MINISTÉRIO DA CIÊNCIA, TECNOLOGIA, INOVAÇÕES E COMUNICAÇÕES
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

DATA CUBE ALGORITHM FOR HIGH SEQUENTIALITY SATELLITE TELEMETRY DATA ANALYSIS

Yuri Matheus Dias Pereira

Dissertação de Mestrado do Curso de Pós-Graduação em Engenharia e Gerenciamento de Sistemas Espaciais. Orientada pelo Dr. Mauricio Gonçalves Vieira Ferreira e pelo Dr. Rodrigo Rocha Silva

URL of the original document:
<http://urlib.net/>

INPE
São José dos Campos
2021

PUBLISHED BY:

Instituto Nacional de Pesquisas Espaciais - INPE
Gabinete do Diretor (GB)
Serviço de Informação e Documentação (SID)
Caixa Postal 515 - CEP 12.245-970
São José dos Campos - SP - Brasil
Tel.:(012) 3945-6923/6921
Fax: (012) 3945-6919
E-mail: pubtc@sid.inpe.br

MINISTÉRIO DA CIÊNCIA, TECNOLOGIA, INOVAÇÕES E COMUNICAÇÕES
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

DATA CUBE ALGORITHM FOR HIGH SEQUENTIALITY SATELLITE TELEMETRY DATA ANALYSIS

Yuri Matheus Dias Pereira

Dissertação de Mestrado do Curso de Pós-Graduação em Engenharia e Gerenciamento de Sistemas Espaciais. Orientada pelo Dr. Mauricio Gonçalves Vieira Ferreira e pelo Dr. Rodrigo Rocha Silva

URL of the original document:
<http://urlib.net/>

INPE
São José dos Campos
2021

Cataloging in Publication Data

Sobrenome, Nomes.

Cutter Data Cube Algorithm for High Sequentiality Satellite Telemetry Data Analysis / Nome Completo do Autor1; Nome Completo do Autor2. – São José dos Campos : INPE, 2021.

xxii + 75 p. ; ()

Dissertação ou Tese (Mestrado ou Doutorado em Nome do Curso) – Instituto Nacional de Pesquisas Espaciais, São José dos Campos, AAAA.

Orientador : José da Silva.

1. Palavra chave. 2. Palavra chave 3. Palavra chave. 4. Palavra chave. 5. Palavra chave I. Título.

CDU 000.000

ATENÇÃO! A FOLHA DE APROVAÇÃO SERÁ INCLUIDA POSTERIORMENTE.

Mestrado ou Doutorado em Nome do Curso

"But I try not to think with my gut. If I'm serious about understanding the world, thinking with anything besides my brain, as tempting as that might be, is likely to get me into trouble. It's OK to reserve judgment until the evidence is in.".

Carl Sagan e Ann Druyan
em *"O Mundo Assombrado pelos Demônios: A Ciência Vista Como Uma Vela no Escuro"*, 1995

ACKNOWLEDGEMENTS

- Mauricio

- Rodrigo

- Família (dos dois lados)

- Bruno e Gabriela

- Italo, Isomar e Danilo

- Comissão WETE e CubeDesign

- Jun, Pascote, Maria (?) e todos do CCS

- Seguranças do INPE (Em especial ao Eduardo)

- Membros do CITAR pelos almoços

- Todos os membros da biblioteca do INPE

Ao INPE e todos os funcionários que proveram todas a infraestrutura necessária para este trabalho.

- CAPES

- Secretaria de Pós-Graduação?

ABSTRACT

Satélites são monitorados pelas equipes de solo via pacotes de telemetria, que informam o estado atual dos equipamentos e permitem avaliar a capacidade do satélite de continuar a sua missão. Esses pacotes de telemetria constituem um corpo de dados de tamanho e alta complexidade, sendo que satélites que operados por vários anos geram dados históricos de grande volume, ainda úteis para as atividades de operação. O volume de dados históricos de telemetria disponíveis ao INPE atualmente é estimado em ao menos 3 *terabytes* no total, com tendência a crescer nos próximos anos. Esta proposta apresenta o uso de cubo de dados como solução para executar consultas e análises sobre esses dados. Os conceitos da área de cubo de dados são apresentados, bem como uma revisão de como outros operadores de satélite estão lidando com grandes volumes, variedades e velocidade de atualização de dados, cenário que define um contexto de *Big Data* para o domínio de controle de satélites. Devido a característica de alta dimensionalidade dos dados de telemetria, algoritmos clássicos da aŕea do cubo de dados tem dificuldade em responder consultas com resultado satisfatório para os operadores de satélite. Assim, neste trabalho é proposto identificar as consultas que são de interesse dos operadores de satélite, criar uma modelagem multidimensional para os dados de telemetria utilizando de cubo de dados, e avaliar quais são os algoritmos de construção do cubo que conseguiriam suprir as necessidades dos dados. Também são apresentados os resultados alcançados até o momento, bem como o planejamento para a continuação do trabalho.

Keywords: cubo de dados. *Big Data*. Satélites. Telemetrias. Operação de Satélites.

DATA CUBE ALGORITHM FOR HIGH SEQUENTIALITY SATELLITE TELEMETRY DATA ANALYSIS

RESUMO

Abstract

Palavras-chave: Atmospheric turbulence. WETAMC campaign. LBA project. Chaotic behavior. Chaotic attractor.

LIST OF FIGURES

Page

LIST OF TABLES

Page

LIST OF ABBREVIATIONS

DW	–	*Data Warehouse* (Armazém de Dados)
OLAP	–	*On-Line Analytical Processing* (Processamento Analítico Online)
OLPT	–	*On-Line Transaction Processing* (Processamento Online de Transações)
NoSQL	–	"Não apenas SQL"
TAD	–	Tipo Abstrato de Dados
ROLAP	–	*Relational OLAP*
MOLAP	–	*Multidimensional OLAP*
HOLAP	–	*Hybrid OLAP*
DBMS	–	*DataBase Management System*
TLE	–	*Two Line Element*
TID	–	*Tuple Identifier* (Identificador de Tupla)
CSV	–	Valores Separados por Vírgula
INPE	–	Instituto Nacional de Pesquisas Espaciais
CCS	–	Centro de Controle de Satélites
SCD	–	Satélite de Coleta de Dados
CBERS	–	Satélite Sino-Brasileiro de Recursos Terrestres
AMZ	–	Amazônia
NASA	–	*National Aeronautics and Space Administration*
NOAA	–	*National Oceanic and Atmospheric Administration*
L-3	–	*Level 3*
ESA	–	*European Space Operations Centre*
EUMETSAT	–	*European Organisation for the Exploitation of Meteorological Satellites*
AWS	–	*Amazon Web Services*
HDFS	–	*Hadoop Distributed FileSystem*
CSMT	–	*China Satellite Marine Track & Control Department*
SISET	–	*Shandong Institute of Space Electronic Technology*

LIST OF SYMBOLS

a	–	primeira contante
b	–	segunda constante
ρ	–	densidade de um fluido
ν	–	viscosidade cinemática
R_e	–	número de Reynolds
α	–	constante de Kolmogorov
k	–	número de onda
K	–	curtose
D_0	–	dimensão de contagem de caixas
D_1	–	dimensão de informação
D_2	–	dimensão de correlação
λ_1	–	expoente de Lyapunov dominante

CONTENTS

Page

1 INTRODUÇÃO

O Centro de Controle de Satélites (CCS) é um departamento pertencente ao Instituto Nacional de Pesquisas Espaciais (INPE) atualmente monitora e controla os seguintes satélites: a família do Satélite de Coleta de Dados (SCD), composta de dois satélites SCD-1 e SCD-2, e a família do Satélite Sino-Brasileiro de Recursos Terrestres (CBERS), com apenas o quinto satélite em operação atualmente, o CBERS-4. Estes satélites realizam passagens sobre as estações terrenas do INPE, durante o qual o CCS recebe dados do estado do satélite, chamados de telemetrias, e envia telecomando, utilizados para controlar o satélite, bem como realiza atividades de manutenção e estimativa, como medidas de velocidade e posição de cada satélite (AZEVEDO; AMBRÓSIO, 2010).

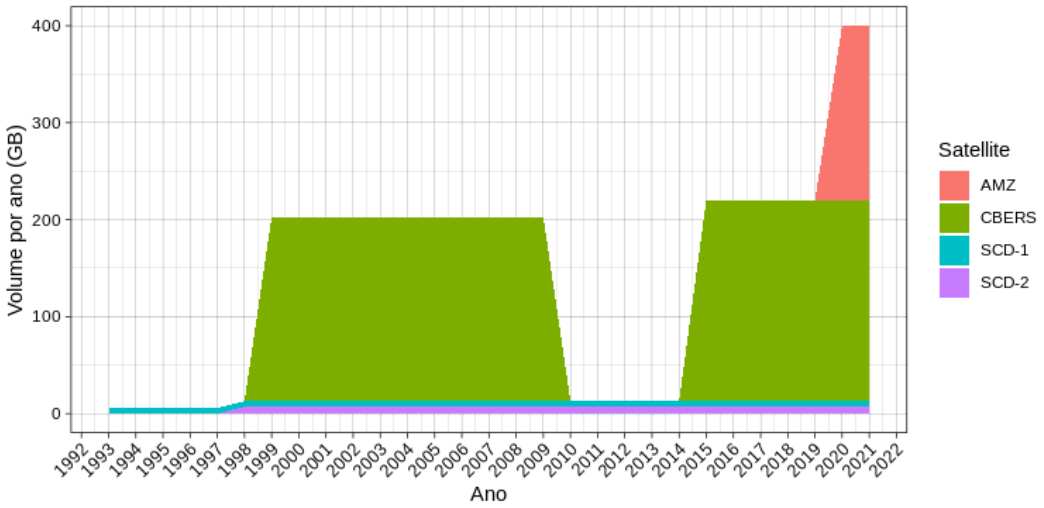
Dados de telemetria geralmente carregam medidas de sensores e verificações de saúde dos instrumentos, como temperatura das baterias, corrente de algum subsistema, se um dado equipamento está ativo ou não, bem como dados que os operadores e engenheiros acham necessários para a operação, entre outros (LARSON; WERTZ, 1999). Estes dados precisam ser guardados por toda a vida do satélite, sendo que para satélites que estão em funcionamento por vários anos adquirem um elevado volume de dados, que deve ser analisado. No caso dos satélites da família SCD, o SCD-1 já estando operacional por mais de 25 anos, e continuando a gerar dados, atualmente gera um volume aproximado de 7GB por ano.

Para satélites mais complexos como os da família CBERS, que possuem mais de 4 mil telemetrias sendo monitoradas. Com os lançamentos futuros do CBERS-4A e do Amazônia-1, o volume de dados e a complexidade da análise dos mesmos deve aumentar, criando novas necessidades de operação (FILHO et al., 2017).

A figura 1.1 mostra uma uma estimativa simples da geração histórica de dados de telemetria no CCS. Essa estimativa foi feita utilizando dos dados não compressos a partir da disponibilidade dos mesmos. Ela também assume que o Amazônia-1 vai gerar um volume de dados de telemetria similar ao gerado do CBERS.

Dessa estimativa, obtemos o total de dados de telemetria disponíveis para a análise no CCS considerando uma taxa constante dos satélites, apresentados na figura 1.2. É importante ressaltar que a grande maioria desses dados não está disponível para consulta pelo usuário, visto que somente os dados de alguns poucos anos da operação estão disponíveis para os operadores e engenheiros, necessitando de trabalho significativo para analisar dados do passado.

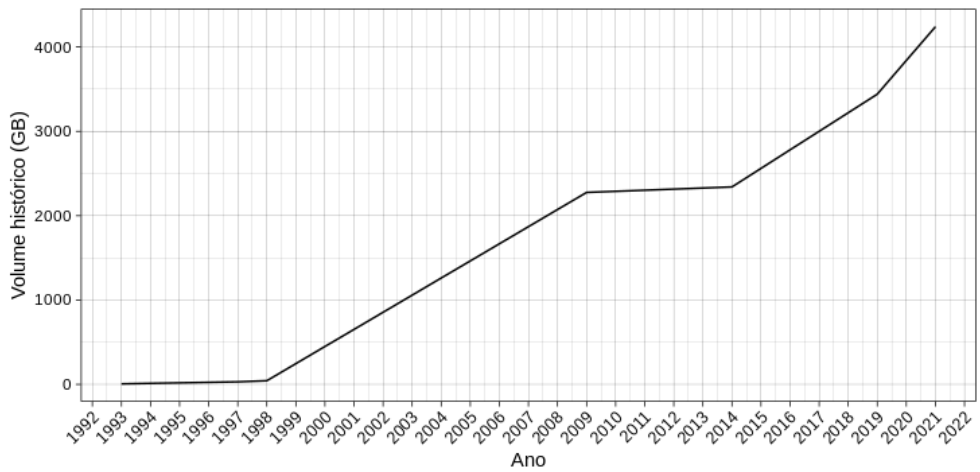
Figure 1.1 - Estimativa de geração anual de dados pelos satélites do INPE



Volume estimado de geração de dados por cada ano de operação de cada satélite.
SOURCE: Produção do autor.

Esses dados devem ser propriamente tratados para que não virem "*dark data*", termo que denota quaisquer tipo de dados que não são de fácil acesso para os seus usuários em potencial (HEIDORN, 2008).

Figure 1.2 - Estimativa do volume de dados histórico de telemetria de todos os satélites



Volume total estimado de dados de telemetria gerados por todos os satélites.
SOURCE: Produção do autor.

Esses dados entram na definição de *Big Data*, pois possuem um grande volume, são gerados continuamente, possuem formatos diversos, sua análise é de alto valor e existe uma incerteza quanto a qualidade dos dados devido a problemas de comunicação e degradação dos instrumentos. Essas características são denotadas pelos cinco Vs do *Big Data*: Volume, Variedade, Velocidade, Valor e Veracidade (EMANI et al., 2015).

Considerando que todos os dados já estivessem no banco de dados, propriamente formatados e prontos para a análise, ainda restariam grandes problemas: com um banco de dados na ordem dos *terabytes*, consultas sobre um número elevado de telemetrias ou que precisem de dados de vários anos poderiam demorar dias, ou mais, para serem executadas.

Deste modo, é necessário criar uma estrutura que permita a análise e consulta desses dados de uma forma estruturada e que tenha desempenho satisfatório. As tecnologias de *Data Warehouse* (DW) e *Online Analytical Processing* (OLAP) tem demonstrado capacidade e experiência para atingir esses objetivos (BIMONTE, 2016), inclusive na área espacial (YVERNES, 2018). Essas tecnologias executam a generalização de dados agregando enormes quantidade de dados em vários níveis de abstração, assim tornam elementos essenciais de apoio à decisão e atraem a atenção tanto da indústria como das comunidades de pesquisa. Sistemas OLAP, que são tipicamente dominados por consultas complexas que envolvem operadores *group-by* e operadores de agregações, são as principais características entre essas ferramentas.

Sistemas OLAP são baseados em um modelo multidimensional chamado de cubo de dados, que é uma generalização do operador *group-by* sobre todas as combinações possíveis das dimensões, com variados níveis de granularidade (GRAY et al., 1996). Cada dimensão é uma perspectiva de decisão sobre os dados, sendo formada por um subconjunto de atributos. Cada combinação é chamada de um subcubo, que correspondem a um conjunto de células descritas como tuplas sobre as dimensões do subcubo. Além das dimensões, cada tupla contém um fato, também chamado de medida, que representa o que será medido no processo de análise.

Cada dimensão pode estar organizada em uma hierarquia para facilitar a análise. Por exemplo, uma dimensão tempo pode ser dividida em "dia $<$ mês $<$ ano", com ano sendo o nível mais genérico. Essa prática visa facilitar a interpretação dos dados pelos usuários. Medidas são atributos atributos associados a uma combinação de dimensões, sendo geradas de forma estatística.

Tecnologias OLAP são caracterizadas pela habilidade em responder consultas de apoio a decisão de forma eficiente (HAN et al., 2011). Para atingir isso, o cubo de dados deve ser materializado antes da execução da consulta. Isso significa que as combinações de dimensões são computadas previamente, assim gerando o cubo de dados completo. Porém, essa abordagem possui um custo computacional exponencial em relação ao número de dimensões, assim a materialização completa do cubo envolve um grande número de células e um tempo substancial para a sua execução.

Dados de satélite são caracterizados pela sua alta dimensionalidade, onde um satélite pode precisar rastrear milhares de telemetrias. Por exemplo, supondo um satélite com $n = 100$ telemetrias, e cada telemetria representando uma dimensão, teremos 2^{100} possíveis subcubos para a implementação de um cubo de dados. Supondo uma cardinalidade, o número de valores diferentes em cada telemetria, como sendo de 100, teremos $101^{100} \approx 10^{200}$ células para cada dimensão. Devido ao controle ativo pelos operadores de satélite, os dados são concentrados em alguns valores que se repetem frequentemente, sendo que isso é chamado de *skew*.

Dessa forma, conseguir calcular e manter um cubo de dados é um problema exponencial, e reduzir o seu consumo de memória e tempo de computação é de fundamental importância para desenvolver um sistema OLAP. Para a área espacial essa necessidade é maior: a maior parte dos algoritmos de computação do cubo tem problemas em lidar com mais do que 15 dimensões (SILVA, 2015).

1.1 Objetivos

Assim, este trabalho tem como objetivo estabelecer um método para processamento de cubos de dados para a área espacial, para que o processamento de consultas OLAP sejam executadas de forma eficiente considerando-se a alta dimensionalidade, elevado número de tuplas, alto *skew* e alta cardinalidade dos dados.

Assim é necessário identificar quais são as consultas de interesse dos operadores de satélite e quais são as análises que devem ser feitas pelos mesmos. Disso será criada uma representação dimensional dos dados de telemetria em uma estrutura do cubo de dados, e algoritmos de construção do cubo devem ser avaliados para identificar qual é o mais apropriado para responder as consultas.

Como resultados esperados deste trabalho teremos a avaliação dos algoritmos de construção do cubo nos dados de alta dimensionalidade, com a adequabilidade do uso de cubo de dados como uma solução para operadores executarem as consultas

analíticas mais estratégicas para as operações de satélites.

