Techniques for indexing large and complex datasets with missing attribute values

Safia Brinis

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito:

Assinatura: _____

Safia Brinis

Techniques for indexing large and complex datasets with missing attribute values

Doctoral dissertation submitted to the Instituto de Ciências Matemáticas e de Computação – ICMC-USP, in partial fulfillment of the requirements for the degree of the Doctorate Program in Computer Science and Computational Mathematics. *FINAL VERSION*

Concentration Area: Computer Science and Computational Mathematics

Advisor: Prof. Dr. Caetano Traina Junior

Co-advisor: Profa. Dra. Agma Juci Machado Traina

USP – São Carlos
July 2016

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados fornecidos pelo(a) autor(a)

B634t Brinis, Safia
Techniques for indexing large and complex datasets
with missing attribute values / Safia Brinis;
orientador Caetano Traina Junior; coorientadora Agma
Juci Machado Traina. - São Carlos - SP, 2016.
136 p.

Tese (Doutorado - Programa de Pós-Graduação em
Ciências de Computação e Matemática Computacional)
- Instituto de Ciências Matemáticas e de Computação,
Universidade de São Paulo, 2016.

1. Missing attribute values. 2. Similarity search.
3. Metric access methods. 4. Fractal dimension. I.
Junior, Caetano Traina, orient. II. Traina, Agma Juci
Machado, coorient. III. Título.

Safia Brinis

**Técnicas de indexação de grandes conjuntos de dados
complexos com valores de atributos faltantes**

Tese apresentada ao Instituto de Ciências
Matemáticas e de Computação – ICMC-USP,
como parte dos requisitos para obtenção do título
de Doutora em Ciências – Ciências de Computação e
Matemática Computacional. *VERSÃO REVISADA*

Área de Concentração: Ciências de Computação e
Matemática Computacional

Orientador: Prof. Dr. Caetano Traina Junior

Coorientadora: Profa. Dra. Agma Juci
Machado Traina

USP – São Carlos
Julho de 2016

ACKNOWLEDGEMENTS

I would like to thank my advisor Prof. Caetano Traina Jr and my co-advisor Prof. Agma Juci Machado Traina for their wonderful inspiration and commitment to this project, without which this work would not be accomplished. I also like to thank my family and my friends for their inestimable support and encouragement during my work on this project. Many special thanks go to the funding agency CAPES¹ for the financial support during this doctoral project. I would also like to thank the funding agencies FAPESP² and CNPq³ that are supporting the ongoing research in the GBdI⁴ laboratory.

¹ <http://www.capes.gov.br>

² <http://www.fapesp.br>

³ <http://cnpq.br>

⁴ <http://www.gbdi.icmc.usp.br>

RESUMO

BRINIS, S.. **Techniques for indexing large and complex datasets with missing attribute values**. 2016. 136 f. Doctoral dissertation (Doctorate Candidate Program in Computer Science and Computational Mathematics) – Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos – SP.

O crescimento em quantidade e complexidade dos dados processados e armazenados torna a busca por similaridade uma tarefa fundamental para tratar esses dados. No entanto, atributos faltantes ocorrem frequentemente, inviabilizando os métodos de acesso métricos (MAMs) projetados para apoiar a busca por similaridade. Assim, técnicas de tratamento de dados faltantes precisam ser desenvolvidas. A abordagem mais comum para executar as técnicas de indexação existentes sobre conjuntos de dados com valores faltantes é usar um "indicador" de valores faltantes e usar as técnicas de indexação tradicionais. Embora, esta técnica seja útil para os métodos de indexação multidimensionais, é impraticável para os métodos de acesso métricos. Esta dissertação apresenta os resultados da pesquisa realizada para identificar e lidar com os problemas de indexação e recuperação de dados em espaços métricos com valores faltantes. Uma análise experimental dos MAMs aplicados a conjuntos de dados incompletos identificou dois problemas principais: distorção na estrutura interna do índice quando a falta é aleatória e busca tendenciosa na estrutura do índice quando o processo de falta não é aleatório. Uma variante do MAM Slim-tree, chamada *Hollow-tree* foi proposta com base nestes resultados. A *Hollow-tree* usa novas técnicas de indexação e de recuperação de dados com valores faltantes quando o processo de falta é aleatório. A técnica de indexação inclui um conjunto de políticas de indexação que visam a evitar distorções na estrutura interna dos índices. A técnica de recuperação de dados melhora o desempenho das consultas por similaridade sobre bases de dados incompletas. Essas técnicas utilizam o conceito de *dimensão fractal* do conjunto de dados e a densidade local da região de busca para estimar um raio de busca ideal para obter uma resposta mais correta, considerando os dados com valores faltantes como uma resposta potencial. As técnicas propostas foram avaliadas sobre diversos conjuntos de dados reais e sintéticos. Os resultados mostram que a *Hollow-tree* atinge quase 100% de precisão e revocação para consultas por abrangência e mais de 90% para k vizinhos mais próximos, enquanto a Slim-tree rapidamente deteriora com o aumento da quantidade de valores faltantes. Tais resultados indicam que a técnica de indexação proposta ajuda a estabelecer a consistência na estrutura do índice e a técnica de busca pode ser realizada com um desempenho notável. As técnicas propostas são independentes do MAM básico usado e podem ser aplicadas em uma grande variedade deles, permitindo estender a classe dos MAMs em geral para tratar dados faltantes.

Palavras-chave: Valores de atributos faltantes, Busca por similaridade, Métodos de acesso métricos, Dimensão fractal.

ABSTRACT

BRINIS, S.. **Techniques for indexing large and complex datasets with missing attribute values**. 2016. 136 f. Doctoral dissertation (Doctorate Candidate Program in Computer Science and Computational Mathematics) – Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos – SP.

Due to the increasing amount and complexity of data processed in real world applications, similarity search became a vital task to store and retrieve such data. However, missing attribute values are very frequent and metric access methods (MAMs), designed to support similarity search, do not operate on datasets when attribute values are missing. Currently, the approach to use the existing indexing techniques on datasets with missing attribute values just use an "indicator" to identify the missing values and employ a traditional indexing technique. Although, this approach can be applied over multidimensional indexing techniques, it is impractical for metric access methods. This dissertation presents the results of a research conducted to identify and deal with the issues related to indexing and querying datasets with missing values in metric spaces. An empirical analysis of the metric access methods when applied on incomplete datasets leads us to identify two main issues: distortion of the internal structure of the index when data are missing at random and skew of the index structure when data are not missing at random. Based on those findings, a new variant of the Slim-tree access method, called *Hollow-tree*, is presented. It employs new techniques that are capable to handle missing data issues when missingness is ignorable. The first technique includes a set of indexing policies that allow to index objects with missing attribute values and prevent distortions to occur in the internal structure of the indexes. The second technique targets the similarity queries to improve the query performance over incomplete datasets. This technique employs the *fractal dimension* of the dataset and the local density around the query object to estimate an ideal radius able to achieve an accurate query answer, considering data with missing values as a potential response. Results from experiments with a variety of real and synthetic datasets show that *Hollow-tree* achieves nearly 100% of precision and recall for Range queries and more than 90% for k Nearest Neighbor queries, while Slim-tree access method deteriorates with the increasing amount of missing values. The results confirm that the indexing technique helps to establish consistency in the index structure and the searching technique achieves a remarkable performance. When combined, the new techniques allow to explore properly all the available data even with high amounts of missing attribute values. As they are independent of the underlying access method, they can be adopted by a broad range of metric access methods, allowing to extend the class of MAMs.

Key-words: Missing attribute values, Similarity search, Metric access methods, Fractal dimension.

LIST OF FIGURES

Figure 1 – Example of objects with missing values projected in a three-dimensional space using an indicator value.	30
Figure 2 – Example illustrating objects assigned to a tree node	31
Figure 3 – Example of data distribution in nodes where representatives are complete and where representatives have missing values.	32
Figure 4 – Examples of similarity queries	41
Figure 5 – An example of a metric space and a range query	43
Figure 6 – Slim-tree index structure.	46
Figure 7 – Sierpinski triangle	49
Figure 8 – Distance plot of the Sierpinski dataset and its fractal dimension.	50
Figure 9 – An example of a range and k -NN _q queries performed over a metric index	51
Figure 10 – How to use the Distance Plot to obtain the final radius r_f	52
Figure 11 – How to use the Distance Plot to obtain the local final radius r'_f	53
Figure 12 – K-NNF _q Query	54
Figure 13 – Example of a data with missing attribute values	59
Figure 14 – Example of an image with missing data and the gray-level histogram	72
Figure 15 – Example of a time series signal with missing data and its reconstruction with Discrete Wavelet Transform	73
Figure 16 – Diagram of the experiments performed to evaluate the effects of missingness on metric access methods.	81
Figure 17 – Precision and recall for NDVI datasets with MAR and MNAR assumptions	82
Figure 18 – Precision and recall for WeathFor datasets with MAR assumption	83
Figure 19 – Precision and recall for WeathFor datasets with MNAR assumption	83
Figure 20 – Query performance for NDVI datasets with MAR and MNAR assumptions	84
Figure 21 – Query performance for WeathFor datasets with MAR assumption	85
Figure 22 – Query performance for WeathFor datasets with MNAR assumption	85
Figure 23 – Probability Density Function of the distances for WeathFor datasets with MAR assumption	86
Figure 24 – Probability Density Function of the distances for WeathFor datasets with MNAR assumption	87
Figure 25 – Example of data distribution in an internal node of an index structure with MAR and MNAR processes	88
Figure 26 – Building a metric tree in two steps following the new indexing technique.	93

