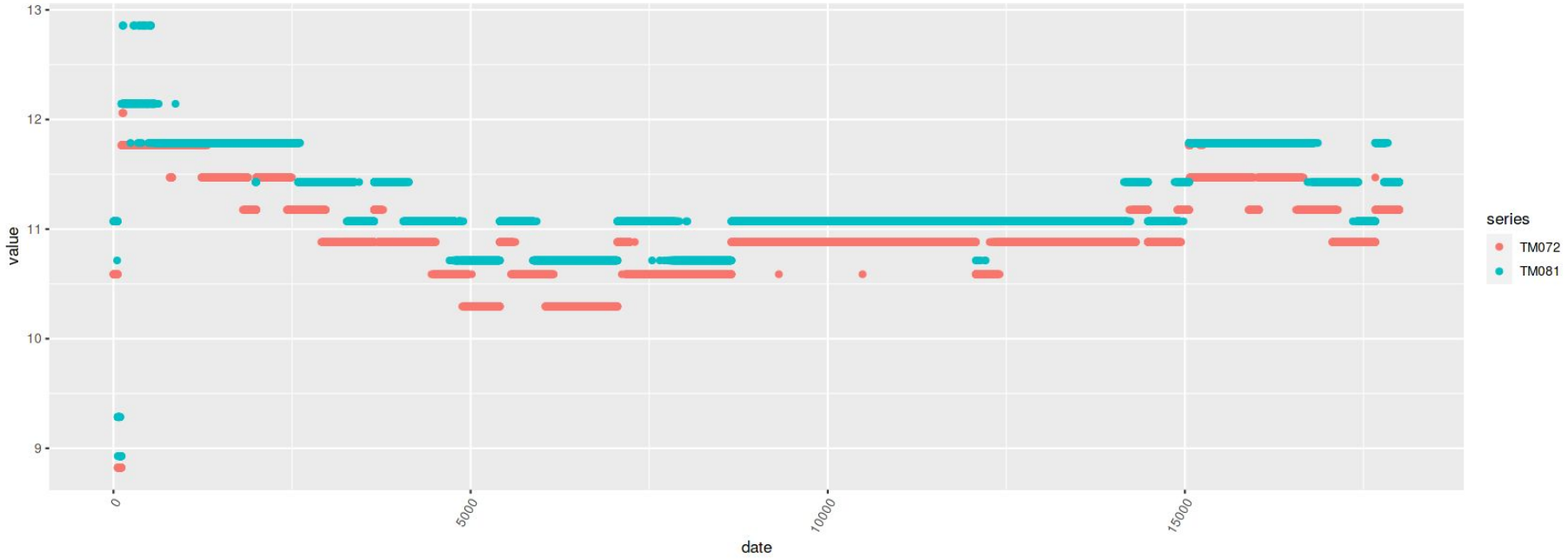
IntervalFrag

Listas de TID e Sequencialidade

- Alguns dados de telemetria de satélite tem uma características: longas repetições dos mesmos valores, então uma longa repetição de outro valor
- Em um índice invertido, isso gera longas listas de TIDs com índices crescentes, e podem ser comprimidos em intervalos
- Para uma lista $\langle d_1, d_2, \cdots, d_k \rangle$, a lista de intervalo seria $\langle [d_1, d_2], [d_3, d_4], \cdots, [d_k, d_{k+1}] \rangle$, onde $d_k < d_{k+1}$ e a diferença de intervalos não pode ser menor que um, assim $d_{k+1} - d_k \geq 1$
- Exemplo: lista [1, 3, 4, 5, 7] teria uma lista de intervalo {[1,1], [3, 5], [7, 7]}, simplificando: {[1], [3, 5], [7]}