1.2 Organização da proposta

Os capítulos restantes desta proposta estão organizados da seguinte maneira:

- Capítulo 2: Este capítulo apresenta os conceitos e fundamentos teóricos desta proposta, como os conceitos relevantes de operação de satélites, *Data Warehouse*, *Big Data* e cubo de dados.
- Capítulo 3: Neste capítulo os trabalhos correlatos de cubo de dados são apresentados, bem como as arquiteturas que outros operadores de satélite estão implementando.
- Capítulo 4: Neste capítulo a proposta é apresentada e seus conceitos principais explicados.
- Capítulo 5: Esse capítulo apresenta os resultados alcançados até o momento, apresentando os *software* utilizados.
- Capítulo 6: Com base nos resultados intermediários alcançados, esse capítulo apresentará as conclusões obtidas, bem como as direções de implementação para o resto do trabalho.

2 FUNDAMENTAÇÃO

Este capítulo apresenta os conceitos fundamentais relacionados a essa proposta, começando pela operação dos satélites na seção 2.1, apresentando a definição de *Big Data* na seção 2.2, e os os conceito de *Data Warehouse* na seção 2.3, *OLAP* na seção 2.4 e cubo de dados na seção 2.5.

2.1 Operação de satélites

Um satélite é dividido em dois módulos: o módulo de serviço e a carga útil. O módulo de serviço compõe tudo necessário para o funcionamento dos equipamentos de bordo, como o sistema de geração de energia, o sistema de comunicação com o solo, o computador de bordo, etc. A carga útil compõe todos os equipamentos necessários para cumprir os objetivos da missão, sendo esses sensores, câmeras, telescópios, etc (LARSON; WERTZ, 1999).

Um satélite gera dois tipos diferentes de dados: dados da carga útil e dados de telemetria. Os dados da carga útil são os dados gerados para cumprir a missão do satélite, sendo que eles podem ser fotos tiradas para o sensoriamento remoto, fotos tiradas por telescópios, dados de comunicação caso este seja o foco da missão entre outros (LARSON; WERTZ, 1999). Os dados de telemetria são os dados de monitoramento do estado de saúde e do funcionamento dos equipamentos do satélite. Esses dados são coletados pelo computador de bordo do satélite, e são enviados para as estações de solo via sistemas de telecomunicação.

Os dados de telemetria compõe usualmente medidas de sensores nos equipamentos do satélite, informações coletadas pelo computador de bordo (como se um instrumento está ligado ou não), e outros dados cuja coleta foi definida como relevante para a operação do satélite. Dependendo da missão, outras medidas podem ser classificadas como telemetria, como por exemplo câmeras voltadas para o satélites, radares para a detecção de possíveis colisões, etc (KRAG et al., 2017).

Esses dados devem ser analisados pelos operadores de satélite, que são os responsáveis pelo monitoramento e operação do satélite, em solo após recebimento no centro de controle. Essa análise visa garantir que o satélite está executando suas tarefas como deveria, e que o seu estado de saúde permite a continuação da missão. Neste trabalho, será utilizada a análise feita pelos operadores de satélite somente nos dados de telemetria, que é uma análise não trivial dadas as características dos dados, classificados como *Big Data*.

2.2 *Big Data*

A aplicação do conceito de *Big Data* vem evoluindo ao longo dos anos, e para este trabalho será utilizada a definição dos *5 Vs*: Volume, Variedade, Velocidade, Valor e Veracidade (EMANI et al., 2015). Em detalhes:

- **Volume**: esse termo geralmente especifica uma quantidade de dados em que um sistema tradicional de gerenciamento de banco de dados é ineficaz. É importante ressaltar que isso não se trata apenas do armazenamento dos dados, mas também do seu processamento (BOUSSOUF et al., 2018). Usar um grande volume de dados geralmente implica em modelos melhores, que então produzem análises melhores, justificando a coleta de uma grande quantidade de dados.

- **Variedade**: dados são provenientes de fontes diferentes, com formatos diferentes, sem um esquema de modelagem padronizado, como dados advindos de *logs* de computadores, dados de sensores, dados multimídia, etc. Como consequência, esses dados devem ser utilizados da forma mais transparente o possível na análise.

- **Velocidade**: dados são disponibilizados de uma forma muito rápida, e devem ser analisados da forma mais rápida o possível. Isso implica que os dados podem ser guardados e analisados até em tempo real.

- **Valor**: os dados devem ser armazenados para criar algum valor para os seus usuários, seja ele econômico, científico, social, organizacional, etc.

- **Veracidade**: os dados não possuem garantias quanto a sua qualidade, como inconsistências e falta de acurácia, porém a análise deve ser de alta qualidade de qualquer forma.

Esses V's estão relacionados com a construção de um *Data Warehouse*, sendo que também podem ser vistos como requisitos para a crição de um para um conjunto de dados caracterizado como *Big Data* (ZHANG et al., 2017). Em especial, existe um certo relacionamento com a ideia de "*NoSQL*" ("Não apenas SQL", em inglês), em que não apenas sistemas de banco de dados relacionais são utilizados, mas também outros paradigmas são utilizados, como orientados a documentos, chave e valor, etc (BIMONTE, 2016).

2.3 *Data Warehouse*

Um Armazém de Dados ou Data Warehouse (DW) é um repositório de dados orientado por assunto, integrado, variado ou particionado em função do tempo e não volátil, que auxilia no gerenciamento do processo de tomada decisões (INMON; HACKATHORN, 1994). Essa definição pode ser dividida em:

- **Orientado por assunto**: o DW é utilizado para a análise de uma área em específico. Por exemplo, é de interesse analisar especialmente os dados da carga útil de uma forma específica.
- **Integrado**: o DW deve integrar dados vindos de múltiplas fontes de uma forma estrutura. Por exemplo, mesmo que existam duas representações diferentes para um mesmo produto, o DW deve possuir apenas uma representação. Isso requer o uso de técnicas de limpeza e integração dos dados, de modo a garantir a consistência dos dados.
- **Variado em função do tempo**: o DW deve conter, explícita ou implicitamente a perspectiva de tempo. Isso quer dizer que o DW possui dados históricos e eles podem ser consultados durante a análise. Por exemplo, pode se querer saber de dados de dias, meses ou anos atrás.
- **Não volátil**: uma vez dentro do DW, os dados não são removidos ou atualizados, sendo um requisito para a consulta de dados históricos.

Essas características diferem o *Data Warehouse* de outros sistemas de repositório, como sistemas de banco de dados, sistemas de processamento de transações e sistemas de arquivos (HAN et al., 2011).

Um DW é geralmente representado por um modelo dimensional que permite eficiência na organização dos dados e na recuperação de informações gerenciais (KIMBALL; ROSS, 2013). Neste modelo são definidos fatos, dimensões e medidas. Um fato corresponde ao assunto de negócio a ser analisado, cada dimensão é uma perspectiva de visualização do assunto de negócio e medidas são valores numéricos que quantificam o assunto de negócio. Uma das dimensões é sempre temporal para permitir a análise do assunto ao longo do tempo (SILVA, 2015).

2.4 *OLAP*

On-line Analytical Processing (OLAP) é um termo que se refere a um conjunto de ferramentas que são utilizadas para resumir, consolidar, visualizar, aplicar formu-

lações e sintetizar dados de acordo com múltiplas dimensões (CODD et al., 1998).

Um sistema OLAP permite a resposta de consultas multidimensionais usando dados armazenados no *Data Warehouse* (KIMBALL; ROSS, 2013), sendo que as características principais são (BIMONTE, 2016):

- **Consultas Online**: as consultas devem ser feitas *Online*, isto é, em tempo real para o usuário.
- **Consultas Multidimensionais**: Consultas são definidas utilizando as dimensões e medidas providas pelo *Data Warehouse*, que esperam dados de alta qualidade.
- **Representação simples**: os resultados das consultas devem ser representados utilizando tabelas e gráficos, pois os usuários finais geralmente são tomadores de decisão que precisam de visualizações relevantes.
- **Exploratórias**: as consultas são utilizadas em carácter exploratório, pois geralmente os usuários não conhecem de antemão todos os dados disponíveis para consultas.

Cada ferramenta OLAP deve manipular um novo tipo abstrato de dados (TAD), chamado de cubo de dados, utilizando estratégias específicas devido ao modo de como os dados são armazenados, sendo classificadas em (MOREIRA; LIMA, 2012):

- ***Relational OLAP* (ROLAP)**: utilizam Sistemas de Gerenciamento de Banco de Dados (*Data base Management System* - DBMS) relacionais para o gerenciamento e armazenamento dos cubos de dados. Ferramentas ROLAP incluem otimizações para cada DBMS, implementação da lógica de navegação em agregações, serviços e ferramentas adicionais;
- ***Multidimensional* OLAP (MOLAP)**: implementam estruturas de dados multidimensionais para armazenar cubo de dados em memória principal ou em memória externa. Não há utilização de repositórios relacionais para armazenar dados multidimensionais e a lógica de navegação já é integrada a estrutura proposta;
- ***Hybrid* OLAP (HOLAP)**: combinam técnicas ROLAP e MOLAP, onde normalmente os dados detalhados são armazenados em base de dados relacionais (ROLAP), e as agregações são armazenadas em estruturas de dados multidimensionais (MOLAP).

É importante ressaltar a diferença entre OLAP e *Online Transaction Processing* (OLPT), visto que sistemas comuns de banco de dados utilizam apenas OLTP, que tem o objetivo de realizar transações e processar consultas online. Isso cobre a grande maioria das operações do dia a dia, como controle de estoque, operações bancárias, etc, servindo a diversos usuários de uma organização. Já o OLAP é utilizado por tomadores de decisão e analistas de dados, sendo voltado para decisões de mais alto nível na organização (HAN et al., 2011).

2.5 Cubo de dados

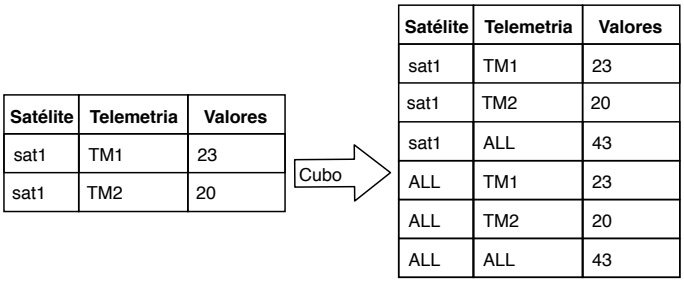
O cubo de dados originalmente foi criado como um operador relacional que gera todas as combinações possíveis de seus atributos de acordo com uma medida (GRAY et al., 1996).

A estrutura do cubo de dados permite que os dados sejam modelados e visualizados em múltiplas dimensões, e ele é caracterizado por dimensões e medidas. Uma medida é um atributo cujos valores são calculados pelo relacionamento entre as dimensões, sendo que esse é calculado utilizando funções de agragação como soma, quantidade, média, moda, mediana, etc. Uma dimensão é feita pelas entidades que compõe os nossos dados, determinando o contexto do assunto em questão (HAN et al., 2011). Uma dimensão pode ainda ser dividida em membros, que podem ter uma hierarquia, como uma divisão da dimensão tempo em dia, mês e ano.

A organização de um cubo de dados possibilita ao usuário a flexibilidade de visualização dos dados a partir de diferentes perspectivas, já que o operador gera combinações através do conceito do valor ALL, onde este conceito representa a agregação de todas as combinações possíveis de um conjunto de valores de atributos. Operações em cubos de dados existem a fim de materializar estas diferentes visões, permitindo busca e análise interativa dos dados armazenados (HAN et al., 2011).

Um cubo de dados é composto por células e cada célula possui valores para cada dimensão, incluindo ALL, e valores para as medidas. A figura 2.1 mostra um exemplo de um cubo de dados. O valor de uma medida é computado para uma determinada célula utilizando níveis de agregação inferiores para gerar os valores dos níveis de agregação superiores na estratégia *Top-down*, com a ordem inversa sendo a *Bottom-up*.

Figure 2.1 - Exemplo de um cubo de dados

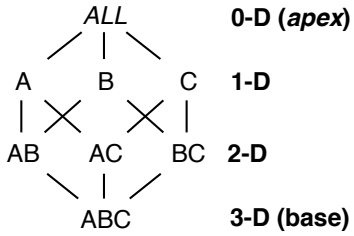


SOURCE: Produção do autor.

2.5.1 Células do cubo de dados

Um cubo de dados é composto de vários subcubos, que são todos os possíveis níveis de agregação nas dimensões especificadas. Subcubos são compostos de células base e células agregadas, sendo uma célular agregada é uma célula que utiliza do valor especial ALL ("$*$") para demonstrar que está agregando valores em uma ou mais dimensões. Uma célula base não utiliza da notação ALL, sendo composta do nível mais baixo de agregação (LIMA, 2009). A figura 2.2 demonstra todos os níveis de agregação de um cubo composto das dimensões A, B e C, do mais genérico (*apex*) ao mais específico(base).

Figure 2.2 - Células de agregação em um cubo de dados



SOURCE: Produção do autor.

Formalmente, supondo um cubo de dados n-dimensional, uma célula a de qualquer subcubo é definida por $a = (a_1, a_2, a_3, \ldots, a_n, medidas)$. A célula é m-dimensional (de um subcubo com m dimensões), se exatamente m, com $(m \leq n)$, valores entre $(a_1, a_2, a_3, \ldots, a_n)$ não são "$*$". Se $m = n$, então a é uma célula base, caso contrário

$(m < n)$, ela é uma célula agregada.

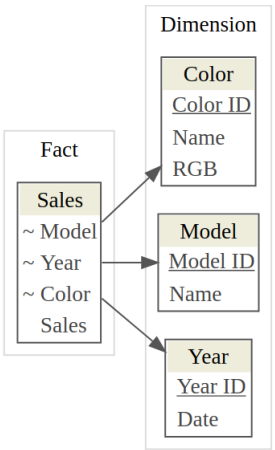
Um relacionamento de descendente-ancestral pode existir entre células. Em um cubo de dados n-dimensional, uma célula $a = (a_1, a_2, a_3, \ldots, a_n, medidas_a)$ de nível i é um ancestral de uma célula $b = (b_1, b_2, b_3, \ldots, b_n, medidbs_b)$ de nível j, e b é um descendente de a, se e somente se $i < j$ e $1 \leq m \leq n$, onde $a_m = b_m$ sempre que $a_m \neq *$. Em particular, uma célula a é chamada de pai de uma célula b, e b de filho de a, se e somente se $j = i + 1$ e b for um descendente de a (HAN et al., 2011).

2.5.2 Modelagem dimensional

Existem três esquemas principais para a modelagem dimensional de um cubo de dados: Esquema Estrela (*Star Schema*), Esquema Floco de Neve (*Snowflake Schema*) e Constelação de Fatos (*Fact Constellation Schema*).

O esquema estrela é o mais utilizado, sendo que ele contém uma tabela central chamada de tabela de fatos, onde reside a maior parte dos dados, com um conjunto menor de tabelas, chamadas de tabelas de dimensão, para as outras dimensões. A figura 2.3 mostra um exemplo de esquema estrela.

Figure 2.3 - Esquema estrela

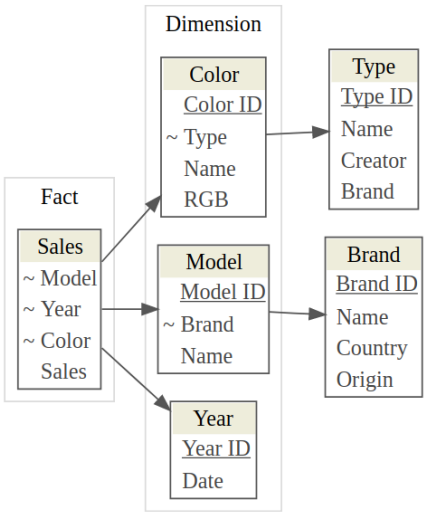


SOURCE: Produção do autor.

O esquema floco de neve é uma variação do esquema estrela, onde algumas dimensões são normalizadas, dividindo os dados das tabelas de dimensão em outras tabelas. Isso possui vantagens de eliminar redundâncias nas tabelas de dimensão, porém cria

problemas durante a execução de consultas, visto que é necessário realizar operações de *join* com as novas tabelas. A figura 2.4 mostra um exemplo de esquema floco de neve.

Figure 2.4 - Esquema floco de neve



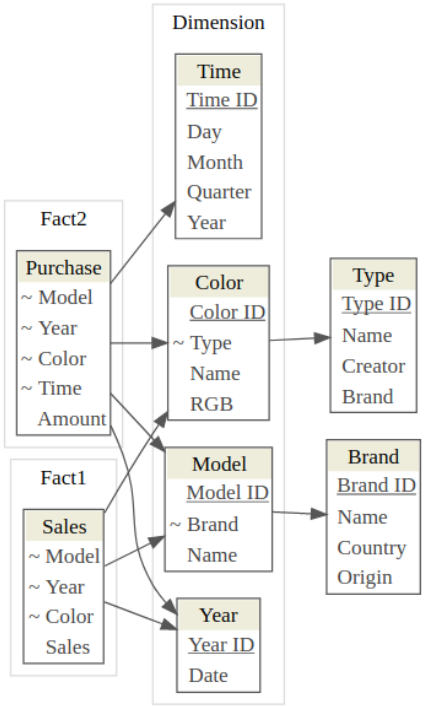
SOURCE: Produção do autor.

O esquema constelação de fatos utiliza de múltiplas tabelas de fato, como se fossem várias tabelas no esquema estrela que compartilham tabelas de dimensão. Isso leva ao seu nome, como um conjunto de estrelas. A figura 2.5 mostra um exemplo de constelação de fatos.

2.5.3 Hierarquias de conceito

Uma hierarquia de conceitos é utilizada para definir uma sequência de mapeamento entre um conjunto de conceitos de baixo nível para um conjunto de conceitos de alto nível, mais gerais. É um estilo de agrupamento e discretização, pois agrupa os valores de modo a reduzir a cardinalidade de uma dimensão (HAN et al., 2011). Elas ajudam a tornar a análise mais fácil de ser entendida, pois as operações traduzem os dados de baixo nível em uma representação que é mais fácil para o usuário final, assim facilitando a execução das consultas e o seu subsequente uso.

