



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 Es-
paciais. Orientada pelo Dr. Mauri-
cio Gonçalves Vieira Ferreira e pelo
Dr. Rodrigo Rocha Silva

URL of the original document:

[<http://urlib.net/>](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

BOARD OF PUBLISHING AND PRESERVATION OF INPE INTELLECTUAL PRODUCTION - CEPPII (PORTARIA Nº 176/2018/SEI-INPE):**Chairperson:**

Marciana Leite Ribeiro - Serviço de Informação e Documentação (SID)

Members:

Dr. Gerald Jean Francis Banon - Coordenação Observação da Terra (OBT)

Dr. Amauri Silva Montes - Coordenação Engenharia e Tecnologia Espaciais (ETE)

Dr. André de Castro Milone - Coordenação Ciências Espaciais e Atmosféricas (CEA)

Dr. Joaquim José Barroso de Castro - Centro de Tecnologias Espaciais (CTE)

Dr. Manoel Alonso Gan - Centro de Previsão de Tempo e Estudos Climáticos (CPT)

Dr^a Maria do Carmo de Andrade Nono - Conselho de Pós-Graduação

Dr. Plínio Carlos Alvalá - Centro de Ciência do Sistema Terrestre (CST)

DIGITAL LIBRARY:

Dr. Gerald Jean Francis Banon - Coordenação de Observação da Terra (OBT)

Clayton Martins Pereira - Serviço de Informação e Documentação (SID)

DOCUMENT REVIEW:

Simone Angélica Del Ducca Barbedo - Serviço de Informação e Documentação (SID)

Yolanda Ribeiro da Silva Souza - Serviço de Informação e Documentação (SID)

ELECTRONIC EDITING:

Marcelo de Castro Pazos - Serviço de Informação e Documentação (SID)

André Luis Dias Fernandes - Serviço de Informação e Documentação (SID)



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 Es-
paciais. Orientada pelo Dr. Mauri-
cio Gonçalves Vieira Ferreira e pelo
Dr. Rodrigo Rocha Silva

URL of the original document:

[<http://urlib.net/>](http://urlib.net/)

INPE
São José dos Campos
2021

Cataloging in Publication Data

Sobrenome, Nomes.

Cutter Data Cube Algorithm for High Sequentiality Satellite Teleme-
try Data Analysis / Nome Completo do Autor1; Nome Completo
do Autor2. – São José dos Campos : INPE, 2021.
xxiii + 81 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



Esta obra foi licenciada sob uma [Licença Creative Commons Atribuição-NãoComercial 3.0 Não Adaptada](#).

This work is licensed under a [Creative Commons Attribution-NonCommercial 3.0 Unported License](#).

**ATENÇÃO! A FOLHA DE
APROVAÇÃO SERÁ INCLU-
IDA 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 AND ANN DRUYAN
in “The Demon-Haunted World:
Science as a Candle in the Dark”, 1995

*Ao meu avô **Antônio Macena***

ACKNOWLEDGEMENTS

Primeiramente gostaria de agradecer a toda a minha família, sem a qual nada disso seria possível, em especial a minha mãe Xarlene e minha madrinha Araújo por sempre acreditarem em mim, minha irmã favorita do mundo Natália, meu pai Irair, minhas avós Genoveva e Maria das Graças e meu padrasto Eudes. Também a todos os inúmeros parentes que me acolheram de alguma forma, seja no Tocantins, em Goiás, no Distrito Federal ou em São Paulo. Em especial ao meu avô Antônio Macena, que nos deixou muito cedo em um acidente logo após o início deste trabalho.

Ao Dr. Maurício, por me acolher no INPE, aceitado como aluno e me abrir as portas do CCS, meu sincero obrigado pela oportunidade e por todo o suporte que me forneceu.

Ao Dr. Rodrigo, por aceitar um completo desconhecido como aluno, e me ensinar tantas coisas sobre computação ao longo desse tempo, e pelos puxões de orelha merecidos.

Aos meus colegas Bruno e Gabriela que aceitaram dividir apartamento comigo, e me aguentarem por todo esse tempo.

Ao Ítalo, Isomar, Danilo e Johnathan pela amizade, tantos almoços compartilhados e por serem estarem disponíveis para uma conversa aleatória sobre algum conceito espacial obscuro de um manual da União Soviética dos anos 70.

As comissões organizadoras do WETE e do CubeDesign que me permitiram ajudar a organizar esses eventos incríveis, e pelas amizades feitas quando todos trabalham por um mesmo objetivo.

Ao Jun, Pascote, Maria do Carmo, secretarias e todos os trabalhadores do CCS, pelas imensa ajuda dentro do prédio do CCS ao longo dos anos e por sempre proverem o melhor suporte para quem está perdido.

Aos seguranças do INPE, em especial ao Eduardo pelas conversas que passavam da meia noite.

Aos membros do projeto CITAR por compartilharem tantos almoços e piadas, bem como informações importantes da área. Nunca iria acreditar que questões do Stack-Overflow poderiam acabar em código de míssil sem vocês.

A todos os membros da biblioteca do INPE, que ao longo dos anos proveram suporte para encontrar os melhores livros, e aguentaram minhas constantes visitas para renovar livros.

Ao INPE e todos os funcionários que proveram todas a infraestrutura necessária para este trabalho, em especial as secretarias da pós-graduação que estão sempre disponíveis para responder perguntas.

我慢してくれた雑種に心から感謝します.

E finalmente, a Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pela bolsa de estudos para executar este trabalho.

ABSTRACT

Satellites are monitored by ground teams via telemetry packages, which report the current status of the equipment and allow them to assess the satellite's ability to continue its mission. These telemetry packages compose a large and complex body of data, with satellites that are operated for several years generating large volumes of historical data that is still useful for operation activities and needs to be archived. The volume of historical telemetry data available to the National Institute for Space Research (INPE) is currently estimated to be at least 3 *terabytes* in total, with a tendency to grow in the coming years. With this volume, and considering that the data analysis on these data is not trivial, requiring expert engineering knowledge, it is necessary to implement **specialized systems** to perform queries and analysis on them. In this work we identify the queries that are of interest to satellite operators, create a multidimensional model for the telemetry data using a data cube model, and then use the Frag-Cubing data cube computation algorithm as a basis for implementation. First an approach that uses pre-processing of the selected queries is implemented, where the dimensions related to the query are filtered out and low-dimensional cubes are created from them. This approach is compared to the high dimensionality approach that uses all available dimensions, and finds that, while queries are restricted to the filtered dimensions, it has a 15% advantage in query time and in the best cases consumes only 10% of the memory used by the high dimensionality approach. So if the queries have a low dimensionality, there is advantage in using a pre-processed cube from disk than running a query on a data cube already built with the high dimensionality approach. Then an approach based on modifying the Frag-Cubing inverted index algorithm is experimentally validated, which consists in using the high-sequentiality characteristic of some satellite telemetry to replace the lists of tuple identifiers (*TID list*) with lists of intervals.. This approach on high dimensional data, tested on the queries defined by the operators, uses on average 20% of the memory that traditional lists use, and is up to 3200% faster to answer queries on dimensions with high sequentiality, while being up to 400% slower to answer queries on dimensions with low sequentiality.

Keywords: Data Cube. Inverted Index. Satellite. Telemetry. Satellite Operations.

ALGORITMO DE CUBO DE DADOS PARA DADOS DE TELEMETRIA DE SATÉLITE COM ALTA SEQUENCIALIDADE

RESUMO

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 elevado tamanho e complexidade, com satélites que são operados por vários anos geram dados históricos de grande volume, ainda úteis para as atividades de operação e que necessitam de ser arquivados. O volume de dados históricos de telemetria disponíveis ao Instituto Nacional de Pesquisas Espaciais (INPE) atualmente é estimado em ao menos 3 *terabytes* no total, com tendência a crescer nos próximos anos. Com este volume, e considerando que as análises de dados sobre esse arquivos não é trivial, necessitando de conhecimento especialista de engenharia, é necessário a implementação de sistemas especializados para realizar consultas e análises sobre esses dados. Neste trabalho é feita a identificação as consultas que são de interesse dos operadores de satélite, uma modelagem multidimensional para os dados de telemetria utilizando de cubo de dados é criada e então os algoritmos de computação do cubo de dados Frag-Cubing é utilizado como base de implementação. Primeiramente uma abordagem que utiliza de pré-processamento das consultas selecionados é implementada, onde as dimensões relacionadas a consulta são filtradas e cubos de baixa dimensionalidade são criados à partir delas. Essa abordagem é comparada com a abordagem de alta dimensionalidade que utiliza de todas as dimensões disponíveis, e encontra que, conquanto que as consultas sejam restritas as dimensões filtradas, tem uma vantagem de 15% no tempo de consulta e nos melhores casos consumindo apenas 10% de memória utilizada pela abordagem de alta dimensionalidade. Assim, se as consultas tiverem uma dimensionalidade baixa, existe vantagem em utilizar um cubo preprocessado do zero do que executar uma consulta em uma cubo de dados construído com abordagem de alta dimensionalidade. Depois uma abordagem baseada na alteração do algoritmo de **índice invertido do Frag-Cubing** é experimentalmente validada, que compõe em utilizar da característica de alta sequencialidade de algumas telemetrias de satélite para substituir as listas de identificadores de tuplas (*TID list*) por listas de intervalos. Essa abordagem sobre os dados de alta dimensionalidade, testada nas consultas definidas pelos operadores anteriormente, usa em média 20% da memória que a listas tradicional utiliza, **e é até 3200% mais rápida para responder consultas em dimensões com alta sequencialidade, porém sendo até 400% mais lenta para responder consultas com dimensões com baixa sequencialidade.**

Palavras-chave: Cubo de Dados. Índice Invertido. Satélite. Telemetria. Operação de Satélites.

LIST OF FIGURES

	<u>Page</u>
1.1 Historic telemetry data generation	2
2.1 Data Cube example	11
2.2 All subcubes for a three dimensional cube	12
2.3 Star schema	13
2.4 Snowflake schema	13
2.5 Fact Constellation scheme	14
2.6 OLAP operations in a Data Cube	16
2.7 Computing the data cube with the Top-Down strategy	18
2.8 Computing the data cube with the Bottom-Up strategy	19
3.1 Data flow in a Big Data architecture	22
3.2 Inverted Index computation example	26
3.3 Shell Fragmentation example	27
4.1 SCD2	30
4.2 General Data Cube Architecture	31
4.3 General Data Cube Architecture	32
5.1 Query 1 results	42
5.2 Query 2 results	43
5.3 Query 3 results	44
5.4 Query 4 results	45
5.5 Query 5 results	46
6.1 IntervalIntersection example	53
6.2 IntervalFrag for Q1 and Q2	56
6.3 IntervalFrag for Q3 and Q4	57
6.4 IntervalFrag for Q5	57
6.5 Comparison: Time to Cube	58
6.6 Comparison: Baseline memory	59
A.1 Set Intersection Algorithm results	79

LIST OF TABLES

	<u>Page</u>
3.1 Operations Data	21
3.2 Satellite Operators and Big Data Architectures	24
5.1 Telemetries overview	37
5.2 Queries overview	39
5.3 Cube representations used in the experiment	41
6.1 IntervalFrag x Frag-Cubing, memory consumption in KiB	55
6.2 IntervalFrag x Frag-Cubing, query response times in ms	55
7.1 Preferred algorithm to use	61
8.1 Resulting published work	64
A.1 Set Intersection Results, in milliseconds	78

LIST OF ABBREVIATIONS

DW	– Data Warehouse
OLAP	– On-Line Analytical Processing
OLPT	– On-Line Transaction Processing
NoSQL	– Not Only SQL
TAD	– Abstract Data Type
ROLAP	– Relational OLAP
MOLAP	– Multidimensional OLAP
HOLAP	– Hybrid OLAP
DBMS	– Data Base Management System
TLE	– Two Line Element
TID	– Tuple Identifier
CSV	– Comma-separated Value
INPE	– National Institute for Space Research
CCS	– Satellite Control Center
SCD	– Data Collection Satellite
CBERS	– China-Brazil Earth Resources Satellite
AMZ	– Amazonia Satellite
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 File System
CSMT	– China Satellite Marine Track & Control Department
SISSET	– Shandong Institute of Space Electronic Technology
CDM	– Conjunction Data Message