Figure 27 – Example of a node split using MSTSplitMiss algorithm	98
Figure 28 – Example of a node split with duplicated representative	98
Figure 29 – Example of equal feature vectors resulting from different images with missing values	99
Figure 30 – Example of a 10-Nearest Neighbour query with $k\text{-}NN_q$ and $k\text{-}NNFM_q$. . .	100
Figure 31 – Desirable number of complete objects and objects with NULL values to be retrieved	102
Figure 32 – Diagram of the experiments performed to evaluate the performance of the <i>Hollow-tree</i> metric access method.	107
Figure 33 – Fractal dimension of the Sierpinski datasets with different amounts of missing data.	108
Figure 34 – Precision for $k\text{-}NN_q$ and $k\text{-}NNFM_q$ queries processed over <i>Hollow-tree</i> and Slim-tree indexes for Weather dataset	110
Figure 35 – Recall for $k\text{-}NN_q$ and $k\text{-}NNFM_q$ queries processed over <i>Hollow-tree</i> and Slim-tree indexes for Weather dataset	111
Figure 36 – Precision for $50\text{-}NN_q$ and $50\text{-}NNFM_q$ queries processed over <i>Hollow-tree</i> and Slim-tree indexes for synthetic datasets.	112
Figure 37 – Recall for $50\text{-}NN_q$ and $50\text{-}NNFM_q$ queries processed over <i>Hollow-tree</i> and Slim-tree indexes for synthetic datasets.	112
Figure 38 – Precision for $100\text{-}NN_q$ and $100\text{-}NNFM_q$ queries processed over <i>Hollow-tree</i> and Slim-tree indexes for synthetic datasets.	113
Figure 39 – Recall for $100\text{-}NN_q$ and $100\text{-}NNFM_q$ queries processed over <i>Hollow-tree</i> and Slim-tree indexes for synthetic datasets.	113
Figure 40 – Precision for $200\text{-}NN_q$ and $200\text{-}NNFM_q$ queries processed over <i>Hollow-tree</i> and Slim-tree indexes for synthetic datasets.	114
Figure 41 – Recall for $200\text{-}NN_q$ and $200\text{-}NNFM_q$ queries processed over <i>Hollow-tree</i> and Slim-tree indexes for synthetic datasets.	114
Figure 42 – Precision for $500\text{-}NN_q$ and $500\text{-}NNFM_q$ queries processed over <i>Hollow-tree</i> and Slim-tree indexes for synthetic datasets.	115
Figure 43 – Recall for $500\text{-}NN_q$ and $500\text{-}NNFM_q$ queries processed over <i>Hollow-tree</i> and Slim-tree indexes for synthetic datasets.	115
Figure 44 – Precision and recall for RM_q query processed over <i>Hollow-tree</i> and Slim-tree indexes for Weather dataset	116
Figure 45 – Precision for RM_q query processed over <i>Hollow-tree</i> and Slim-tree indexes for synthetic datasets.	117
Figure 46 – Recall for RM_q query processed over <i>Hollow-tree</i> and Slim-tree indexes for synthetic datasets.	117
Figure 47 – Efficiency parameters for $50\text{-}NN_q$ and $50\text{-}NNFM_q$ queries for Weather dataset	118

Figure 48 – Efficiency parameters for 200- NN_q and 200- $NNFM_q$ queries for Weather dataset	119
Figure 49 – Average searching radius for 50- NN_q and 50- $NNFM_q$ queries for synthetic datasets.	120
Figure 50 – Average searching radius 200- NN_q and 200- $NNFM_q$ queries for synthetic datasets.	120
Figure 51 – Total execution time for 50- NN_q and 50- $NNFM_q$ queries for synthetic datasets.	121
Figure 52 – Total execution time for 200- NN_q and 200- $NNFM_q$ queries for synthetic datasets.	121
Figure 53 – Average number of disk accesses for 50- NN_q and 50- $NNFM_q$ queries for synthetic datasets.	122
Figure 54 – Average number of disk accesses for 200- NN_q and 200- $NNFM_q$ queries for synthetic datasets.	122
Figure 55 – Average number of distance calculations for 50- NN_q and 50- $NNFM_q$ queries for synthetic datasets.	123
Figure 56 – Average number of distance calculations for 200- NN_q and 200- $NNFM_q$ queries for synthetic datasets.	123

LIST OF ALGORITHMS

Algorithm 1 – randomSplit	47
Algorithm 2 – minMaxSplit	47
Algorithm 3 – MSTSplit	48
Algorithm 4 – $k\text{-NNF}_q(s_q, k)$	54
Algorithm 5 – randomSplitMiss	95
Algorithm 6 – minMaxSplitMiss	96
Algorithm 7 – MSTSplitMiss	97
Algorithm 8 – $RM_q(s_q, r)$	99
Algorithm 9 – $k\text{-NNFM}_q(s_q, k, p)$	102

LIST OF TABLES

Table 1 – Illustration of the different mechanisms of missingness	61
Table 2 – Bitmaps encoding with missing data	67
Table 3 – Bitmaps index	67
Table 4 – Database with missing values and VA-Files representation	67
Table 5 – Summary of the techniques for missing data treatment at the index level . . .	68
Table 6 – Example of biased parameters of MNAR mechanism	75
Table 7 – Distribution of missing data for $k = 5$	79
Table 8 – MAR datasets	80
Table 9 – MNAR datasets	80
Table 10 – Datasets used in the experiments and their characteristics.	106
Table 11 – Datasets with missing attribute values.	106

LIST OF ABBREVIATIONS AND ACRONYMS

k -NN _q	k Nearest Neighbor query.
k -NNF _q . . .	k Nearest Neighbor query based on Fractal dimension.
k -NNFM _q . . .	k Nearest Neighbor query based on Fractal dimension for Missing data.
BKT	Burkhard-Keller Tree.
BR-tree . . .	Bit-string augmented R-tree.
BST-tree . .	Bisector tree.
CPU	Central Processing Unit
DBM-tree . . .	Density-Balanced Metric tree.
DBMS . . .	Database Management System
DWT	Discrete Wavelet Transform.
FD	Fractal Dimension.
FQ-tree . . .	Fixed Queries tree.
GHP-tree . .	Generalized Hyperplane tree.
GNAT	Geometric Near-Neighbor Access Tree.
Hollow-tree . .	A variant of the Slim tree for missing data.
I/O	Input / Output.
ICkNNI . . .	Incomplete Case k -Nearest Neighbors Imputation.
IDWT	Inverse Discrete Wavelet Transform.
INMET . . .	Instituto Nacional de Meteorologia.
kAndRange . .	k nearest neighbor And Range query.
kd-tree . . .	k -dimensional tree.
KDD	Knowledge Discovery and Data Mining.
kRingRange . .	k nearest neighbor Ring Range query.
M-tree	Metric tree.
MAM	Metric Access Method.
MAR	Missing At Random.
MBT	Monotonous Bisector Tree.
MCAR . . .	Missing Completely At Random.
minDist . . .	Minimum Distance.
minMaxSplit . .	Minimum Maximum Split.
minMaxSplitMiss . .	Minimum Maximum Split for Missing data.

minOccup Minimum Occupancy.
 MM-tree . . Memory-based Metric tree.
 MNAR . . . Missing Not At Random.
 MOSAIC . Multiple One-dimensional Single-Attribute Indexes.
 MSTSplit . Minimal Spanning Tree Split.
 MSTSplitMiss Minimal Spanning Tree Split for Missing data.
 MVP-tree . Multi-Vantage-Point tree.
 NDVI Normalized Difference Vegetation Index.
 R_q Range query.
 randomSplit Random Split.
 randomSplitMiss Random Split for Missing data.
 RM_q Range query for Missing data.
 SAM Spacial Access Method.
 VP-tree . . . Vantage-Point tree.
 VT Voronoi Tree.

LIST OF SYMBOLS

m — Number of attributes.

s_q — Query element $s_q \in \mathbb{S}$.

r — Covering radius.

$\mathbb{R}^m$ — m-dimensional space.

$\mathbb{S}$ — Data domain.

S — Dataset $S \subset \mathbb{S}$.

d — Distance function.

s_1, s_2, s_3 — Elements of the data domain $\mathbb{S}$.

∞ — Infinite values.

$=, \neq$ — Exact matching comparison operators.

$<, \leq, >, \geq$ — Relational comparison operators.

$\times$ — Cartesian product operator.

L_p — Minkowski distance function family.

L_2 — Euclidean distance function.

L_1 — Manhattan distance function.

d_E — Edit distance function.

s, t — String elements.

d_Q — Quadratic distance.

d_C — Canberra distance.

k — Number of nearest neighbors.

bp — Ball partitioning.

ghp — Generalized hyperplane partitioning.

rep, rep_1, rep_2 — Representative elements.

c — Minimum occupancy.

C — Maximum occupancy.

A, A_1, A_2 — Tree nodes.

$\mathcal{D}$ — Distance Exponent.

n — Cardinality of the dataset.

$PC(r)$ — Number of pairs of objects within a distance r .

K_p — Proportionality constant.

R — Diameter of the dataset.

r_f — Final radius.

$\log$ — Logarithm function.

e — Exponential function.

k' — Number of elements, $k' < k$.

r'_f — Local final radius, $r'_f > r_f$.

k'' — Number of elements, $k'' \leq k$.

L, L_1, L_2 — Lists of elements.

I — Indicator variable of missingness.

X, Y — Random variables.

X_{obs}, Y_{obs} — Observed values of the random variables X and Y , respectively.

X_{miss}, Y_{miss} — Missing values of the random variables X and Y , respectively.

α — Parameter estimation.

ε — Gap of parameter estimation.

f — Probability Density Function.

k_{com} — Number of complete objects.

k_{miss} — Number of objects with missing values.