Figure 2.5 - Esquema constelação de fatos



SOURCE: Produção do autor.

2.5.4 Medidas

Cada célula de um cubo é definida como um par $\langle (d_1, d_2, \ldots, d_n), medidas \rangle$, onde $(d_1, d_2, \ldots, d_n)$ representam as combinações possíveis de valores de atributos sobre as dimensões. Uma medida é calculada para uma certa célula agregando os dados correspondentes a combinação de dimensões e valores (HAN et al., 2011). Medidas podem ser classificadas em três tipos: distributiva, algébrica e holística.

Uma medida distributiva é uma medida cujo cálculo pode ser particionado e depois combinado, e o resultado seria o mesmo se o cálculo fosse executado em todo o conjunto de dados. Por exemplo, a função de soma é distributiva: dividindo os dados N em conjuntos n, e fazendo a soma de cada conjunto n, teremos o mesmo resultado que se a fosse feita diretamente sobre N.

Uma medida algébrica é uma medida cujo cálculo pode ser feito sobre duas ou mais medidas distributivas. Por exemplo, uma medida de média pode ser calculada com a divisão da medida *soma* pela a medida *contagem*, que são ambas distributivas.

Uma medida é holística se não existe uma medida algébrica com M argumentos que caracterize a computação. Isso quer dizer que a computação não pode ser particionada, com valores exatos obtidos apenas se a medida for aplicada em todos os dados. Alguns exemplos são as medidas de moda, desvio padrão e mediana (HAN et al., 2011).

2.5.5 Operações OLAP

Para realizar consultas no *Data Warehouse*, é necessário utilizar de algumas operações sobre o cubo de dados para obter os resultados adequados. Essas consultas também devem conseguir passar na hierarquia de conceitos de cada dimensão, bem como seguir o modelo dimensional do cubo definido, para conseguir oferecer uma interface amigável com o usuário para análise interativa (HAN et al., 2011). Algumas operações estão exemplificados na figura 2.6, porém elas geralmente são:

- *Roll-up*: realiza agregação no cubo de dados, seja navegando na hierarquia de conceitos de nível específico para um mais genérico, ou reduzindo uma dimensão.

- *Drill-down*: o inverso da operação *roll-up*, navega na hierarquia de conceitos do nível mais genérico para o nível mais específico, ou adiciona dimensões ao cubo atual. Essa operação visa aumentar o nível de detalhes dos dados.

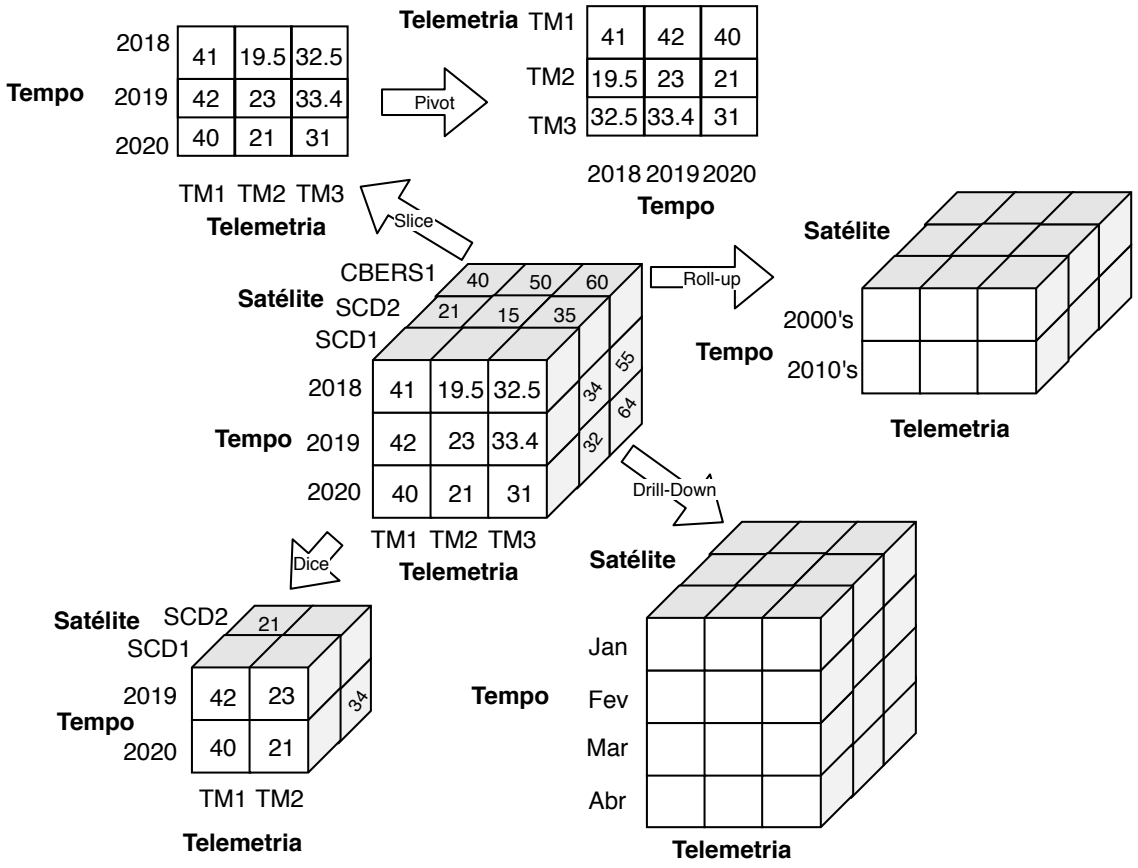
- *Slice*: ou "fatiamento", realiza uma seleção em uma dimensão do cubo, resultando em um subcubo.

- *Dice*: define um subcubo realizando uma seleção (*slice*) em duas ou mais dimensões.

- *Pivot*: também chamada de rotação, permite mudar a posição das dimensões na visualização, portanto alterando linhas por colunas e vice-versa.

Dependendo do sistema OLAP, é possível que outras operações sejam possíveis, como *drill-across* que passa por mais do que uma tabela de fatos, e *drill-through* que permite executar consultas direto na representação em baixo nível do cubo (HAN et al., 2011).

Figure 2.6 - Operações *OLAP* em um cubo de dados



SOURCE: Produção do autor.

2.5.6 Computação do cubo de dados

A computação do cubo de dados é uma tarefa essencial, pois a pré-computação de todo ou parte de um cubo de dados pode aumentar significativamente o desempenho do DW. Porém, essa tarefa possui complexidade exponencial em relação ao número de dimensões, sendo chamada de materialização, com a materialização completa exigindo uma grande quantidade de células, e portanto um elevado consumo de memória e tempo (HAN et al., 2011).

O cálculo original da computação do cubo de dados foi proposta por (GRAY et al., 1996), sendo: dada uma relação de relação de entrada R com tuplas de tamanho n, o número de subcubos que podem ser gerados é 2^n, onde n é o número de dimensões do cubo. Por exemplo, supondo um cubo com três dimensões *Satélite, Telemetria, Valor*, teremos $2^3 = 8$ subcu-

bos possíveis: $\{(satélite, telemetria, valor), (satélite, valor), (satélite, telemetria), (telemetria, valor), (telemetria), (valor), (satélite), ()\}$, com () denotando o agrupamento vazio (célula base, as dimensões não estão agrupadas).

Porém, na prática, as dimensões podem possuir hierarquias de conceito associadas, como para a dimensão tempo: "dia<mês<trimestre<semestre<ano". Para um cubo com n dimensões com múltiplas hierarquias de conceito, o número total de subcubos é apresentado na equação 2.1.

$$subcubos = \prod_{i=1}^{n}(L_i + 1) \tag{2.1}$$

Onde L_i é o número de níveis de conceito da dimensão i. É necessário adicionar um a equação 2.1 para denotar o nível virtual ALL. O tamanho de cada subcubo também depende da cardinalidade de cada dimensão, isto é, o número de valores distintos. Enquanto o número de dimensões, hierarquias de conceito e cardinalidade do cubo aumenta, também aumentam os seus requisitos de espaço de forma exponencial, sendo conhecida como a **maldição de dimensionalidade** na computação do cubo (HAN et al., 2011).

Para conseguir responder as consultas de maneira apropriada, é necessário escolher um método para a computação dos subcubos: a não materialização, a materialização completa e a materialização parcial.

Na não materialização, os subcubos agregados não são pré-computados, assim as agregações são computadas imediatamente, que podem ser extremamente lentas, porém tem o menor consumo de memória.

A materialização completa computa todos as agregações possíveis do cubo, gerando um cubo de dados completo. Esse método gera os melhores tempos de resposta, pois as agregações já foram computadas, porém necessita de uma grande quantidade de espaço de memória.

A materialização parcial computa apenas um subconjunto selecionado de subcubos, sendo que existem diversas técnicas diferentes de seleção dos subcubos que serão computados. Uma delas é computar todos os subcubos que contém apenas células que satisfazem um dado critério, especificado pelo usuário. Esses cubos são chamados de *iceberg* (BEYER; RAMAKRISHNAN, 1999).

Outra técnica é computar cubos pequenos, geralmente entre 3 e 5 dimensões, para formar cubos completos. Para responder consultas com mais dimensões, as combinações entre os subcubos pequenos são agregadas. Esta técnica é chamada de *shell fragment*, e o cubo é chamado de *cube shell* (LI et al., 2004).

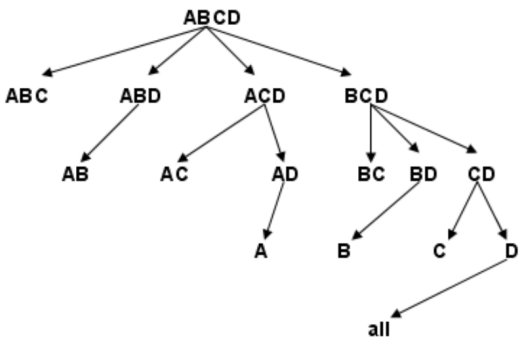
Um cubo de dados onde as células com medidas idênticas são encapsuladas em uma única abstração, chamada de célular fechada (*closed cell*) é chamado de cubo fechado, ou cubo quociente. Esta técnica foi apresentada com o cubo fechado (*closed cube*) (Dong Xin et al., 2006) e com o cubo quociente (*quotient cube*) (LAKSHMANAN et al., 2002).

A escolha da materialização parcial depende do equilíbrio necessário entre tempo de resposta e espaço de armazenamento. Porém, a computação do cubo completo continua sendo relevante, sendo que os avanços na computação dos cubos parciais são geralmente adotados na computação do cubo completo. Existe ainda o problema de atualização do cubo, pois cada atualização pode causar uma recomputação parcial ou completa do cubo para manter as medidas corretas.

A partir de um cubo base, a computação do cubo de dados pode utilizar a estratégia *Top-down* ou *Bottom-up* para a geração dos subcubos remanescentes (HAN et al., 2011).

A figura 2.7 mostra a geração de um cubo de dados de quatro dimensões pela estratégia *Top-down*. Sendo ABCD um cubo base, os subcubos de três dimensões são: ABC, ABD, ACD e BCD; que podem utilizar os resultados do cubo base para serem computados.

Figure 2.7 - Computação de cubo de dados através da estratégia *Top-Down*

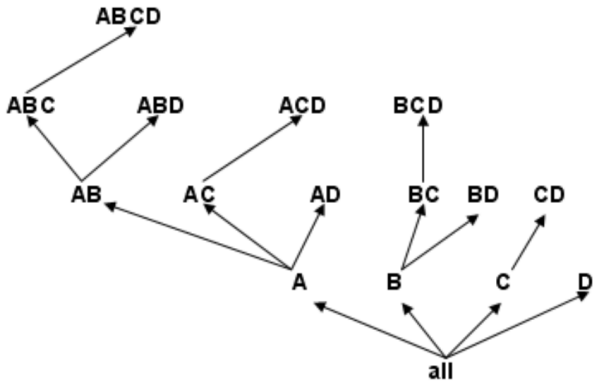


SOURCE: (SILVA, 2015).

Os resultados da computação do subcubo ACD podem ser utilizados para computar AD, que consequentemente podem ser utilizados para computar A. Essa computação compartilhada permite que a estratégia *Top-down* compute agregações em múltiplas dimensões. Os valores agregados intermediários podem ser reutilizados para a computação de subcubos descendentes sucessivos.

A Figura 2.8 mostra a geração de um cubo de dados de 4 dimensões por meio da estratégia *Bottom-up*. Subcubos de poucas dimensões tornam-se pais de subcubos com mais dimensões. Infelizmente, a computação compartilhada, utilizada na estratégia *Top-down*, não pode ser aplicada quando utilizada a estratégia *Bottom-up*, então cada subcubo descendente necessita ser computado do início.

Figure 2.8 - Computação de cubo de dados através da estratégia *Bottom-up*



SOURCE: (SILVA, 2015).

3 RELATED WORKS

In this section the related works will be presented, and they can be divided into two sections: the Big Data solutions from other operators, and the algorithms for the construction of data cubes with high cardinalities.

3.1 Operations Data

Table 3.1 shows the common data types used and generated by satellite operators, their origin and the common format for communication. These data are either available to the satellite, or to the satellite operators in some way.

Table 3.1 - Operations Data

Data Type	**Source**	**Format**
On-Board sensors	Satellite Equipment	Tables, CSV
Computer Logs	On-Board computer	Text (*Logs*)
Multimedia	Camera	MP4, JPG, RAW
Orbital Parameters	Operations and Tracking	TLE, text, tables
Associated documentation	Operators, Engineering	Text (Word, Excel, PDF)
Space Weather	Space or ground based information	Text, tables, warnings
Situational Awareness	Radars, US-STRACOM, etc	CDM, text, tables, warnings

SOURCE: Adapted from (ZHANG et al., 2017)

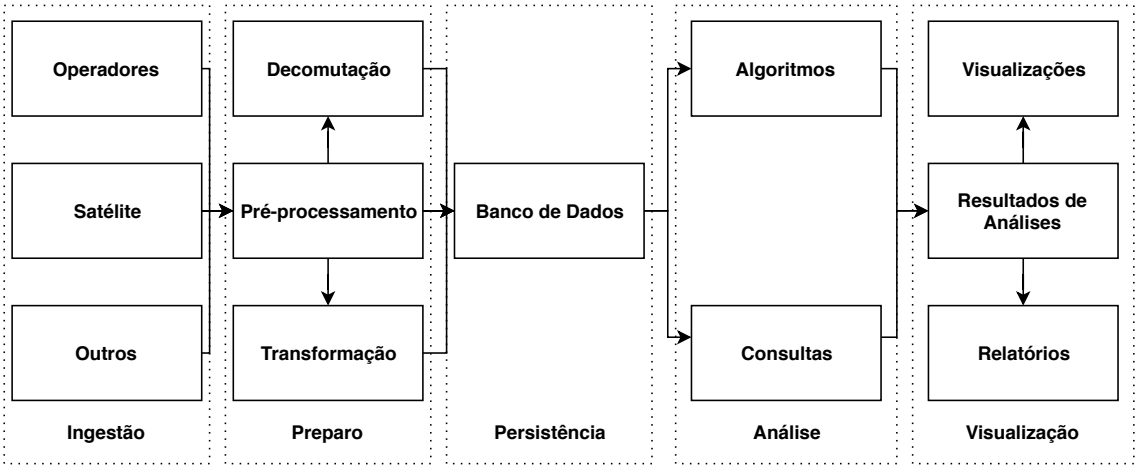
For this work, only the data from the on-board sensors will be considered, as the other data in this table can considered for a larger data management effort by the operator organization, however they are out of scope.

3.1.1 Data Flow

Based on previous works and on the relevant data, Figure 3.1 shows a sample data flow expected of a Big Data architecture for satellite operators. This flow is split into five phase that go from the source of the data to their final product analysis, and this work will only showcase the flow based on the correlated works, with each phase being:

Can't find the original source. Working on it.

Figure 3.1 - Data flow in a Big Data architecture



SOURCE: Adapted from (ZHANG et al., 2017)

.

- **Collection**: where the data are collected at their source (satellite, ground sensors, etc). This phase treat **where** and **how** to collect the data, as well as **which** data are important enough to be collected. The source here can be a third party, another department or available through other means.
- **Preparation**: the relevant data are selected and the necessary transformations to insert them into the database are peformed. This phase treats the specific format of the data, doing cleaning, quality control and relevance for analysis, among others. The objective is to guarantee the quality, relevance and adequacy of the data to the database.
- **Storage**: after processing, the high quality data are stored in a database, where they will be available for analysis. At this point any data storage means are used, treating only how these data are stored and how they will be available for analysis.
- **Analysis**: queries and algorithms will be executed on the data, with the assurance and availability of high quality data. This can range from simple queries ("what's the value of telemetry X during pass Y?"), to the use of complex algorithms ("predict the values of telemetry X for the next three passes on station Z").
- **Presentation**: analysis and algorithmic results are displayed to their end users. These can range from simple graphs to complex reports, as well as

the result of algorithms. Anything that reaches the decision makers.

The works of (ZHANG et al., 2017), (MATEIK et al., 2017) and (BOUSSOUF et al., 2018) have this process the best well defined out of all presented articles.

3.2 Data Analysis by Satellite Operators

Table 3.2 shows some recent published work made by satellite operators about the architectures that they use to process and analyze satellite data.

The common objective in these is to ease the satellite operator activities by means of anomaly detection algorithms and telemetry bounds checking. Some operators in this list are responsible for constellations of complex satellites, like remote sensing and communications, which are only cost-effective with a certain degree of automation for continuous operations.

These articles are filtered by only technologies that are used by the operators and not necessarily by the mission exploitation, as the housekeeping telemetry is on a different data ingestion pipeline than the payload data, as shown in (MATEIK et al., 2017) and (ADAMSKI, 2016).

Some of these do not shown complete infrastructure for the entire data flow, like (FERNÁNDEZ et al., 2017) and (TROLLOPE et al., 2018) that use *ad-hoc* scripting, and focus on only one part of the data flow.