CONTENTS

	<u>Page</u>
1 INTRODUCTION	1
1.1 Research Objectives	4
1.2 Method	5
1.3 Contributions	5
1.4 Document Structure	5
2 THEORETICAL BACKGROUND	7
2.1 Satellite Operations	7
2.2 Big Data	7
2.3 Data Warehouse	8
2.4 OLAP	9
2.5 Data Cube	10
2.5.1 Data Cube Cells	11
2.5.2 Dimensional Modelling	12
2.5.3 Concept Hierarchies	13
2.5.4 Measures	14
2.5.5 OLAP Operations	15
2.5.6 Data Cube Computation	15
3 RELATED WORKS	21
3.1 Operations Data	21
3.1.1 Data Flow	21
3.2 Data Analysis by Satellite Operators	23
3.2.1 Data Analysis at INPE	23
3.3 Data Cube Computation	25
3.3.1 <i>Frag-Cubing</i>	25
3.3.2 Other Algorithms	27
4 METHOD	29
4.1 Objectives	29
4.2 Case Study: SCD2	29
4.3 Proposed Architecture	30
4.3.1 Proposed changes	31

5	QUERY PARTITION	33
5.1	Algorithm	33
5.1.1	Aggregation Generator	33
5.1.2	Relationship Strength Calculation	34
5.2	Queries	36
5.2.1	Q1	36
5.2.2	Q2	37
5.2.3	Q3	38
5.2.4	Q4	38
5.2.5	Q5	38
5.2.6	Summary	39
5.3	Experimental Validation	39
5.3.1	Dataset and Method	39
5.3.2	Results	41
5.3.2.1	Q1	42
5.3.2.2	Q2	42
5.3.2.3	Q3	43
5.3.2.4	Q4	44
5.3.2.5	Q5	45
5.4	Summary and Analysis	46
6	INTERVALFRAG	49
6.1	Using Intervals in Inverted Indexes	49
6.2	Algorithm	50
6.2.1	IntervalInsertion	51
6.2.2	IntervalIntersection	51
6.3	Results	54
6.4	Summary	59
7	ANALYSIS AND DISCUSSION	61
8	CONCLUSIONS	63
8.1	Main contributions	63
8.2	Future work	64
8.3	Final thoughts	65

REFERENCES	67
APPENDIX A - INTERSECTION ALGORITHMS	75
A.1 Problem	75
A.2 Algorithms	75
A.3 Experiments	78

1 INTRODUCTION

The Satellite Control Center (CCS) at the National Institute for Space Research (INPE) currently operates the following satellites: the Data Collection Satellite (SCD) family, comprised of the satellites SCD1 and SCD2; and the China-Brazil Earth Resources Satellite (CBERS) family, currently operating the fifth and the sixth satellites in the family, CBERS-4 and CBERS-4A. Each satellite pass by INPE's ground stations, a period called **overpass in which CCS receives satellite health via telemetry data, and chooses to send commands via telecommands to control the satellite.** This entails maintenance and operation capabilities, as the orbit and equipment health need to be measured and adjusted for each satellite, with decisions being made by engineers (AZEVEDO; AMBRÓSIO, 2010).

Telemetry data are composed of on-board sensors and equipment health measures gathered by the On-Board Computer (OBC), like battery current, system voltage, whether a given equipment is active or not, and any other data that is necessary for the satellite operators to execute the operation procedures (LARSON; WERTZ, 1999).

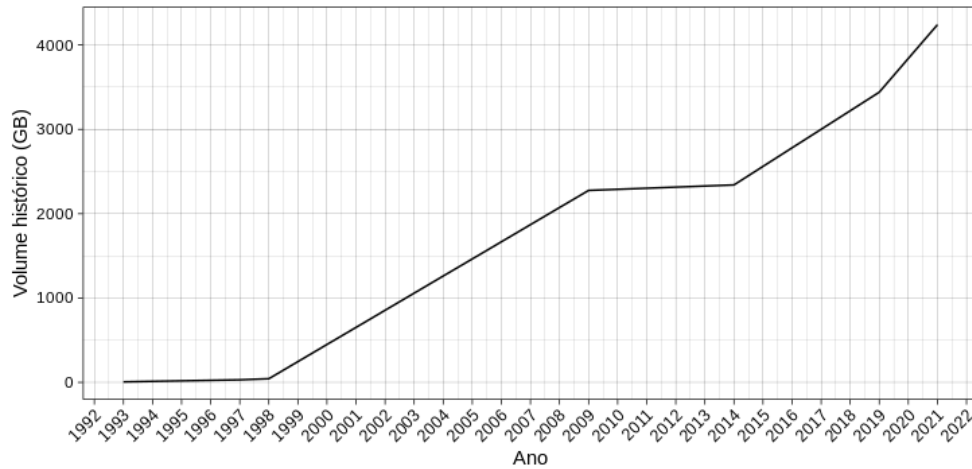
These data need to be stored for the entire life cycle of the satellite, as long-lived satellites that work for decades generate a considerable amount of data, which must be analyzed by the operators. For the SCD family, SCD1 has been operational for 25 years, even if in an degraded state, and keep generating data of about 7GB per year, with an average of 8 station passes per day.

However SCD1 is only a small satellite, tracking only about **100 telemetry points,** and more complex satellites like the ones from the CBERS family will track over **4 thousand telemetries at once.** With the future launch of the Amazônia-1 satellite, INPE will be operating 5 satellites at once (CBERS-4, CBERS-4A, AMZ-1, SCD1 and SCD2), which generates pressure to the operation activities (FILHO et al., 2017).

Figure 1.1 shows a simple statistic of the expected volume of telemetry data being handled by CCS since the launch of SCD1 in 1993, and with the added volume of the CBERS and AMZ programs. This assumes a constant rate for the telemetry data, however closely tracks the average raw engineering data generation. **The majority of these data are not available for analysis, as every few years it is archived on tapes by INPE, and thus the operators in general only deal with data that goes a few months back for each satellite, and not with the full volume for each.**

It is necessary to be careful so that the data does not become “dark data”, a concept meaning data is not available for all users that could benefit from it (HEIDORN, 2008).

Figure 1.1 - Historic telemetry data generation



Estimated telemetry data volume generated by INPE since the first satellite launch.

SOURCE: Author

These data can be classified as Big Data, as they represent a considerable volume, are constantly generated, have differing formats, their analysis is of high value and there is some uncertainty as to quality of the data, as not only communication problems occurs, the natural degradation of equipments can mislead users. These characteristics are encapsulated by the five Vs of Big Data: Volume, Variety, Velocity, Value and Veracity (EMANI et al., 2015).

Even if all of these data were available for the satellite operators at once, there is still the problem of querying a database with terabytes of data, in which queries over a long timespan or over too many telemetries would not be trivial and could take hours to complete.

In order to make the operator work feasible, database systems are deployed that enable the visualization of telemetry points over time, get the data in a specific format and execute query operations on it. However, to go beyond previously defined queries and visualizations, these systems require the use of a query language, which further requires operator training and experience to know what and how to ask, and how to format the result as to make it understandable for other mission members. This is further complicated by each satellite and telemetry format being different: it is necessary to understand how each individual spacecraft behaves and how the telemetries communicate that behavior to perform analysis on the data (UHLIG et al., 2015).

An efficient solution is to create a Data Warehouse (DW) suited to the analysis of the data, tailored to answer the most relevant and frequent domain questions about the data, which is extended by implementing **On-Line Analytical Processing (OLAP)**, **that aims to enable the exploratory analysis of the data in a fast and user-centric way** (HAN et al., 2011; VISWANATHAN; SCHNEIDER, 2014). This has been proposed and implemented for the satellite operations and ground segment with good results (ADAMSKI, 2016; YVERNES, 2018).

In order to allow for fast OLAP operations on a Data Warehouse, the data cube has been created as a query operator to pre-compute and store multidimensional aggregations, enabling users to perform multidimensional analysis on-line, and have since played an essential role in creating stable solutions for Data Warehouses (GRAY et al., 1996). However, it is often necessary to materialize a part of the data cube beforehand, as this provides a way to pre-compute and store multi-dimensional aggregates and operations, enabling for multi-dimensional analysis to be performed on the fly (on-line).

A data cube is constituted of dimensions and measures: dimensions relate to the constituent attributes of the data, determining its context; while measures are the calculated relationships between the dimensions. Each dimension can have different values at different data points, the number of distinct values in the dimension is defined as the dimension's cardinality (denoted by C_i for a dimension i). The data cube is based on the tuples of the data, as the cube operator will generalize groups of values in each dimension as part of the concept of aggregate cells that have those values in those dimensions, thus generating the cells of the cube (HAN et al., 2011). The tuples can be of any value, and the distribution of these values within a dimension is called the skew: if values are repeated too often ($C_i \ll n$, for a dimension i), the dimension is said to have a high skew.

The cells that are related to some dimensions form a cuboid: one of the possible subsets of the combination of dimensions, with a resulting measure computed between the data, resulting in the relationship between those dimensions. These cuboids will have the size of the cardinalities of each constituent dimensions multiplied, and they can be 0-dimensional up to n -dimensional, for a number of n dimensions.

For that reason it is necessary to choose **an appropriate data cube computation algorithm**, of which several have been proposed to partially, or fully, materialize a data cube, like (DOKA et al., 2011; Dong Xin et al., 2006; LI; WANG, 2005; LI et al., 2004; XIN et al., 2007). To say that a cube is fully materialized means that all

the 1-dimensional to n -dimensional cuboids have been pre-computed, with partial materialization pre-computing some of those cuboids, and no materialization pre-computing 0 cuboids.

However, one of the main limitations that these algorithms face is keeping system memory consumption low: as it is a 2^d storage operation, with d being the number of dimensions used, it is often not possible to fully materialize the data cube, making partial materialization strategies necessary.

The work of Li et al. (2004) presents Frag-Cubing, an algorithm to materialize the minimal data cube necessary to answer queries with few dimensions, while using less memory to answer the queries that need more dimensions. Frag-Cubing uses an inverted index schema that excels at answering queries from data that has a high skew: data that focuses on values that are repeated often are compressed into fewer indices, that can then be used to more efficiently answer queries. In Frag-Cubing, each attribute value of a tuple is associated with $1 - n$ tuple identifiers (TID). Point queries with two or more attribute values are answered by intersecting tuple identifiers of these attribute values.

Though the choice of the data cube construction algorithm can reduce the memory and storage space usage of the data warehouse, it is still necessary to adapt the data into schemas to further reduce the dimensionality and improve the organization of the data.

1.1 Research Objectives

The goal of this thesis is to create a data cube base architecture to represent satellite telemetry data along a mission, using the distribution of the telemetry values to ease analysis and querying of the satellite's state by satellite engineers.

Specific objectives of this work include:

- To test two approaches of reducing the memory usage of the Frag-Cubing algorithm, with the goal of reducing implementation requirements for a data warehouse based on a data cube, using the telemetry value distribution of satellite telemetry data. It will use the information gathered by analysing the satellite telemetries to select and optimize queries, taking into account the high dimensionality, high number of tuples, high skew and high cardinality of the base data.

- To probe whether the use of inverted index compression via list intervals can improve the memory consumption and query response times for the Frag-Cubing algorithm. With the results being evaluated against the Frag-Cubing original implementation, it will be possible to know when to use each of the alternatives and decide for which dimensions and/or kinds of data they are applicable.

1.2 Method

In order to evaluate whether the approaches are useful for a satellite operator, first an algorithm will be implemented to find **relationships between telemetries** and then this will be evaluated **with an experienced satellite operator as to how useful it is**. The queries selected will be filtered, and an approach of pre-processing the input data will be executed, where the data are filtered to only the dimensions that will be queried and then evaluated as to whether that improves query response times and memory consumption for each type of query.

This will be experimentally evaluated on data from one of INPE's satellite, notably SCD2, for which the Satellite Control Center (CCS) has allowed the use of over 4 years of satellite telemetry data, totalling over *24GB* of raw data and associated documentation, **with 135 telemetries being tracked**.

1.3 Contributions

This work aims to use open source software to define an improved data cube algorithm, and thus it is expected to save money and time for space organizations that operate satellites and need to implement their own telemetry data analysis structures to analyse data. This work is being performed using open source software, open literature and aims to publish the satellite telemetry data used as a case study for others in the end. Furthermore, all analyses and experiments are designed to be reproducible via open source software repository.

For the Frag-Cubing-derived algorithms, this also shows that simpler strategies can **be used to improve memory consumption and query response times**, first by simple pre-processing of frequent queries and then by changing the inverted index compression strategy of the algorithm to drastically reduce memory usage.

1.4 Document Structure

The remainder of this document is structured as follows:

- Chapter 2: Presents the theoretical background on satellite operations, Data Warehouse, Big Data and Data Cube approaches necessary for the understanding of this work;
- Chapter 3: Related works in the literature will be presented and reviewed, as well as data cube concepts close to this one, and how other satellite operators are solving these problems with what technologies;
- Chapter 4: Presents the method, the case study experimental setup with the SCD2 satellite and showcases the sample data warehouse-based architectural vision for this work and operation activities;
- Chapter 5: Presents the Query Partitioning algorithm, and the results of trying to filter the data by the query related dimensions;
- Chapter 6: Presents the IntervalFrag algorithm and the experimental validation, with an analysis of replacing Frag-Cubing's inverted index architecture;
- Chapter 7: Presents a critical analysis of the results, where each algorithm is better suited and where they excelled or had drawbacks;
- Chapter 8: Concludes the thesis, summarizing the usefulness of the results and future work that can be discussed from them.