CONTENTS

1	INTRODUCTION	27
1.1	Motivation	27
1.2	Problem Definition and Main Objectives	30
1.3	Summary of the Contributions	33
1.4	Final Considerations	34
2	SIMILARITY SEARCH IN METRIC SPACES	37
2.1	Introduction	37
2.2	Metric Spaces	38
2.3	Distance Functions	38
2.3.1	<i>Examples</i>	38
2.3.2	<i>Complexity Evaluation</i>	40
2.4	Similarity Queries	40
2.5	Metric Access Methods	42
2.6	Fractal Dimension for Similary Search	48
2.6.1	<i>Fractals and Fractal Dimension</i>	48
2.6.2	<i>Fractal Dimension for k-NN_q Queries</i>	51
2.7	Final Considerations	55
3	MISSING DATA AND RELATED WORK	57
3.1	Introduction	57
3.2	Missing Data Theory	58
3.2.1	<i>Missing Data</i>	58
3.2.2	<i>Legitimate or Illegitimate Missing Data</i>	59
3.2.3	<i>Missing Data Mechanisms</i>	60
3.2.4	<i>Ignorable and Non-ignorable Missingness</i>	61
3.2.5	<i>Consequences of MCAR, MAR and MNAR</i>	62
3.3	Missing Data Treatment	62
3.3.1	<i>Treatment at the Data Level</i>	63
3.3.1.1	<i>Case Deletion</i>	63
3.3.1.2	<i>Missing Data Imputation</i>	63
3.3.1.3	<i>Model-based Imputation</i>	64
3.3.2	<i>Treatment at the Index Level</i>	65

3.3.2.1	<i>Conventional Method</i>	65
3.3.2.2	<i>BR-tree</i>	66
3.3.2.3	<i>MOSAIC</i>	66
3.3.2.4	<i>Extended Bitmaps</i>	66
3.3.2.5	<i>Extended VA-Files</i>	66
3.4	Final Considerations	68
4	THEORY OF MISSING DATA IN METRIC SPACES	71
4.1	Introduction	71
4.2	Formal Model for Missing Data in Metric Spaces	74
4.2.1	<i>Bias of Parameter Estimation</i>	74
4.2.2	<i>Formal Model for Missing Data</i>	75
4.3	Experimental Analysis	77
4.3.1	<i>Experimental Set-up</i>	78
4.3.1.1	<i>Data Description</i>	78
4.3.1.2	<i>Data Preparation</i>	78
4.3.1.3	<i>Methodology</i>	79
4.3.2	<i>Performance Results</i>	81
4.3.2.1	<i>Effectiveness Parameters</i>	82
4.3.2.2	<i>Efficiency Parameters</i>	84
4.3.2.3	<i>Distance Concentration</i>	86
4.4	Final Considerations	88
5	HOLLOW-TREE	91
5.1	Introduction	91
5.2	The Hollow-tree Access Method	92
5.2.1	<i>Indexing Technique</i>	92
5.2.1.1	<i>SplitNode Policies</i>	94
5.2.1.2	<i>ChooseSubtree Policies</i>	96
5.2.2	<i>Similarity Queries</i>	97
5.2.2.1	<i>The RM_q Query</i>	98
5.2.2.2	<i>The k-NNFM$_q$ Query</i>	98
5.3	Final Considerations	102
6	HOLLOW-TREE EVALUATION	105
6.1	Introduction	105
6.2	Experimental Set-up	105
6.3	Performance Metrics	108
6.3.1	<i>Effectiveness Parameters</i>	108
6.3.2	<i>Efficiency Parameters</i>	109

6.4	Experimental Results	109
6.4.1	<i>Effectiveness Assessment for k-NN_q and k-NNFM_q Queries</i>	109
6.4.2	<i>Effectiveness Assessment for RM_q Query</i>	116
6.4.3	<i>Efficiency Assessment for k-NN_q and k-NNFM_q Queries</i>	118
6.5	Final Considerations	124
7	CONCLUSION	125
7.1	Main Contributions	125
7.1.1	<i>Background of Missing Data</i>	125
7.1.2	<i>Formal Model for Missing Data in Metric Spaces</i>	126
7.1.3	<i>Empirical Analysis for Missing Data in Metric Spaces</i>	126
7.1.4	<i>Indexing Technique for Datasets with Missing Values</i>	127
7.1.5	<i>Similarity Queries for Datasets with Missing Values</i>	127
7.1.6	<i>Hollow-tree Access Method</i>	128
7.1.7	<i>Extended Family of Metric Access Methods</i>	129
7.2	Future Work	129
7.3	Publications Generated in this Ph.D. Work	130
	BIBLIOGRAPHY	131

INTRODUCTION

1.1 Motivation

Missing data is a common fact in real world applications. This may happen at the time of data collection or preprocessing due to errors in sampling, failures in transmission, in recording, storing, etc. For example, medical data often have missing diagnostic tests that would be relevant for estimating the likelihood of diagnoses or for predicting the effectiveness of a treatment. In spatial data, measurements can be missing because of sensor malfunction, low battery levels or insufficient resolution. Another example are census data where participants choose not to answer sensitive questions for personal or safety reasons. There are two forms of missingness: missing records, i.e., objects, and missing attribute values, i.e., some but not all attribute values from an existing object. While missing records may only be problematic in restricted situations (e.g., smaller size of the sample provides lower statistical power of the data), missing attribute values can influence the quality of data and the performance of the tasks that operate on such data.

For any type of data, missingness can be *legitimate* or *illegitimate* (OSBORNE, 2012). Legitimate missingness is an absence of data where there is no suitable value for the missing data. Illegitimate missingness refers to the absence of data when it is possible to get the value in principle. Legitimate missingness is very common in surveys. For example, when filling out a survey that asks whether the participant is employed, and if so, what is the value of his/her salary. If the participant is not employed, then, it is legitimate to skip the value of the salary. Illegitimate missingness also exists in all types of data, nevertheless, it is more present in modern databases because of the exotic nature of the data, such as images, time series and videos. It occurs, for instance, when an equipment accidentally stops from recording data. Therefore, the diversity of equipments used to collect data, such as X-ray generators, sensors, video recorders and finger print scanners, and their susceptibility to malfunctions and failures often lead to illegitimate missing data. This latter is mainly the one that most of researches are concerned with, because it has the potential of *bias*, also called *skew*.

One of the major aspects of missingness is the degree of randomness in which missing data occurred. Rubin (RUBIN, 1976) developed a framework of inference about missing data and defined three classes of missingness, also known as *mechanisms of missingness*. *Missing Completely At Random* (MCAR) refers to the scenario where missingness of attribute values is independent of other attribute values (observed or missing). *Missing At Random* (MAR) corresponds to the scenario where missingness of attribute values depends only on observed values. *Missing Not At Random* (MNAR) corresponds to the scenario where missingness of attribute values depends on missing values. For example, consider a population of n individuals who had their blood pressure measured and a random sample of size $n' < n$ who also had their weight measured. Missingness on weight measure is MCAR because it is randomly sampled and does not depend on any other value. Now, if we consider the same population that had their blood pressure measured and only those individuals with high blood pressure had their weight measured; missingness on weight measure is MAR because it depends on the observed values of blood pressure. Note that, if individuals with high blood pressure are likely to be overweight, it would appear that only overweight individuals had their weight measured and thus, missingness is MNAR, because missing values on weight depend on the measurements of weight where missing values occurred. As a consequence, the average value of the weight will be overestimated since low values are missing. Therefore, any parameter estimation of the weight measure will yield bias (i.e., underestimate or overestimate) in the results. The latter example shows how MNAR data can cause bias in the parameter estimation, therefore, determining the mechanism of missingness is very important, because it can influence the choice of an appropriate treatment for missing data.

The past few decades have witnessed a great development of techniques for missing data treatment. The main interest appeared first in the problems caused in surveys and census data (RUBIN, 1976; SCHAFER, 1997; LITTLE; RUBIN, 2002; RUBIN, 2004). Traditional approaches for dealing with missing data essentially include *Case Deletion* and *Missing Data Imputation*. Case deletion consists of deleting the instances with missing values and performing the analysis using only complete instances. Despite being simple and amenable for implementation, this method suffers from an important loss of relevant information. On the other hand, imputation methods aim at estimating missing values using relationships that can be identified in the observed values of the dataset. Several methods of imputation have been proposed in the literature, ranging from the most simple *Mean or Mode Substitution* to the most sophisticated *Multiple Imputation*. However all these methods are either biased or computationally expensive. In addition, in many cases, some attributes have missing values because they are not relevant for certain objects, therefore, any imputation method would become inappropriate.

Modern scientific and engineering applications generated large amounts of complex data. These data are growing both in the number of objects (cardinality) and the number of attributes (dimensionality), and also in the complexity of the features that describe each attribute. Therefore, it is very common to have missing values occurring in some of the attributes. Both non-

dimensional and high-dimensional data are usually stored and retrieved following a similarity criteria, and the process is called *Similarity Search*. Similarity search involves finding objects that are similar to a given query object based on a similarity measure. The searching process is usually performed with range query or nearest neighbor query, often using an index structure to speed up the searching process. There are two families of access methods that provide indexing support for similarity search: *Multidimensional Access Methods* and *Metric Access Methods* (MAMs). Multidimensional access methods, such as R-tree (GUTTMAN, 1984) and kd-tree (BENTLEY, 1975; BENTLEY, 1979), are created to support efficient selection of objects based on spacial properties. Many multidimensional access methods have been proposed in the literature (SAMET, 1984; GAEDE; GÜNTHER, 1998) aiming at improving the performance of the similarity queries over multimedia datasets. However, they are not appropriate for datasets where the only available information is the distances between objects, like DNA datasets. This fact is one of the reasons that led to the development of metric access methods.