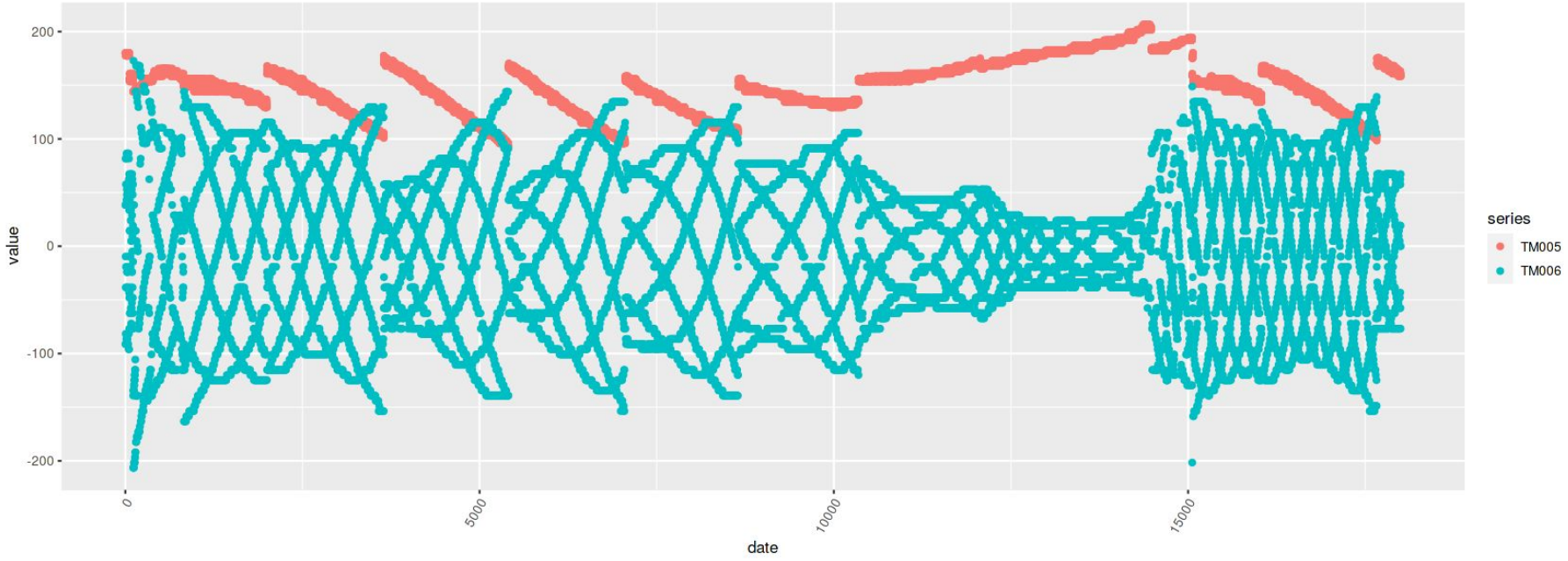
Sequencialidade - Exemplo

- TMs 72 e 81: sequência das telemetrias



Sequencialidade - Exemplo

- Sequência TM005 x TM006: quase 0 sequencialidade



Algoritmo - Inserção

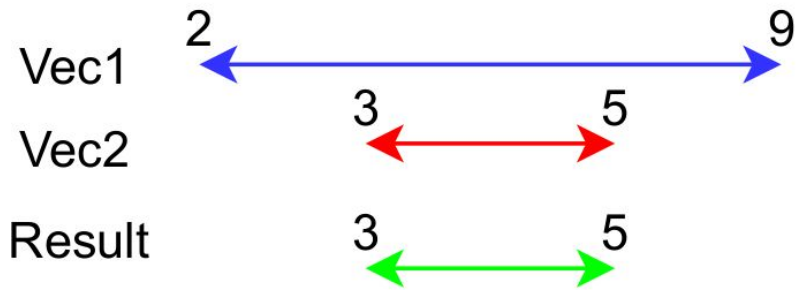
- Inserção de um novo elemento no intervalo pode simplesmente ser feita no final da lista
- Se o último elemento for um intervalo, verifica se o elemento da direita (maior) é igual a elemento que será inserido + 1, se sim, só atualiza o elemento da direita já existente
- Caso não seja, adiciona um novo intervalo como elemento da esquerda
- Implementação utiliza de inteiro positivos para os intervalos, e negativos quando não há um intervalo (necessário para diminuir consumo de memória)

Algoritmo - Intersecção

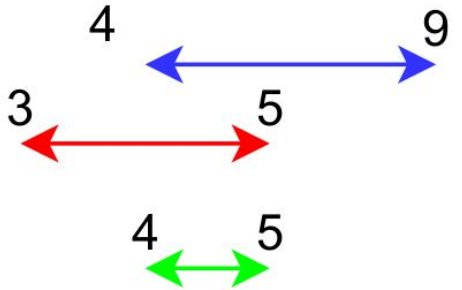
- Para mais detalhes sobre intersecção de conjuntos, ver o anexo A
- Primeiro é necessário fazer a intersecção intervalo por intervalo: identificar qual elemento é maior na esquerda e qual elemento é menor na direita, e se há uma intersecção válida ou não
- Um algoritmo de dois ponteiros pode adicionar o resultado em uma lista auxiliar, utilizando essa intersecção como elemento
- Complexidade ótima: $\mathcal{O}(n+m)$