2 THEORETICAL BACKGROUND

This chapter presents the theoretical background necessary to understand the concepts in this thesis, starting with satellite operations, the definitions of Big Data, Data Warehouse, OLAP and Data Cubes.

2.1 Satellite Operations

A satellite is divided into two modules: the service module and the payload. The service module is composed of everything necessary for the operation of the on-board equipment, such as the power supply system, the ground communication system, the on-board computer, etc. The payload is composed of all the necessary equipment to fulfill the mission objectives, these being sensors, cameras, telescopes, etc (LARSON; WERTZ, 1999).

A satellite generates two different types of data: payload data and telemetry data. Payload data is the data generated to fulfill the mission of the satellite, and it can be photos taken for remote sensing, photos taken by telescopes, communication data if this is the focus of the mission among others (LARSON; WERTZ, 1999). The telemetry data are the monitoring data of the health status and good operation of the satellite systems. These data are collected by the satellite's on-board computer, and are sent to the ground stations via telecommunication systems.

The telemetry data usually consist of sensor measurements on the satellite equipment, information collected by the on-board computer (such as whether an instrument is turned on or not), and other data whose collection has been defined as relevant to the operation of the satellite. Depending on the mission, other measurements can be classified as telemetry, e.g. satellite-facing cameras, radars for the detection of possible collisions, etc (KRAG et al., 2017).

This data must be analyzed by satellite operators, who are responsible for monitoring and operating the satellite, on the ground after reception at the control center. This analysis aims to ensure that the satellite is performing the tasks as it should, and that its state of health allows the continuation of the mission.

2.2 Big Data

The concept of Big Data is still evolving, and in this work the 5 Vs definition will be used: Volume, Variety, Velocity, Value and Veracity (EMANI et al., 2015).

- **Volume:** This term generally specifies an amount of data in which a traditional database management system is ineffective. It is important to note that this is not only about the storage of data, but also about its processing (BOUSSOUF et al., 2018). Using a large volume of data usually implies in better models, which are then hoped to produce better analyses, justifying the collection of a large amount of data.
- **Variety:** The data has multiple sources, with different formats, without a standardized modeling scheme, such as data coming from computertexts, sensor data, multimedia data, etc. As a consequence, these data should be used as transparently as possible in the analysis.
- **Velocity:** data is made available very quickly, and should be analyzed as quickly as possible. This implies that the data might be stored and analyzed even in real time.
- **Value:** data must be stored to create some value for its users, be it economic, scientific, social, organizational, etc.
- **Veracity:** the data has no guarantees as to its quality, such as inconsistencies and lack of accuracy, but the analysis must be of high quality anyway.

These V's are related to the construction of a Data Warehouse, and can also be seen as requirements for the creation of one for a data set characterized by Big Data (ZHANG et al., 2017). In particular, there is a certain relationship with the idea of “NoSQL” (“Not only SQL”), where not only relational database systems are used, but also other paradigms are used, such as document oriented, key and value, etc (BIMONTE, 2016).

2.3 Data Warehouse

A Data Warehouse (DW) is a subject oriented, integrated, time-varying or time partitioned, non-volatile data repository that assists the decision making process (INMON; HACKATHORN, 1994). This definition can be divided into:

- **Subject oriented:** the DW is used for the analysis of a specific area. For example, a DW for telemetry data might just be useful for the operators, but not specific enough for engineering.
- **Integrated:** the DW must integrate data from multiple sources in a structured way. For example, even if there are two different representations for the same product, the DW should have only one representation. This re-

quires the use of data cleaning and integration techniques in order to ensure data consistency.

- **Time-varying or time partitioned:** the DW must explicitly or implicitly contain the time perspective. This means that DW has historical data and they can be consulted during the analysis. For example, one might want to know data from days, months or years ago.
- **Non-volatile:** once inside DW, the data is not removed or updated, being a requirement for consulting historical data.

These features differentiate the Data Warehouse from other repository systems such as database systems, transaction processing systems and file systems (HAN et al., 2011).

A DW is generally represented by a dimensional model that allows efficiency in data organization and management information retrieval (KIMBALL; ROSS, 2013). Furthermore, this model has a few definitions: facts, dimensions and measures. A fact corresponds to the business subject to be analyzed, each dimension is a visualization perspective of the business subject and measures are numerical values that quantify the business subject. One of the dimensions is always temporal to allow the analysis of the subject over time.

2.4 OLAP

On-line Analytical Processing (OLAP) is a term that refers to a set of tools that are used to summarize, consolidate, visualize, apply formulations and synthesize data according to multiple dimensions (CODD et al., 1998).

An OLAP system allows the response of multi-dimensional queries using data stored in the Data Warehouse (KIMBALL; ROSS, 2013), and the main features are (BIMONTE, 2016):

- **Online queries:** must be made *Online*, that is, in time for the user.
- **Multidimensional queries:** They are defined using the dimensions and measures provided by the Data Warehouse, which expect high quality data.
- **Simple representation:** Query results must be represented using tables and graphs, because end users are usually decision makers who need visualizations that are relevant to the subject.
- **Exploratory:** queries are used in an exploratory way, since users generally do not know in advance all the data available for queries.

Each OLAP tool must manipulate a new **Abstract Data Type** (ADT), called a data cube, **using specific strategies due to the way the data is stored**, being classified in (MOREIRA; LIMA, 2012):

- **Relational OLAP (ROLAP)**: use relational Database Management Systems (Data base Management System - DBMS) for the management and storage of data cubes. ROLAP tools include optimizations for each DBMS, implementation of navigation logic in aggregations, services and additional tools;
- **Multidimensional OLAP (MOLAP)**: implement multidimensional data structures to store data cubes in main memory or in external memory. No relational repositories are used to store multidimensional data and the navigation logic is already integrated into the proposed structure;
- **Hybrid OLAP (HOLAP)**: combine ROLAP and MOLAP techniques, where detailed data is stored in a relational database (ROLAP), and aggregations are stored in multidimensional data structures (MOLAP).

It is important to highlight the difference between OLAP and Online Transaction Processing (**OLPT**), since common database systems use only OLTP, which has the purpose of performing valid transactions and processing queries. This covers the vast majority of day-to-day operations, such as stock control, banking operations, etc., serving various users of an organization. **OLAP is used by decision makers and data analysts and is geared towards higher level decisions in the organization** (HAN et al., 2011).

2.5 Data Cube

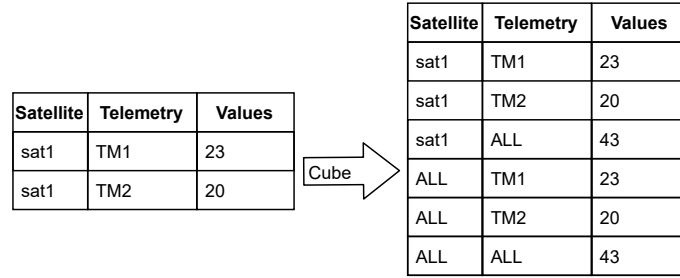
The data cube was originally created as a relational operator that generates all the possible combinations of its attributes according to a measure (GRAY et al., 1996).

The structure of the data cube allows the data to be modeled and visualized in multiple dimensions, and it is characterized by dimensions and measures. A measurement is an attribute whose values are calculated by the relationship between the dimensions, which is calculated using aggregation functions such as sum, count, average, mode, median, etc. A dimension is made by the entities that compose the data, determining the context of the subject in question. A dimension can also be divided into members, which can have a hierarchy, such as a time dimension divided in day, month and year (HAN et al., 2011).

The organization of a data cube allows the user the flexibility to visualize the data from different perspectives, since the cube operator generates combinations through the concept of the value *ALL*, where this concept represents the aggregation of all possible combinations of a set of attribute values. Operations in data cubes exist in order to materialize these different views, allowing search and interactive analysis of stored data (HAN et al., 2011).

A data cube is composed of cells and each cell has values for each dimension, including *ALL*, and values for the measurements. Figure 2.1 shows an example of a data cube. The value of a measure is computed for a given cell using lower aggregation levels to generate the values of the higher aggregation levels in the *Top-down* strategy, with the inverse order used in *Bottom-up*.

Figure 2.1 - Data Cube example



Satellite	Telemetry	Values
sat1	TM1	23
sat1	TM2	20

Cube →

Satellite	Telemetry	Values
sat1	TM1	23
sat1	TM2	20
sat1	ALL	43
ALL	TM1	23
ALL	TM2	20
ALL	ALL	43

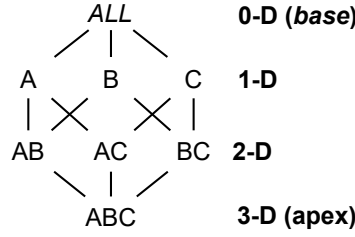
SOURCE: Author

2.5.1 Data Cube Cells

A data cube is composed of several subcubes, which are all possible levels of aggregation in the specified dimensions. Subcubes are composed of base cells and aggregated cells, and an aggregated cell is a cell that uses the special value *ALL* (“*”) to demonstrate that it is adding values in one or more dimensions. A base cell does not use the *ALL* notation, being composed of the lowest level of aggregation (LIMA, 2009). Figure 2.2 shows all levels of aggregation of a cube composed of dimensions A, B and C, from the most generic (*apex*) to the most specific (*base*).

Formally, assuming a n -dimensional data cube, a cell a of any subcube is defined by $a = (a_1, a_2, a_3, \dots, a_n, measures)$. The cell is m -dimensional (from a subcube with m dimensions), if exactly m , with $(m \leq n)$, values between $(a_1, a_2, a_3, \dots, a_n)$ are not “*”. If $m = n$, then a is a base cell, otherwise $(m < n)$, it is an aggregate cell.

Figure 2.2 - All subcubes for a three dimensional cube



A Hasse Diagram of a data cube with three dimensions A, B and C.

SOURCE: Author

A descending-ancestral relationship can exist between cells. In a n -dimensional data cube, a cell $a = (a_1, a_2, a_3, \dots, a_n, measures_a)$ with level i is an ancestor of a cell $b = (b_1, b_2, b_3, \dots, b_n, measures_b)$ of level j , and b is a descendant of a , if and only if $i < j$ and $1 \leq m \leq n$, where $a_m = b_m$ whenever $a_m \neq *$. In particular, a a cell is called the parent of a b cell, and b the child of a , if and only if $j = i + 1$ and b is a descendant of a (HAN et al., 2011).

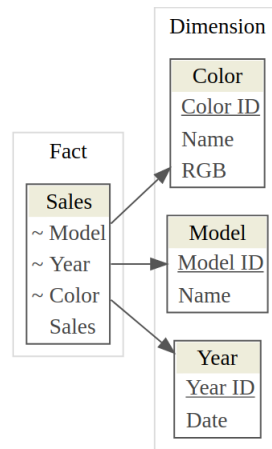
2.5.2 Dimensional Modelling

There are three main schemes for the dimensional modeling of a data cube: Star Schema, Snowflake Schema and Fact Constellation. The star schema is the most used, as it contains a central table called the fact table, where most of the data is located, with a smaller set of tables, called dimension tables, for the other dimensions. Figure 2.3 shows an example of a star scheme.

The snowflake scheme is a variation of the star scheme, where some dimensions are normalized, dividing the data of the dimension tables into other tables. This has the advantages of eliminating redundancies in the dimension tables, but it creates problems during the execution of queries since it is necessary to perform join operations with the new tables. Figure 2.4 shows an example of a snowflake scheme.

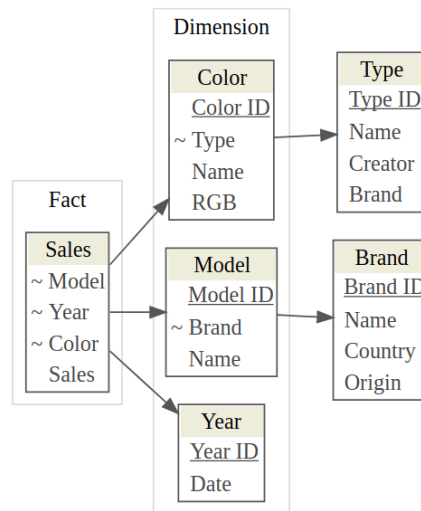
The Fact Constellation uses multiple fact tables, like multiple star schemas but sharing dimension tables, leading to its name as a group of stars. Figure 2.5 shows an example of a fact constellation scheme.