Metric access methods are fundamental to organize and access both non-dimensional and high-dimensional data. They employ index structures to organize the data based on a distance function and the space is divided into regions using a set of chosen objects, called *representatives*, and their distances to the rest of the objects in the space. A metric space is a pair $(\mathbb{S}, d)$, where $\mathbb{S}$ denotes a data domain and $d : \mathbb{S} \times \mathbb{S} \rightarrow \mathbb{R}^+$ is a distance function that satisfies the symmetry, the non-negativity and the triangle inequality properties. A dataset S is in a metric space when $S \subset \mathbb{S}$. When a query is issued, the properties of the metric space are used to prune the searching space in order to speed up the query process. Many metric access methods have been proposed in the literature (HJALTASON; SAMET, 2003; SAMET, 2006), however, they were all designed to perform only on complete datasets; and when applied on datasets with missing values, they lose their functionalities and suffer from poor query performance.

Missing attribute values are a major concern to perform similarity search over data indexed in a metric space. Similarity search involves comparison operations based on a distance function that helps comparing pairs of objects in a data domain $\mathbb{S}$ and returns a numerical value that estimates the degree of similarity between the objects. The distance functions are designed to measure the similarity between fully specified objects. They cannot be applied with data having missing values. The problem stems from the fact that the distance measures between objects with missing values are undefined. For the metric indexes built on pairwise distances between objects, objects with missing values are dismissed from the index structure. One way to deal with the problem is to ignore missing attribute values and consider only observed attribute values when measuring the distances. Nevertheless, distance measures are prone to distortion and skew when relevant attributes of the objects are missing, leading to inconsistency in the metric space.

Figure 1 – Example of objects with missing values projected in a three-dimensional space using an indicator value.

1.2 Problem Definition and Main Objectives

Advanced database applications often deal with incomplete data. Such applications usually involve large datasets of non-dimensional or high-dimensional data objects. Therefore, there is always a possibility of having erroneous or missing data.

The most popular approach used to make the existing indexing techniques operational on incomplete datasets is to use an indicator variable to identify the missing values, then employ a traditional indexing technique to index the dataset. For example, if the domain of an attribute are the positive integers, then, a value of -1 can be used to denote the missing values. Data stored in relational Database Management Systems (DBMS) employ the attribute state of NULL values to indicate missing data. With this scheme and for a multidimensional indexing technique, all the objects with missing values on a given dimension will be projected into the same value of the hyper-region (see Figure 1), causing skew in the indexed space. When the proportion of missing values is large, it will result in a highly skewed data space.

Metric access methods are even more susceptible to skew despite the fact that there is no direct concept of projection. The reason is that metric indexes are built on pairwise distances between the objects, and the indicators of missing values will be certainly involved in the estimation of the distances. But since the indicators are invalid values of the attributes, this can cause distortion in the distances between pairs of objects. Therefore, indicator values are only

Figure 2 – Example illustrating objects s_i and s_j assigned to a tree node A , where $s_i \in A$ because $d(s_i, Rep) < r$, and $s_j \notin A$ because $d(s_j, Rep) > r$.

useful for multidimensional indexing techniques and they are impractical for metric indexing techniques.

On the other hand, most of the distance functions are designed to measure the similarity between complete objects, and it is not feasible to apply them on data with unknown (i.e., missing) values. For instance, if we consider the objects $s_1, s_2 \in \mathbb{S}$ with attribute values denoted by $s_{1,i}, s_{2,i}$ for $1 < i < m$, where m is the number of attributes. The squared Euclidean distance between s_1 and s_2 is the sum of the squared differences between the mutual attribute values, i.e., $\sum_{i=1}^m (s_{1,i} - s_{2,i})^2$. If one of the objects has missing values on some attributes, the differences between the mutual attributes with missing values are unknown, and consequently the distance measure becomes undefined.

Index structures are essential to accelerate similarity search and tree structures are commonly employed to index data in metric spaces. In such structures, the objects of the dataset are organized in a hierarchical structure where one of the stored objects is elected to be the representative of each sub-tree. The representative is the center of each minimum bounding region that covers the objects in the sub-tree. An object can be assigned to a node if the covering radius of the representative covers it, that is, the distance between the object and the representative of the node is smaller than its covering radius (see Figure 2). Hence, any object with missing values will be dismissed from the metric tree because the distance between the object and the representative is undefined. In addition, representatives can be chosen among objects with missing values and the entire node will remain empty because the distance between the representative and any other object is undefined. Therefore, the more representatives are chosen with missing values the less objects are indexed, which can lead to a severe loss of information. A practical solution is to force the distance function to ignore missing attribute values and consider only observed attributes. However, if two objects do not have any observed attribute in common, it becomes unfeasible to measure the distance between them. Nevertheless, missingness can underestimate or overestimate the distances between pairs of objects, and when data are missing at random, distance measures are prone to distortions when relevant attributes are missing, but when data are missing not at

Figure 3 – Data distribution in nodes where (a) representative is complete, (b) representative has missing values.

random, distance measures are prone to skew. In particular, objects with missing values tend to shrink the subspaces where missingness occurred, causing *Distance Concentration*. The major problem here is that if the representatives are chosen among the objects that have missing values, the bounding regions of the associated nodes are likely to shrink because of smaller distances (see Figure 3). Moreover, for the same reason, representatives with missing values potentially hold more objects during the indexing and searching processes, independently from if they are similar or not. This fact can lead to inconsistency in the index structure. Therefore, when similarity queries are issued, it is more likely to have irrelevant objects in the query response (false positives), having relevant ones not included in the response (false negatives), which leads to poor query performance.

The performance of a metric access method regards its overall execution time, which in turn depends mainly on the number of distance calculations and the number of disk accesses it needs to execute the similarity queries. However, when dealing with data that have missing attribute values, the retrieval accuracy (i.e., effectiveness of the similarity queries) becomes another major measurement of the index quality. The reason is that missing attribute values may introduce changes to the distribution of data in the metric space and, subsequently, cause inconsistency in the data structure, leading to inaccurate query response. In the course of this Doctoral work, we seek to demonstrate that problem, thus, we established three major hypotheses to guide our work:

1. Data with the MAR mechanism cause distortion in the internal structure of the metric index and affect significantly the effectiveness parameters of the query process.
2. Data with the MNAR mechanism cause skew (i.e., bias) in the internal structure of the metric index and affect significantly the efficiency parameters of the query process.

3. The performance of a metric access method in incomplete high-dimensional spaces are influenced differently when data are MAR and when data are MNAR.

To demonstrate the validity of these hypotheses, this research was conducted with the following specific goals:

1. Investigate the key issues involved when indexing and searching datasets with missing attribute values in metric spaces.
2. Identify the effects of each mechanism of missingness on the metric access methods when applied on incomplete datasets.
3. Based on these effects, formalize the problem of missing data in metric spaces, proposing a "Model of Missingness".
4. Based on the formal model of missingness, develop new techniques to support similarity search over large and complex datasets with missing values, in order to overcome the limitations of metric access methods and allow traditional methods to explore the available data while preserving their functionalities and their characteristics.

1.3 Summary of the Contributions

In this Ph.D. work, we identified the issues related to missing data when indexed and searched in metric spaces, and we used them to establish a formal Model of Missingness able to describe missingness in metric spaces. The analyzes of the metric access methods when applied on incomplete datasets (see the upcoming chapter 4) led us to identify two main issues. The first issue is related to distortions that occur in the internal structure of the index when the mechanism of missingness is random. The second issue concerns the skew of the index structure caused by the MNAR mechanism. The findings suggest that the effects of the MAR and the MNAR mechanisms described in the statistics literature apply also to the metric spaces when incomplete datasets are indexed. With regard to the MAR mechanism, objects with missing values may change the distribution of data in the space while keeping the overall sparseness (i.e., volume) of the metric space. With regard to the MNAR mechanism, objects with missing values are capable to skew the distribution of data in the space causing a significant distance concentration.

Based on these findings, new techniques for indexing and querying incomplete datasets in metric spaces are proposed. Assuming that missingness mechanism is MAR, our new indexing technique is capable of building consistent indexes by preventing objects with missing values from causing distortion in the internal structure of the indexes. It also guarantees that no object is missing from the metric index. Another technique targets the similarity queries to improve the query performance. We defined two new query types that perform over datasets of objects

with missing values. Both return two separated lists: one for the complete objects retrieved and another for objects with missing values. The new query types are:

- The range query for missing data – $RM_q(s_q, r)$: It receives a query center s_q and a query radius r as parameters, and returns the complete objects and the objects with missing values that are distant at most at the covering radius r from s_q .
- The k Nearest Neighbor query based on Fractal Dimension for Missing data – $k\text{-NNFM}_q(s_q, k)$: It employs the fractal dimension of the dataset and the local density of the query region to search for similar objects and filter data in the query response. The query response is composed of two lists: A list of complete objects and a list of objects with missing values.

We applied these techniques to the Slim-tree access method ([TRAINA et al., 2002](#)), which led to the development of a new variant, called *Hollow-tree*, to support datasets with missing values.

Existing metric access methods do not support indexing data with missing attribute values. The proposed *Hollow-tree* can efficiently index and search data with missing attribute values. Prior approaches for missing data treatment provided support for multidimensional access methods, which resulted in a severe performance that is often worse than the sequential scan when the number of missing values is significantly high ([OOI; GOH; TAN, 1998](#)). Other approaches supported indexing and searching incomplete datasets in non-hierarchical indexing techniques ([CANAHUATE; GIBAS; FERHATOSMANOGLU, 2006](#)). These techniques exhibit a linear performance with respect to dimensionality and amount of missing data, however, they are not suitable to process large and complex datasets. The ability to accommodate non-dimensional and high-dimensional datasets with missing data makes the *Hollow-tree* more appropriate for nowadays applications.