Algoritmo - Intersecção - Elemento

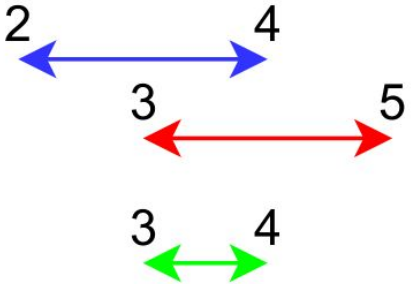
$[2, 9] \cap [3, 5] = \{[3, 5]\}$



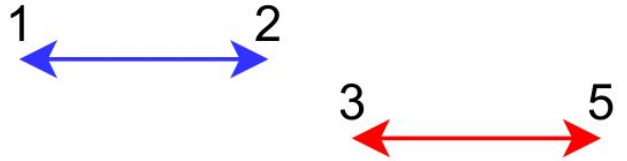
$[4, 9] \cap [3, 5] = \{[4, 5]\}$



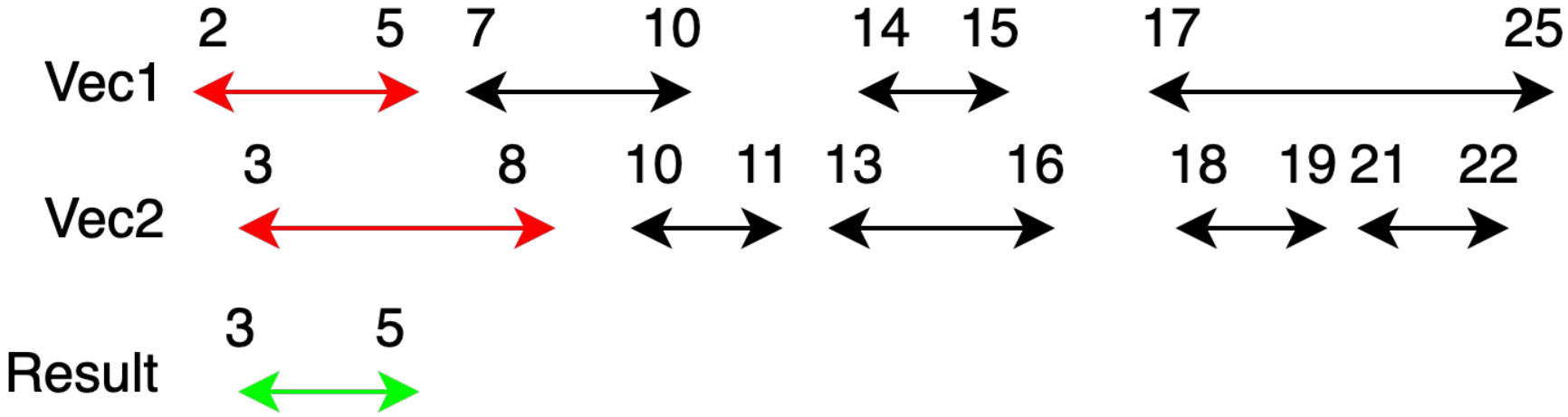
$[2, 4] \cap [3, 5] = \{[3, 4]\}$



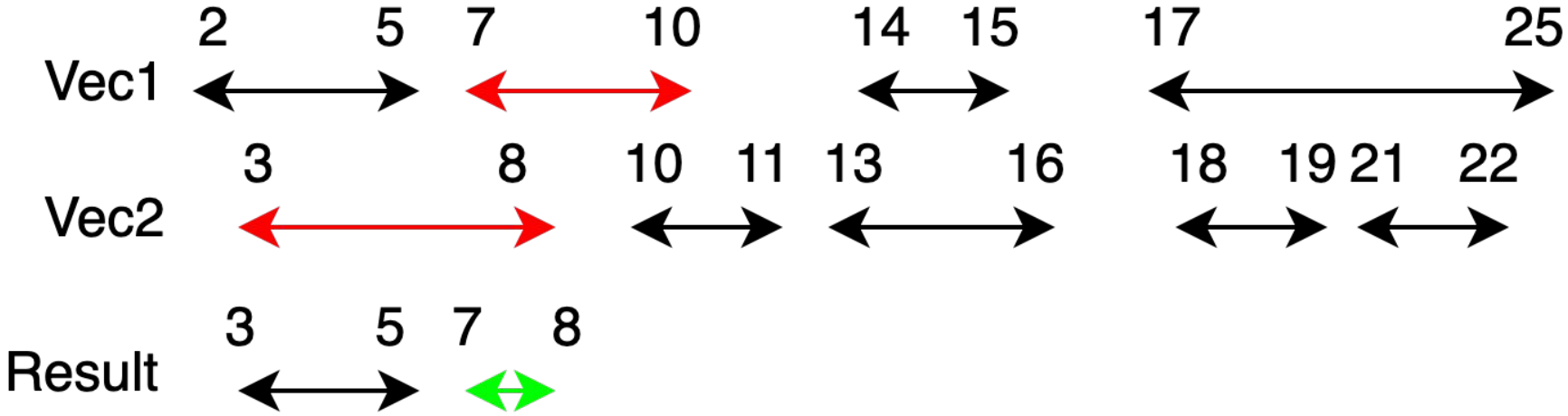
$[2, 4] \cap [3, 5] = \{\emptyset\}$