Figure 2.3 - Star schema



SOURCE: Author

Figure 2.4 - Snowflake schema

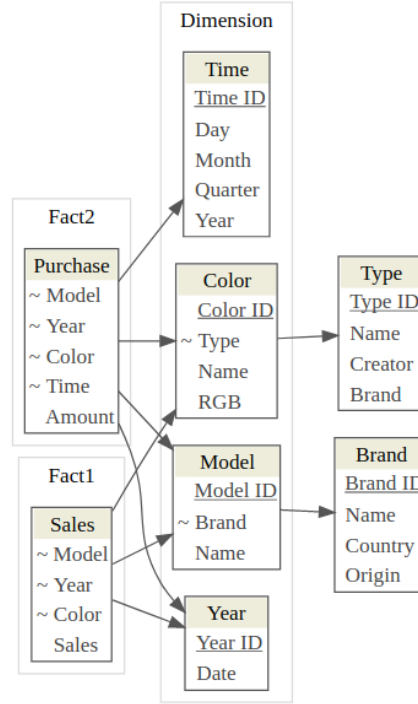


SOURCE: Author

2.5.3 Concept Hierarchies

A concept hierarchy is used to define a mapping sequence between a set of low-level concepts and a set of high-level, more general concepts. It is a style of grouping and discretization, because it groups the values in order to reduce the cardinality of a dimension (HAN et al., 2011). They help make the analysis easier to understand, as

Figure 2.5 - Fact Constellation scheme



SOURCE: Author

the operations translate the low-level data into a representation that is easier for the end user, thus facilitating the execution of queries and their subsequent use.

2.5.4 Measures

Each cell of a cube is defined as a pair $\langle (d_1, d_2, \dots, d_n), measures \rangle$, where (d_1, d_2, \dots, d_n) represent the possible combinations of attribute values over dimensions. A measure is calculated for a certain cell by aggregating the data corresponding to the combination of dimensions and values (HAN et al., 2011). Measurements can be classified into three types: distributive, algebraic and holistic.

A distributive measure is a measure whose calculation can be partitioned and then combined, and the result would be the same if the calculation was performed on the entire data set. For example, the sum function is distributive: dividing the N data into n sets, and making the sum of each n set, will have the same result as if it were done directly over N .

An algebraic measure is a measure that can be calculated with two or more distribu-

tive measures. For example, a measure of average can be calculated by dividing the measure sum by the measure count, which are both distributive.

A measure is holistic if there is no algebraic measure with M arguments that performs the computation. This means that computing cannot be partitioned, with exact values obtained only if the measure is applied to all data. Some examples are the measures of mode, standard deviation and median (HAN et al., 2011).

2.5.5 OLAP Operations

To perform queries in the Data Warehouse, it is necessary to use some operations on the data cube to obtain the appropriate results. These queries must also be able to go through the hierarchy of concepts of each dimension, as well as follow the dimensional model of the cube, in order to offer a user-friendly interface for interactive analysis (HAN et al., 2011). Some operations are exemplified in Figure 2.6, but they usually are:

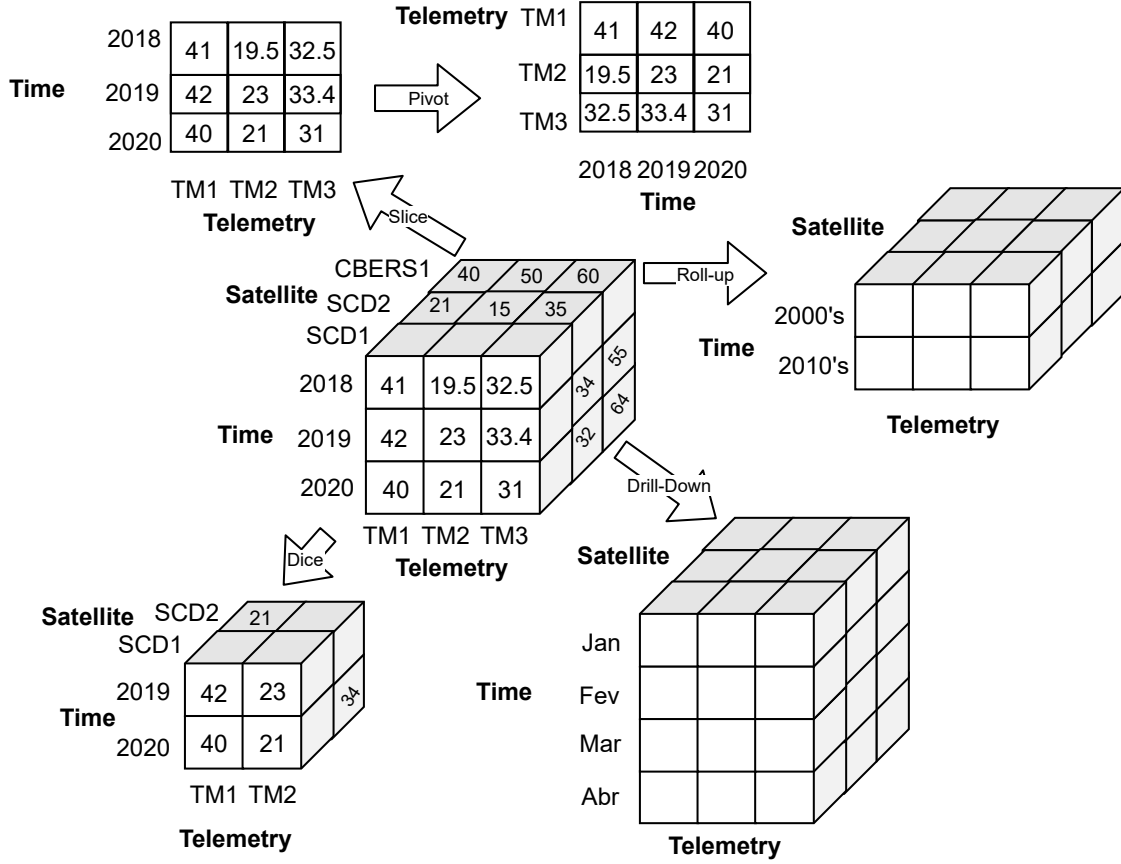
- *Roll-up*: performs aggregation in the data cube, either by navigating the concept hierarchy from a specific level concepts to a more generic one, or by reducing a dimension.
- *Drill-down*: the inverse of the roll-up operation, navigates in the hierarchy of concepts from the most generic level to the most specific level, or adds dimensions to the current cube.
- *Slice*: performs a selection in a cube dimension, resulting in a subcube focused on that dimension.
- *Dice*: defines a subcube by making a selection (slice) in two or more dimensions.
- *Pivot*: also called rotation, it allows changing the position of the dimensions in view, changing rows to columns and vice versa.

Depending on the OLAP system, it is possible to execute other operations, like drill-across between multiple fact tables, and drill-through where queries are executed directly on the low-level representation of the data (HAN et al., 2011).

2.5.6 Data Cube Computation

Data cube computation is an essential task, as pre-computing all or part of a data cube can significantly increase DW performance. However, this task has exponential complexity in relation to the number of dimensions, being called materialization,

Figure 2.6 - OLAP operations in a Data Cube



Common OLAP operations on a three-dimensional data cube with Satellite, Time and Telemetry dimensions

SOURCE: Author

with complete materialization requiring a large number of cells, and therefore a high consumption of memory and time (HAN et al., 2011).

The original calculation of the data cube computation was proposed by Gray et al. (1996), being: given an input relation R with tuples of size n , the number of subcubes that can be generated is 2^n , where n is the number of dimensions of the cube. For example, assuming a cube with three dimensions *Satellite*, *Telemetry*, *Value*, we have $2^3 = 8$ possible subcubes: $\{(satellite, telemetry, value), (satellite, value), (satellite, telemetry), (telemetry, value), (telemetry), (value), (satellite), \emptyset\}$, with \emptyset denoting the empty grouping (base cell, dimensions are not grouped).

However, in practice, dimensions can have associated concept hierarchies, as for the time dimension: “day<month<quarter<semester<year”. For a cube with n dimensions with multiple concept hierarchies, the total number of subcubes is shown in Equation 2.1.

$$subcubes = \prod_{i=1}^n (L_i + 1) \quad (2.1)$$

Where L_i is the number of concept levels of the i dimension. It is necessary to add one to the equation 2.1 to denote the virtual level *ALL*. The size of each subcube also depends on the cardinality of each dimension, that is, the number of distinct values. While the number of dimensions, concept hierarchies, and cardinality of the cube increases, they also increase the space requirements exponentially, being known as the **Curse of Dimensionality** in computing.

To be able to answer the queries properly, it is necessary to choose a method for subcube computing: non-materialisation, complete materialisation and partial materialisation.

In non-materialisation, the aggregated subcubes are not pre-computed, so the aggregations are computed in query time, which can be extremely slow, but has the lowest memory consumption.

Complete materialization computes all possible aggregations of the cube, generating a complete data cube. This method generates the best response times, because the aggregations have already been computed, but it requires a large amount of memory.

Partial materialization only computes a selected subset of subcubes, and there are several different techniques for selecting the subcubes to be computed. One of them is to compute all the subcubes that contain only cells that satisfy a given criterion, specified by the user. These cubes are called iceberg (BEYER; RAMAKRISHNAN, 1999).

Another technique is to compute small cubes, usually between 3 and 5 dimensions, to form complete cubes. To answer queries with more dimensions, the combinations between the small subcubes are aggregated. This technique is called shell fragment, and the cube is called cube shell (LI et al., 2004).

A data cube where cells with identical measurements are encapsulated in a single

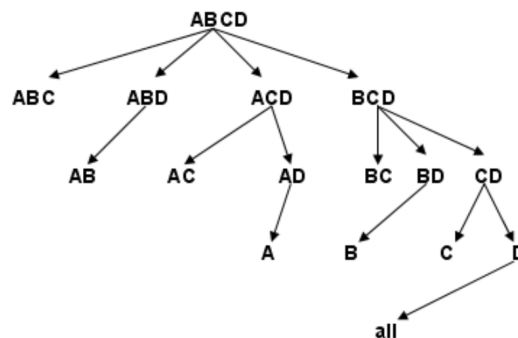
abstraction, called a closed cell is called a closed cube (Dong Xin et al., 2006), or quotient cube (LAKSHMANAN et al., 2002).

The choice of partial materialization depends on the required balance between response time and storage space. However, full cube computation remains relevant, and advances in partial cube computation are generally adopted in full cube computation. There is also the problem of updating the cube, as each update can cause a partial or complete recomputation of the cube to maintain the correct measurements.

From a base cube, the data cube computation can use the Top-down or Bottom-up strategy for the generation of the remaining subcubes (HAN et al., 2011).

Figure 2.7 shows the generation of a four-dimensional data cube by the Top-down strategy. ABCD being a base cube, the three dimension subcubes are: ABC, ABD, ACD and BCD; which can use the results of the base cube to be computed.

Figure 2.7 - Computing the data cube with the Top-Down strategy



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.

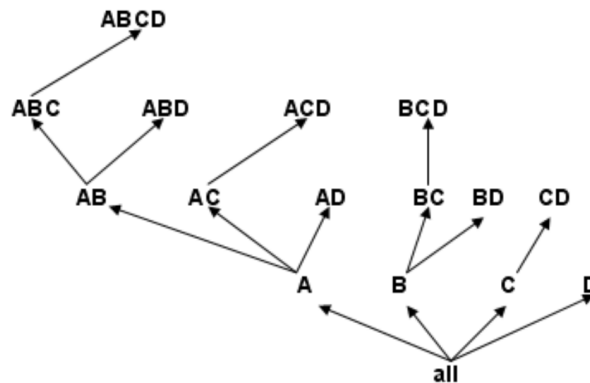
A Figura 2.8 mostra a geração de um cubo de dados de 4 dimensões por meio da estratégia *Bottom-up*. Subcubos de poucas dimensões tornam-se pais de subcubos

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.

The results of the ACD subcube computation can be used to compute AD, which consequently can be used to compute A. This shared computing allows the Top-down strategy to compute aggregations in multiple dimensions. The intermediate aggregated values can be reused for the computation of successive descending subcubes.

Figure 2.8 shows the generation of a 4-dimensional data cube through the Bottom-up strategy. Low dimensional subcubes become parents of more dimensioned subcubes. Unfortunately, the shared computing used in the Top-down strategy cannot be applied when using the Bottom-up strategy, so each descending subcube needs to be computed from the beginning.