1.4 Final Considerations

This chapter presented an overview of this Doctoral dissertation with a description of the facts that motivated this project, the definition of the problem, the main objectives and contributions of this work. The remaining chapters of the thesis are as follows:

- Chapter 2 gives an introduction to the basic concepts of the similarity search in metric spaces, including an overview of the Fractal Theory concepts used to boost similarity queries in metric spaces. This chapter gives a background for the rest of the thesis.
- Chapter 3 provides a description of different concepts related to missing data, including some relevant works found in the literature for missing data treatment.

-
- Chapter 4 studies the effect of missing data when indexed in metric spaces. A formal model for missing data is defined to provide an understanding of the effects of distortion and skew caused by missingness in metric spaces.
 - Chapter 5 describes the proposed techniques for indexing and querying incomplete datasets.
 - Chapter 6 is dedicated to the performance evaluation of the proposed techniques.
 - Chapter 7 summarizes the results obtained in the dissertation and indicates some directions for a future research.

SIMILARITY SEARCH IN METRIC SPACES

2.1 Introduction

In classical databases systems, where most of the attributes are either textual or numerical, the fundamental search operation is matching, i.e., given a query object, the system finds the objects in the database whose attributes match those specified in the query object. The result of this type of queries is, typically, a set of objects that are, in some sense, the same as the query object. In multimedia databases, this kind of operation is not appropriate. In fact, due to the complexity of multimedia objects, such as images, audio and video collections, DNA or protein sequences, among others; matching is not sufficiently expressive. Instead, objects should be searched by similarity criteria. Similarity search involves a collection of objects (e.g., documents, images, videos) that are characterized by a set of relevant features and represented as points in the high-dimensional space. Given a query in the form of a point in the space, similarity queries search for the nearest (most similar) object to the query object. The particular case of the Euclidean space is of a major importance for a variety of applications, like databases and data mining, information retrieval, data compression, machine learning and pattern recognition, etc. Typically, the features of the objects are represented as points in the $\mathbb{R}^m$ and a distance function is used to measure the degree of similarity between them.

In this chapter, we provide the necessary concepts for the understanding of similarity search in metric spaces. A definition of the metric spaces is presented in Section 2.2, while a description of the distance functions, including examples, are provided in Section 2.3. Section 2.4 includes a definition of the main queries employed by similarity search operations, and Section 2.5 describes the metric access methods and provides a classification based on partitioning approaches. Also, an overview of the Fractal Theory concepts used to boost similarity queries in metric spaces is presented in Section 2.6. The last section concludes the chapter.

2.2 Metric Spaces

A metric space is defined as a pair $(\mathbb{S}, d)$, where $\mathbb{S}$ denotes a domain (universe) of valid objects (elements, points) and $d : \mathbb{S} \times \mathbb{S} \rightarrow \mathbb{R}^+$ a distance function. The distance function is called a metric if it satisfies the following properties:

Given any objects $s_1, s_2, s_3 \in \mathbb{S}$:

$$\begin{aligned} \forall s_1, s_2 \in \mathbb{S}, \quad d(s_1, s_2) &= d(s_2, s_1) && \text{symmetry,} \\ \forall s_1, s_2 \in \mathbb{S}, \quad 0 < d(s_1, s_2) &< \infty && \text{non-negativity,} \\ \forall s_1 \in \mathbb{S}, \quad d(s_1, s_1) &= 0 && \text{identity,} \\ \forall s_1, s_2 \in \mathbb{S}, \quad d(s_1, s_2) + d(s_2, s_3) &\geq d(s_1, s_3) && \text{triangle inequality.} \end{aligned}$$

The distance function d measures the distance between two objects and returns a real non-negative value, which assesses the similarity degree between them. The smaller the distance measure is, the closer or more similar the objects are; the larger the distance measure is, the less similar the objects are.

2.3 Distance Functions

Distance functions are often associated to a specific application or a specific data type. Depending on the returned value, typically, there are two groups of distances:

- **Discret** - the distance function returns a small set of discrete values, such as the Edit distance,
- **Continuous** - the distance function returns a set of values whose cardinality is very large or infinite, such as the Euclidean distance.

2.3.1 Examples

Some of the most popular distance functions include the Minkowski distances, Edit distance and the Quadratic Form distance.

Minkowski Distance

Minkowski distances represent the family of metric distances, called L_p metrics. They are defined on m -dimensional vectors of real numbers X and Y as follows:

$$L_p(X, Y) = \sqrt[p]{\sum_{i=1}^m |x_i - y_i|^p} \quad p \geq 1 \quad (2.1)$$

Special cases of such metrics are the Euclidean (L_2) and the Manhattan, or city bloc, (L_1) distances. When $p \rightarrow \infty$, we obtain the L_∞ metric defined as $L_\infty = \max_{i=1}^m \{|x_i - y_i|\}$. L_p metrics

are very appropriate to measure the distance between gray-level or color histograms and between vectors of features extracted from images.

Levenshtein Distance

The Levenshtein (LEVENSHTEIN, 1965), or edit, distance $d_E(s, t)$ for strings counts the minimum number of edit operations (insertions, deletions, substitutions) needed to transform a string s into a string t .

Quadratic Form Distance

The Euclidean distance L_2 , used to measure the distance between color histograms, does not identify the correlations between the components of the histograms. A distance function that has been successfully applied to image databases (FALOUTSOS *et al.*, 1994) and that has the power to model dependencies between different components of the histogram is provided by the class of quadratic form distance functions (HAFNER *et al.*, 1995; SEIDL; KRIEGEL, 1997). Therefore, the distance measure of two m -dimensional vectors is based on an $m \times m$ matrix $A = [a_{ij}]$, where the weights a_{ij} denote the similarity between the components i and j of the vectors $\vec{x}$ and $\vec{y}$, respectively:

$$d_Q(\vec{x}, \vec{y}) = \sqrt{(\vec{x} - \vec{y})^T \cdot A \cdot (\vec{x} - \vec{y})} \quad (2.2)$$

When the matrix A is equal to the identity matrix, then we have the Euclidean distance. The quadratic form distance function is computationally expensive as the image color histograms are typically high-dimensional vectors, consisting of 64 or 256 distinct colors.

Canberra Distance

Similar to the Manhattan distance, Canberra distance is however more restrictive, since the absolute difference of the feature values is divided by the sum of their absolute values. It is formally defined for m -dimensional vectors X and Y as follows:

$$d_C(X, Y) = \sum_{i=1}^m \frac{|x[i] - y[i]|}{|x[i]| + |y[i]|} \quad (2.3)$$

This distance is very sensitive to small changes, and it has been used in different areas such as DNA sequences in bioinformatics, and also in computer intrusion detection.

Note that, choosing the most adequate distance function to the type of data (or features) is one of the key aspects to improve the retrieval performance over multimedia databases when answering similarity queries.

2.3.2 Complexity Evaluation

The time required to compute the L_p and the Canberra distances is linear with respect to the number of components in the feature vectors. The Levenshtein distance employs a dynamic algorithm, thus, it is executed with quadratic complexity on the number of letters in both strings. The Quadratic form distance is even more expensive because it requires multiplications by a matrix. Thus, it is evaluated in $\mathcal{O}(n^3)$. Other distance functions that measure the distance between sets of objects can be up to $\mathcal{O}(n^4)$. Therefore, distance functions are generally considered as time consuming operations, and most of the indexing techniques for similarity search try to reduce the number of distance evaluations to speed up the searching process.

2.4 Similarity Queries

Similarity queries are defined by a query object and some selectivity criterion. The answer to the query is the set of objects that are close to the query object and satisfy the selectivity criterion. There are two types of similarity queries:

Range Query

This is the most common type of query that is used in almost every application. Provided an object s_q and a covering radius r , the $R_q(s_q, r)$ query retrieves all the objects which are within the covering radius r from s_q (see Figure 4a). Formally:

Definition 1. Given a query center $s_q \in \mathbb{S}$ and a query distance r , a **Range query** $R_q(s_q, r)$ in a metric space $(\mathbb{S}, d)$ retrieves all the objects $s_i \in \mathbb{S}$ such that $d(s_q, s_i) < r$.

Example: "Find all the restaurants that are within a distance of two miles from here".

k-NN_q Query

Provided an object s_q and an integer k , the $k\text{-NN}_q(s_q, k)$ query retrieves the k nearest neighbors to the objects s_q (see Figure 4b). Formally:

Definition 2. Given a query center $s_q \in \mathbb{S}$ and an integer $k > 0$, a **k-Nearest Neighbor query** $k\text{-NN}_q(s_q, k)$ in a metric space $(\mathbb{S}, d)$ retrieves the k objects closest to s_q , based on the distance function d .

Example: "Find the five closest restaurants to this place".

The major problem of similarity search is how to efficiently process similarity queries. When the size of the dataset is relatively small (e.g., in the order of hundreds of objects) and the distance function is not computationally expensive, a sequential scan of the entire dataset followed by a similarity assessment can be an adequate solution. However, when the size of the

Figure 4 – (a) Range query $R_q(s_q, r)$, and (b) k -nearest neighbor query $k-NN_q(s_q, k)$, with $k = 5$.

dataset is very large (e.g., in the order of hundreds of thousands or millions of objects) and the distance function is computationally expensive, the sequential scan of the entire dataset is not an adequate option. One good alternative is to structure the objects of the dataset in a way that allows to find the required objects efficiently without looking at every object of the dataset. In traditional database management systems, where sorting (or ordering) is the basis for efficient searching, index structures, such as B-tree (BAYER; MCCREIGHT, 1973), are used to search through the objects of the dataset. Although, sorting numbers, text strings or dates is easier, it is not obvious to order complex data, like images. Thus, the traditional index structures cannot be used to search complex data in general.