In (YVERNES, 2018) the authors showcase some OLAP queries and the use of a data cube, having used dimensional modelling to aid the operation of a satellite constellation. However, this work only showcases at a very high level the used methodology, and mention that the work was performed only on at the modelling level to be integrated with other tools, not showcasing the rest of the infrastructure needed for this to work.

3.2.1 Data Analysis at INPE

INPE already perform data analysis on satellite telemetry, and in fact in many departments other than satellite operations. The satellite operators must monitor the telemetries, analyse the incoming data, and act based on engineering guidance in case a problem is identified (TOMINAGA et al., 2017). An example is in (MAGALHÃES, 2012), made about a fault on the CBERS-2 satellite, where the proposed model would enhance knowledge about a phenomenon known as thermal avalanche that

Table 3.2 - Satellite Operators and Big Data Architectures

Reference	Operator	Tool	Technologies
(ADAMSKI, 2016)	L3 (EUA)	InControl	Hadoop, Spark, HBase, MongoDB, Cassandra, Amazon AWS
(BOUSSOUF et al., 2018)	Airbus	Dynaworks	Hadoop, Spark, HDFS, HBase, PARQUET, HIVE
(DISCHNER et al., 2016)	SwRI + NOAA	CYGNSS MOC	SFTP, -
(EDWARDS, 2018)	EUMETSAT	MASIF	FTP, RESTful service, JMS Messague Queue, PostgreSQL
(EVANS et al., 2016)	S.A.T.E + ESA/ESOC	-	Java, CSV
(FEN et al., 2016)	CSMT (China)	-	-
(FERNÁNDEZ et al., 2017)	NASA	MARTE	R, CSV, ad-hoc
(GILLES, 2016)	L-3	InControl	Amazon EC2, LXC, Nagios
(HENNION, 2018)	Thales Alenia	AGYR	Logstash, Kafka, InfluxDB, ElasticSearch, Kibana, Grafana
(MATEIK et al., 2017)	Stinger, NASA	-	Logstash, ElasticSearch, Kibana, HDF5, CSV, R, Python, AWS, Excel
(SCHULSTER et al., 2018)	EUMETSAT	CHART	MATLAB, MySQL, Oracle
(TROLLOPE et al., 2018)	EUMETSAT	CHART	ad-hoc algorithms, a case study only
(YVERNES, 2018)	Telespazio France	PDGS	OLAP (DataCube), Saiku, Pentaho, Jaspersoft OLAP
(ZHANG et al., 2017)	SISET (China)	-	Hadoop, HDFS, PostgreSQL, MongoDB, Logstash, Kibana, ElasticSearch, Kafka, MapReduce

SOURCE: Author.

can make a satellite inoperable and thus identify and be able to stop that from happening again. Furthermore, as in line with Table 3.2, anomaly detection is a common area of study (AZEVEDO et al., 2011).

Other departments will mostly use data from the payload or from external agents, like remote sensing data, which analysis is also not trivial and are definitely a Big Data problem. Monteiro (2017) use Big Data concepts to the trajectory analysis; Ramos et al. (2016) showcase the use of tools like Hadoop to analyse Space Weather data, and Simões et al. (2018) show an architecture based on data cubes for remote sensing time series analysis, to name a few.

3.3 Data Cube Computation

The selective computation of data cubes has a rich history, due to the curse of dimensionality, most algorithms need a way to treat high dimensional data and deal with limited available memory (HAN et al., 2011).

The *Frag-Cubing* algorithm (LI et al., 2004) uses *cube shells*, where subcubes with few dimensions (generally from 2 to 5) will be computed using an inverted index, thus using the fragments as a compromise between used memory and how many dimensions can be answered with a multidimensional query. Frag-Cubing is one of the most efficient and popular data cube computation algorithms, and it's internal workings will be explored in the next section.

Using distributed computing, (DOKA et al., 2011) shows the Brown Dwarf, a Peer-to-Peer system that allows for cells to be updated and to reduce the redundancy in distributed cubes.

PopUp-Cubing is shown in (HEINE; ROHDE, 2017), using icebergs to deal with streaming data, and having superior results to FTL and Star-Cubing. This work specially deals with streaming data, which is of interest for real time data analysis of telemetries, however that that is not an scenario explored by this work.

Using MapReduce as a base, (WANG et al., 2013) presents HaCube, seeking a balance between parallel cube computation between various MapReduce nodes, allowing updates and incremental computation of measures. Due to its own distributed nature, it needs some fault tolerance to work, and also the tests were limited to only 5 dimensions, with up to 2,4 billion tuples. Also using MapReduce Yang and Han (2017) demonstrates the computing of holistic measures by presenting Multi-RegionCube, however with less testing than the previous method.

In (ZHAO et al., 2018) it is presented Closed Frag-Shells Cubing, which combines the C-Cubing approach (Dong Xin et al., 2006) of creating closed cube cells that losslessly compress drill-down/roll-up semantics, with the Frag-Cubing approach of using shell fragments, adding their own bitmap-based indexing on top to enhance query response times. This work is probably the closest to this own thesis, however with much less implementation requirements.

qCube (SILVA et al., 2013) extend the Frag-Cubing approach by adding value interval queries, allowing for other comparison operators besides equality to be used in queries. *HFrag* (SILVA et al., 2015) uses of external memory to aid in the computation of the inverted index, using a hybrid system to partition the cube in both main and secondary memory by keeping the most frequent used values in the main memory and the least used in the secondary memory. Hybrid Inverted Cubing (HIC) (SILVA et al., 2016) is an extension to the HFrag algorithm that uses a critical accumulated frequency parameter, which ends up having better results in practice.

Of these works *Frag-Cubing* keeps being a base robust algorithm to compute the cube, as the inverted index techniques are still relevant and the results are adequate. However, Li et al. (2004) show the exponential memory usage of the different cube computation schemes using only 12 dimensions, and there is a clear saturation when cubes with 20, 50, 100 or more dimensions are used to compute the cube (SILVA, 2015).

3.3.1 *Frag-Cubing*

Therefore, this work will focus on ways of improving the performance of the Frag-Cubing algorithm, both in query response time and memory usage. For that, it is necessary to understand how the algorithm computes the cube and answer queries. It does this by using the concept of an inverted index, where each inverted tuple iT has an attribute value, an Tuple Identifier (TID) list, and some computed measure values. For example, let us consider four tuples: $t_1 = (tid_1, a_1, b_2, c_2, m_1)$, $t_2 = (tid_2, a_1, b_3, c_3, m_2)$, $t_3 = (tid_3, a_1, b_4, c_4, m_3)$, and $t4 = (tid_4, a_1, b_4, c_1, m_4)$. These four tuples will generate eight inverted tuples: $iTa_1, iTb_2, iTb_3, iTb_4, iTc_1, iTc_2, iTc_3$ and iTc_4, shown in Figure 3.2.

For each attribute value an occurrence list is built, so for a_1 there is $iTa_1 = (a_1, tid_1, tid_2, tid_3, tid_4, m_1, m_2, m_3, m_4)$ where the attribute value 1 is associated with TIDs: tid_1, tid_2, tid_3, and tid_4. Tuple identifier tid_1 has the measure value m_1, tid_2 has the measure value m_2 and so on. Query $q = (a_1, b_4, COUNT)$ can be answered

by intersecting the lists $iTa_1 \cap iTb_4 = (a_1b_4, tid_3, tid_4, COUNT(m_3, m_4))$, in which $q, iTa_1 \cap iTb_4$ indicates the common TIDs between iTa_1 and iTb_4, their set intersection.

Figure 3.2 - Inverted Index computation example

TID	A	B	C	m
tid1	a1	b2	c2	m1
tid2	a1	b3	c3	m2
tid3	a1	b4	c4	m3
tid4	a1	b4	c1	m4

Valor	Lista de TIDs	Medidas
a1	tid1, tid2, tid3, tid4	m1, m2, m3, m4
b2	tid1	m1
b3	tid2	m2
b4	tid3, tid4	m3, m4
c1	tid4	m4
c2	tid1	m1
c3	tid2	m2
c4	tid3	m3

SOURCE: Author.

Can't find this source, I'll update it later

The complexity of the intersection is proportional to the number of times that an attribute value occurs in the input data, but bounded by the size of the smallest list, and in this example iTb_2 with a single TID is the smallest list. The number of TIDs associated to an attribute value can then be enormous, with low cardinality dimensions and high number of tuples needing a high processing capacity. Smaller TID lists allow for queries to be quickly answered, so relations with a low skew, or uniformity of values in the relation attributes, and high cardinality are more suitable to be computed by approaches with Frag-Cubing.

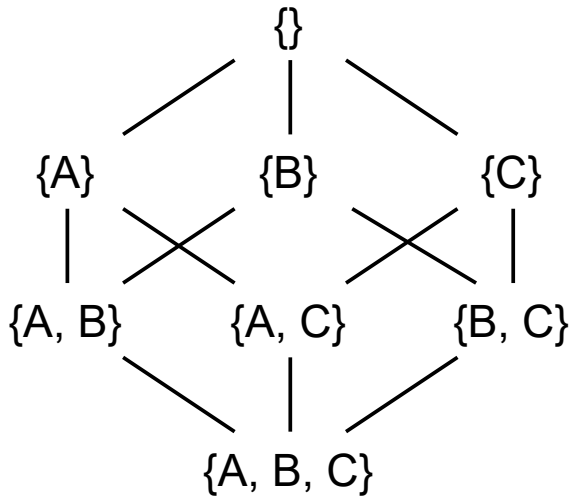
Furthermore, a skew close to zero can indicate a relation with uniformly distributed values, and the higher this value the less uniform the relation list will be.

The algorithm also introduces the concept of cube shells: instead of computing every combination of low-dimensional cubes, only a thin shell of dimensions is computed.

This translates to dividing the number of dimensions in the data by the supplied fragment size parameter $\mathcal{F}$ and generating all subcubes with that amount of dimensions, but not repeating the dimensions that have already been generated. Figure 3.3 shows a typical full cube with three dimensions A, B and C, with the top being the raw cell data (0-dimensional cube) and the base being the most aggregated cell that summarizes all data.

If we set $\mathcal{F} = 1$, then the Frag-Cubing algorithm will only pre-compute dimensions A, B and C, as they are the lowest dimensions that fit the fragment size, as is the standard of a cube that doesn't compute any aggregation besides the single dimensions. However, if we set $\mathcal{F} = 2$, then the algorithm will compute from left to right the subcube $\{A, B\}$, then see that there aren't enough dimensions for another cube with the same size as the fragment and compute the next remaining not computed subcube $\{C\}$, thus computing the subcubes $[\{A, B\}, \{C\}]$. If $\mathcal{F} = 3$, then the full cube would be precomputed, equating a fully materialized cube with all children subcubes being computed too.

Figure 3.3 - Shell Fragmentation example



SOURCE: Author.

4 QUERY PARTITION

In this chapter, an algorithm for aggregating the satellite telemetries into relationship groups is presented, which is then used to selected useful queries by a satellite operator. These queries are then compared with the Frag-Cubing algorithm: one set is called High-Dimensional, in which the queries are executed on the full data dimensionality; and the other is Low-Dimensional, in which the input data for the data cube algorithm is made of only the dimensions that the query will execute. The aim of this work is then to see if the query response time and memory consumption can be improved by pre-filtering the data that will be queried with just the dimensions related to that query.

4.1 Case Study: SCD2

The case study used in this thesis will be the satellite SCD2 (Data Collection Satellite in Portuguese), which has over 20 years of continuous operations by the Satellite Control Center at INPE (ORLANDO; KUGA, 2007). It is one of the first satellites designed, tested and assembled in Brazil, the second in the SCD family of data collection satellites (OLIVEIRA, 1996). The mission is goal is to retransmit to assigned receiver stations (Cuiabá and Alcântara, for example), data obtained from a network of Automatic Environmental Data Collection Platforms (PCD), which are distributed throughout the Brazilian territory. Figure 4.1 shows a commemorative mission patch for the satellite.

Each PCD is composed of a transmitter in UHF band (about $400Mhz$) that collects environmental data and continuously transmits them back into space. This transmission is then relayed by the PCD transponder to one of the receiver stations, and thus the data is gathered. This relaying can only happen when the satellite is visible by both the receiver stations and the PCDs, which happens around eight times per day (MIGUEZ et al., 1993).

The SCD2 satellite is composed of ten subsystems, including the DCP payload. With over 20 years of operation, more than 135 telemetry data points and generating over 8GB of data per year, there is a lot to be analyzed from the housekeeping telemetry data alone.

Figure 4.1 - SCD2

Data Collection Satellite 2 (SCD2) mission patch
SOURCE: Taken from http://www.inpe.br/noticias/galeria/

4.2 Algorithm

The objective of this algorithm is to easily classify the rate of change between groups of telemetries, as from previous data science work conducted on the telemetry data, just by identifying whether a group of telemetries change on a similar rate, it is possible to find a relationship between them. The algorithm is separated into two parts: the groups generation and the strength calculation.

4.2.1 Aggregation Generator

The algorithm works by creating telemetry groups with all possible dimensional combinations, considering that each telemetry is treated as a dimension. The combination of a given set of **n** elements taken **k** at a time is given by formula 4.1.

$$C_k^n = \binom{n}{k} \tag{4.1}$$

Since the number of telemetries is usually on the order of hundreds to thousands, it's best to limit the algorithm to combinations taken from 2 to 5 at a time. This is equivalent of computing a all subcubes with those dimensions. That gives us the following number of combinations, with **kmax** being the biggest k that we want, on formula 4.2.

$$\sum_{2 \leq k \leq n}^{kmax} \binom{n}{k} = 2^{n-2} \tag{4.2}$$

Each of these combinations is generated from a vector of n telemetry names that we're interested. For each of the generated combinations, we then execute aggregation measures:

- Group the available telemetry readings by the combination groups
  - for the combination "TM001", "TM002", group the table by "TM001" and "TM002"
- Each aggregate is counted for frequency that the values appear, called *count* henceforth
  - TM001 = ["01"] && TM002 = ['02'] -> count = 25
- For each of the telemetries used, compute the cardinality of the telemetry, called C_t for telemetry t
  - TM001 = ['01', '02','03'], then it'll have cardinality 3
- Compute descriptive statistics over all the values of *count*
  - Number of aggregates (length of the vector)
  - Mean
  - Median
  - Standard deviation

4.2.2 Relationship Strength Calculation

We can then use the descriptive statistics to calculate the strength of the relationship between the telemetries. This involves the use of conditionals and some parameters from the algorithm.

The initial condition is that if the cardinality of any telemetry is 1, it means that it didn't change in the time period, hence any aggregate with $C_t = 1$ will be marked with *NONE* on relationship. If the number of groups is 1 it also means that no changes were observed in the period, so we can't infer any relationships from the data, and the relationship is marked *NONE*.

If that condition passes, then we compute some values to help with classifying the other cases:

The biggest possible number of groups expected is the product of the sequence of cardinalities, called maxc is given by equation 4.3.

$$maxc = \prod_{t} C_t = C_1 * ... * C_t \tag{4.3}$$

The biggest cardinality in the combination, to us the *minimum* possible value for the number of groups, called *minc* on equation 4.4.

$$minc = \max(C_1, ..., C_t) \tag{4.4}$$

The proportion of the number of groups by the maximum cardinality, called cratio on equation 4.5.

$$cratio = \frac{numgroups}{maxc} \tag{4.5}$$

The absolute cardinality difference, as it is more representative of the discrepancy between bigger cardinalities, called *abscdiff* on equation 4.6.

$$abscdiff = cratio - \frac{minc}{maxc} \tag{4.6}$$

The coefficient of variation, from the standard deviation σ and the mean μ, called CV, is used to check the variability of the number of groups inside an aggregate on equation 4.7.

$$CV = \frac{\sigma}{\mu} \tag{4.7}$$

After all of those values, we are left with the choice of some parameters: the absolute cardinality ratio cutoff; the CV minimum and maximum cutoff and the CV minimum cutoff for the medium case.

Each of these parameters is necessary to characterize the distribution of each combination. A high number of groups does not tells us much about it's distribution: we need more statistics to know if the groups are evenly spread or if they are focused

on few values. Knowing that is essential to be able to distinguish the strength of the relationships.

So, the cardinality ratio cutoff is the first: it tells us how the cardinalities change in relation to each other. The number *cratio* will be closer to 1 if there's a relationship of 1 to 1 for each telemetry. This means that every time that one telemetry changes, the others changes too.

In contrast, a number closer to 0 means that the telemetries have very little variability, and that they're using the minimum expected cardinality. This means that the number of groups is closer to *minc*, and the variability is low.

The CV is then used to peer into the distribution of aggregates, by telling us if they're focused on few values or more spread evenly. A value close to, or bigger than, one means that the data are very spread, and thus might have a strong relationship, as that means that they tend to change together. A value closer to the CV minimum cutoff has data with low variability, which means that they're probably clustered together on few values. If it's within the absolute cardinality variability cutoff, then this value also denotes a strong relationship. If the value is within both cutoffs, then it's neither very clustered nor much variable, so we adopt a medium strength relationship.

From each of these paths, we have a single strength relationship, however the relative adoption of the relationship strength calculation is subjective, as the cutoff points need to be manually defined. With this algorithm, some 2x2 and 3x3 relationships were generated by grid searching all possible relationships and then showed to a satellite operator for evaluation.

4.3 Queries

With the satellite operator help, some sample queries that are frequent to the satellite operation procedures were filtered, not only related by their relationship but how useful the operator found them for their activities. The related telemetries are summarized in Table 4.1, with their identification, brief description and the calculated cardinality from the historic database. In this table, the cardinality of each telemetry is defined as the number of unique values that the telemetry can take.