Figure 2.8 - Computing the data cube with the Bottom-Up strategy



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 (<i>Logs</i>)
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. \(2017\)](#)

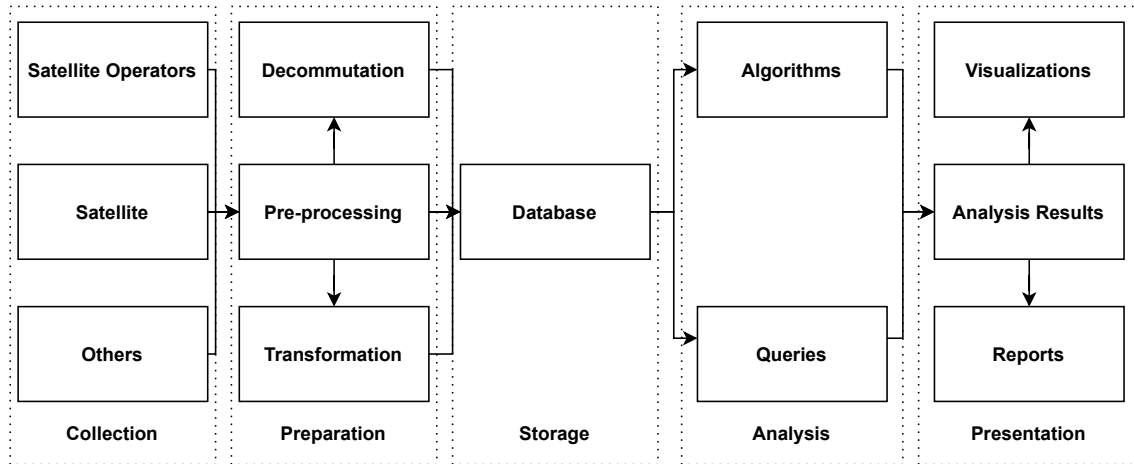
For this work, only the data from the on-board sensors will be considered, as the other data in this table can be considered for a larger data management effort by the operator organization, however they are out of scope.

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 phases 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:

- **Collection:** where the data are collected at their source (satellite, ground

Figure 3.1 - Data flow in a Big Data architecture



SOURCE: Adapted from [Zhang et al. \(2017\)](#)

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 performed. 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 performs 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 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

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 Message 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)	SISSET (China)	-	Hadoop, HDFS, PostgreSQL, MongoDB, Logstash, Kibana, ElasticSearch, Kafka, MapReduce

SOURCE: Author.

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 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 (LI et al., 2004). *Frag-Cubing* is one of the most efficient and popular data cube computation algorithms, and its internal workings will be explored in the next section.

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 $t_4 = (tid_4, a_1, b_4, c_1, m_4)$. 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 $iTa_1 \cap iTb_4 = (a_1 b_4, tid_3, tid_4, COUNT(m_3, m_4))$, in which $q, iTa_1 \cap iTb_4$ indicates the common TIDs between iTa_1 and iTb_4 , their set inter-

section.

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

Value	TID list	Measures
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.

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.

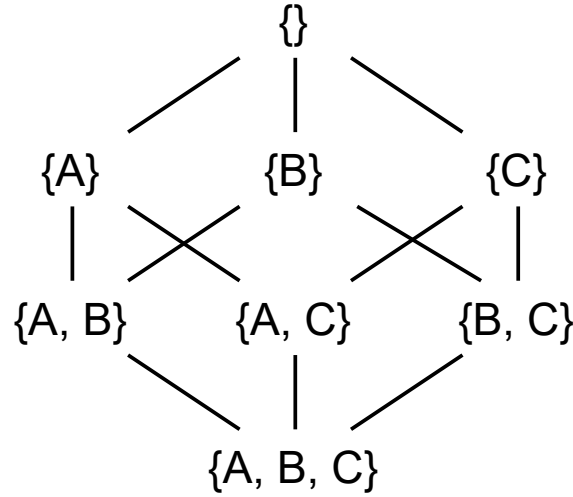
Furthermore, a skew close to zero can indicate a relation with uniformly distributed values, and the higher this value the less uniform the relation list will be.

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.

3.3.2 Other Algorithms

There are of course, more algorithms than Frag-Cubing, and they need some review of how they are applicable. 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 and 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) is presented Closed Frag-Shells Cubing, which combines the C-Cubing approach by 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 by 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* by 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) also by 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).

4 METHOD

In this section, the objectives will be reviewed, the case study data will be presented and also the proposed architecture.

4.1 Objectives

Recalling from [section 1.1](#), the main objective of this thesis is to "create a data cube base architecture to represent satellite telemetry data along a mission, using the distribution of the telemetry values to ease analysis and querying of the satellite's state by satellite engineers". In order to achieve that, and as specific goals, two different architectures will be proposed and implemented: one tries to reduce main memory consumption by pre-processing the data to be queried into high-dimensional and low-dimensional, and then filters the resulting data cubes by the dimensions related to each pre-defined query; the other uses an inverted index compression technique of changing the TID lists to hold interval lists instead of raw number lists, and tries to see how that affects query response time and memory consumption.

4.2 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 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. This work has access to data taken between 2014 and 2018, and amounts to about 23GB of CSV files. These data were obtained by exporting the telemetry database to CSV format via the SatCS program developed at INPE (AZEVEDO et al., 2015), taking over three weeks to process the data.

Figure 4.1 - SCD2



Data Collection Satellite 2 (SCD2) mission patch

SOURCE: Taken from <http://www.inpe.br/noticias/galeria/>

4.3 Proposed Architecture

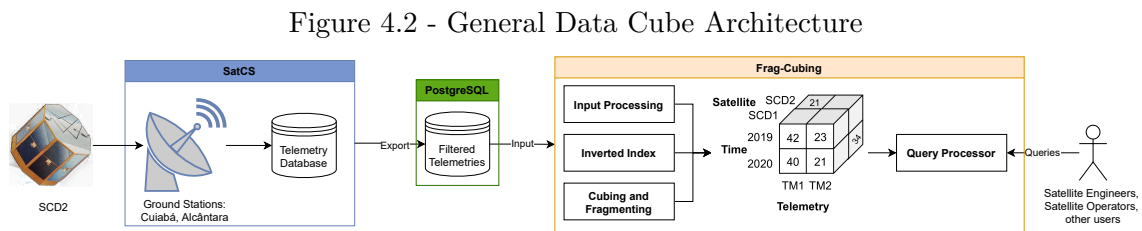
One of the using necessities of different data cube algorithms is in the number of dimensions that a certain cube can perform queries: queries with more than 15 dimensions are not generally (or practically) executed in some algorithms, as the work of Silva et al. (2016) demonstrates, however demonstrating some works that do execute queries with more dimensions than that.

The telemetry data of interest has many more dimensions than the common limit for data cube with up to 60 dimensions: with over 135 telemetries for the SCD family satellites, and thousands for larger satellites like CBERS and Amazonia, running high-dimensional queries would typically be infeasible in traditional cube construction algorithms.

It is then needed to first propose an overall architecture that takes the data from input, processes it and then spits out an analysis on the other side. This is presented as a data cube answering queries, after having been inputted sufficient data and statistical information about this data. An overview of this architecture is present in 4.2, showing how the data is based on databases The figure 4.2 shows an overview

of the proposed structure. The biggest gain of using the data cube is, plainly, to easily execute group-by operators on multiple aggregation levels and using multiple dimensions at once to compute the relationship measures between them.

In this figure, the Frag-Cubing algorithm has been arbitrarily divided into the steps of reading the input, building the inverted index, building the shell fragments in accord to the algorithm and then the cubing step that generates the data cube aggregations and is shown in the middle of the picture. The users can then query into this structure via a Query Processor built-in.



Data Cube-based architecture showing the data being generated by the satellite, being processed by SatCS, exported to a raw CSV format into PostgreSQL, which is then used as a Data Warehouse base to build a Data Cube using the Frag-Cubing algorithm. The user can then query the data cube directly.

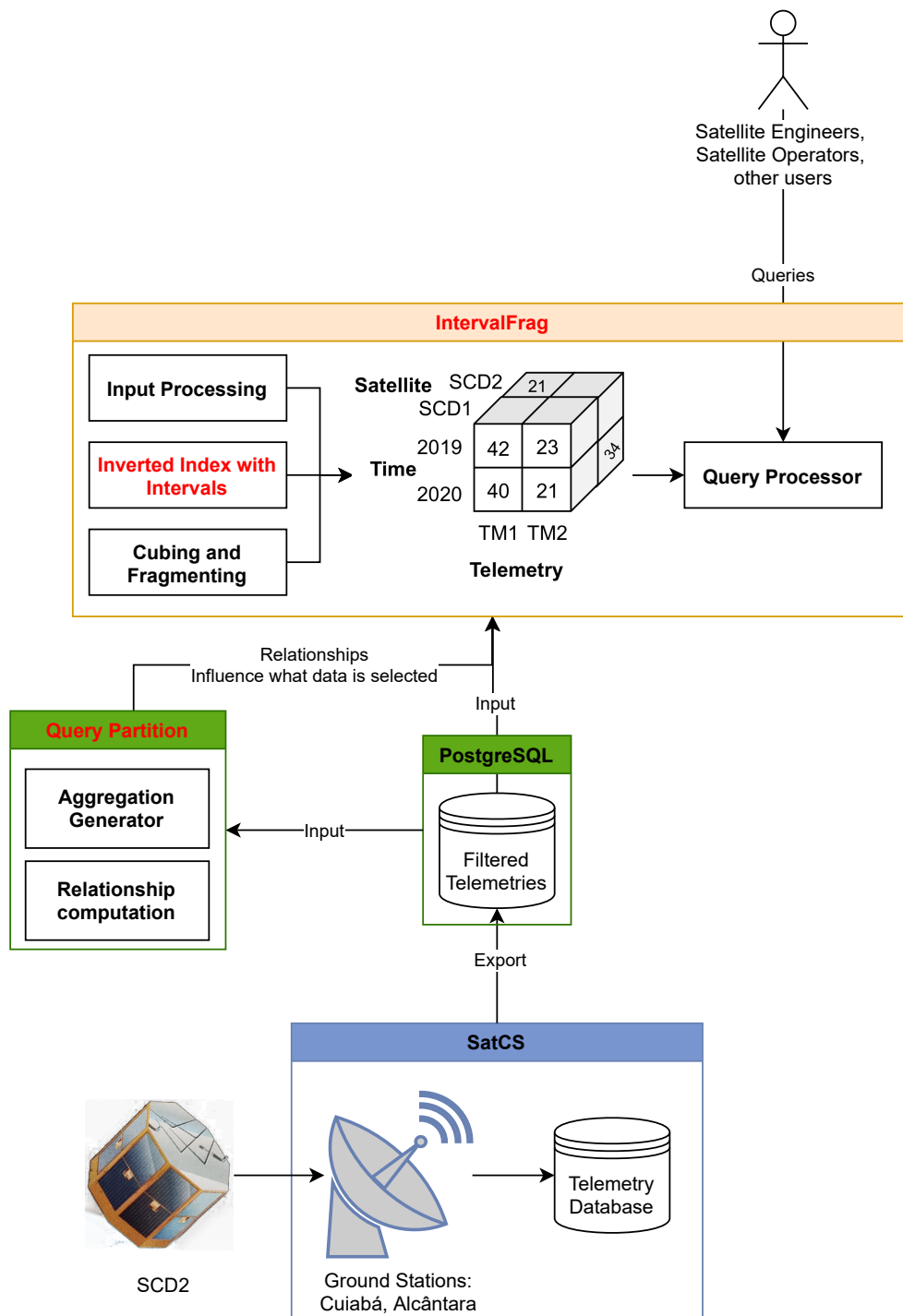
SOURCE: Author

4.3.1 Proposed changes

In order to achieve the objectives of this thesis, the following changes are then proposed, illustrated by Figure 4.3: **change the Frag-Cubing algorithm into the IntervalFrag algorithm by changing the inverted index step into an inverted index with intervals**; add the Query Partition step in which the telemetry value distribution is used to filter out relationships between the telemetries with the help of a Satellite Engineer, and have those relationships influence what data is then input into IntervalFrag. The overall objective is to show that only some changes are made into the Frag-Cubing algorithm, and the Query Partition is an added optional step. The changes have been highlighted in red lettering.

In the next chapters, the Query Partition algorithm will be presented and the results of using the approach will be shown, then the IntervalFrag algorithm will be explained, the constituent algorithms that diverge from Frag-Cubing explained and the experimental results shown.

Figure 4.3 - General Data Cube Architecture



Data Cube-based architecture showing the data being generated by the satellite, being processed by SatCS, exported to a raw CSV format into PostgreSQL, which is then used as a Data Warehouse base to build a Data Cube using the IntervalFrag algorithm. The Query Partition algorithms can then use the information obtained from the data to filter out dimensions to be input into IntervalFrag. The user can then query the data cube directly, or the relationships if necessary.

SOURCE: Author

5 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.

5.1 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 changing 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.