Multimedia applications require that access methods satisfy the following properties:

- **Efficiency:** Unlike traditional access methods which only aim at minimizing the number of disk accesses, index structures for similarity search need also to consider the CPU costs given by the number of distance calculations, since the distance functions are generally considered as time consuming operations.
- **Scalability:** Due to the size of modern multimedia data repositories which are usually in the order of millions of objects, the access methods should also perform well when the size of the database grows.
- **Dynamicity:** Access methods need to support insertions and deletions of objects from the index structures.
- **Data type independency:** Access methods need to perform well on all possible data types.
- **Use of secondary storage:** Due to the huge amounts of data, it is not feasible to store the entire dataset in main memory. Therefore, the access methods should be able to efficiently exploit secondary storage devices.

There are two families of access methods that support similarity search on multimedia datasets; *Multidimensional Access Methods*, also well-known as spacial access method (SAMs), and *Metric Access Methods* (MAMs). Multidimensional access methods are created to support efficient selection of objects based on spacial properties. These methods use the idea of partitioning the indexed space into subsets, called equivalent classes, and group into sets objects that have some property in common, e.g., the first coordinate of a vector is lower than three. For example, kd-trees (BENTLEY, 1975; BENTLEY, 1979) or quad-trees (SAMET, 1984) use a threshold (value of a coordinate) to divide the given space into subsets. On the other hand, R-tree (GUTTMAN, 1984) groups objects in an hierarchical organization using the minimum bounding rectangle. Many spacial access methods have been proposed in the literature (see (SAMET, 1984; GAEDE; GÜNTHER, 1998; SAMET, 2006) for a comprehensive survey), aiming at improving the performance of the similarity queries over multimedia datasets. However, they do not satisfy the properties described above. In fact, spacial access methods can only index multidimensional vector spaces; thus, they are not appropriate for datasets where the only available information is the distances between objects, like DNA datasets. Moreover, with regard to query performance, only I/O cost given by the number of disk accesses is considered, while CPU cost of the distance calculations is ignored. In the case of high-dimensional vector spaces, the computation of the distances should not be ignored, instead, an additional optimization is needed in order to minimize the number of computed distances. These considerations led to the development of metric access methods (UHLMANN, 1991).

2.5 Metric Access Methods

Metric access methods (MAMs) employ an index structure to organize the objects of a given dataset in an hierarchical tree structure, called *Metric Tree*; based on a distance function that satisfies the metric properties; symmetry, non-negativity and triangle inequality. Metric trees aim at dividing the space into regions, where objects that are close to each other are grouped together (see Figure 5a). Ideally, each region in the space is associated to a node in the tree, where leaf nodes store the data objects, the internal nodes store the pointers to the lower levels of the tree, and the root node holds the entire data space. Similarity search in metric trees involves accessing only the nodes where the relevant objects can be found and disposing of the nodes that do not contain relevant objects. This is often achieved by applying the triangle inequality property of the distance function, in order to speed up the searching process and reduce the number of disk accesses and the number of distance calculations (see Figure 5b). The performance of similarity queries depend mainly on the pruning ability of the underlying access method, which in turn depends on the space partitioning technique.

Partitioning metric spaces is more difficult because there are no spatial coordinates that can be used for a "geometric" split. In metric spaces, objects are chosen from the data space and are promoted as representatives (pivots), then the remaining objects are grouped according

Figure 5 – An example of a metric space and a range query: (a) A metric index structure, (b) a range query.

to their distances to the representatives. Burkhard and Keller ([BURKHARD; KELLER, 1973](#)) defined two basic partitioning techniques:

1. the *ball* partitioning *bp*, and
2. the *generalized hyperplane* partitioning *ghp*.

In the ball partitioning, one representative *rep* is chosen and a radius *r* is defined to split the objects s_i , $1 < i < n$, into two groups as follows:

$$bp(s_i, rep, r) = \begin{cases} 0 & \text{if } d(s_i, rep) \leq r, \\ 1 & \text{if } d(s_i, rep) > r. \end{cases}$$

In the hyperplane partitioning, two representatives rep_1, rep_2 are chosen and the objects s_i , $1 < i < n$, are split into two groups according to their distances to the representatives as follows:

$$ghp(s_i, rep_1, rep_2) = \begin{cases} 0 & \text{if } d(s_i, rep_1) \leq d(s_i, rep_2), \\ 1 & \text{if } d(s_i, rep_1) > d(s_i, rep_2). \end{cases}$$

The Burkhard-Keller Tree (BKT) ([BURKHARD; KELLER, 1973](#)) was the first access method that provided a recursive approach to build a metric tree, which further developed into several access methods (see ([HJALTASON; SAMET, 2003](#)) and ([SAMET, 2006](#)) for a detailed and comprehensive surveys). The Ball Decomposition of Uhlmann ([UHLMANN, 1991](#)) and the

Vantage-Point tree (VP-tree) of Yianilos (YIANILOS, 1993) are based on the ball partitioning technique, as they partition the objects into two groups according to a representative, called "vantage point". In order to reduce the number of distance calculations between the query object and the representatives, Baeza-Yates (BAEZA-YATES *et al.*, 1994) proposed an extension of the VP-tree, called the Fixed Queries tree (FQ-tree), which uses the same vantage point in all the nodes that belong to the same level. However, this method suffers space complexity that grows super-linearly because the objects selected as representatives are duplicated. Another extension of the VP-tree is the Multi-Vantage-Point tree (MVP-tree) (BOZKAYA; ÖZSOYOGLU, 1997; BOZKAYA; ÖZSOYOGLU, 1999). While VP-trees use different representatives at the lower level of an internal node, MVPT-trees employ only one, that is, all children at the lower level use the same representative. This allows to have less representatives while the number of partitions is preserved.

The Bisector tree (BST-tree)(KALANTARI; MCDONALD, 1983) is the first access method that uses the generalized hyperplane partitioning technique. BST-tree is a binary tree that is built recursively, choosing two representatives at each node and applying hyperplane partitioning. For each node, the covering radius is established as the maximum distance between the representative and the objects in its subtree and it is stored in the node. The first improvement of the BST-tree is the Voronoi Tree (VT) (DEHNE; NOLTEMEIER, 1987). VT-trees use two or three representatives in each internal node and have the property that the covering radii are reduced when moving to the bottom of the tree. This provides better packing of objects in the subtrees. Another variant of BST-tree, called Monotonous Bisector Tree (MBT), is proposed by Noltemeier *et al.* (NOLTEMEIER; VERBARG; ZIRKELBACH, 1992a; NOLTEMEIER; VERBARG; ZIRKELBACH, 1992b). The idea of this structure is that the representative of each internal node, except the root node, is inherited from its parent node. This technique allows to use less representatives and, consequently, reduces the number of distance calculations during the searching process. The Generalized Hyperplane tree (GHP-tree) proposed by Uhlmann (UHLMANN, 1991) is nearly the same as the BST-tree. It consists of partitioning the dataset recursively using the generalized hyperplane technique. The main difference is that, during a searching process, the covering radii are not used as pruning criterion, instead, the hyperplane between the representatives of two nodes is used to choose which subtree should be visited. The Geometric Near-Neighbor Access Tree (GNAT) of Bin (BRIN, 1995) is considered as a refinement of the Ball Decomposition tree. In addition to the representatives and the maximum distance, the distance between pairs of representatives are also stored, in order to be used by the triangle inequality for further pruning in the searching space.

All of these indexing methods are either static, unbalanced, or both. Thus, they are not suitable for dynamic environments where data are subject of permanent changes. The M-tree (CIACCIA; PATELLA; ZEZULA, 1997) was the first height-balanced and dynamic tree where data objects are stored in leaf nodes, and internal nodes keep pointers to nodes at the lower level along with additional information about their subtrees. An internal node is a set of entries where

each entry includes a pointer to the subtree and its covering radius r . Each node contains at least c and at most C entries, where $c \leq C/2$. The Slim-tree (TRAINA *et al.*, 2002) is an extension of the M-tree that includes a technique, called "Slim-Down", which aims at reducing the amount of overlap between the nodes. Other dynamic MAMs include the Omni-family (FILHO *et al.*, 2001), the Density-balanced Metric tree (DBM-tree) (VIEIRA *et al.*, 2010), the Memory-based Metric tree (MM-tree) (VENTURINI; TRAINA; TRAINA, 2007) and the Onion-tree (CARÉLO *et al.*, 2011).

Slim-tree

The Slim-tree (TRAINA *et al.*, 2002) is a dynamic and balanced metric tree that aims to speed up insertion and node splitting while reducing overlap between the nodes. The metric tree is built from the bottom to the root and objects are grouped into fixed size disk pages, each page corresponding to a node. A Slim-tree is organized in a hierarchical structure, where a node holds a representative object, a covering radius and a subset of objects that are inside the covering radius. Since the size of a page is fixed, the nodes hold a limited number of objects C . Slim-trees support two types of nodes, leaf nodes and index (internal) nodes (see Figure 6). The leaf nodes hold all the objects stored in the index and the structure is:

$$leaf_node[array\ of\ < Oid_i, d(s_i, s_{rep}), s_i >],$$

where Oid_i is the identifier of the object s_i and $d(s_i, s_{rep})$ is the distance between the object s_i and the representative of the leaf node s_{rep} . The index nodes hold the representatives of the nodes in the lower level and pointers that point to these nodes. The structure is:

$$index_node[array\ of\ < s_i, r_i, d(s_i, s_{rep}), Ptr(s_i), NEntries(Ptr(s_i)) >],$$

where s_i is the representative of the subtree pointed by $Ptr(s_i)$ and r_i is the covering radius of its region. The distance between the entry s_i and the representative s_{rep} of this node is kept in $d(s_i, s_{rep})$. The pointer $Ptr(s_i)$ points to the root node of the subtree rooted by s_i , for which the number of entries is kept in $NEntries(Ptr(s_i))$.