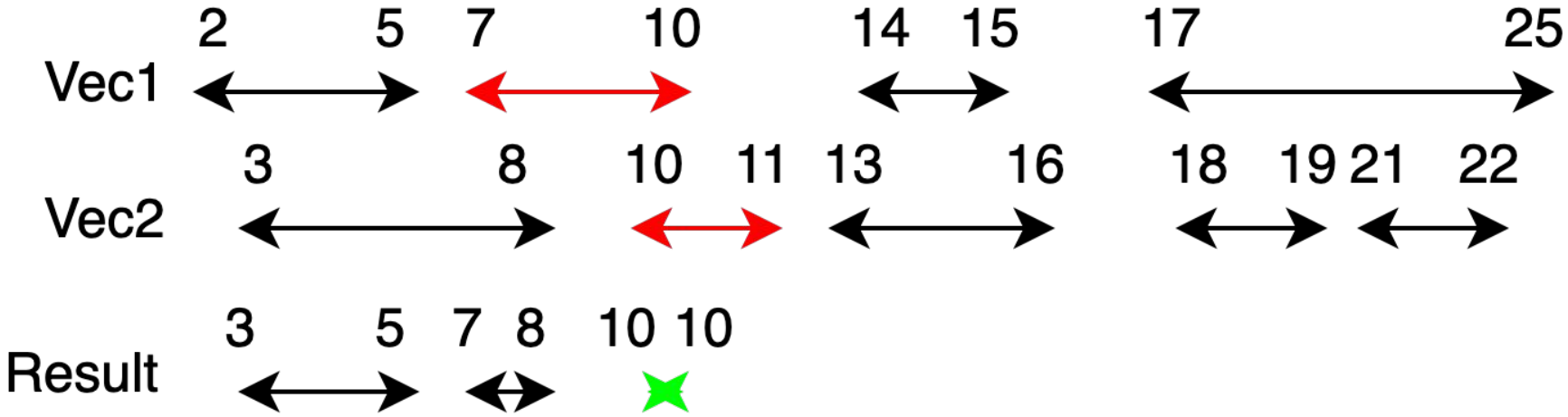
Algoritmo - Intersecção



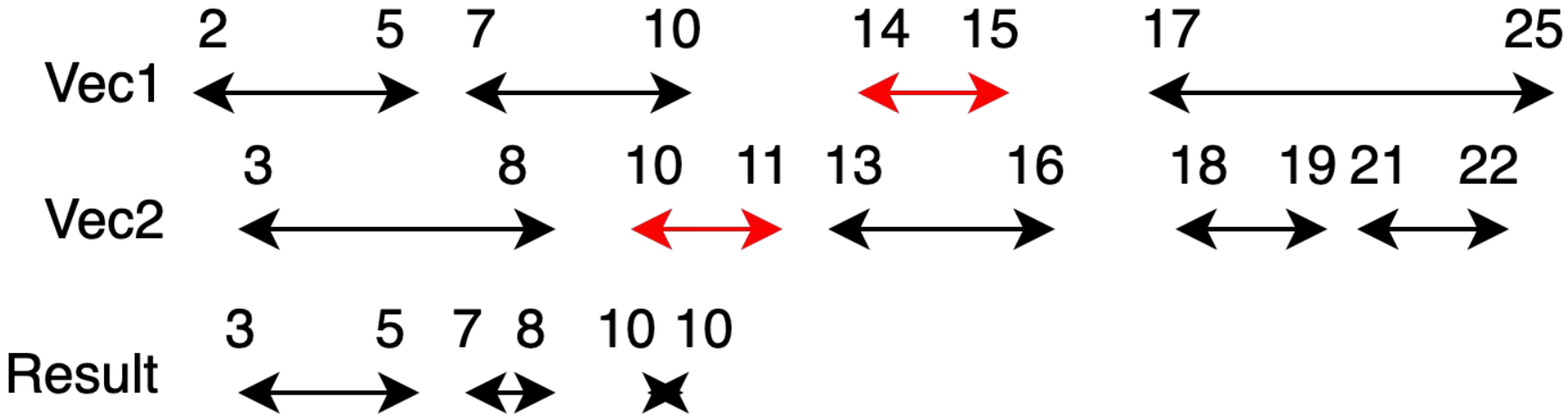
Algoritmo - Intersecção



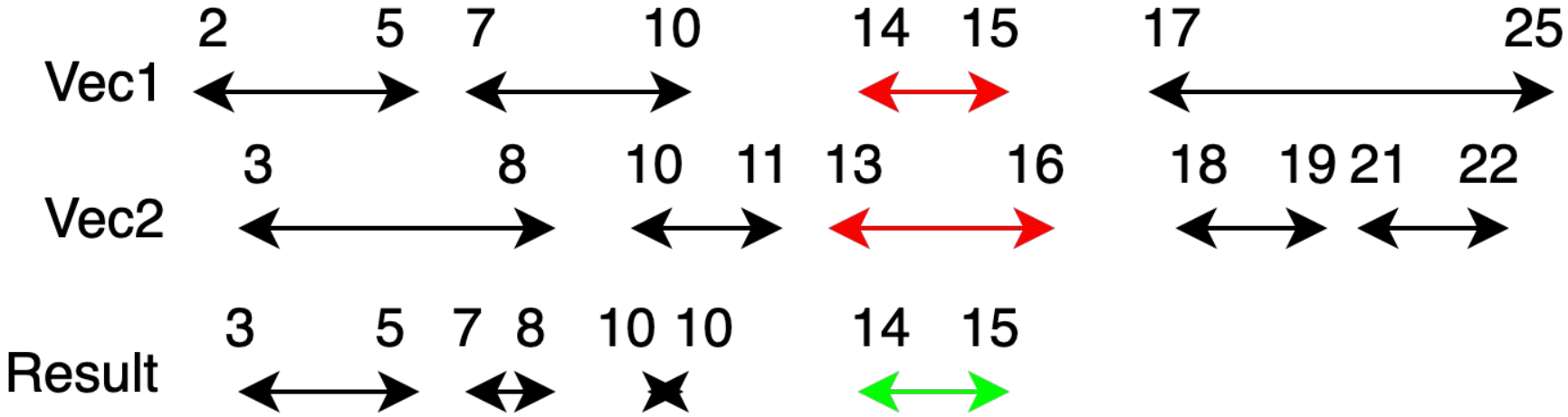
Algoritmo - Intersecção



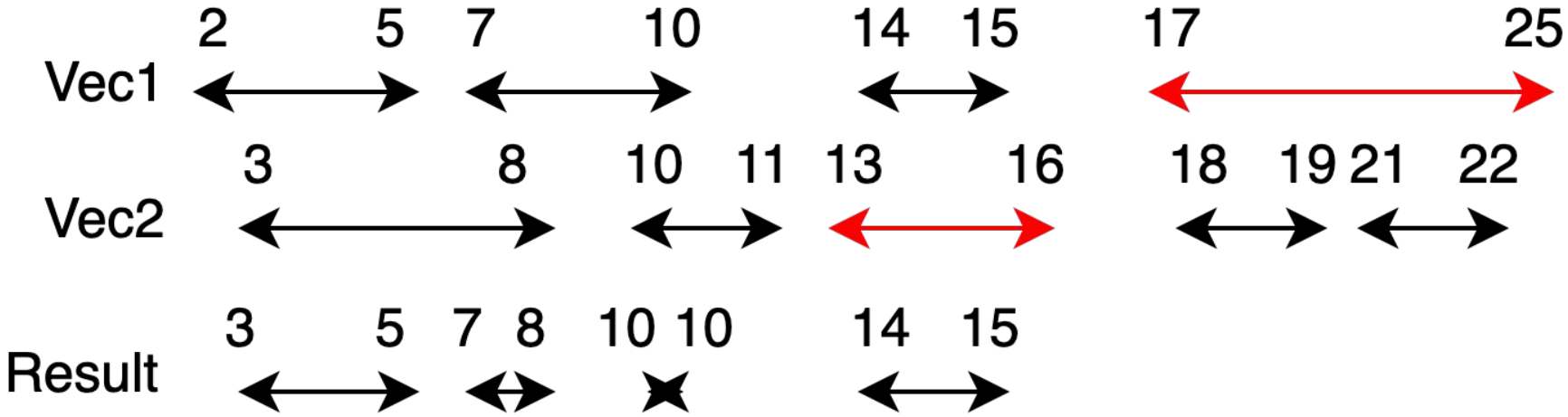
Algoritmo - Intersecção



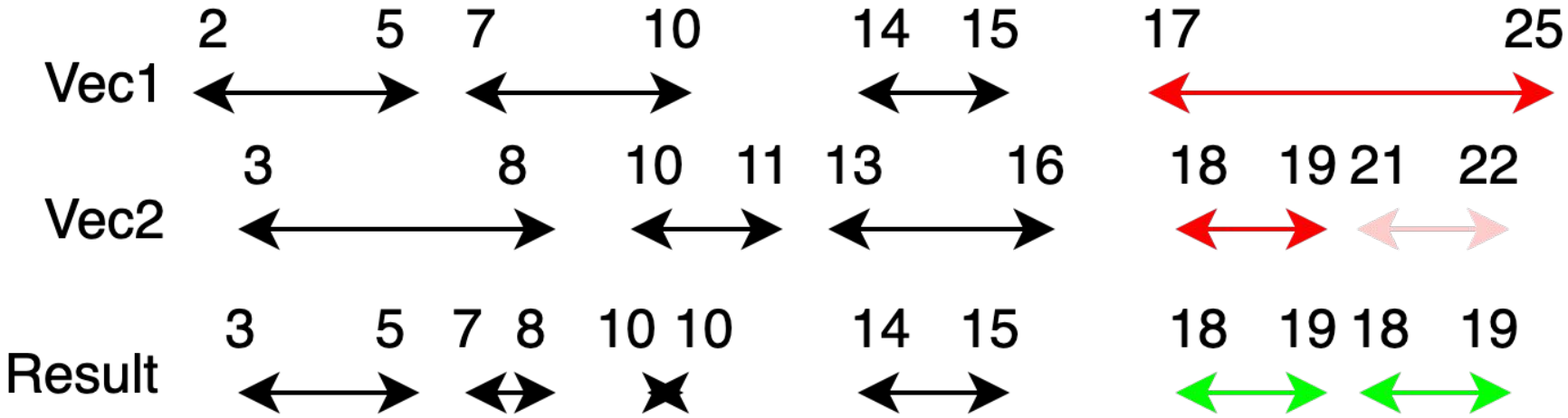
Algoritmo - Intersecção



Algoritmo - Intersecção