4.3.1 Q1

Question: **are the batteries being charged or discharged?**

Table 4.1 - Telemetries overview

ID	Description	Cardinality
TM001	Payload receiver voltage	149
TM002	Payload RF output power	175
TM003	Magnetometer 1, Y axis	251
TM004	Magnetometer 1, -X axis	251
TM005	Magnetometer 1, Z axis	251
TM006	Magnetometer 2, Y axis	251
TM072	Battery Temperature 1	251
TM075	Solar Panels Current	251
TM077	Battery Charge Regulator 1	2
TM078	Battery Charge Regulator 2	2
TM081	Battery Temperature 2	251
TM082	Battery Discharge Regulator 1	2
TM083	Battery Discharge Regulator 2	2
TM130	Solar sensor temperature 1	233
TM131	Solar sensor temperature 2	233

The related telemetries are: TM072 and TM081 are each of the satellite's battery thermistor readings, TM077 and TM078 are charge regulator telemetries for each of the batteries, and TM080 and TM081 are discharge regulator indicators for each battery. The regulator telemetries simply indicate whether each battery is being charged or discharged as seen by the OBC, and take the form of "ON" and "OFF" values, while the thermistor telemetries indicate the thermal behavior of each battery.

This seems trivial at first glance: TMs 77, 78, 80 and 81 already display this information as each batteries' charge regulators, directly as collected by the OBC. However, in the case of this satellite, the thermal behavior of the batteries is important to verify whether the batteries are actually being charged or not. Furthermore, an overloading of one of the batteries might cause the relationship between the regulators to change and not show an accurate picture of what is happening, relating the query to anomaly discovery.

4.3.2 Q2

Question: **what is the current satellite orientation?**

The three telemetries are related to the magnetometer measurements, each (3, 4, 5) being of one axis (Y, X, Z) and with 300 mGauss precision.

This query has a simple objective of showcasing one of the most frequent operator activities: determining the satellite attitude. The strongest magnetic field will be the Earth's, and for this satellite, it has the express goal of deciding whether the satellite's antennas are still pointed in the correct direction to earth, and to verify the satellite's rotation rate. This satellite is stabilized by spin, and so verifying the speed and direction of spin is crucial for operations.

4.3.3 Q3

This question is meant as a comparative between the previous query: **is there any difference between the magnetometer readings in the satellite?**

As mentioned, TM003 is related to the magnetometer in the Y axis at 300 mGauss, and TM006 is just a redundant instrument with 600 mGauss precision for the same axis. This is meant to both create a redundancy in the instrument readings, as there are two instruments to measure the attitude that can be directly compared to see if there is any discrepancy in the sensors.

4.3.4 Q4

This question means to probe the data collection antenna: **is the payload antenna working as expected?**.

These telemetries are related to the primary payload, the Data Collection Payload. TM001 measures the voltage of the data collection antenna, while TM002 measures the output transmission gain of the antenna. This subsystem works by retransmitting the data from data collection platforms on various places of the earth to INPE's Mission Exploitation Center, and thus is relatively simpler to maintain. This query aims to see if the antenna is working as it should: the output gain is generally very stable, and the voltage is meant to just monitor if the antenna electronics are working.

4.3.5 Q5

Question: **are there any discrepancies between the measured currents and the solar panels temperatures?**

Telemetries 130 and 131 are thermistor readings for the solar panels, 75 measures the total output current, and 76 measures the shunt current for the solar charging system. The shunt aims to regulate the current that is measured in TM075, that is

the main output of the solar panels, and used to charge the batteries and to power the satellite. If the temperature telemetries (130 and 131) have readings that are too hot or too cold, the solar panels might fail and not provide the necessary power to the satellite anymore, which would be catastrophic failure, as the satellite would not be able to recharge its batteries and would stop working.

4.3.6 Summary

Table 4.2 has an overview of the queries presented, with the query identification, the telemetries that are queried and the product of the cardinalities involved.

Table 4.2 - Queries overview

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

4.4 Experimental Validation

To validate whether the pre-selection effectively reduces query memory consumption and response times, it is needed to test it against the Frag-Cubing algorithm. This section details how this selection was performed, the used algorithms and presents a simple overview of the results.

4.4.1 Dataset and Method

The Frag-Cubing algorithm used the Illimine project implementation (ILLIMINE, 2004) that was coded in C++ and compiled on a Linux Kernel 5.0.0-29 machine, with gcc 7.4.0. Some adaptations were made to the original code to allow for better output formatting, however these were minimal format changes and didn't impact on the performance or changed how the algorithm works. All of the experiments were executed on an Intel(R) Core(TM) i7-8550U CPU @ 1.80 GHz, with 16 GB of DDR4 @ 2400 MHz system memory and on an Adata XPG SX8200 Pro Solid State Drive using PCIe Gen 3x4 interface.

The experiments were designed to measure:

- Base cube main memory;
- Runtime to build the base cube representation;
- Query response time;
- Query memory increase, which measures how much memory was needed to answer the query beyond what was used by the base cube.

As a notational convention, we use $\mathcal{D}$ to denote the number of dimensions, $\mathcal{C}$ the cardinality of each dimension, $\mathcal{T}$ the number of tuples in the database, $\mathcal{F}$ the size of the shell fragment, $\mathcal{I}$ the number of instantiated dimensions, $\mathcal{Q}$ the number of inquired dimensions, and $\mathcal{S}$ the skew or zipf of the data. Minimum support level is 1, as well as $\mathcal{F} = 1$ for all experiments.

Each test was executed 5 times, with the average value of the five runs being taken. Additionally, before each test a baseline with no performed queries was executed, just computing the time to cube: how long, and using how much memory, it takes for the algorithm to compute the initial cube. This is meant to ease the comparison of the results.

The central idea of this experiment is to partition the input data with the dimensions with the expected dimensions used in a query, to see if that is a better or worse cube construction strategy. To achieve that, the 4 year of data resulted in to 24 M (2.4×10^7) tuples over the satellite's 135 telemetries, saved in a relational database. Those were separated into files for each query and each data size. To better provide comparisons, each data was separated into datasets of equal interval: 2M, 4M, 6M, 8M and 10M tuples (2×10^6, 4×10^6, 6×10^6, 8×10^6 and 10^7).

In a first test run it was found that the different data distributions at those levels were interfering with the experiment, and so, to evaluate only the general distribution of the data and how it was organized, each tuple of each dataset was sampled from the full 2.4×10^7 original data.

In the end, this resulted in $12,83$ GB of data converted to Frag-Cubing's format, counting the datasets with the full 135 telemetries and the datasets with the filtered telemetries, resulting in 30 different data files (5 for the high-dimensional case, and 5 for each query).

For this paper, the names in Table 4.3 will be used to refer to each of these cubes.

The cubes with 135 dimensions will be treated as “C0”, with “C1” to “C5” being the cubes with dimensions filtered for the telemetries in “Q1” to “Q5”.

Table 4.3 - Cube representations used in the experiment

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

The process to separate the data was performed as follows:

a) Select from telemetry database (PostgreSQL) the dimensions that are used in the query (ex. ‘SELECT TM001, TM002 FROM telemetries’);
b) Filter first n tuples from that selection, where n is in 2×10^6, 4×10^6, 6×10^6, 8×10^6 and 10^7;
c) Save the results to a file and convert it to Frag-Cubing’s input format, naming it cube i (eg. "Ci"), where i is one of the query identifiers;
d) Load the file into Frag-Cubing and execute the relevant queries.

4.4.2 Results

For the algorithm to partition the queries, it was quickly apparent that the output was too broad and the difference between the queries was too hard to classify by an operator, as most relationships are not clear and would all require further investigation to validate, which would defeat the purpose of the algorithm. This led to using the most frequent queries as detailed in the previous sections, and the total abandonment of the algorithm, as the output could not be validated in a scientific manner.

Furthermore, INPE has only a few satellite operator experts, and for this satellite that has spanned multiple years of operations, the knowledge amounted to a single available person available for questioning, which is not enough for an objective scientific inquiry. Therefore, the separation in queries used was used as per the operator’s

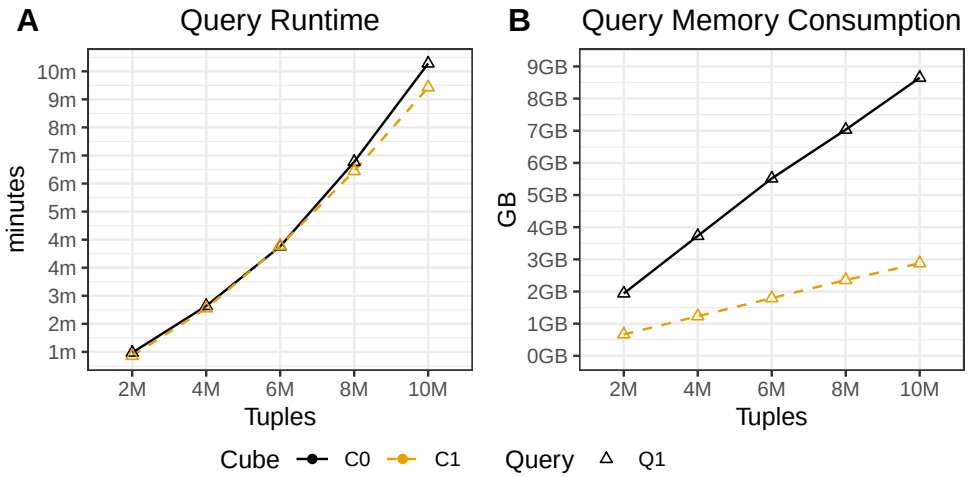
experience, and thus are inherently biased. The algorithm needs more study and a robust dataset to be validated, and there are some hints in the literature trying to do, but data is sparse and the necessary information first needs to be made after human analysis. The use of the algorithm is then not recommended, and further improvements to it will be out of the scope of this work.

Thus, this section will deal with the results from the experiment detailed in section 4.4.1. Each defined is compared with their execution in $C0$ and the relevant cube, with their memory and response times being measured by each.

4.4.2.1 Q1

Figure 4.2-A shows the query response times for query Q1, with the execution of the query in the C0 and C1 cubes. There's a speedup of about 10% with C1 for the $\mathcal{T} = 10^7$ case, for the query that uses the highest amount of dimensions on all the studied queries. Figure 4.2-B shows the query memory difference between C0 and C1 for query Q1. There's a clear advantage of C1 taking only one third of the memory that C0 takes to answer the same query.

Figure 4.2 - Query 1 results



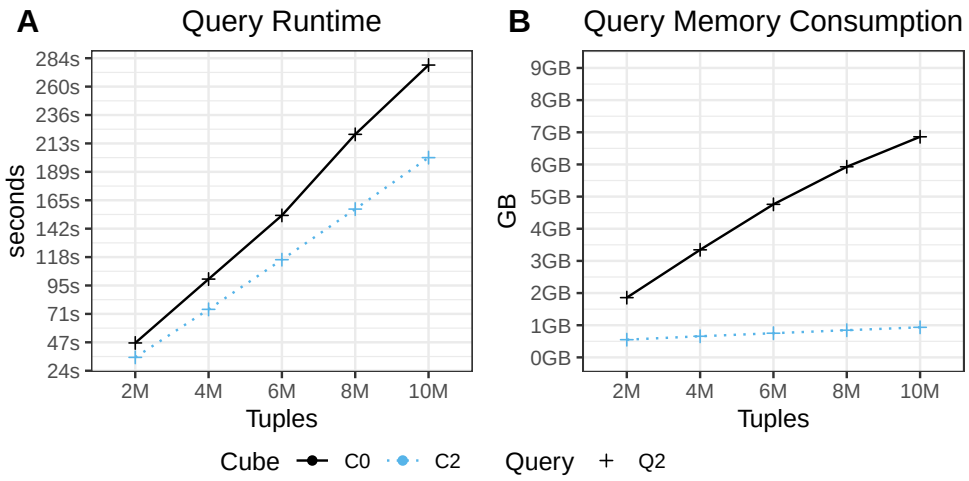
(**A**) Total time to answer a query, including the time to read the data from the disk, build the cube and then answer the query Q1, with cubes C0 and C1. (**B**) Total memory consumption to answer Q1 with cubes C0 and C1.

SOURCE: Author

4.4.2.2 Q2

Figure 4.3-A shows the query response times for query Q2, with the execution of the query in the C0 and C2 cubes. Here the difference when $\mathcal{T} = 10^7$ is C2 having a runtime 40% faster than C0. Figure 4.3-B shows the query memory difference between C0 and C2 for query Q2. Again the result is C2 taking only a fraction (14%) of the same query under C0.

Figure 4.3 - Query 2 results



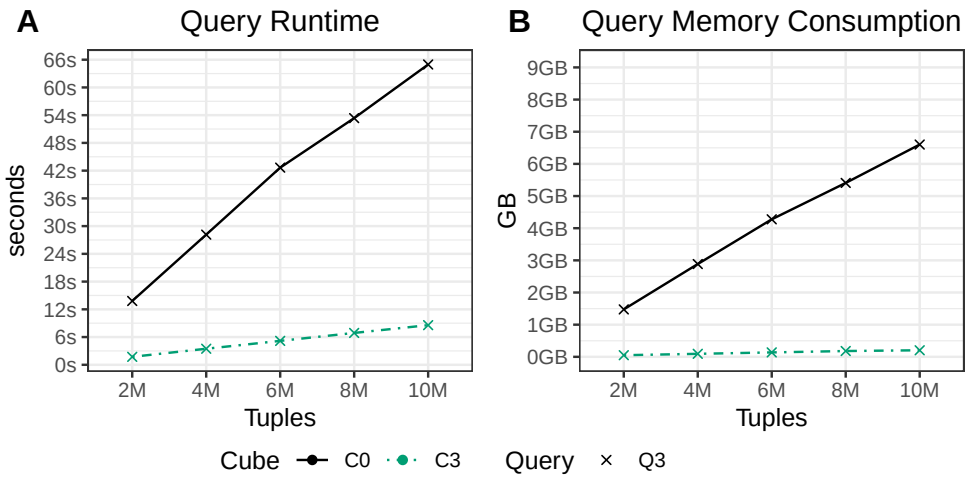
(**A**) Total time to answer a query, including the time to read the data from the disk, build the cube and then answer the query Q2, with cubes C0 and C2. (**B**) Total memory consumption to answer Q2 with cubes C0 and C2.

SOURCE: Author

4.4.2.3 Q3

Figure 4.4-A shows the query response times for query Q3, with the execution of the query in the C0 and C3 cubes. With less inquires and dimensions this operation is expected to be faster, however the speedup is even greater: Q3 under C3 takes only 12% of the memory used by C0. Figure 4.4-B shows the query memory difference between C0 and C3 for query Q3, with C3 needing only 7% of the memory used by C0.

Figure 4.4 - Query 3 results



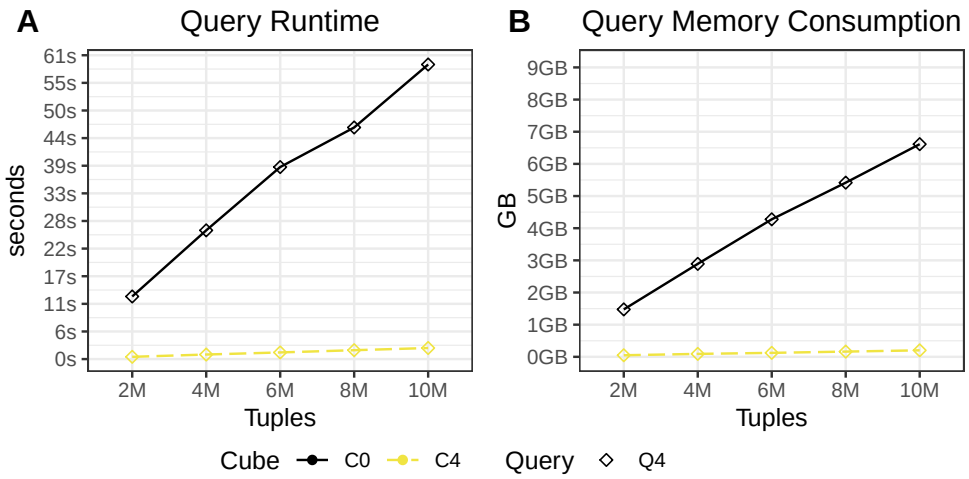
(**A**) Total time to answer a query, including the time to read the data from the disk, build the cube and then answer the query Q3, with cubes C0 and C3. (**B**) Total memory consumption to answer Q3 with cubes C0 and C3.

SOURCE: Author

4.4.2.4 Q4

Figure 4.5-A shows the query response times for query Q4, with the execution of the query in the C0 and C4 cubes. Another query with only two dimensions, and expected to be faster, with C4 taking 5% of the time used by C0. Figure 4.5-B shows the query memory difference between C0 and C4 for query Q4, showing the same speedup pattern.

Figure 4.5 - Query 4 results



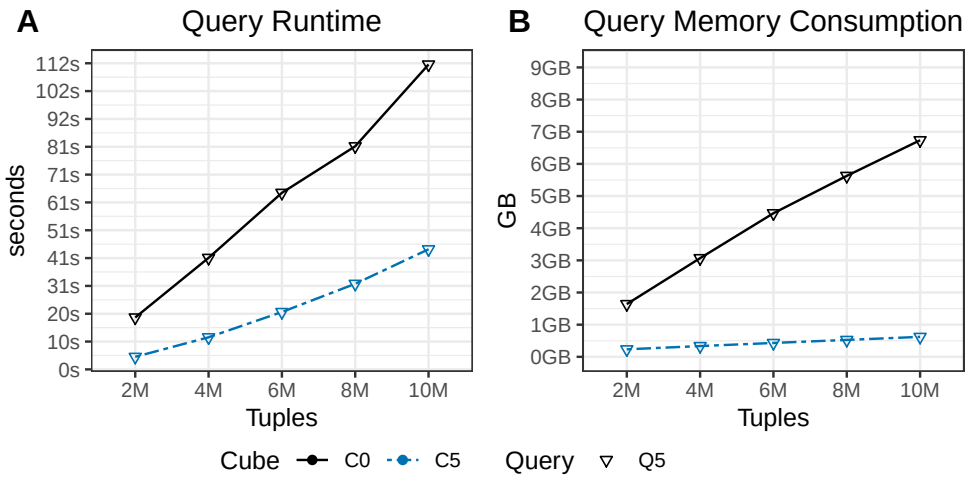
(**A**) Total time to answer a query, including the time to read the data from the disk, build the cube and then answer the query Q4, with cubes C0 and C4. (**B**) Total memory consumption to answer Q4 with cubes C0 and C4.