When a metric tree is built, the regions corresponding to the nodes can overlap each other. However, when nodes overlap, the number of paths traversed when a query is issued increases, which increases the number of distances computed. The Slim-tree was developed aiming at reducing the overlap between the nodes at each level. This is achieved by a set of policies used together to efficiently build the slim-tree and the Slim-Down algorithm used to reduce the overlap and produce thinner trees. When an object is inserted into the tree, the nodes that cover the new object are selected. If none qualifies, the node whose representative is nearest to the new object is selected. If more than one qualifies, **ChooseSubtree** procedure is executed to select the most appropriate node. This process is recursively applied for all the levels of the tree until the new

Figure 6 – Slim-tree index structure.

object is inserted in a leaf node. When a node overflows, **SplitNode** procedure is executed to split the node into two nodes. Subsequently, a new node is allocated at the same level and the objects are distributed among the nodes. Slim-tree provides three policies for *ChooseSubtree* procedure:

1. **random** - Chooses randomly one of the qualifying nodes.
2. **minDist** - Chooses the node for which the representative has the shortest distance to the new object.
3. **minOccup** - Chooses the node that has less objects (minimum occupancy) among the qualifying nodes. This is the default method used by Slim-tree, as it has, in general, the best performance.

The *minOccup* policy requires the number of objects in the qualified nodes. This is provided by the *NEntries* component stored in the parent node that points to these nodes. The splitting policies of the Slim-tree are illustrated in the Algorithms 1, 2 and 3.

1. **randomSplit** - The representatives are chosen randomly among the objects in the node and the remaining objects are assigned to the node having the closest representative (see the details in Algorithm 1).

2. **minMaxSplit** - Each pair of objects are considered as potential representatives and the pair that minimizes the covering radii for both nodes is chosen (see the details in Algorithm 2).

Algorithm 1: randomSplit

Data: Node A to be split
Result: Nodes A_1 and A_2 with their representatives rep_1 and rep_2 , respectively

```

1 Choose randomly  $rep_1$  from  $A$ ;
2 Choose randomly  $rep_2$  from  $A$ ;
3 while  $rep_2 = rep_1$  do
4   | Choose randomly  $rep_2$ ;
5 end
6 Set the new representatives,  $rep_1$  for node  $A_1$  and  $rep_2$  for node  $A_2$ ;
7 for each object  $s_i$  of the node  $A$  do
8   | if  $d(s_i, rep_1) < d(s_i, rep_2)$  then
9     |   Add  $s_i$  to node  $A_1$ ;
10    | else
11      |   Add  $s_i$  to node  $A_2$ ;
12    | end
13  | end
14 end
15 Return  $(A_1, rep_1), (A_2, rep_2)$ ;
```

Algorithm 2: minMaxSplit

Data: Node A to be split
Result: Nodes A_1 and A_2 with their representatives rep_1 and rep_2 , respectively

```

1 for each object  $s_i$  of the node  $A$  do
2   | for each object  $s_j \neq s_i$  of the node  $A$  do
3     |   Set  $s_i$  for  $rep_1$  and set  $s_j$  for  $rep_2$ ;
4     |   Create temporary nodes  $A'_1$  and  $A'_2$  with  $rep_1$  and  $rep_2$ , respectively;
5     |   Distribute the objects of the node  $A$  between  $A'_1$  and  $A'_2$ ;
6     |   Save the covering radii of the resulting nodes;
7   | end
8 end
9 Select the pair  $(rep_1, rep_2)$  that minimizes the covering radii of the resulting nodes;
10 Set the new representatives  $rep_1$  for node  $A_1$  and  $rep_2$  for node  $A_2$ ;
11 for each object  $s_i$  of the node  $A$  do
12   | if  $d(s_i, rep_1) < d(s_i, rep_2)$  then
13     |   Add  $s_i$  to node  $A_1$ ;
14     | else
15       |   Add  $s_i$  to node  $A_2$ ;
16     | end
17   | end
18 end
19 Return  $(A_1, rep_1), (A_2, rep_2)$ ;
```

3. **MSTSplit** - With the *MSTSplit* policy, a *Minimal Spanning Tree* of the objects is generated, where the weight of each arc is the distance among the connecting nodes. The longest arc of the tree is removed, obtaining two sub-trees that constitute the new nodes (see the details in Algorithm 3). This is the default method used to build a Slim-tree due to its overall better

performance.

Algorithm 3: MSTSplit

Data: Node A to be split

Result: Nodes A_1 and A_2 with their representative objects rep_1 and rep_2 , respectively

```

1 Create a temporary node  $A'$ ;
2 for each object  $s_i$  of the node  $A$  do
3   | Add object  $s_i$  to  $A'$ 
4 end
5 Perform the MST algorithm on  $A'$ ;
6 Set the new representatives  $rep_1$  for node  $A_1$  and  $rep_2$  for node  $A_2$ ;
7 for each object  $s_i$  of the node  $A$  do
8   | if  $d(s_i, rep_1) < d(s_i, rep_2)$  then
9     | Add  $s_i$  to node  $A_1$ ;
10    | else
11      | Add  $s_i$  to node  $A_2$ ;
12    | end
13  | end
14 end
15 Return  $(A_1, rep_1), (A_2, rep_2)$ ;
```

minOccup policy for *ChooseSubtree* procedure allows to generate nodes with higher occupancy rates, leading to a smaller number of disk accesses. *MST* policy for *SplitNode* procedure is the fastest splitting algorithm among the available policies when the capacity of the nodes C is greater than 20 (TRAINA *et al.*, 2002). Slim-Down algorithm allows producing thinner trees, reducing the number of distance calculations during the searching process. Therefore, when combined, these algorithms allow to Slim-tree to achieve a performance significantly better than the M-tree.

2.6 Fractal Dimension for Similary Search

In this section, we present concepts involving the use of the Fractal theory to allow improving algorithms that perform similarity search.

2.6.1 Fractals and Fractal Dimension

Fractal geometry provides a mathematical model for many complex objects found in the nature (MANDELBROT, 1983; PENTLAND, 1984), such as coastlines, mountains, and clouds. These objects are too complex to possess characteristic sizes and to be described by traditional Euclidean geometry. Self-similarity is an essential property of fractals in the nature and can be quantified by their "Fractal Dimension" (FD).

A set of points is a fractal if it exhibits self-similarity over all scales. This is illustrated in Figure 7, which shows the first few steps in the recursive construction of the so-called "Sierpinski

Figure 7 – Sierpinski triangle: first few steps of its recursive construction.

triangle”. Each small triangle is a miniature replica of the whole triangle. In general, the essence of fractals is this self-similarity property: parts of the fractal are similar to the whole fractal.

There is an infinite family of fractal dimensions, each one making sens in a different situation. Among them, we find the *Hausdorff Fractal Dimension*, the *Information Fractal Dimension* and the *Correlation Fractal Dimension*. For similarity search, the employed fractal dimension FD is commonly associated to the correlation fractal dimension $\mathcal{D}$. Correlation fractal dimension measures the probability that two randomly chosen objects are within a certain distance from each other. Different methods have been proposed to estimate $\mathcal{D}$, such as pair-counting and box-counting. The pair-counting approach involves calculating the number of pairs of elements within a given distance from each other. The average number of neighbors C_r within a given radius r is exactly twice the total number of pairs within the distance r , divided by the number of elements N in the dataset. The box-counting approach (SCHROEDER, 1991) consists of imposing a nested hypercube grids on the data, followed by counting the occupancy of each grid cell, thus focusing on individual elements instead of pairs of elements. The sum of the squared occupancies $S_2(r)$ for a particular grid side length r is defined as $S_2(r) = \sum_i C_i^2$, where C_i is the count of elements in the i th cell. The box-counting approach can be employed to measure $\mathcal{D}$ of any dataset and has been employed in various application fields. The reason for its dominance lies in its simplicity and automatic computability.

The FD has been used as a powerful support for texture analysis and segmentation (CHAUDHURI; SARKAR, 1995; IDA, 1998), shape measurement and classification (NEIL; CURTIS, 1997; BRUNO *et al.*, 2008), attribute selection and dimensionality reduction (BERCHTOLD; BÖHM; KRIEGAL, 1998; PAGEL; KORN; FALOUTSOS, 2000; TRAINA *et al.*, 2010), analysis of spatial access methods (FALOUTSOS; KAMEL, 1994; KAMEL; FALOUTSOS, 1994; BELUSSI; FALOUTSOS, 1995), analysis of metric trees (TRAINA; TRAINA; FALOUTSOS, 2000), indexing (BÖHM; KRIEGEL, 2000), join selectivity estimation (FALOUTSOS *et al.*, 2000) and selectivity estimation for nearest neighbor queries (BERCHTOLD *et al.*, 1997; PAPADOPOULOS; MANOLOPOULOS, 1997; VIEIRA *et al.*, 2007).