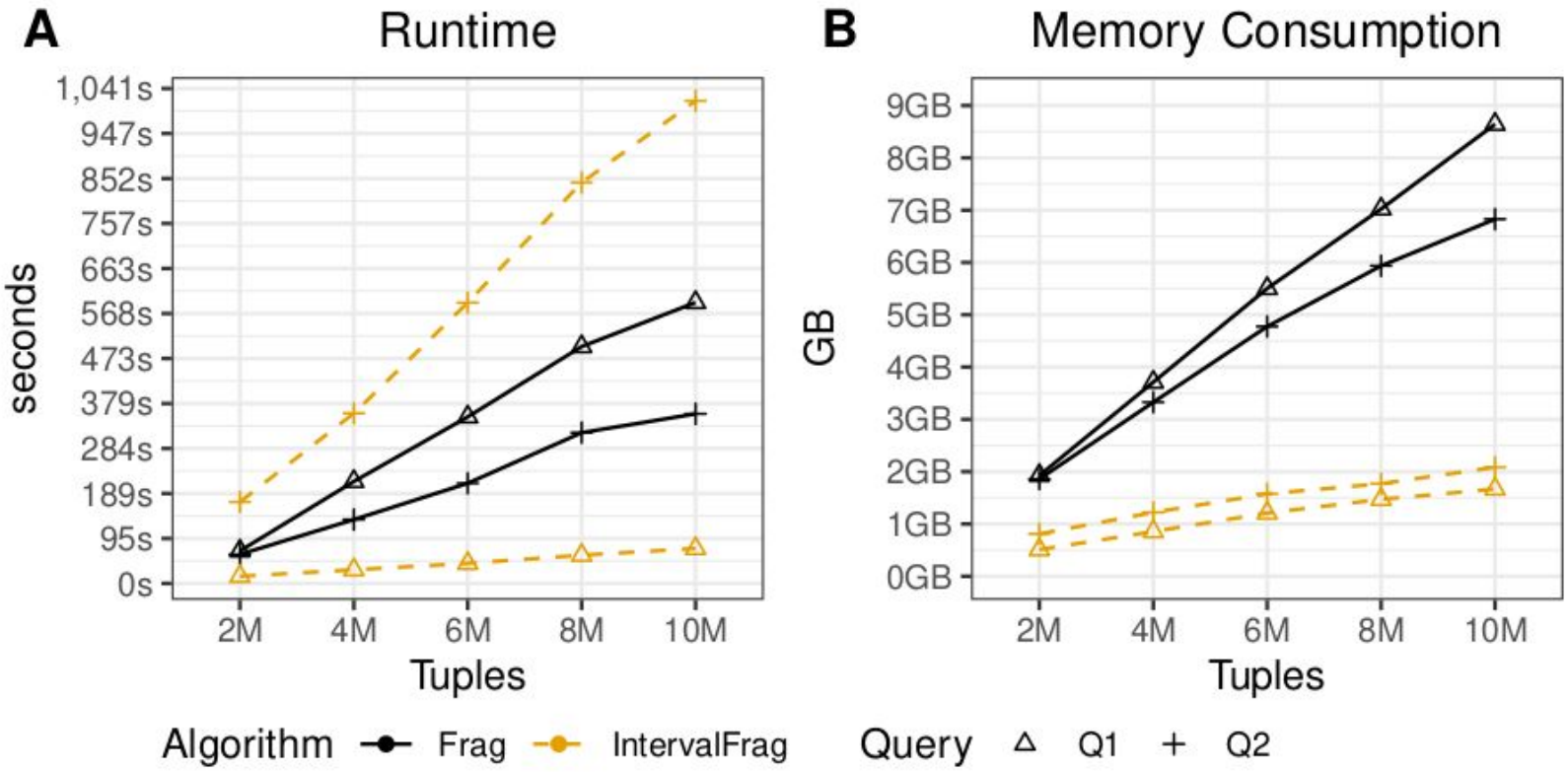
Algoritmo - Intersecção



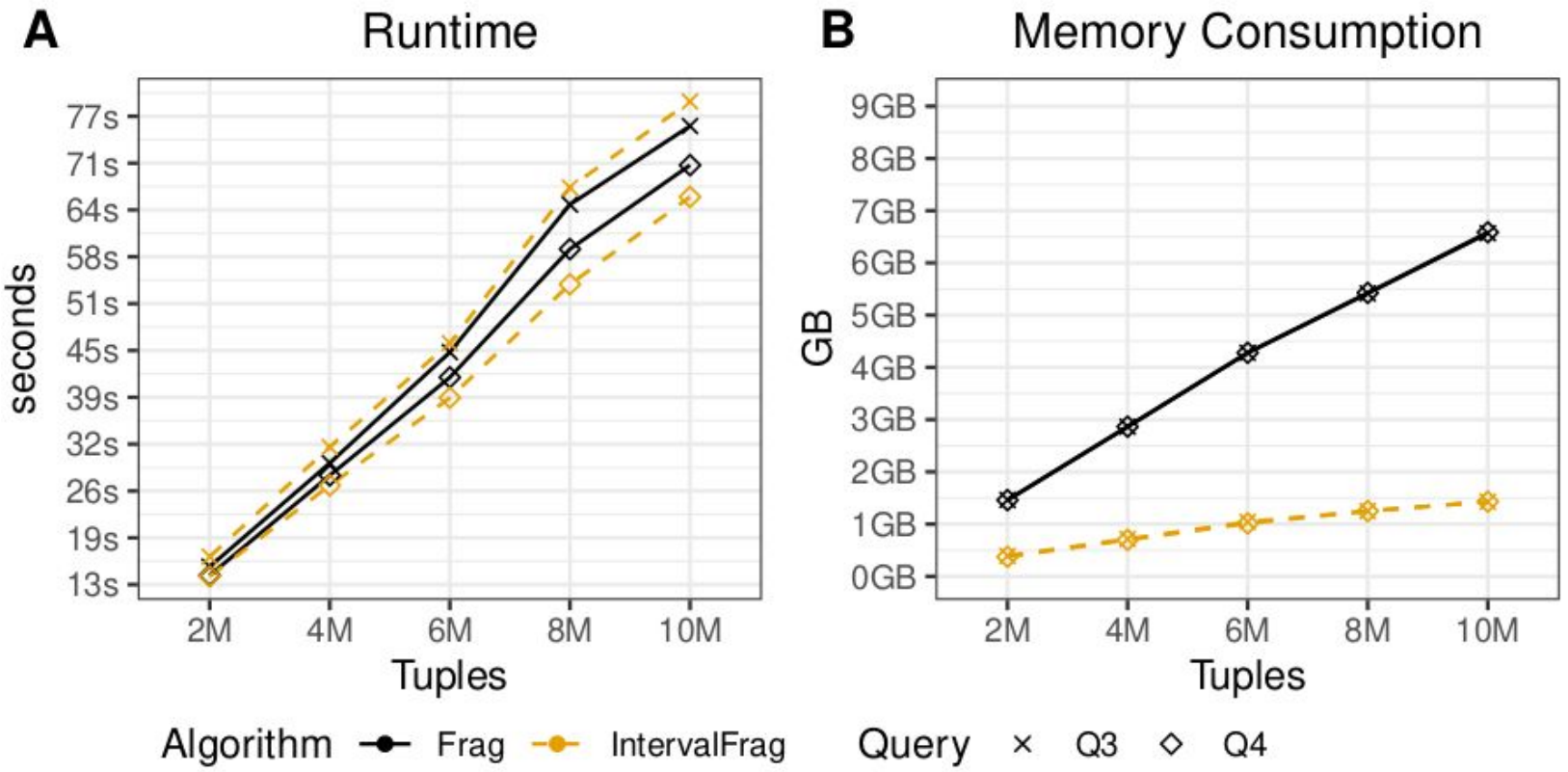
Resultados

- Mesma base de testes do particionando por consulta, com as mesmas consultas Q1 a Q5.
- Diferença é a comparação do C0 apenas, a execução no Frag-Cubing versus a execução no IntervalFrag.
- Mesmos parâmetros foram medidos.