5.1.1 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 5.1.

$$C_k^n = \binom{n}{k} \quad (5.1)$$

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 all subcubes with those dimensions. That gives us the following number of combinations, with k_{max} being the biggest k that we want, on formula 5.2.

$$\sum_{2 \leq k \leq n}^{k_{max}} \binom{n}{k} = 2^{n-2} \quad (5.2)$$

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

5.1.2 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 5.3.

$$maxc = \prod_t C_t = C_1 * \dots * C_t \quad (5.3)$$

The biggest cardinality in the combination, to us the *minimum* possible value for the number of groups, called *minc* on equation 5.4.

$$minc = \max(C_1, \dots, C_t) \quad (5.4)$$

The proportion of the number of groups by the maximum cardinality, called *cratio* on equation 5.5.

$$cratio = \frac{numgroups}{maxc} \quad (5.5)$$

The absolute cardinality difference, as it is more representative of the discrepancy between bigger cardinalities, called *absdiff* on equation 5.6.

$$absdiff = cratio - \frac{minc}{maxc} \quad (5.6)$$

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 5.7.

$$CV = \frac{\sigma}{\mu} \quad (5.7)$$

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 tell us much about its 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.

5.2 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 5.1, 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.

5.2.1 Q1

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

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

Table 5.1 - 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

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.

5.2.2 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.

5.2.3 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.

5.2.4 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.

5.2.5 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.

5.2.6 Summary

Table 5.2 has an overview of the queries presented, with the query identification, the telemetries that are queried and the product of the cardinalities involved.

Table 5.2 - 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

5.3 Experimental Validation

In order 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.

5.3.1 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 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. In order 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. In order 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 5.3 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 5.3 - 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.

5.3.2 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.

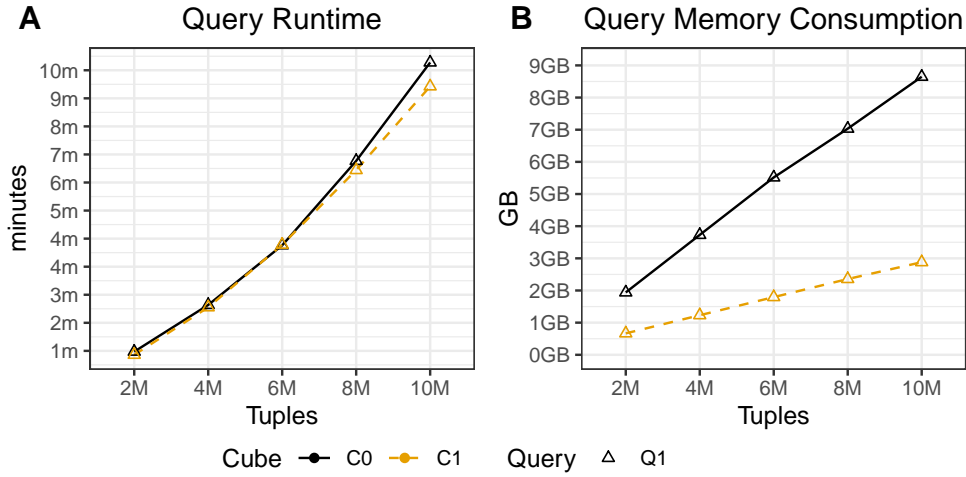
Furthermore, INPE has only a few satellite operator experts, and for this satellite that has spanned multiple years of operations, the knowledge amounted to a single available person available for questioning, which is not enough for an objective scientific inquiry. Therefore, the separation in queries 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 5.3.1. Each defined query is compared with their execution in $C0$ and the relevant cube, with their memory and response times being measured by each.

5.3.2.1 Q1

Figure 5.1-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 5.1-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 5.1 - 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 $C1$.

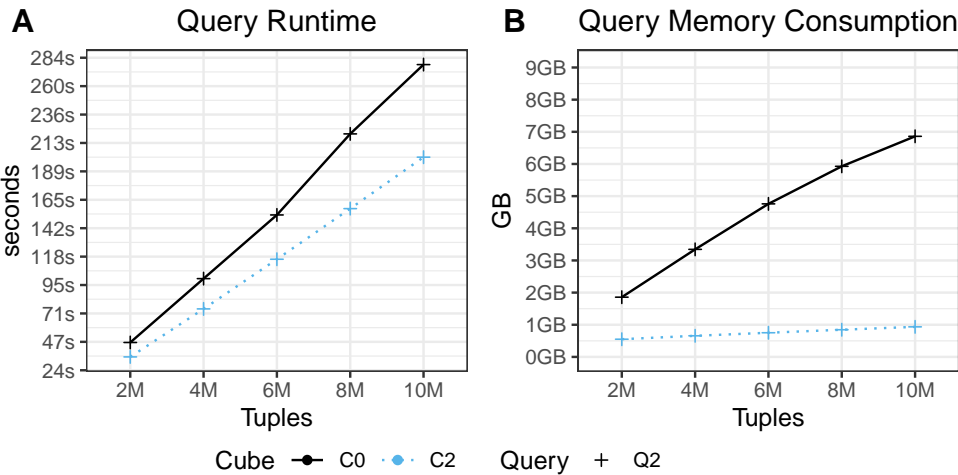
SOURCE: Author

5.3.2.2 Q2

Figure 5.2-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 5.2-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 5.2 - Query 2 results



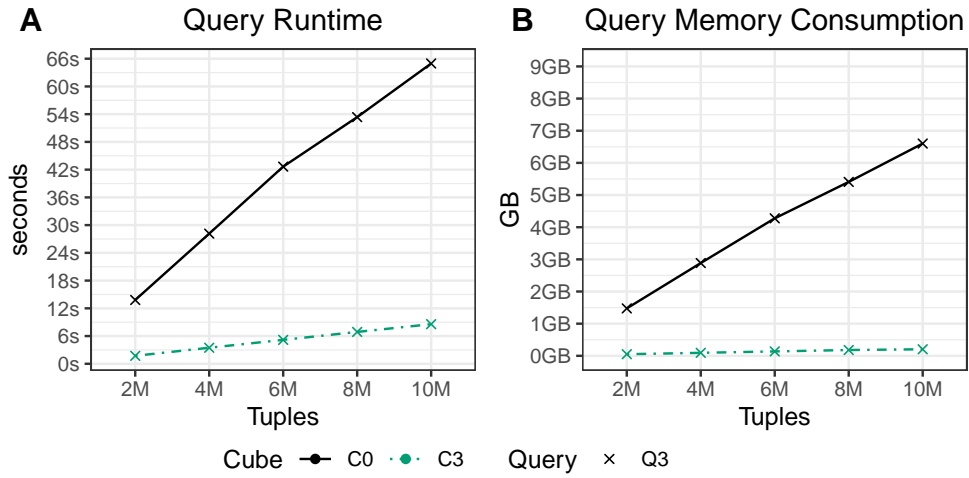
(A) Total time to answer a query, including the time to read the data from the disk, build the cube and then answer the query Q2, with cubes C0 and C2. (B) Total memory consumption to answer Q2 with cubes C0 and C2.

SOURCE: Author

5.3.2.3 Q3

Figure 5.3-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 5.3-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 5.3 - Query 3 results



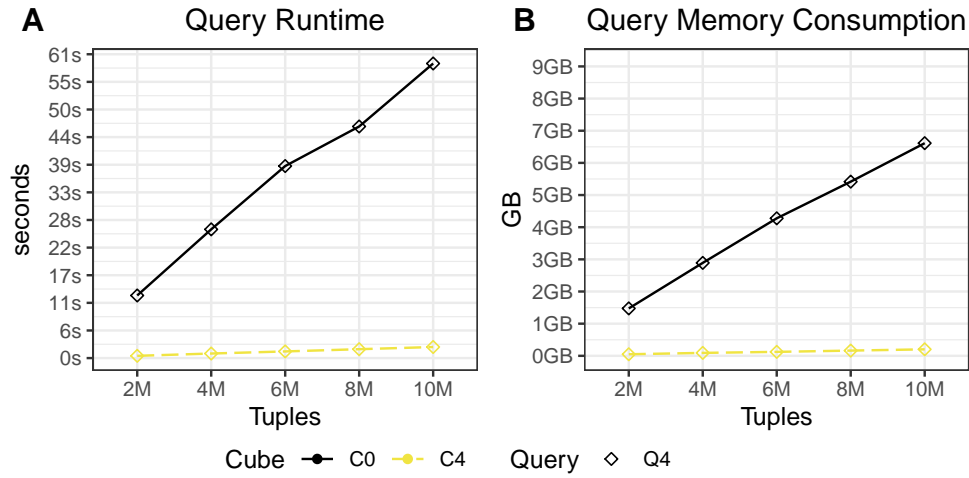
(A) Total time to answer a query, including the time to read the data from the disk, build the cube and then answer the query Q3, with cubes C0 and C3. (B) Total memory consumption to answer Q3 with cubes C0 and C3.

SOURCE: Author

5.3.2.4 Q4

Figure 5.4-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 5.4-B shows the query memory difference between C0 and C4 for query Q4, showing the same speedup pattern.

Figure 5.4 - Query 4 results



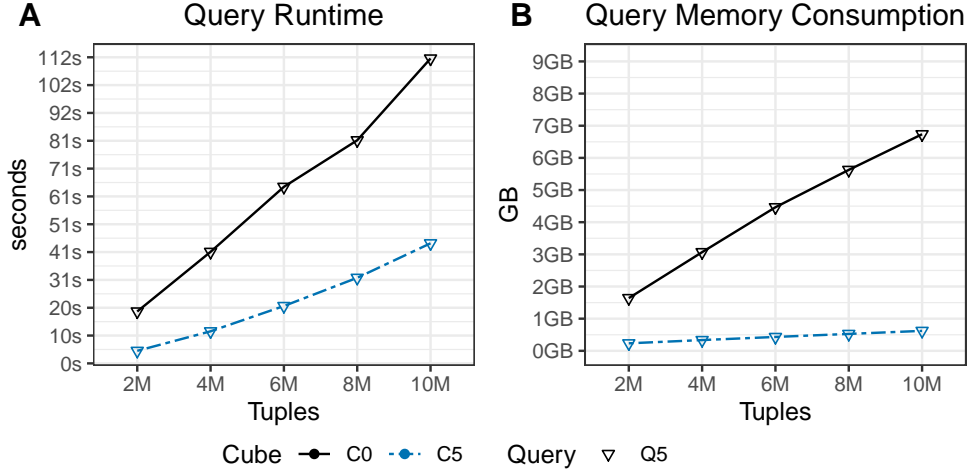
(A) Total time to answer a query, including the time to read the data from the disk, build the cube and then answer the query Q4, with cubes C0 and C4. (B) Total memory consumption to answer Q4 with cubes C0 and C4.

SOURCE: Author

5.3.2.5 Q5

Figure 5.5-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 5.5-B shows the query memory difference between C0 and C5 for query Q5, where the same figure of speedups are maintained.

Figure 5.5 - 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

5.4 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.

6 INTERVALFRAG

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

6.1 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 know 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 $\langle d_1, d_2, \dots, d_k \rangle$, the *d-gap* list would be $\langle d_1, d_2 - d_1, \dots, d_k - d_{k-1} \rangle$. 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. More options, and a more complex implementation of the above method can be found into Elias-Fano encoding and the others mentioned in Pibiri and 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 $\langle d_1, d_2, \dots, d_k \rangle$, the interval list would be $\langle [d_1, d_2], [d_3, d_4], \dots, [d_k, d_{k+1}] \rangle$, 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 number of dimensions have attribute values that are repeated on long sequences of the same telemetry data points, and Frag-Cubing generates a long list of these repetitions for some dimensions. This work 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.

6.2 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.

Furthermore, it is necessary to define the concept of sequentiality, as it quantifies how much a list can be compressed by transforming it into a list of intervals. A high sequentiality means that the data are repeated in long sequences, for example $\langle 1, 2, 3, 4, 5, 6, 7, 8, 9 \rangle$ can be compressed into the sequences $\langle [1, 9] \rangle$, being of high sequentiality. However, the list $\langle 1, 3, 5, 6, 9 \rangle$ would have a low sequentiality, as the resulting list interval would be only $\langle [1], [3], [5, 6], [9] \rangle$, and would consume exactly as much space as the raw list, with the conversion to an interval list being unfavorable.

6.2.1 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. Else none of these we can safely append b to the end of the list after flipping the signal to negative, and thus this algorithm is implemented at $\Theta(1)$, shown in [algoritmo 1](#).

6.2.2 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 so the IntervalIntersection algorithm is created.