SOURCE: Author

4.4.2.5 Q5

Figure 4.6-A shows the query response times for query Q5, with the execution of the query in the C0 and C5 cubes. Here with more dimensions than the previous two queries the speedup is less, but still substantial, with C5 taking only 43% of the runtime used by C0. Figure 4.6-B shows the query memory difference between C0 and C5 for query Q5, where the same figure of speedups are maintained.

Figure 4.6 - Query 5 results



(**A**) Total time to answer a query, including the time to read the data from the disk, build the cube and then answer the query Q5, with cubes C0 and C5. (**B**) Total memory consumption to answer Q5 with cubes C0 and C5.

SOURCE: Author

4.5 Summary and Analysis

For this experiment, the time dimension has the property of having a cardinality that is approximately equal to the amount of tuples ($\mathcal{C}_d = \mathcal{T}$, for a cardinality of dimension d and with a database of $\mathcal{T}$ tuples), as each observation is time-stamped and therefore unique. This creates a considerable skew to the results with all telemetries, as with 10^7 tuples, a single dimension with $\mathcal{C}_d = 10^7$ is not suitable to be computed entirely, and very different from the other dimensions that have a maximum cardinality of $\mathcal{C}_d = 256$. As this time dimension was not considered in any query, this is expected to have been one of the reasons for the great difference in memory consumption between the queries and $C0$.

In summary, these results show that building the cube with only the dimensions for a given query can yield up to 80 times less memory for the cube construction, and using between 1% to 33% less memory to answer the same query on the same data, with variations depending on the amount of inquired dimensions. The speedup is so apparent that it'd be *faster* to read, build and then answer a query using one of the low-dimensional cubes ($C1{-}5$) than to directly query an already built data cube with all dimensions ($C0$), and is the main contribution of this work. Furthermore, the shell

fragment size parameter shows little benefit in reducing the memory consumption of the queries when they are performed in this manner, and should be kept at 1.

It is also important to highlight the query definitions with the help of a domain expert: most queries are low dimensional in nature, in line with Frag-Cubing's strengths. These queries are non-exhaustible and meant to be samples, but show that the common algorithmic approaches are suitable for the domain and can be made to improve memory implementation requirements.

5 IntervalFrag

This section describes the IntervalFrag algorithm, and the proposed architecture needed to implement the enhancements to the Frag-Cubing's algorithm.

5.1 Using Intervals in Inverted Indexes

In chapter 3.3.1 the Frag-Cubing algorithm was explained, as per the original implementation by (LI et al., 2004). One of the key parts of the algorithm is the use of an inverted index to query directly into the attribute values and allow for the iceberg pruning of cells that fall below minimum support. The algorithm depends on the intersection of TID lists to work, as that is how it can known what tuples have that specific value and where to find them to compute the relevant measures. This part of the algorithm is mentioned by the original work as being done naively, and the original authors suggest some ways of compressing the index and speeding up the intersection of the lists, as this will be one of the most frequent operations that the algorithms needs to do to answer queries, and a speedup on this part would greatly enhance performance.

Furthermore, as most of the data is directly loaded into memory, using some form of compression on the inverted index would also reduce memory implementation requirements, and hopefully also query response times. One of the strategies that they mention is by compressing the TID list into *d-gaps*, in which each element is encoded by being the sum of the current element plus the previous element. In general, for a list of numbers $\langle d_1, d_2, \cdots, d_k \rangle$, the *d-gap* list would be $\langle d_1, d_2 - d_1, \cdots, d_k - d_{k-1} \rangle$. The compression then would come from encoding the elements into a smaller number of bits, hopefully much less than the standard of 32 for an integer. This approach takes advantage of the ordered nature of the TID list, as it is encoded from the tuples as they read and thus are naturally sorted non-zero positive integers.

This approach requires heavily changing the binary operations of the inverted index, however it is not the only one: since publication there are various different techniques to compress an inverted index, and they are an active branch of research to this day. More options, and a more complex implementation of the above method can be found into Elias-Fano encoding and the others mentioned in (PIBIRI; VENTURINI, 2019). However, most of them require the use of dictionaries or heavy bit-encoding to achieve better results, which in general sacrifices the update operation.

One much simpler technique that has not been explored is the use of intervals instead of raw numbers in the TID list. The inverted index structure is then kept, but each TID list now instead of containing the ordered numbers, contains an ordered list of intervals between each element. For a list of numbers $\langle d_1, d_2, \cdots, d_k \rangle$, the interval list would be $\langle [d_1, d_2], [d_3, d_4], \cdots, [d_k, d_{k+1}] \rangle$, where $d_k < d_{k+1}$, and the difference between the intervals cannot be smaller than one, thus $d_{k+1} - d_k \geq 1$. Example: for the TID list $\langle 1, 3, 4, 5, 7 \rangle$, the Interval List would be $\langle [1, 1], [3, 5], [7, 7] \rangle$, where we can represent 3 and 7 that have no difference between their intervals as the singles $\langle [1], [3 - 5], [7] \rangle$.

This has implications only for the intersection of the lists, and is overall a much simpler implementation to execute. By previous experience with the data, it was found that a great of dimensions have attribute values that are repeated on long sequences of the same telemetry data points being repeated, and Frag-Cubing generates a long list of these repetitions for some dimensions. This thesis then seeks to answer the question: **Can the use of the Interval algorithm instead of the raw list reduce memory requirements and query response times for long sequences of real world satellite telemetry data?**

Thus, in this chapter this implementation will be detailed, as well as an experiment to evaluate whether there is any advantage to using these intervals over the standard technique used by Frag-Cubing.

5.2 Algorithm

In practice, the algorithm cannot be implemented using two integers for each interval, as in the case where there is not a substantial interval (difference bigger than one) all elements will take the space of two integers when in Frag-Cubing they would take the place of only one integer. This would lead to the worst case for the Interval algorithm to be double the memory of Frag-Cubing, but this can be worked around in practice by encoding the elements without an interval in the negative range of the integer, which isn't used in the TID list as each element indicates a unique identifier that is positive.

In case the identifier to be inserted would be close to zero, it is necessary to add one to each of the elements in the TID list as they are inserted into the IntervalIndex, as checking for a negative zero is not recommended and not guaranteed to work the same on every computer architecture (IEEE, 2019). With this, the entire integer bit space is used, including the signal bit for the algorithm. For the scope of this work,

and since the experiment does not require the update of cells, only the insertion and intersection algorithms will be detailed.

5.2.1 IntervalInsertion

Insertions to the index can be done always by appending to the current list, as the TID are read in sequence and are naturally ordered positive integers. Supposed that we're inserting an element b to the list. The insertion can be done by checking if the last element in the list is positive, if it is, then there is an interval and it is necessary to check the penultimate element in the list. If the last element is negative c, then we can check if $c*-1+1 = b$, and if it is we flip the signal of the position c and insert b to the list as it is. If the last element is not negative, then we check if the element c, is $c + 1 = b$ and update the position c if it is true, else we flip the signal of b and insert it at the end of the list. If none of these we can safely append b to the end of the list after flipping the signal to negative. Thus this algorithm is implemented at $\Theta(1)$, shown in algoritmo 1.

Algoritmo 1: IntervalInsertion

```
Result: Element inserted into the list
b element to be inserted;
L the interval list to be inserted;
if empty(L) then
    AppendList(L, b * -1);
    return;
end
c = LastElement(L);
if c < 0 then
    if c * -1 + 1 = b then
        L[c] = L[c] * -1 ;              // Update the last to be positive
        AppendList(L, b);
        return;
    end
else
    if b + 1 = c then
        L[c] = b ;                                  // Update last element
        return;
    end
end
AppendList(L, b * -1) ;     // None of the previous cases, just append
 the element as a negative
```

5.2.2 IntervalIntersection

The problem of intersecting two sets of elements is a big one, and will not be fully explored by this work, however a simple exploration of some algorithms that was a byproduct of this work are available in Appendix A.1. However, due to the change in how the index uses the TID list, it is necessary to adapt the Frag-Cubing set intersection algorithm to still calculate the intersection between two intervals, and thus the IntervalIntersection algorithm is created.

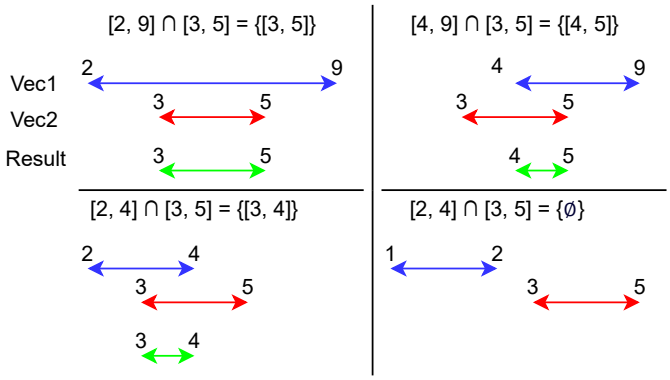
To check if two intervals intersect: assuming two intervals $I_a = \langle a_l, a_r \rangle$ and $I_b = \langle b_l, b_r \rangle$, where a_l is the smallest element in the interval, and a_r the biggest element in the interval. The output intersection can be defined as $I_o = \langle a_l, a_r \rangle$, and the intersection $I_o = I_a \bigcap I_b$ can be computed by computing the biggest element between a_r and b_r (function max) and the smallest element between a_l and b_l (function min), or algoritmo 2.

Algoritmo 2: IntersectTwoIntervals, adapted from (spektr, 2017)

Result: The result intersection I_o, or $\emptyset$ in case there is no intersection
I_a and I_b two intervals;
if $b_l > a_r$ *or* $a_l > b_l$ **then**
    return $\emptyset$;
else
    $o_l = max(a_l, b_l)$;
    $o_r = min(a_r, b_r)$;
    return $[o_l, o_r]$;
end

Figure 5.1 shows four examples of using this algorithm to select for intersections, including the empty case when there is no intersection. For the list interval intersection, it can be naively implemented using the scalar merge strategy: two pointers walk by each interval, check if there is an intersection between then and, if there is, create a new element in the result list with the intersection between these two intervals. The algorithm to perform this is shown below, and takes the same complexity of the parent algorithm of $\mathcal{O}(n + m)$, where n and m are the sizes of the interval lists being intersected. This resulting algorithm is based on the implementations by (LI et al., 2004) and (SILVA, 2015), shown in algoritmo 3.

Figure 5.1 - IntervalIntersection example



IntervalIntersection example with four operations

SOURCE: Author

Algoritmo 3: IntervalIntersection

Result: The resulting list intersection between two interval lists, or $\emptyset$ if there is no intersection

L_a and L_b two input interval lists;
L_c result list, with maximum size $min(length(L_a), length(L_b))$;
$a_i, b_i, c_i = 0$;
while $a_i < length(L_a)$ *and* $b_i < length(L_b)$ **do**
    $A[a_l, a_r] = \text{interval}(L_a[a_i])$; `// Gets the interval at this position`
    $B[b_l, b_r] = \text{interval}(L_b[a_i])$;
    if $b_l \leq a_r$ *and* $a_l \leq b_r$ **then**
        IntervalInsertion(L_c, IntersectTwoIntervals(A, B)) ; `// Insert into the result list the intersection between the elements`
        NextIntervalPosition(c_i) ; `// Skips to the next available list space`
    end
    if $a_r \leq b_r$ **then**
        NextIntervalPosition(a_i);
    else
        NextIntervalPosition(b_i);
    end
end
return L_c;

5.3 Results

Table 5.1 and Table 5.2 show the results of executing both algorithms to answer the queries defined in section 4.3, while using the cube structure defined as $C0$ in subsection 4.4.1, where all telemetries were used as a single file for each test, and then the query was executed, measuring the memory consumption in the first and query response time in the latter.

Table 5.1 - IntervalFrag x Frag-Cubing, memory consumption in KiB

		Tuples				
Algorithm	**Query**	2×10^6	4×10^6	6×10^6	8×10^6	1×10^7
Frag-Cubing	Q1	1.908.708	3.674.784	5.447.864	6.953.424	8.557.348
	Q2	1.842.396	3.294.628	4.727.816	5.877.016	6.760.080
	Q3	1.448.280	2.836.592	4.236.496	5.362.128	6.502.628
	Q4	1.444.816	2.836.696	4.236.404	5.372.104	6.520.176
	Q5	1.607.456	3.024.996	4.445.800	5.591.052	6.650.456
IntervalFrag	Q1	504.428	845.804	1.196.504	1.455.472	1.651.552
	Q2	801.864	1.207.760	1.560.652	1.752.292	2.062.560
	Q3	388.652	707.588	1.030.392	1.237.324	1.415.472
	Q4	370.624	690.456	1.003.236	1.237.292	1.415.456
	Q5	540.788	895.272	1.232.520	1.412.280	1.604.288

Table 5.2 - IntervalFrag x Frag-Cubing, query response times in ms

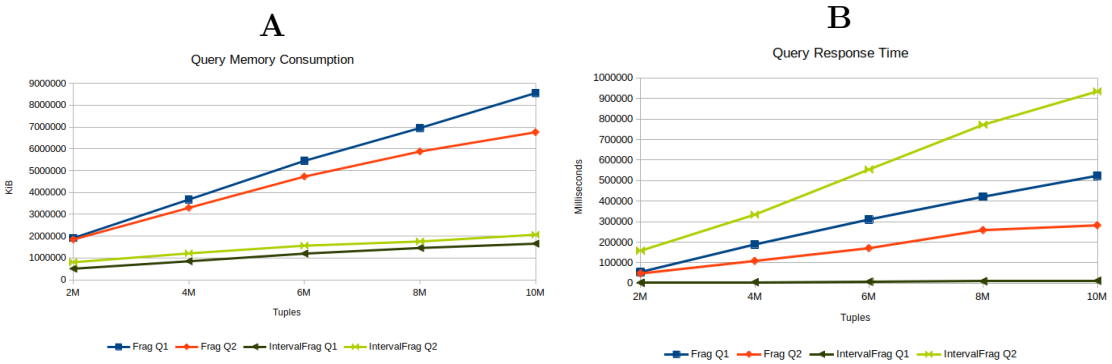
		Tuples				
Algorithm	**Query**	2×10^6	4×10^6	6×10^6	8×10^6	1×10^7
Frag-Cubing	Q1	5.4691	188.634	310.455	421.409	523.772
	Q2	47.391	108.405	170.515	258.585	281.877
	Q3	1.557	3.089	4.637	6.497	7.597
	Q4	399	817	1.193	1.573	1.990
	Q5	7.138	21.668	33.483	49.590	59.428
IntervalFrag	Q1	1.946	4.712	6.981	9.198	11.508
	Q2	158.050	333.838	554.111	772.125	934.793
	Q3	3.570	7.064	10.655	14.812	17.714
	Q4	995	2.011	2.952	3.860	4.916
	Q5	4.871	11.837	18.477	26.176	32.649

To make the comparisons easier to understand, let's focus on only queries $Q1$ and $Q2$, that were the highest in number of dimensions and the highest in number of cardinalities respectively. Figure 5.2 shows those results, and it is clear that the

memory usage is always much lower under IntervalFrag. $Q1$ has four dimensions with low cardinality and high sequentiality, and these are quickly answered by IntervalFrag, while Frag-Cubing takes a long time to answer the queries that involve those dimensions.

However, in the case of $Q2$ where all dimensions have both cardinality and low sequentiality, IntervalFrag takes 331% longer to answer the query on the worst case, being the biggest drawback of this algorithm. This is due to the inherently slower set intersection algorithm used by IntervalFrag, that needs to execute more comparisons than Frag-Cubing, even when the algorithms have the same complexity and are similar.

Figure 5.2 - IntervalFrag for Q1 and Q2

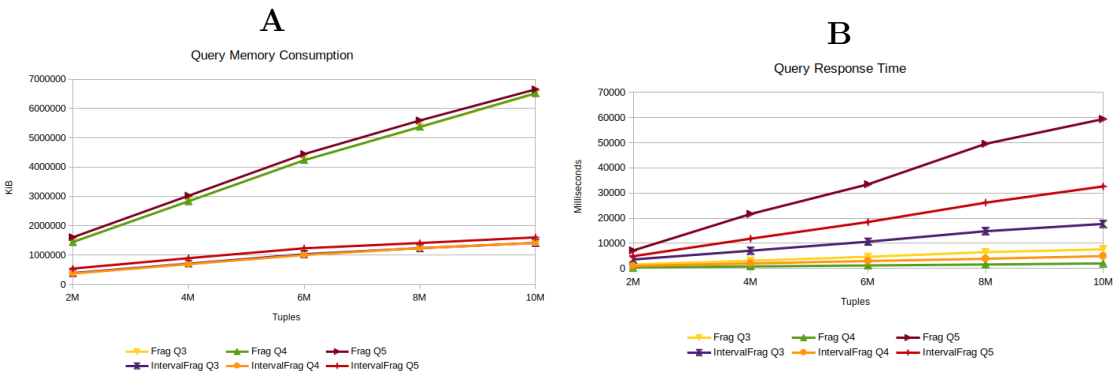


(**A**) Memory consumption of IntervalFrag and Frag-Cubing for queries $Q1$ and $Q2$ under the cube $C0$. (**B**) Query response times of IntervalFrag and Frag-Cubing for queries $Q1$ and $Q2$ under the cube $C0$

SOURCE: Author

In the case of queries $Q3$, $Q4$ and $Q5$, IntervalFrag is slower to answer queries $Q3$ and $Q4$, both that have a high cardinality and low sequentiality, but takes only about 53% of the time to answer $Q5$, that also has a high cardinality but presents a high sequentiality. Figure 5.3 shows those results, and it is also clear that the memory usage is much lower with IntervalFrag than with Frag-Cubing.