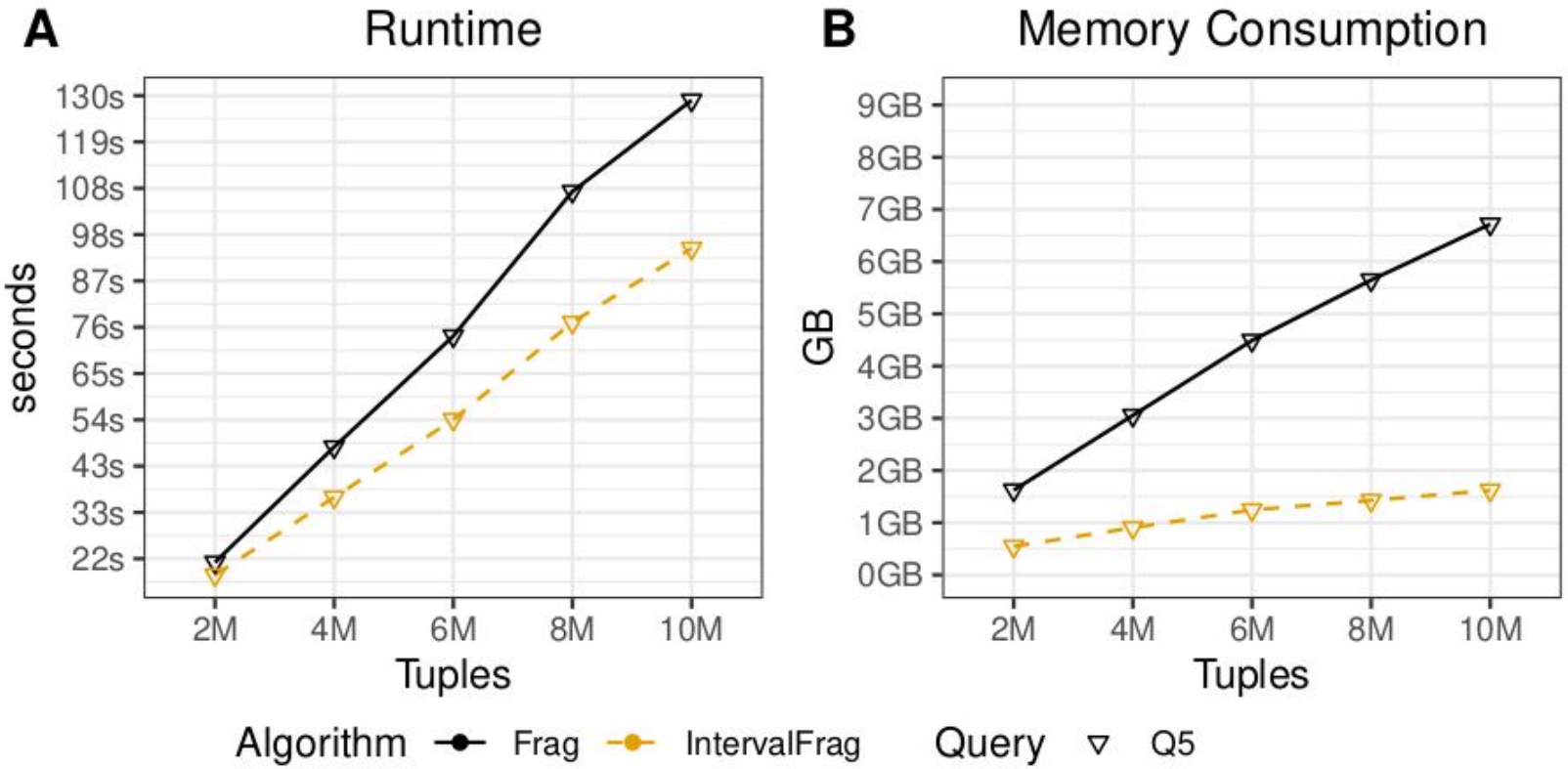
Resultados - Q1 e Q2



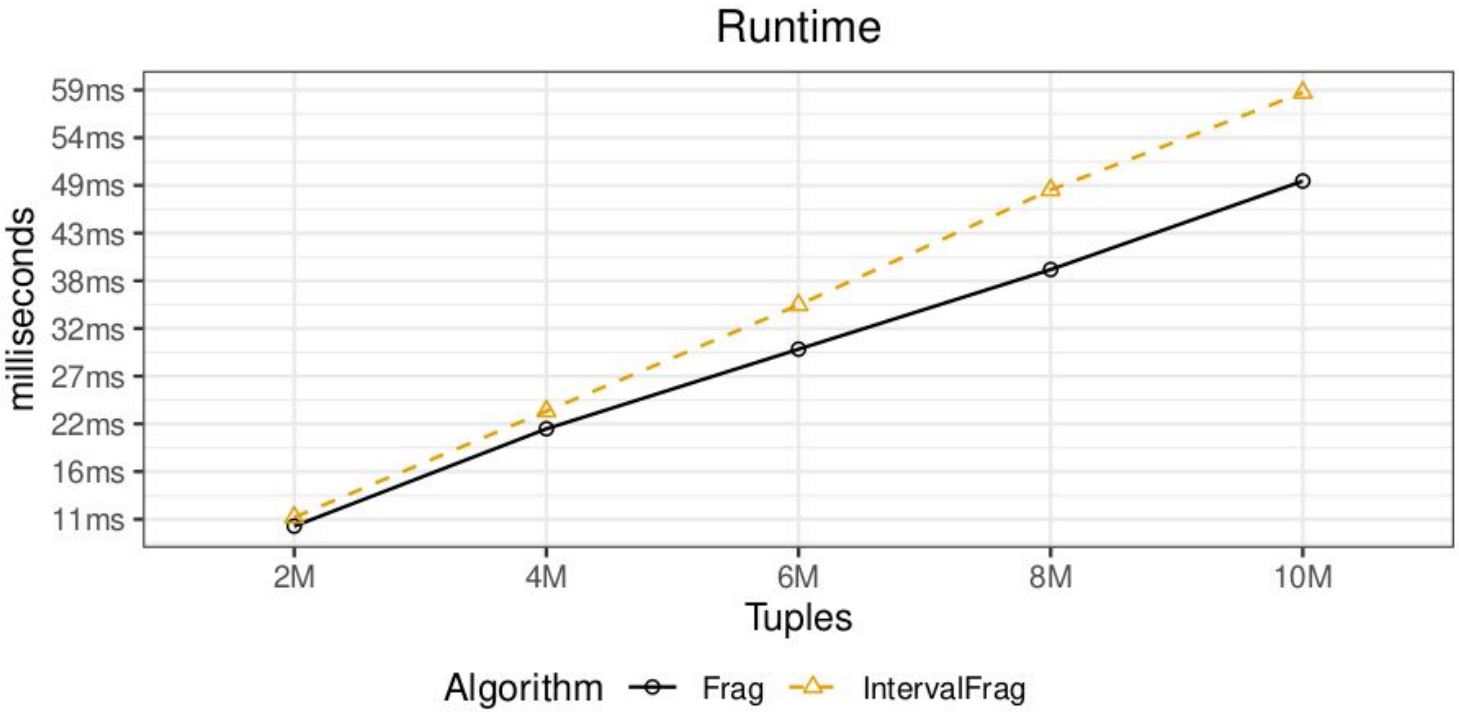
Resultados - Q3 e Q4



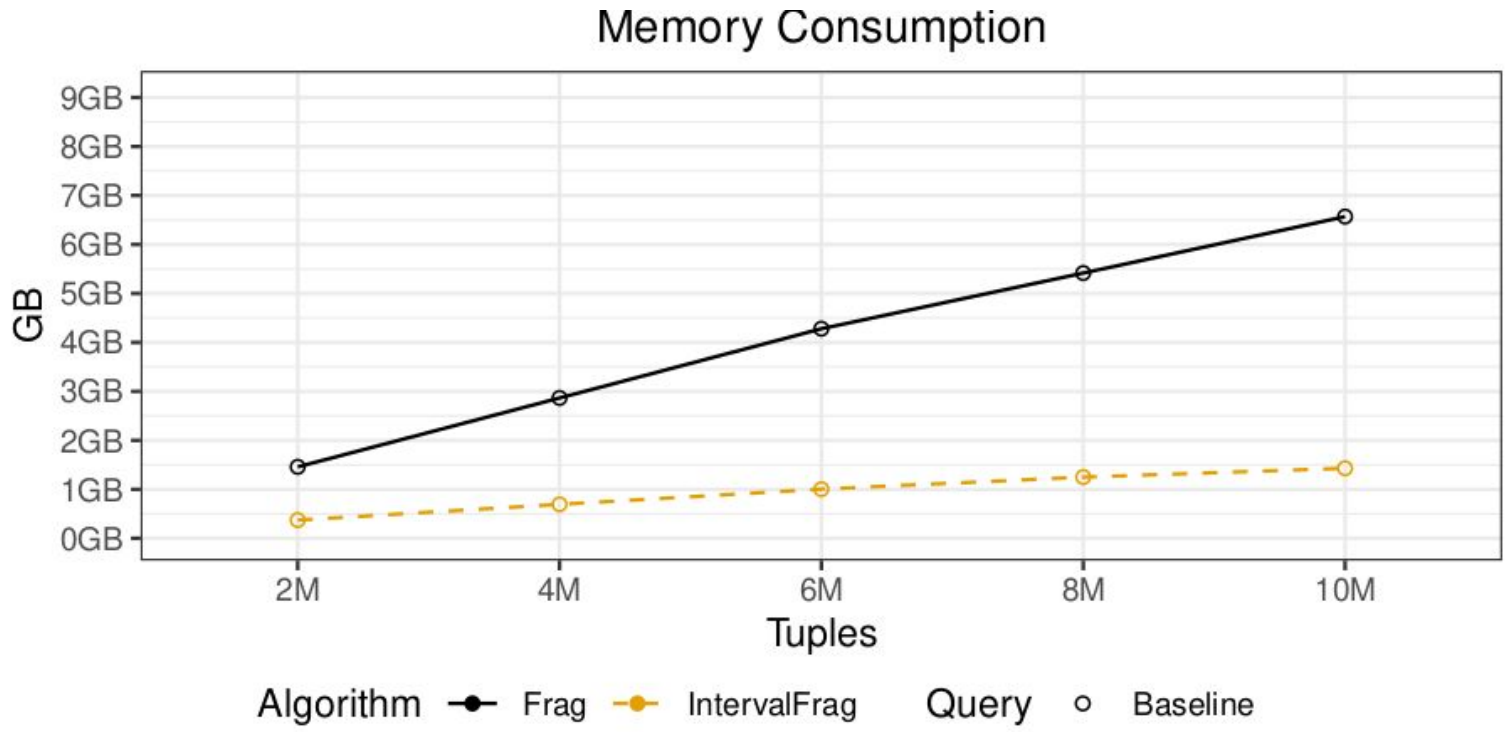
Resultados - Q5



Resultados - Time to Cube



Resultados - Memória base



Sumário

- IntervalFrag pode ser até 3x mais lento que o Frag-Cubing se os dados tiverem uma sequencialidade baixa, e o número de intervalos tender ao número de elementos
- Porém, com alta sequencialidade o IntervalFrag é mais rápido para responder consultas que o Frag-Cubing
- Em todos os casos, IntervalFrag utilizou apenas 20 a 24% da memória utilizada pelo Frag-Cubing
- IntervalFrag foi em média 10% mais lento que o Frag-Cubing para computar o cubo base, porém utilizando apenas 22% da memória utilizada pelo Frag-Cubing

Análise

Table 7.1 - Preferred algorithm to use

	Low Sequentiality	High Sequentiality	High Dimensionality	High Cardinality	High Skew
Computing the base cube	IntervalFrag	IntervalFrag	IntervalFrag	IntervalFrag	IntervalFrag
Subcube query	Frag-Cubing	IntervalFrag	IntervalFrag	Frag-Cubing[1]	FragCubing
Point query	Frag-Cubing	IntervalFrag	IntervalFrag	Frag-Cubing[1]	FragCubing
Low available RAM	IntervalFrag	IntervalFrag	IntervalFrag	IntervalFrag	IntervalFrag
Fast Query Reponse	Frag-Cubing	IntervalFrag	Frag-Cubing[1]	Frag-Cubing[1]	Frag-Cubing

[1]If there's high sequentiality, prefer IntervalFrag

Discussão

- IntervalFrag só é melhor quando as dimensões possuem um alto grau de sequencialidade
- Porém, em todos os casos utiliza muito menos memória que o Frag-Cubing, e deve ser priorizado em situações com pouca memória disponível