In order 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 $I_o = I_a \cap I_b$ 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

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

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

$o_l = max(a_l, b_l)$;

$o_r = min(a_r, b_r)$;

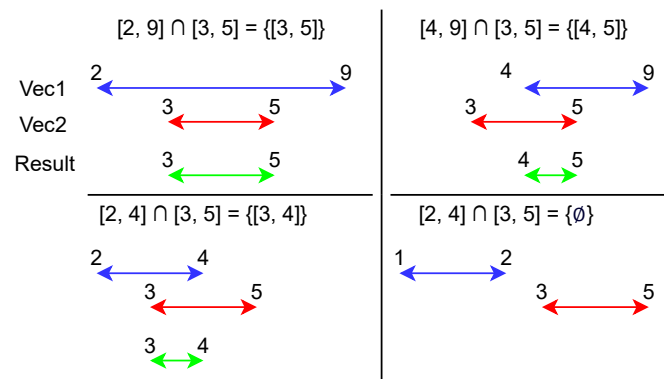
return $[o_l, o_r]$;

end

Figure 6.1 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 6.1 - 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 $\min(\text{length}(L_a), \text{length}(L_b))$;

$a_i, b_i, c_i = 0$;

while $a_i < \text{length}(L_a)$ and $b_i < \text{length}(L_b)$ **do**

$A[a_l, a_r] = \text{interval}(L_a[a_i])$; // Gets the interval at this position

$B[b_l, b_r] = \text{interval}(L_b[b_i])$;

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 ;

6.3 Results

Table 6.1 and Table 6.2 show the results of executing both algorithms to answer the queries defined in section 5.2, while using the cube structure defined as $C0$ in subsection 5.3.1, 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.

In order to make the comparisons easier to understand, let's focus on only queries $Q1$ and $Q2$, that were the highest number of dimensions and the highest number of cardinalities respectively. Figure 6.2 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.

Table 6.1 - IntervalFrag x Frag-Cubing, memory consumption in KiB

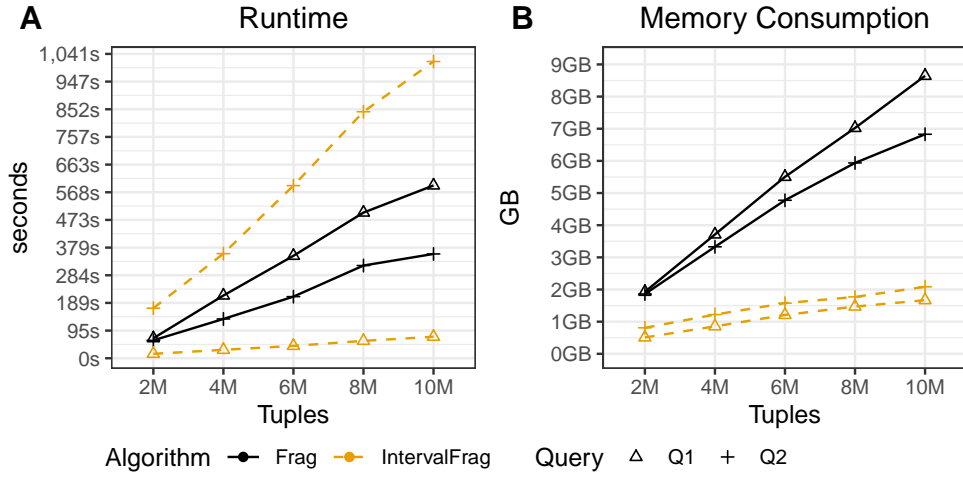
Algorithm	Query	Tuples				
		2×10^6	4×10^6	6×10^6	8×10^6	1×10^7
Frag-Cubing	Q1	1.908.708	3.674.784	5.447.864	6.953.424	8.557.348
	Q2	1.842.396	3.294.628	4.727.816	5.877.016	6.760.080
	Q3	1.448.280	2.836.592	4.236.496	5.362.128	6.502.628
	Q4	1.444.816	2.836.696	4.236.404	5.372.104	6.520.176
	Q5	1.607.456	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 6.2 - IntervalFrag x Frag-Cubing, query response times in ms

Algorithm	Query	Tuples				
		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

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 6.2 - IntervalFrag for Q1 and Q2

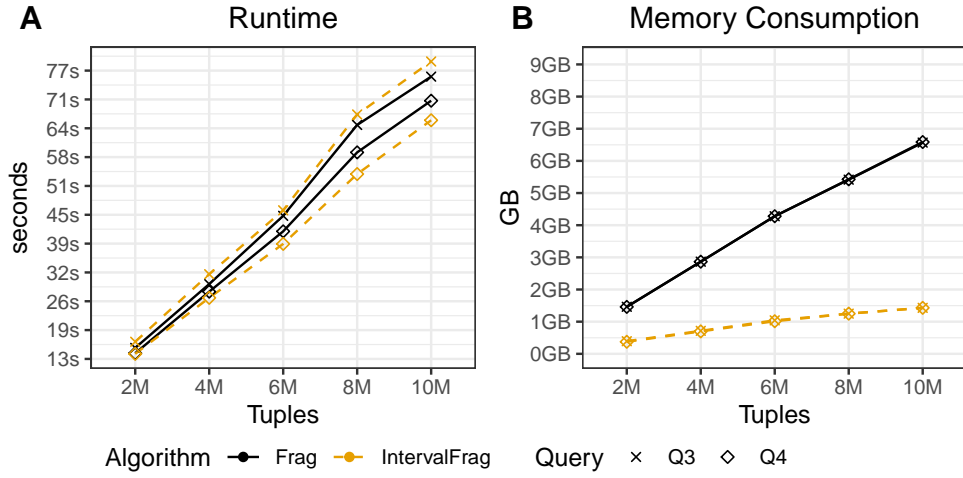


(A) Query response times of IntervalFrag and Frag-Cubing for queries $Q1$ and $Q2$ under the cube $C0$ (B) Memory consumption of IntervalFrag and Frag-Cubing for queries $Q1$ and $Q2$ under the cube $C0$.

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 6.3 shows those results for $Q3$ and $Q4$, Figure 6.4 for $Q5$, and it is also clear that the memory usage is much lower with IntervalFrag than with Frag-Cubing.

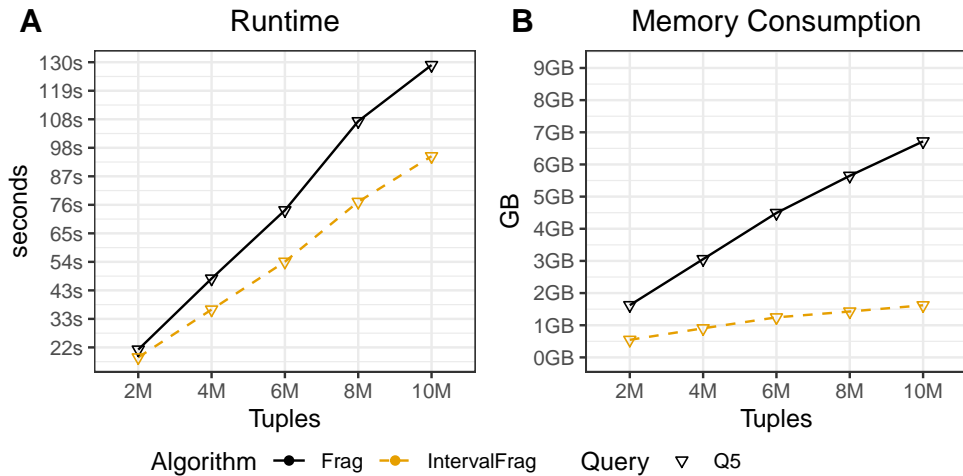
Figure 6.3 - IntervalFrag for Q3 and Q4



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

SOURCE: Author

Figure 6.4 - IntervalFrag for Q5

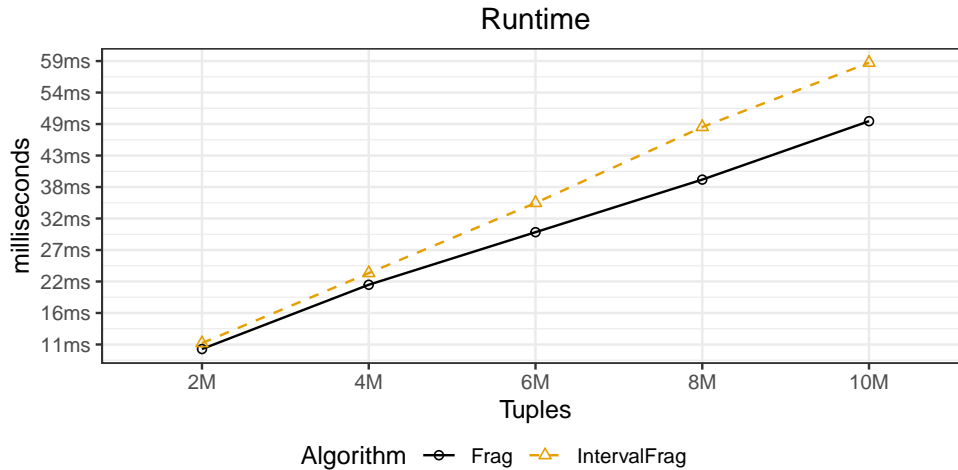


(A) Query response times of IntervalFrag and Frag-Cubing for queries $Q5$ under the cube $C0$ (B) Memory consumption of IntervalFrag and Frag-Cubing for queries $Q5$ under the cube $C0$.

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 pruning this would need to be counted against it, however as Figure 6.5 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.

Figure 6.5 - Comparison: Time to Cube

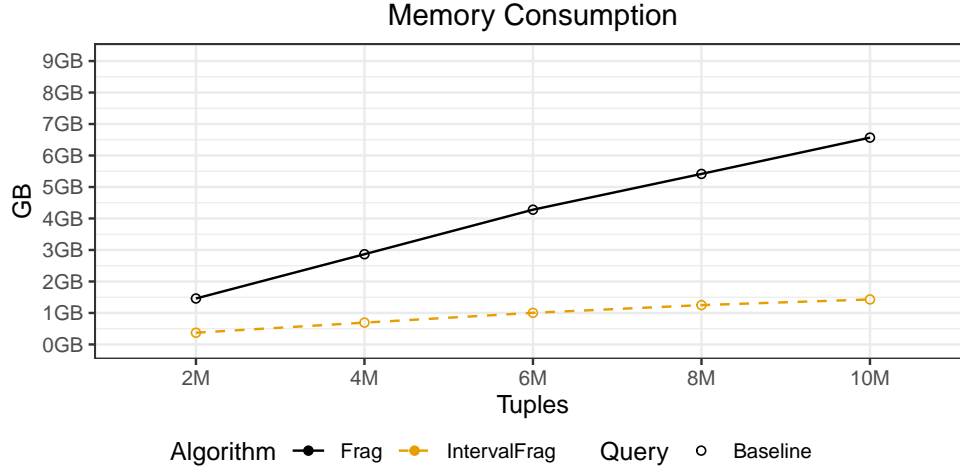


Time necessary to compute the cube after the data is read into memory.

SOURCE: Author

Another important metric is the baseline memory used before queries can be performed on the data. Figure 6.6 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 6.6 - Comparison: Baseline memory



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

SOURCE: Author

6.4 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.

7 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 6 and 5 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 7.1 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 7.1 - Preferred algorithm to use

	Low Sequentiality	High Sequentiality	High Dimensionality	High Cardinality	High Skew
Comput- ing 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

¹If 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 uncommon. 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 5 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.

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 trivially ported to Frag-Cubing. Further details on the intersection algorithms tested and their performance differences can be found on Appendix A.1.

8 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.

8.1 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 day to day activities, and this work has achieved three main results:

The goal of this thesis to create a **data cube base architecture** to represent satellite telemetry was achieved and the value distribution of the data was also used to improve upon query response times and memory consumption. As for specific contributions, they are as follows:

- 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;
- Using the heuristic to enhance Frag-Cubing’s query response times and memory **by pre-partitioning the data;**
- Improving upon Frag-Cubing’s Inverted Index memory model by saving only intervals instead of the entire values, creating IntervalFrag, and thus reducing memory and query response times for some queries;

For academia, the contribution of IntervalFrag is more important, as it showcases a way of improving the memory consumption of an inverted index with a $\Theta(1)$ insertion operation, even when keeping the same complexity as the traditional intersection. **This work also ascertained the downside of the Intervals being slower to intersect when the sequentiality is low, showing that experimentally.**

For INPE this work showcases a data cube architecture that can be implemented to help the Satellite Engineers and operators to execute their day to day analysis, and furthermore shows how can it be implemented as to reduce **memory consumption and query response times.**

For the correlated works, this improves upon Frag-Cubing’s original implementation and shows that other algorithms can be improved to by using the value distribution

of the data as a base for improvements. This change would not be too hard to implement on other correlated works that aren't based on bitmaps. For other satellite operator organizations, this can be incorporated to their Big Data architecture, in replacement for similar Data cube computation schemes.

Furthermore, [Table 8.1](#) shows the summary of the published, and currently expecting to be published articles that resulted either directly from this work or from the wider master's effort. In this table, two main conference results are shown, as well as an article that has been submitted already, and another that is currently being written, containing the results of the IntervalFrag algorithm.

Table 8.1 - Resulting published work

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

8.2 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 format for the use of the community in the near future pending liberation.