Figure 5.3 - IntervalFrag for Q3, Q4 and Q5



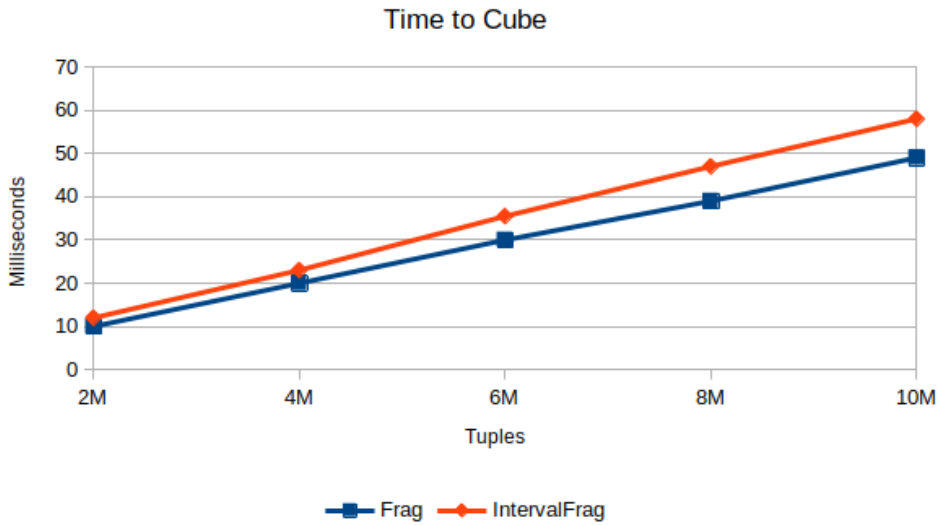
(**A**) Memory consumption of IntervalFrag and Frag-Cubing for queries $Q3$, $Q4$ and $Q5$ under the cube $C0$. (**B**) Query response times of IntervalFrag and Frag-Cubing for queries $Q3$, $Q4$ and $Q5$ under the cube $C0$

SOURCE: Author

Additionally, we need to compare the time necessary to create the cube under each of the algorithms, here called Time to Cube. If one algorithm takes too long to transverse the cube structure and execute the minimum support prunning this would need to be counted against it, however as Figure 5.4 shows, the difference between IntervalFrag and Frag-Cubing is not that big, with IntervalFrag being on average 10% slower than Frag-Cubing. This however is just a simple comparison, as in that step each subcube of the fragments would be computed and for these tests that use $\mathcal{F} = 1$ they are almost exactly the same computation, needing higher fragment sizes to appreciate the difference. However, since this computation would involve list intersections at higher fragment sizes, the speed of the intersection algorithm would heavily influence this computation, being more comparable to the query response times compared above.

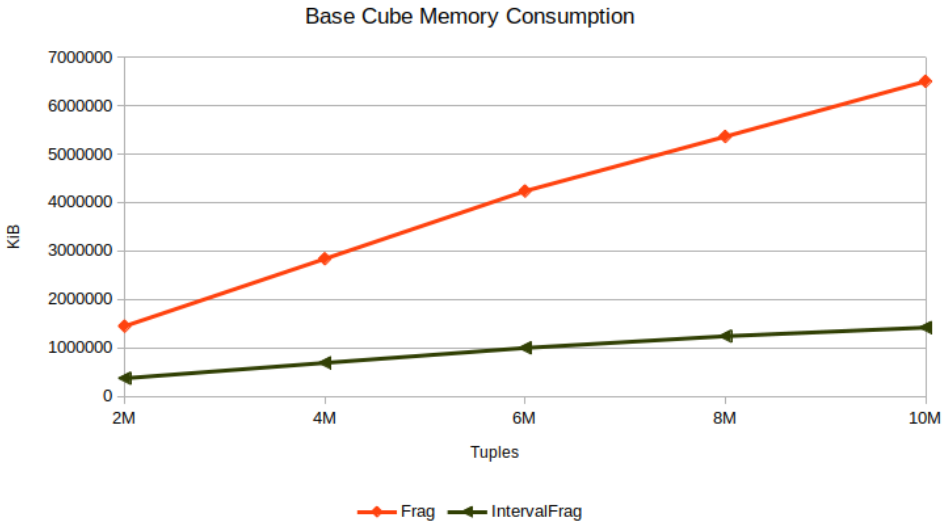
Another important metric is the baseline memory used before queries can be performed on the data. Figure 5.5 shows these values for the files used in this experiment, and IntervalFrag consumes on average 22% of the memory that Frag-Cubing needs. This is important as it allows IntervalFrag to be implemented while requiring much less computational resources.

Figure 5.4 - Comparison: Time to Cube



Time necessary to compute the cube after the data is read into memory.
SOURCE: Author

Figure 5.5 - Comparison: Baseline memory



Memory used by the baseline cube, when it can start to answer queries
SOURCE: Author

5.4 Summary

FragInterval was up to 3 times slower to answer queries on dimensions with low sequentiality and high cardinality, but was faster to answer queries on dimensions with high sequentiality. On all queries FragInterval used only between 20% and 24% of the memory that Frag-Cubing used, being the biggest improvement that IntervalFrag brings. Furthermore, FragInterval also needed only on average 22% of the memory that Frag-Cubing used to compute the baseline cube, before the queries can be answered, while being only 10% slower to compute.

Thus FragInterval shows to be preferred on environments with low available RAM, as long as the slower queries with higher cardinalities are acceptable. FragInterval achieves the objective of answering the queries with much less memory, however fails at having comparable response times than Frag-Cubing on high dimensionality queries. This slowness is due to the set intersection algorithm in IntervalFrag having to execute more comparisons in practice than Frag-Cubing's, and thus being slower when the sets have the same size and the data has low sequentiality to favor IntervalFrag's algorithm.

6 Analysis and Discussion

In this chapter, a critical analysis of the algorithms is presented, as well as an overview of how useful are the results and what are their shortcomings. The results from chapters 5 and 4 show that simple approaches can be used to enhance the query response time for the selected queries, and can be easily ported to other domains and styles of computation.

Table 6.1 shows the characteristics in which each algorithm has showed to excel at. Frag-Cubing is still preferred when the data has a low degree of sequentiality, as there's little advantage in using the IntervalFrag scheme when the intervals are closer to the size of the original list. On those cases, IntervalFrag is discouraged, as the algorithm will be slower than Frag-Cubing's by simple virtue of needing more instructions to answer the same query, being up to 400% slower than the same query under Frag-Cubing.

Table 6.1 - Preferred algorithm to use

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

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

In summary, in case the RAM available is low and the worst case query response times can be up to 4x slower than Frag-Cubing's, then IntervalFrag should be preferred as the main method of computing the data cube. In case the data have a very low sequentiality, and RAM is available, then using Frag-Cubing should still be preferred for the faster response times. IntervalFrag will excel at any dimension that has a high sequentiality, even if it also has a high cardinality, however dimensions that have a high cardinality will tend to have a low sequentiality in practice, and this usage might be rarer. The Skew parameter can influence the algorithm both

ways, as it does not necessarily mean that the dimension will have a higher or lower sequentiality and cardinalities.

When the dimensions have a high degree of sequentiality, then IntervalFrag excels, as it can not only answer the same queries much faster, but also using only a fraction of the memory used by Frag-Cubing. Furthermore, Frag-Cubing used much less memory to answer queries $Q1$, $Q2$ and $Q5$, with $Q4$ having a small difference and $Q3$ having no difference in memory usage in the end, when compared with cubes $C1$ to $C5$. All queries executed on the $C0$ cube with all dimensions used only a fraction of the memory needed to answer queries with IntervalFrag, they were however in general slower to answer.

From the tests made using Frag-Cubing and the different cubes ($C1$ to $C5$) tailored to specific queries, it was shown that the best algorithms can be further enhanced by doing some simple pre-processing of the queries, and depending on the type of query used they can drastically improve upon memory usage requirements, allowing for some frequent queries to be optimized and even allowing for queries that could not be answered under a $C0$ cube to be answered by smaller cubes. In chapter 4 it was shown that it is faster to load a smaller subset of the data in memory as prepared files when needed and then computing the answer from that file instead of querying a cube that was already loaded in memory, but that used the full dimensional capability of the data.

ADD AT LEAST ONE GRAPH OR TABLE COMPARING THEM. THE ONE USED FOR THE PRESENTATION IS FINE

It is important to note IntervalFrag had faster file reading speeds (about 12% faster) due to improvements made on the implementation, as well as a slightly more efficient set intersection algorithm, which were not backported to Frag-Cubing. This was done to preserve the Frag-Cubing algorithm's performance, as the original code was made for the C language in 2002 and the updated IntervalFrag implementation uses C++17 standards. Nonetheless, it was possible to compile Frag-Cubing using the same flags as IntervalFrag under the GNU C++ compiler, with minimal performance differences.

The difference in query response times from IntervalFrag and Frag-Cubing, even when using the same intersection algorithm, was due to IntervalFrag having to do more comparisons to answer the same query, and this implementation could not be further optimized without heavily skewing the response times to IntervalFrag's side,

by using other techniques that could also be ported to Frag-Cubing trivially. Further details on the intersection algorithms tested and their performance differences can be found on Appendix A.1.

7 CONCLUSIONS

This work shows that it is possible to further optimize data cube algorithms by gathering information from the underlying data, and how this can be made to aid the end user's experience by decreasing implementation requirements and improving response times.

7.1 Main contributions

One of the stated purposes of this work was to find ways of using the data's domain characteristics to improve the satellite operator's day to day activities, and this work has achieved three main results:

a) A heuristic to discover related telemetries between satellite time series data and how to use this with the help of an operator to validate the relevant queries;

b) Using the previous heuristic to enhance Frag-Cubing's query response time and memory by pre-partitioning the data;

c) Improving upon Frag-Cubing's Inverted Index memory model by saving only intervals instead of the entire values, and thus reducing memory and query response times for some queries;

This is still all wrong. What do?

7.2 Future work

The natural evolution of this work would be to test it using other data cube algorithms, as there's a great variety of them mentioned in section 3 and not all of them might be applicable to satellite telemetry data, or showcase useful performance metrics. On that note the use of bCubing (SILVA, 2015) will be interesting, as the inverted index separation into blocks can further improve upon the memory usage as described in this chapter.

The use of the gathered satellite data on other projects is also of interest, as there's no public reliable dataset of satellite telemetry data that contains all relevant data and not just a subset of a subsystem, and this work showcases a volume that has information enough for the training of Machine Learning and Artificial Intelligence projects. Only projects that release full telemetry data are relatively simple CubeSat

projects, who do not generate a significant volume that is enough for the use of these algorithms. The author plans to release the dataset in a citable format for the use of the community in the near future.

This work also has the potential of improving query execution when dealing with multiple satellites, constellations and/or formations, it needing only the data to be gathered and the suitable cube format defined to be tested.

The relationship algorithms mentioned in section 4.2 can be remade to use other different solutions, and combined with the shell-cubing method to generate only shells that have relationships above a certain strength. This was also one of the ideas to be developed during this work, which however had not enough time to be fully developed. This idea is best when paired with known "best available" techniques, like using the project (LIBRESPACEFOUNDATION..., 2021).

Furthermore the Set Intersection problem defined in chapter 5 can be further optimized with recent advances not only in computer architectures, but also with regards to complexity and the validation of the algorithms in real world datasets. A preliminary investigation was performed, as a simple but not rigorous overview of the results is detailed in Appendix .

7.3 Final thoughts

The approach and architecture detailed in this work...

This work was developed entirely with open source software, and they will be made available later at `https://github.com/Yuri-M-Dias/SCD2`. Furthermore, there is a lack of good datasets that deal with satellite telemetry data available, perhaps this work can further contribute by allowing the usage of the dataset by making it public. The volume available here is much bigger than what is currently used by Machine Learning competitions, and the publication can further enhance work in this area.

REFERENCES

ADAMSKI, G. Data Analytics for Large Constellations. In: **SpaceOps 2016 Conference**. [S.l.]: American Institute of Aeronautics and Astronautics, 2016. (SpaceOps Conferences). 23, 24

AZEVEDO, D. N. R.; AMBRÓSIO, A. M. Dependability in Satellite Systems: An Architecture for Satellite Telemetry Analysis. In: WORKSHOP EM ENGENHARIA E TECNOLOGIA ESPACIAIS, 1. (WETE)., 30 mar. - 1 abr. 2010, São José dos Campos. **Anais...** São José dos Campos: INPE, 2010. IWETE2010-1065, p. 6. ISSN 2177-3114. 1

AZEVEDO, D. N. R.; AMBRÓSIO, A. M.; VIEIRA, M. **Estudo sobre técnicas de detecção automática de anomalias em satélites**. PhD Thesis (PhD) — Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2011. 25

BEYER, K.; RAMAKRISHNAN, R. Bottom-up Computation of Sparse and Iceberg CUBE. In: **Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data**. New York, NY, USA: ACM, 1999. (SIGMOD '99), p. 359–370. ISBN 978-1-58113-084-3. 18

BIMONTE, S. Open issues in Big Data Warehouse design. **Revue des Nouvelles Technologies de l'Information**, p. 10, 2016. 3, 8, 10

BOUSSOUF, L.; BERGELIN, B.; SCUDELER, D.; GRAYDON, H.; STAMMINGER, J.; ROSNET, P.; TAILLEFER, E.; BARREYRE, C. Big Data Based Operations for Space Systems. In: **2018 SpaceOps Conference**. [S.l.]: American Institute of Aeronautics and Astronautics, 2018. 8, 23, 24

CODD, E. F.; CODD, S.; SALLEY, C. Providing olap to user-analysts: An it mandate. In: . [S.l.: s.n.], 1998. 10

DING, B.; KÖNIG, A. C. Fast Set Intersection in Memory. **arXiv:1103.2409 [cs]**, mar. 2011. 71

DISCHNER, Z.; REDFERN, J.; ROSE, D.; ROSE, R.; RUF, C.; VINCENT, M. CYGNSS MOC; Meeting the challenge of constellation operations in a cost-constrained world. In: **2016 IEEE Aerospace Conference**. [S.l.: s.n.], 2016. p. 1–8. 24

DOKA, K.; TSOUMAKOS, D.; KOZIRIS, N. Brown Dwarf: A fully-distributed, fault-tolerant data warehousing system. **Journal of Parallel and Distributed Computing**, v. 71, n. 11, p. 1434–1446, nov. 2011. ISSN 0743-7315. 25

Dong Xin; Zheng Shao; Jiawei Han; Hongyan Liu. C-Cubing: Efficient Computation of Closed Cubes by Aggregation-Based Checking. In: **22nd International Conference on Data Engineering (ICDE'06)**. [S.l.: s.n.], 2006. p. 4–4. 19, 26

EDWARDS, T. Dealing with the Big Data - The Challenges for Modern Mission Monitoring and Reporting. In: **15th International Conference on Space Operations**. Marseille, France: American Institute of Aeronautics and Astronautics, 2018. ISBN 978-1-62410-562-3. 24

EMANI, C. K.; CULLOT, N.; NICOLLE, C. Understandable Big Data: A survey. **Computer Science Review**, v. 17, p. 70–81, aug. 2015. ISSN 1574-0137. 3, 8

EVANS, D. J.; MARTINEZ, J.; Korte-Stapff, M.; VANDENBUSSCHE, B.; ROYER, P.; RIDDER, J. D. Data Mining to Drastically Improve Spacecraft Telemetry Checking: A Scientist?s Approach. In: **SpaceOps 2016 Conference**. [S.l.]: American Institute of Aeronautics and Astronautics, 2016, (SpaceOps Conferences). 24

FEN, Z.; YANQIN, Z.; CHONG, C.; LING, S. Management and Operation of Communication Equipment Based on Big Data. In: **2016 International Conference on Robots Intelligent System (ICRIS)**. [S.l.: s.n.], 2016. p. 246–248. 24

FERNÁNDEZ, M. M.; YUE, Y.; WEBER, R. Telemetry Anomaly Detection System Using Machine Learning to Streamline Mission Operations. In: **2017 6th International Conference on Space Mission Challenges for Information Technology (SMC-IT)**. [S.l.: s.n.], 2017. p. 70–75. 23, 24

FILHO, A. C. J.; AMBRÓSIO, A. M.; FERREIRA, M. G. V.; LOUREIRO, G. The Amazonia-1 satellite's ground segment - challenges for implementation of the space link extension protocol services. In: INTERNATIONAL ASTRONOMICAL CONGRESS, 68. (IAC), 25-29 Sept., Adelaide, Australia. **Proceedings...** [S.l.], 2017. p. 1–12. 1

GILLES, K. Flying Large Constellations Using Automation and Big Data. In: **SpaceOps 2016 Conference**. [S.l.]: American Institute of Aeronautics and Astronautics, 2016, (SpaceOps Conferences). 24

GRAY, J.; BOSWORTH, A.; LYAMAN, A.; PIRAHESH, H. Data cube: A relational aggregation operator generalizing GROUP-BY, CROSS-TAB, and SUB-TOTALS. In: . [S.l.]: IEEE Comput. Soc. Press, 1996. p. 152–159. ISBN 978-0-8186-7240-8. 3, 11, 17

HAN, J.; KAMBER, M.; PEI, J. **Data Mining: Concepts and Techniques, Third Edition**. 3 edition. ed. Haryana, India; Burlington, MA: Morgan Kaufmann, 2011. ISBN 978-93-80931-91-3. 4, 9, 11, 13, 14, 15, 16, 17, 18, 19, 25

HEIDORN, P. B. Shedding Light on the Dark Data in the Long Tail of Science. **Library Trends**, v. 57, n. 2, p. 280–299, 2008. ISSN 1559-0682. 2

HEINE, F.; ROHDE, M. PopUp-Cubing: An Algorithm to Efficiently Use Iceberg Cubes in Data Streams. In: **Proceedings of the Fourth IEEE/ACM International Conference on Big Data Computing, Applications and Technologies**. New York, NY, USA: ACM, 2017. (BDCAT '17), p. 11–20. ISBN 978-1-4503-5549-0. 25

HENNION, N. Big-data for satellite yearly reports generation. In: **2018 SpaceOps Conference**. [S.l.]: American Institute of Aeronautics and Astronautics, 2018. 24

IEEE. IEEE Standard for Floating-Point Arithmetic. **IEEE Std 754-2019 (Revision of IEEE 754-2008)**, 2019. 46

ILLIMINE. **Software and Data Repository from Data Mining Research Group, Data and Information Systems (DAIS) Research Laboratory**. 2004. 36