- O algoritmo de intersecção do IntervalFrag precisa de mais comparações e executa mais operações, explicando o fato de ser mais lento quando a sequencialidade é baixa

Conclusões

- É possível de utilizar técnicas que utilizam das características do domínio dos dados de telemetria de satélite para reduzir requisitos de implementação de Data Warehouses;
- Dados de telemetria de satélite podem ser modelados em uma arquitetura de análise de dados baseada em cubo de dados para facilitar a análise dos dados históricos

Contribuições

- Um algoritmo para descobrir telemetrias relacionadas em dados de telemetria de satélite, e a validação desse algoritmo com um operador de satélites para selecionar consultas de interesse.
- Utilizando esses resultados para diminuir o consumo de memória e tempo de resposta do Frag-Cubing particionando os dados pelas consultas;
- Criação do IntervalFrag, que melhora o consumo de memória do Frag-Cubing para algumas consultas, utilizando de listas de intervalo para a computação do cubo.
- Uma execução em todos os testes: ~85h, com centenas de horas de testes durante o desenvolvimento

Publicações - 1

- Artigo sobre o particionamento por consultas inicialmente aprovado pelo Information - MDPI
- Retraído devido a falta de verba para publicação

Dear Mr. Dias Pereira,

We are pleased to inform you that the following paper has been officially accepted for publication:

Manuscript ID: information-782822
Type of manuscript: Article
Title: Reducing the memory usage of data cube queries for satellite telemetry data
Authors: Yuri Matheus Dias Pereira *, Mauricio Gonçalves Vieira Ferreira, Rodrigo Rocha Silva