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 5.1](#) can be remade to use 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 Polaris [Librespacefoundation...](#) (2021).

The Set Intersection problem defined in [chapter 6](#) can be further optimized with recent advances not only in computer architectures, but also in relation 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 A.1](#).

8.3 Final thoughts

This work was developed entirely with open source software, and they will be made available 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 other competitions, and the publication can further enhance work in this area, like contributing to education and serving a benchmark for future implementations of projects like CubeDesign.

REFERENCES

- ADAMSKI, G. Data Analytics for Large Constellations. In: **SpaceOps 2016 Conference**. [S.l.]: American Institute of Aeronautics and Astronautics, 2016. (SpaceOps Conferences). 3, 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
- FRAMEWORK DE SISTEMA DE CONTROLE DE SATÉLITES - SATCS**. dec. 2015. BR 51 2014 001516 5. 30
- 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. 17
- BIMONTE, S. Open issues in Big Data Warehouse design. **Revue des Nouvelles Technologies de l'Information**, p. 10, 2016. 8, 9
- 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. 9
- DING, B.; KÖNIG, A. C. Fast Set Intersection in Memory. **arXiv:1103.2409 [cs]**, mar. 2011. 79
- 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. [3](#), [27](#)

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. [3](#), [18](#), [28](#)

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. [2](#), [7](#)

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](#), [10](#), [16](#)

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. [3](#), [9](#), [10](#), [11](#), [12](#), [13](#), [14](#), [15](#), [16](#), [18](#), [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. [1](#)

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. [27](#)

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

HWANG, F. K.; LIN, S. A Simple Algorithm for Merging Two Disjoint Linearly Ordered Sets. **SIAM Journal on Computing**, Society for Industrial and Applied Mathematics, v. 1, n. 1, p. 31–39, mar. 1972. ISSN 0097-5397. [75](#), [76](#)

IEEE. IEEE Standard for Floating-Point Arithmetic. **IEEE Std 754-2019 (Revision of IEEE 754-2008)**, 2019. [50](#)

ILLIMINE. **Software and Data Repository from Data Mining Research Group, Data and Information Systems (DAIS) Research Laboratory**. 2004. [39](#)

INMON, W. H.; HACKATHORN, R. D. **Using the Data Warehouse**. Somerset, NJ, USA: Wiley-QED Publishing, 1994. ISBN 978-0-471-05966-0. [8](#)

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. [75](#), [76](#), [77](#)

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](#)

KRAG, H.; SERRANO, M.; BRAUN, V.; KUCHYNKA, P.; CATANIA, M.; SIMINSKI, J.; SCHIMMERHORN, 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. 18

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, S.-E.; WANG, S. Semi-Closed Cube: An Effective Approach to Trading Off Data Cube Size and Query Response Time. **Journal of Computer Science and Technology**, v. 20, n. 3, p. 367–372, may 2005. ISSN 1860-4749. 3

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. 3, 4, 17, 25, 28, 49, 53, 75, 77

LIBRESpaceFOUNDATION / Polaris / Polaris. 2021.
<https://gitlab.com/librespacefoundation/polaris/polaris>. 65

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. 11

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. [64](#)

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. [64](#)

PIBIRI, G. E. Fast and Compact Set Intersection through Recursive Universe Partitioning. In: **IEEE Data Compression Conference**. [S.l.: s.n.], 2021. p. 10. [79](#)

PIBIRI, G. E.; VENTURINI, R. Techniques for Inverted Index Compression. **arXiv:1908.10598 [cs]**, aug. 2019. [49](#), [78](#)

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. 18, 19, 28, 53, 64

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. [28](#)

_____. Computing BIG data cubes with hybrid memory. **Journal of Convergence Information Technology**, v. 11, n. 1, p. 18, jan. 2016. [28](#), [30](#)

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. [28](#)

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. [52](#)

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. [79](#)

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](#)

UHLIG, T.; SELLMAIER, F.; SCHMIDHUBER, M. (Ed.). **Spacecraft Operations**. Wien: Springer-Verlag, 2015. ISBN 978-3-7091-1802-3. [2](#)

VISWANATHAN, G.; SCHNEIDER, M. User-centric spatial data warehousing: A survey of requirements and approaches. **International Journal of Data Mining, Modelling and Management**, v. 6, n. 4, p. 369, 2014. ISSN 1759-1163, 1759-1171. [3](#)

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. [28](#)

XIN, D.; HAN, J.; LI, X.; SHAO, Z.; WAH, B. W. Computing Iceberg Cubes by Top-Down and Bottom-Up Integration: The StarCubing Approach. **IEEE Transactions on Knowledge and Data Engineering**, v. 19, n. 1, p. 111–126, jan. 2007. ISSN 1041-4347. [3](#)

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. [28](#)

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. [79](#)

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. 28

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 and unique (non-repeating), as they represented index positions and are thus always non-zero positive integers.

A.2 Algorithms

This section details the compared algorithms, giving more information on the ones that deviate from literature further.

The UnorderedSet algorithm is an implementation of the optimum complexity algorithm: iterate over all elements of one vector and use a hashing function to check for equality, adding the elements that are equal. This is an $\mathcal{O}(n + m)$ operation, with n and m the size of both lists. For this work, the UnorderedSet functions from the C++ standard were used for brevity.

The Scalar algorithm, also called the naïve or the tape-merge algorithm, is the standard way of computing the intersection between two lists (HWANG; LIN, 1972). It is detailed in Algorithm 4, and works by creating two pointers for each list, then incrementing only one of those until a matching element is found or it reaches the end of the list.

Algorithm 5 is based on the Figure 2 algorithm by Inoue et al. (2014), and consists on a branchless version of the scalar algorithm above. Its only aim is to reduce the branch mispredictions during runtime, and removing the two separate ifs with a constant operation to decide which element to advance next.

The algorithm called "Li" is based on the original Frag-Cubing implementation code (LI et al., 2004). It is based off the scalar version, however it uses a look-ahead heuristic in which it verifies the next ten elements instead of just the next one, advancing ten spots if the present element is still smaller. It is detailed in Algorithm 6.

The BinaryLi algorithm is just the same as the Li algorithm mentioned earlier,

Algoritmo 4: Scalar intersection algorithm, adapted from [Hwang and Lin \(1972\)](#)

Result: The intersection between two lists, or \emptyset if there is no intersection

L_a and L_b two input lists;

L_c result list, with maximum size $\min(\text{length}(L_a), \text{length}(L_b))$;

$a_i, b_i, c_i = 0$;

while $a_i < \text{length}(L_a)$ and $b_i < \text{length}(L_b)$ **do**

$A_{data} = L_a[a_i]$;

$B_{data} = L_b[b_i]$;

if $A_{data} == B_{data}$ **then**

$L_c[c_i++] = A_{data}$;

a_i++ ; b_i++ ;

else

if $a_i \leq b_i$ **then**

a_i++ ;

else

b_i++ ;

end

end

end

return L_c ;

Algoritmo 5: ScalarBranchless, adapted from [Inoue et al. \(2014\)](#)

Result: The intersection between two lists, or \emptyset if there is no intersection

L_a and L_b two input lists;

L_c result list, with maximum size $\min(\text{length}(L_a), \text{length}(L_b))$;

$a_i, b_i, c_i = 0$;

while $a_i < \text{length}(L_a)$ and $b_i < \text{length}(L_b)$ **do**

$A_{data} = L_a[a_i]$;

$B_{data} = L_b[b_i]$;

if $A_{data} == B_{data}$ **then** ; // easy-to-predict branch

$L_c[c_i++] = A_{data}$;

else

$a_i = (A_{data} < B_{data})$;

$b_i = (B_{data} < A_{data})$;

end

end

return L_c ;

however using a binary-search based skip parameter instead of the fixed 10-elements

Algoritmo 6: "Li" intersection algorithm, adapted from [Li et al. \(2004\)](#)

Result: The intersection between two lists, or \emptyset if there is no intersection
 L_a and L_b two input lists. For brevity, L_a is assumed as the smaller of the two;
 L_c result list, with maximum size $\min(\text{length}(L_a), \text{length}(L_b))$;
 $LOOK = 10$; // how many elements to look-ahead
 $b_i, c_i = 0$;
for $a_i = 0$ **to** $\text{length}(L_a)$ **do**
 if $b_i + LOOK < \text{length}(L_b)$ **and** $L_b[b_i + LOOK] < L_a[a_i]$ **then**
 | $b_i += LOOK$;
 end
 while $L_b[b_i] < L_a[a_i]$ **and** $b_i < \text{length}(L_b)$ **do**
 | $b_i ++$;
 end
 if $L_b[b_i] == L_a[a_i]$ **then**
 | $L_c[c_i ++] = A_{data}$;
 | $b_i ++$;
 end
 if $b_i == \text{length}(L_b)$ **then**
 | **break**;
 end
end
return L_c ;

heuristic.

The `std::set_intersect` version is using C++'s language features, and has a very good native performance due to being vectorized. This however requires that the data uses the `vector` class and be in iterator format to work.

The SIMD implementation chosen here is based of the scalar implementation: one element is put into the first lane, and then four other integers are read on the other lane. The SIMD instruction used is less-than, and a mask is used to verify if all four elements on the second data lane are **-1**, meaning that they are lesser than the element on the first lane. If any of them are, then all four elements are iterated and then ones equal to the element in the first lane are added to the resulting vector. For brevity, this is loosely based on the algorithm used in [Inoue et al. \(2014\)](#), without the adaptive parts.

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 5.3.1: 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, these results being the median time necessary to compute the intersection.

Table A.1 - Set Intersection Results, in milliseconds

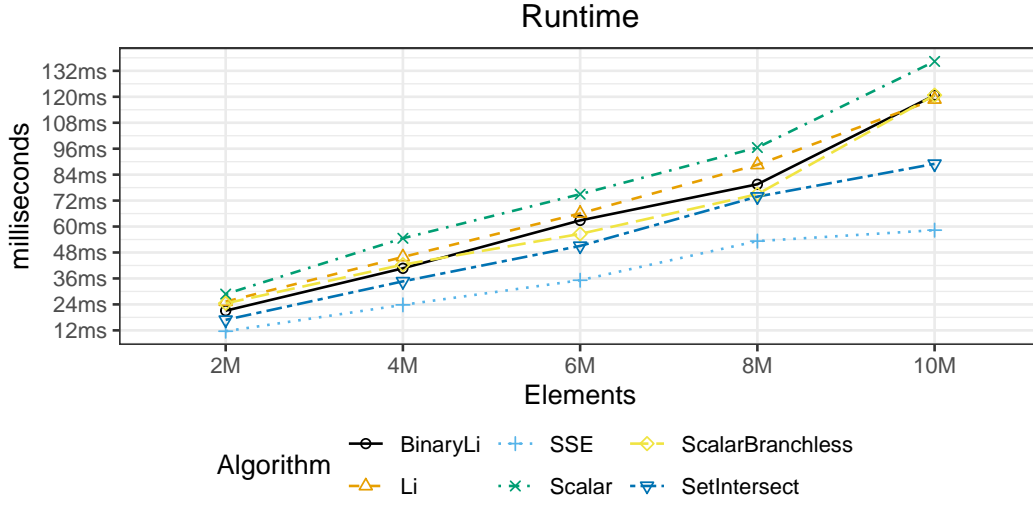
Algorithm - N	2×10^6	4×10^6	6×10^6	8×10^6	1×10^7
SSE	11,6	23,7	35,1	53,1	58,2
SetIntersect	16,8	34,6	50,9	73,6	88,8
BinaryLi	21,0	40,6	62,7	79,3	120
ScalarBranchless	24,4	42,3	56,5	74,7	120
Li	25,2	45,8	65,9	88,3	119
Scalar	28,7	54,4	74,8	96,3	136
UnorderedSet	948	2101	3062	3993	4999

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.

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 and Venturini (2019) show. Furthermore, the SIMD instruction set used here (SSE2) is limited even if its support is widespread, with other instruction sets (SSE3, AVX, AVX512, etc) being

Figure A.1 - Set Intersection Algorithm results



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

SOURCE: Author

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 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 $\Theta(1)$ by using a quantum computer and extending from the Bernstein-Vazirani algorithm (TIAN et al., 2019), however due to needing $\mathcal{O}(n)$ quantum storage space, 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. Some of the cited documents make decisions of which algorithm to use based on the ratio between the sizes of the two input sets, and this should also be incorporated on future algorithms.

PUBLICAÇÕES TÉCNICO-CIENTÍFICAS EDITADAS PELO INPE

Teses e Dissertações (TDI)

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

Manuais Técnicos (MAN)

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

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.

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.

Propostas e Relatórios de Projetos (PRP)

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

Publicações Didáticas (PUD)

Incluem apostilas, notas de aula e manuais didáticos.

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.

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.

Pré-publicações (PRE)

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