INMON, W. H.; HACKATHORN, R. D. **Using the Data Warehouse**. Somerset, NJ, USA: Wiley-QED Publishing, 1994. ISBN 978-0-471-05966-0. 9

INOUE, H.; OHARA, M.; TAURA, K. Faster set intersection with SIMD instructions by reducing branch mispredictions. **Proceedings of the VLDB Endowment**, v. 8, n. 3, p. 293–304, nov. 2014. ISSN 2150-8097. 71

KIMBALL, R.; ROSS, M. **The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling**. Edição: 3rd. Indianapolis, IN: John Wiley & Sons, 2013. ISBN 978-1-118-53080-1. 9, 10

KRAG, H.; SERRANO, M.; BRAUN, V.; KUCHYNKA, P.; CATANIA, M.; SIMINSKI, J.; SCHIMMEROHN, M.; MARC, X.; KUIJPER, D.; SHURMER, I.;

O'CONNELL, A.; OTTEN, M.; MUÑOZ, I.; MORALES, J.; WERMUTH, M.; MCKISSOCK, D. A 1 cm space debris impact onto the Sentinel-1A solar array. **Acta Astronautica**, v. 137, p. 434–443, aug. 2017. ISSN 0094-5765. 7

LAKSHMANAN, L. V. S.; PEI, J.; HAN, J. Quotient Cube: How to Summarize the Semantics of a Data Cube. In: **Proceedings of the 28th International Conference on Very Large Data Bases**. [S.l.]: VLDB Endowment, 2002. (VLDB '02), p. 778–789. 19

LARSON, W. J.; WERTZ, J. R. (Ed.). **Space Mission Analysis and Design, 3rd Edition**. 3rd edition. ed. El Segundo, Calif. : Dordrecht ; Boston: Microcosm, 1999. ISBN 978-1-881883-10-4. 1, 7

LI, X.; HAN, J.; GONZALEZ, H. High-dimensional OLAP: A Minimal Cubing Approach. In: **Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30**. [S.l.]: VLDB Endowment, 2004. (VLDB '04), p. 528–539. ISBN 978-0-12-088469-8. 19, 25, 26, 45, 48

LIBRESPACEFOUNDATION / Polaris / Polaris. 2021. Https://gitlab.com/librespacefoundation/polaris/polaris. 60

LIMA, J. d. C. **SEQUENTIAL AND PARALLEL APPROACHES TO REDUCE THE DATA CUBE SIZE**. PhD Thesis (PhD) — Instituto Tecnológico de Aeronáutica, São José dos Campos, 2009. 12

MAGALHÃES, R. O. de. **Estudo de avalanche térmica em um sistema de carga e descarga de bateria em satélites artificiais**. PhD Thesis (PhD) — Instituto Nacional de Pesquisas Espaciais, São José dos Campos, feb. 2012. 23

MATEIK, D.; MITAL, R.; BUONAIUTO, N. L.; LOUIE, M.; KIEF, C.; AARESTAD, J. Using Big Data Technologies for Satellite Data Analytics. In: . [S.l.]: American Institute of Aeronautics and Astronautics, 2017. ISBN 978-1-62410-483-1. 23, 24

MIGUEZ, R. R. B.; SILVA, M. M. Q. da; KONO, J. **SCD2 Operation Handbook**. [S.l.], 1993. 29

MONTEIRO, D. V. **A FRAMEWORK FOR TRAJECTORY DATA MINING**. PhD Thesis (PhD), 2017. 25

MOREIRA, A. A.; LIMA, J. d. C. Full and partial data cube computation and representation over commodity PCs. In: **2012 IEEE 13th International**

Conference on Information Reuse Integration (IRI). [S.l.: s.n.], 2012. p. 672–679. 10

OLIVEIRA, F. **O Brasil Chega Ao Espaco: SCD 1 Satelite de Coleta de Dados**. Sao Paulo, 1996. 972 p. 29

ORLANDO, V.; KUGA, H. K. Os satélites SCD1 e SCD2 da Missão Espacial Completa Brasileira - MECB. In: Othon Cabo Winter; PRADO, A. F. B. d. A. (Ed.). **A Conquista Do Espaço: Do Sputnik à Missão Centenário**. cap. 5. São Paulo: Editora Livraria da Física, 2007. p. . ISBN 978-85-88325-89-0. 29

PEREIRA, Y. M. D.; AMAURI, S. C.; JUNQUEIRA, B. C.; RAIMUNDI, L. R.; GUIMARÃES, S. G.; LOUREIRO, G. Lessons learned on Systems of Systems Engineering: Systems Concurrent Engineering of a Constellation of Cubesat Formations. In: **70 Th International Astronautical Congress (IAC)**. Washington DC: [s.n.], 2019. 73

PEREIRA, Y. M. D.; FERREIRA, M. G. V.; SILVA, R. R. A study for the application of OLAP in satellite telemetry data. In: **Workshop em Engenharia e Tecnologia Espaciais, 9. (WETE)**. São José dos Campos: Instituto Nacional de Pesquisas Espaciais (INPE), 2018. ISSN 2177-3114. 73

PIBIRI, G. E. Fast and Compact Set Intersection through Recursive Universe Partitioning. In: **IEEE Data Compression Conference**. [S.l.: s.n.], 2021. p. 10. 71

PIBIRI, G. E.; VENTURINI, R. Techniques for Inverted Index Compression. **arXiv:1908.10598 [cs]**, aug. 2019. 45, 71

RAMOS, M. P.; TASINAFFO, P. M.; de Almeida, E. S.; ACHITE, L. M.; da Cunha, A. M.; DIAS, L. A. V. Distributed Systems Performance for Big Data. In: LATIFI, S. (Ed.). **Information Technology: New Generations**. [S.l.]: Springer International Publishing, 2016, (Advances in Intelligent Systems and Computing). p. 733–744. ISBN 978-3-319-32467-8. 25

SCHULSTER, J.; EVILL, R.; PHILLIPS, S.; FELDMANN, N.; ROGISSART, J.; DYER, R.; ARGEMANDY, A. CHARTing the Future – An offline data analysis and reporting toolkit to support automated decision-making in flight-operations. In: **15th International Conference on Space Operations**. Marseille, France: American Institute of Aeronautics and Astronautics, 2018. ISBN 978-1-62410-562-3. 24

SILVA, R. R. **Abordagens para Cubo de Dados Massivos com Alta Dimensionalidade Baseadas em Memória Principal e Memória Externa: HIC e BCubing**. PhD Thesis (PhD) — Instituto Tecnológico de Aeronáutica, São José dos Campos, 2015. 4, 9, 19, 20, 26, 48, 59

SILVA, R. R.; HIRATA, C. M.; LIMA, J. d. C. A Hybrid Memory Data Cube Approach for High Dimension Relations. In: HAMMOUDI, S.; MACIASZEK, L. A.; TENIENTE, E. (Ed.). **ICEIS 2015 - Proceedings of the 17th International Conference on Enterprise Information Systems, Volume 1, Barcelona, Spain, 27-30 April, 2015**. [S.l.]: SciTePress, 2015. p. 139–149. ISBN 978-989-758-096-3. 26

_____. Computing BIG data cubes with hybrid memory. **Journal of Convergence Information Technology**, v. 11, n. 1, p. 18, jan. 2016. 26

SILVA, R. R.; LIMA, J. d. C.; HIRATA, C. M. qCube: Efficient integration of range query operators over a high dimension data cube. **JIDM**, v. 4, n. 3, p. 469–482, 2013. 26

SIMÕES, R. E. d. O.; CAMARA, G.; QUEIROZ, G. R. de. Sits: Data analysis and machine learning using satellite image time series. In: Workshop de Computação Aplicada, 18. (WORCAP), 21-23 ago., São José dos Campos, SP. **Resumos...** [S.l.], 2018. p. 18. 25

spektr. **The Easiest Way to Find Intersection of Two Intervals**. 2017. Computational Science Stack Exchange. 48

TIAN, S.; WANG, P.; CHEN, X. Quantum Algorithm for Finding Sets Intersection. In: **2019 International Conference on Intelligent Computing, Automation and Systems (ICICAS)**. Chongqing, China: IEEE, 2019. p. 843–847. ISBN 978-1-72816-106-8. 71

TOMINAGA, J.; FERREIRA, M. G. V.; AMBRÓSIO, A. M. Comparing satellite telemetry against simulation parameters in a simulator model reconfiguration tool. In: CERQUEIRA, C. S.; BÜRGER, E. E.; YASSUDA, I. d. S.; RODRIGUES, I. P.; LIMA, J. S. d. S.; OLIVEIRA, M. E. R. de; TENÓRIO, P. I. G. (Ed.). **Anais...** São José dos Campos: Instituto Nacional de Pesquisas Espaciais (INPE), 2017. ISSN 2177-3114. 23

TROLLOPE, E.; DYER, R.; FRANCISCO, T.; MILLER, J.; GRISO, M. P.; ARGEMANDY, A. Analysis of automated techniques for routine monitoring and

contingency detection of in-flight LEO operations at EUMETSAT. In: **2018 SpaceOps Conference**. Marseille, France: American Institute of Aeronautics and Astronautics, 2018. ISBN 978-1-62410-562-3. 23, 24

WANG, Z.; CHU, Y.; TAN, K.-L.; AGRAWAL, D.; ABBADI, A. E.; XU, X. Scalable Data Cube Analysis over Big Data. **arXiv:1311.5663 [cs]**, nov. 2013. 25

YANG, H.; HAN, C. Holistic and Algebraic Data Cube Computation Using MapReduce. In: **2017 9th International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC)**. [S.l.: s.n.], 2017. v. 2, p. 47–50. 25

YVERNES, A. Copernicus Ground Segment as a Service: From Data Monitoring to Performance Analysis. In: **15th International Conference on Space Operations**. Marseille, France: American Institute of Aeronautics and Astronautics, 2018. ISBN 978-1-62410-562-3. 3, 23, 24

ZHANG, J.; LU, Y.; SPAMPINATO, D. G.; FRANCHETTI, F. FESIA: A Fast and SIMD-Efficient Set Intersection Approach on Modern CPUs. In: **2020 IEEE 36th International Conference on Data Engineering (ICDE)**. Dallas, TX, USA: IEEE, 2020. p. 1465–1476. ISBN 978-1-72812-903-7. 71

ZHANG, X.; WU, P.; TAN, C. A big data framework for spacecraft prognostics and health monitoring. In: **2017 Prognostics and System Health Management Conference (PHM-Harbin)**. [S.l.: s.n.], 2017. p. 1–7. 8, 21, 22, 23, 24

ZHAO, Q.; ZHU, Y.; WAN, D.; TANG, S. A Closed Frag-Shells Cubing Algorithm on High Dimensional and Non-Hierarchical Data Sets. In: **Proceedings of the 12th International Conference on Ubiquitous Information Management and Communication**. New York, NY, USA: ACM, 2018. (IMCOM '18), p. 6:1–6:8. ISBN 978-1-4503-6385-3. 26

APPENDIX A - INTERSECTION ALGORITHMS

A.1 Problem

This is only a simple overview to show the importance of the problem and how different algorithms stack against each other.

The problem can be stated as follows: given sets S_1 and S_2, find the elements that are present in both sets, their intersection, represented as $S_1 \cap S_2$. Furthermore, each element is ordered as unique, for they represented index positions and are thus always non-zero positive integers.

A.2 Algorithms

All wrong for now

UnorderedSet

Scalar

Li

BinaryLi

std::set_intersect

SIMD (SS2)

A.3 Experiments

To search for the best algorithm with real world data, an experiment was performed, much on the same framework as detailed in **??**: C++ code, each test was executed 5 times and the median of the values was taken. As each technique is efficient with the memory usage, there wasn't much difference to be measured, so only the time necessary to intersect each list was measured. Each list was generated in interval from 2×10^6 to 1×10^7, and all algorithms work on randomized ordered lists with the same size. This spread was intentional to mirror the worst cases in the Frag-Cubing algorithm.

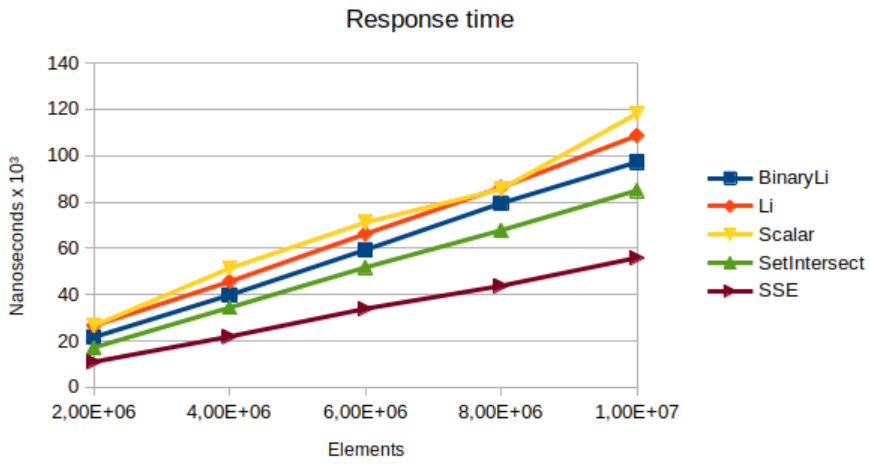
Table A.1 showcases a summary of the experiment's results, this time caring only for the necessary time to answer the queries.

Table A.1 - Set Intersection Results, in milliseconds

Algorithm - N	2×10^6	4×10^6	6×10^6	8×10^6	1×10^7
UnorderedSet	867,802	1806,19	2586,04	3448,11	4213,15
Scalar	26,57	51,346	71,109	85,531	118,114
Li	26,531	45,596	66,19	86,234	108,603
BinaryLi	21,601	39,776	59,27	79,416	97,155
std::set_intersect	17,125	34,392	51,682	67,717	84,933
SIMD (SS2)	10,941	21,854	33,866	43,739	55,814

As the HashSet approach was the slowest approach, Figure A.1 showcases the comparison between the other algorithms, that have comparable performances and are easier to visualize.

Figure A.1 - Set Intersection Algorithm results



Set intersection algorithms response times in milliseconds, ordered by the size of the input relation.

SOURCE: Author

The SIMD approach is clearly the fastest, however it is also dependant on processor architecture and even though the used SIMD instructions (SS2) are available on almost all modern CPUs, other newer instruction sets might not be, or might have different implementations depending on the CPU vendor, which complicates widespread implementation.

While these tests are interesting, there was not enough time to test some recent benchmarks that found the use of different algorithms and could improve the performance of each, as the use of compressed indexes in (PIBIRI; VENTURINI, 2019) show. Furthermore, the SIMD instruction set used here (SSE2) is limited even if it's support is widespread, with other instruction sets (SSE3, AVX, AVX512, etc) being available on modern CPUs and also available for use.

Some recent results that have not been properly explored in the data cube context as of yet: Recursive Universe Partitioning, a technique that uses the possible search space to partition the sets and execute the intersection (PIBIRI, 2021); FESIA, which combines the use of previous techniques with different SIMD computation techniques and a bitmap to decide which algorithm is more suitable for use depending on the set size (ZHANG et al., 2020); and simpler implementations that reduce branch mispredictions (INOUE et al., 2014), and the use of pre-processed dictionaries to greatly aid in the computation (DING; KÖNIG, 2011). There's even an algorithm to compute the set intersection in $\mathcal{O}(1)$ by using a quantum computer (TIAN et al., 2019), however that approach is likely not implementable for any significant dataset in the near future.

Further testing is necessary when dealing with lists of varying sizes, that would showcase the improvements of certain algorithms over others, and the incorporation of these algorithms into a real-world dataset for accurate tests.

ANNEX A - PUBLICATIONS

This annex showcases the publications that resulted from this work, and from the general Master's effort. Table A.1 shows the summary of the published, and currently expecting to be published articles.

Table A.1 - Resulting published work

Name	QUALIS	SCOPUS Percentile	Source	Status
WETE 2018	NA - Conference	-	(PEREIRA et al., 2018)	Published
IAC 2019	NA - Conference	-	(PEREIRA et al., 2019)	Published
MDPI Information	B2 (A4)	52%	-	Accepted, later retracted
IEEE Latin America Transactions	B2	61%	-	Submitted
IntervalFrag	-	-	-	Writing

Furthermore, the table shows that the last two articles are being written, and will be published shortly with the results derived from this work.

Tem os artigos do CubeDesign + Jenny/LIT que faltam aqui, coloco eles ou não? Eu tive pouca atuação, não sei se é relevante colocar aqui

PUBLICAÇÕES TÉCNICO-CIENTÍFICAS EDITADAS PELO INPE

Teses e Dissertações (TDI)

Teses e Dissertações apresentadas nos Cursos de Pós-Graduação do INPE.

Notas Técnico-Científicas (NTC)

Incluem resultados preliminares de pesquisa, descrição de equipamentos, descrição e ou documentação de programas de computador, descrição de sistemas e experimentos, apresentação de testes, dados, atlas, e documentação de projetos de engenharia.

Propostas e Relatórios de Projetos (PRP)

São propostas de projetos técnico-científicos e relatórios de acompanhamento de projetos, atividades e convênios.

Publicações Seriadas

São os seriados técnico-científicos: boletins, periódicos, anuários e anais de eventos (simpósios e congressos). Constam destas publicações o Internacional Standard Serial Number (ISSN), que é um código único e definitivo para identificação de títulos de seriados.

Pré-publicações (PRE)

Todos os artigos publicados em periódicos, anais e como capítulos de livros.

Manuais Técnicos (MAN)

São publicações de caráter técnico que incluem normas, procedimentos, instruções e orientações.

Relatórios de Pesquisa (RPQ)

Reportam resultados ou progressos de pesquisas tanto de natureza técnica quanto científica, cujo nível seja compatível com o de uma publicação em periódico nacional ou internacional.

Publicações Didáticas (PUD)

Incluem apostilas, notas de aula e manuais didáticos.

Programas de Computador (PDC)

São a seqüência de instruções ou códigos, expressos em uma linguagem de programação compilada ou interpretada, a ser executada por um computador para alcançar um determinado objetivo. Aceitam-se tanto programas fonte quanto os executáveis.