Publicações - 2

- IEEE LATAM: em revisão
- IEEE Systems: em escrita

Name	QUALIS	SCOPUS Percentile	Source	Status
WETE 2018	Conference	-	(PEREIRA et al., 2018)	Published
IAC 2019	Conference	-	(PEREIRA et al., 2019)	Published
IEEE Latin America Transactions	B2	61%	-	Submitted
IEEE Systems Journal	A2	88%	-	Writing

Trabalhos Futuros

- Problema da intersecção de conjuntos pode ser adaptado de acordo com o índice invertido utilizado, e como é uma operação muito frequente, melhorias são sentidas facilmente (ver anexo A)
- Diferentes técnicas de compressão da lista de TIDs podem ser testadas, uma técnica de compressão que permitisse realizar operações algébricas sem descompressão iria melhorar bastante o consumo de memória do algoritmo

Trabalhos Futuros

- O uso de algoritmos de aprendizado de máquina para gerar os relacionamentos de interesse para os operadores, que serviria de entrada para a construção dos cubos
- Adaptar o uso de cubo de dados para constelações de satélites
- Publicação dos dados deste trabalho: algumas ressalvas quanto a segurança do satélite

Obrigado!

Extra

Anexo A

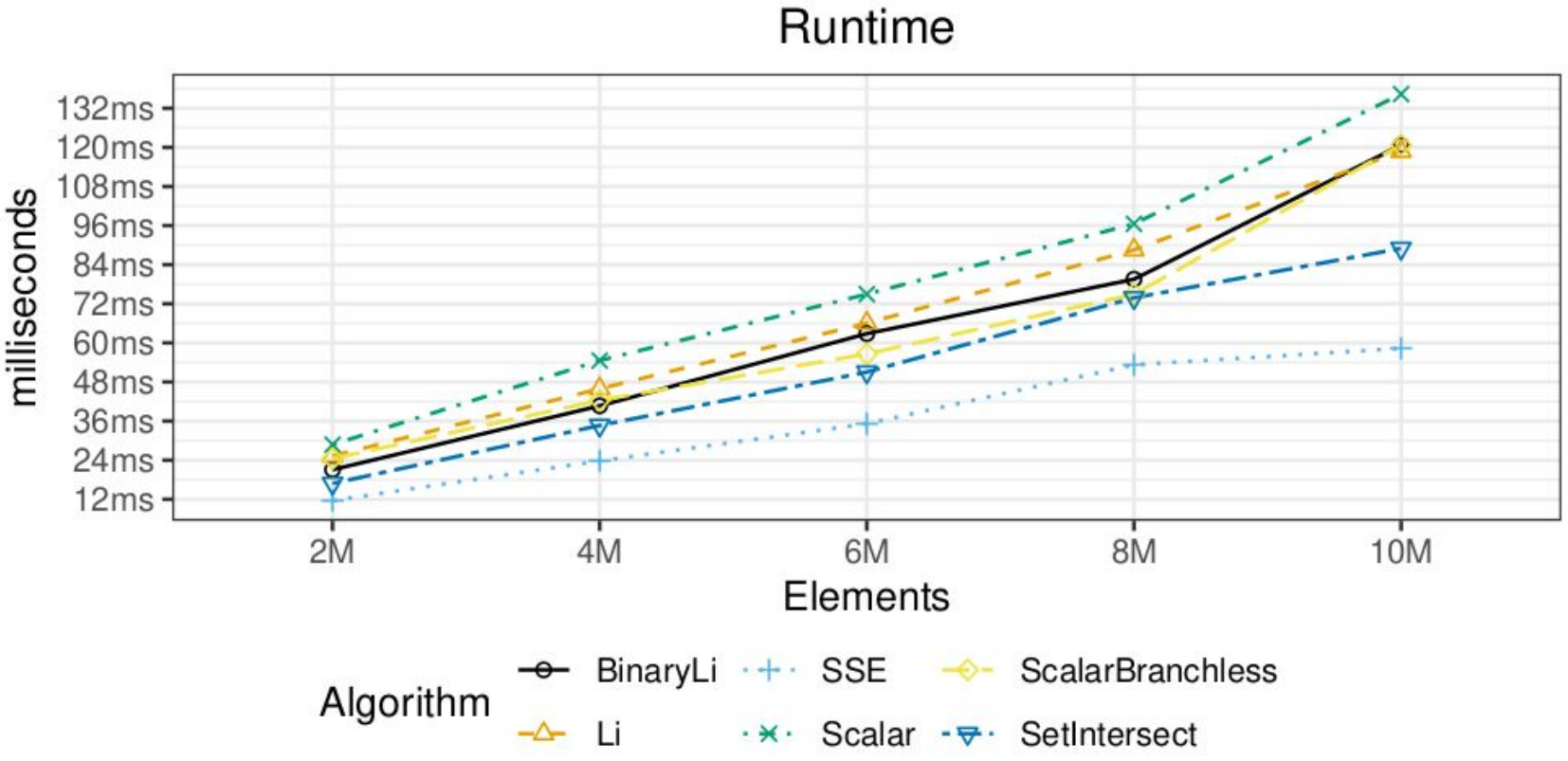
Intersecção de Conjuntos

- Dados conjuntos S1 e S2, encontre os elementos que estão presentes em ambos os conjuntos, representados pela intersecção S1 ∩ S2. Adicionalmente, ambos os conjuntos são ordenados e apenas com elementos únicos dentro de cada conjunto.
- Vários algoritmos foram testados para melhorar essa operação, visto que ela é executada várias vezes para responder consultas e criar os cubos de dados baseline

Intersecção de Conjuntos

- UnorderedSet - Complexidade ótima ($\mathcal{O}(n+m)$ ssar por todos os elementos e adicionar numa tabela hash
- Scalar - Dois ponteiros fazendo uma comparação por elemento em estilo merge
- BranchlessScalar - Versão do algoritmo Scalar, porém sem ramos de decisão
- Li - Versão FragCubing, versão do scalar com lookahead fixo
- BinaryLi - Li com lookahead baseado em busca binária
- std::set_intersect - Implementação do scalar nativa C++
- SSE - Versão do algoritmo Scalar baseada em instruções SIMD, baseada em Inoue et al. (2014) e restrita a SSE2

Intersecção de Conjuntos



Conclusões

- Versão SIMD limitada a implementação SSE2, pode ser adaptada para medir qual conjunto de instruções é mais adequado em tempo de compilação, porém tem tempo melhor que todas as outras
- Algoritmos na literatura requerem pré-processamento anterior dos dados para melhorar a intersecção, todos os algoritmos testados funcionam direto na listas
- SSE2 e SSE3 estão presentes em todas as CPUs feitas desde pelo menos 2009, com implementação diferentes disponíveis, porém que podem ser otimizadas pela arquitetura executora -> “Embarassingly parallel”