Many real datasets are fractals (SCHROEDER, 1991; TRAINA; TRAINA; FALOUTSOS, 2000), i.e., their parts are self-similar. Self-similar datasets, by definition, obey a power law as follows (FALOUTSOS; KAMEL, 1994):

Definition 3. Given a set of n objects in a dataset with a distance function d , the average

Figure 8 – Distance plot of the Sierpinski dataset and its fractal dimension.

number k of neighbors within a given distance r is proportional to r raised to $\mathcal{D}$, where $\mathcal{D}$ is the correlation fractal dimension of the dataset:

$$PC(r) = K_p \times r^{\mathcal{D}} \quad (2.4)$$

where K_p is a proportionality constant.

The graph of the number of pairs of objects within a distance r versus the distance r is called “Distance Plot”. For any dataset associated with a metric that estimates the pairwise distances between its objects, the graph of the distance plot can be drawn, even if the dataset is not in a dimensional domain. Plotting this graph in log-log scale, for the majority of real datasets, results in an almost straight line for a significant range of distances. The slope of the line that best fits the resulting curve in the distance plot is the “Distance Exponent $\mathcal{D}$ ”. Note that, measured in this way, $\mathcal{D}$ closely approximates the theoretical correlation fractal dimension of the dataset. Figure 8 shows the distance plot of the Sierpinski dataset and the resulting fractal dimension $\mathcal{D}$ obtained measuring the slope of the line.

The following observation states a useful property of the distance exponent.

Observation 1. The distance exponent is invariant to sampling, i.e., the power law holds for subsets of the dataset (FALOUTSOS *et al.*, 2000).

Consider a dataset S with n objects and a random sampling rate p ($0 < p < 1$). That is, the sample has $n * p$ objects. Consider an object s from the dataset S and let $s(r)$ be the number of its neighbors within a distance r . Intuitively, after sampling, the object s will have $s(r) * p$ neighbors on the average. Thus, the total number of pairs within distance r will be the original

Figure 9 – An example of the similarity queries performed over a metric index: (a) range query, (b) $k\text{-NN}_q$ query.

$PC(r)$ multiplied by the sampling rate p , on the average. Therefore, the slope of the line in the plot will not change; it will only lower the position of the plot by $\log(p)$.

2.6.2 Fractal Dimension for $k\text{-NN}_q$ Queries

Nearest Neighbor and Range are the most commonly used queries for similarity search. Due to the high computational cost to calculate the distances between the objects in complex domains, similarity queries are usually processed over index structures to accelerate the processing. The queries employ the properties of the index structure to prune the searching space using a limiting radius that is specified beforehand or calculated during the searching process. Range queries are provided with limiting radii to help prune subtrees that do not intersect the query region, which increases their pruning ability (see Figure 9a). A $k\text{-NN}_q$ query starts collecting a list of k objects in the dataset by computing the distances between the query object and all the objects of the dataset. The list is ordered by distances of the objects to the query object. A dynamic radius is first set to the maximum distance between any pair of objects in the dataset, that is, the diameter of the dataset (see Figure 9b). When a nearer object is found, it is added into the list, removing the object with the largest distance to the query object while keeping the list ordered, and thus adjusting (i.e., reducing) the dynamic radius accordingly. The dynamic radius is progressively reduced allowing to dispose of the irrelevant objects initially considered as a potential response. The searching process is stopped when at least k objects are found and no further pruning can be performed. During the searching process, a priority queue is also used to order by distances all

Figure 10 – How to use the Distance Plot to obtain the final radius r_f .

the subtrees to be visited. However, managing the priority queues and the dynamic radius can be very expensive and, consequently, can reduce the performance of the algorithm.

In order to speed up $k\text{-}NN_q$ queries, Vieira et al. (VIEIRA *et al.*, 2007) proposed a new variant, called $k\text{-}NNF_q(s_q, k)$, that estimates a final radius r_f for the $k\text{-}NN_q$ queries, allowing to prune most of the objects that are not likely to be relevant. $k\text{-}NNF_q(s_q, k)$ query performs in three steps:

1. Estimate a limiting radius for the $k\text{-}NN_q$ query,
2. Perform a composed Range and $k\text{-}NN_q$ queries,
3. If the required number k of objects were not obtained, refine the searching procedure.

In the first step of $k\text{-}NNF_q$ algorithm, the final radius r_f is estimated using the distance plot $\mathcal{D}$ of the dataset and a point $Point_0 = \langle \log(R), \log(Pairs(N)) \rangle$. Figure 10 illustrates how to use the distance plot to estimate the final radius r_f . It is obtained by converting the number k of objects into the corresponding covering radius as follows:

$$r_f = R \times e^{(\log(k-1) - \log(N-1)) / \mathcal{D}} \quad (2.5)$$

where R is the diameter of the dataset, N is the total number of objects in the dataset and $\mathcal{D}$ is the fractal dimension of the dataset. The diameter R can be obtained from the index structure as the covering radius of the root node in the tree.

In the second step, a composed Range and $k\text{-}NN_q$ queries, called $k\text{AndRange}$, is performed.

Figure 11 – How to use the Distance Plot to obtain the local final radius r'_f .

Definition 4. In a metric space $(\mathbb{S}, d)$, given a query center $s_q \in \mathbb{S}$, an integer $k > 0$ and a range $r_f \in \mathbb{R}^+$, a $kAndRange(s_q, k, r_f)$ query retrieves at most k objects $s_i \in \mathbb{S} \mid d(s_i, s_q) \leq r_f$, such that for every non retrieved object s_j , $d(s_j, s_q) > d(s_i, s_q)$ for any retrieved objects s_i .

The radius r_f estimated in equation 2.5 is only an approximation as the exact radius depends on the local density of the objects around the query center s_q . Therefore, when r_f is underestimated, $kAndRange(s_q, k, r_f)$ algorithm will return fewer objects than required, that is, a quantity $k' < k$, and another call to the $kAndRange(s_q, k, r_f)$ algorithm with a larger radius is required.

The third step of $k-NNF_q(s_q, k)$ consists in a local estimation of the final radius r'_f , where $r'_f > r_f$, using the local density of the objects around s_q . Figure 11 illustrates how to use the distance plot to estimate the local final radius r'_f . This time, the local radius r'_f is estimated using the distance plot $\mathcal{D}$ of the dataset and a point $Point_1 = \langle \log(r_f), \log(Pairs(k')) \rangle$ as follows:

$$r'_f = r_f \times e^{(\log(k-1) - \log(k'-1)) / \mathcal{D}} \quad (2.6)$$

Once the local radius r'_f is calculated, another query, called $kRingRange$, is performed to retrieve the remaining $k - k'$ objects. The reason is that, calling $kAndRange$ query again will perform all the search again and consequently reduce the overall performance.

Definition 5. In a metric space $(\mathbb{S}, d)$, given a query center $s_q \in \mathbb{S}$, the inner radius r_f , the outer radius r'_f and the maximum number of objects $k - k'$ to be retrieved, the $kRingRange(s_q, k - k', r_f, r'_f)$ algorithm searches the dataset to find at most $k - k'$ objects s_i in the ring centered at s_q , so that $r_f < d(s_q, s_i) \leq r'_f$.

The $kRingRange$ algorithm is basically a $kAndRange$ algorithm modified to prune not only objects and nodes that are outside the region limited by the outer radius r'_f , but also the nodes

Figure 12 – Illustration of $kAndRange()$ and $kRingRange()$ algorithms.

and objects that are inside the region limited by the inner radius r_f (see Figure 12). The first call to $kRingRange$ algorithm retrieves k'' objects, where $k' \leq k'' \leq k$. If $k'' < k$, the point $Point_2 = \langle \log(r'_f), \log(Pairs(k'')) \rangle$ can be used to estimate another radius r''_f , and a another call to $kRingRange(s_q, k - k'', r'_f, r''_f)$ algorithm is performed. This last step is repeated until the number k of objects is retrieved. Notice that, the efficiency of the $k-NNF_q$ query depends mainly on the number of times $kRingRange$ algorithm is called. But, if the estimated final radius is sufficiently accurate, one call of the $kAndRange$ algorithm will be sufficient to retrieve the k objects. Algorithm 4 shows the different steps of the $k-NNF_q$ query.

Algorithm 4: $k-NNF_q(s_q, k)$

Data: A query center s_q and an integer number k

Result: The list L of k objects $\langle Oid_i, d(s_q, s_i) \rangle$

- 1 Obtain N as the number of elements in the dataset;
 - 2 Obtain R as the diameter of the dataset indexed;
 - 3 Clear list L ;
 - 4 Estimate the final radius r_f ;
 - 5 Process $KAndRange(s_q, k, r_f)$, store the answer in L , and set k' as the number of retrieved elements;
 - 6 **while** $k' < k$ and $r_f < R$ **do**
 - 7 Estimate the local final radius r'_f ;
 - 8 Execute $kRingRange(s_q, k - k', r_f, r'_f)$, store the answer in L , and set k' as the number of retrieved elements;
 - 9 Set $r_f = r'_f$;
 - 10 **end**
 - 11 Return L ;
-

2.7 Final Considerations

Similarity search has gained a lot of popularity in recent years which resulted in a lot of research in the area. Metric access methods are fundamental to process similarity queries over complex and large datasets. In this chapter we presented an overview of the concepts related to similarity search in metric spaces. We provided a definition of the metric spaces and the similarity queries. We discussed the performance parameters of the similarity queries and we presented a classification of the metric access methods based on the partitioning techniques. We also introduced the fractal dimension concept, a powerful tool that has proven to offer a valuable support to speed up similarity queries. Finally, we showed how the correlation fractal dimension can be employed to accelerate k - NN_q queries in metric spaces.

All the concepts presented in this chapter are valid only over datasets of complete objects. The next chapter describes the concepts related to missing data and discusses the most relevant missing data treatment methods available in the literature.

MISSING DATA AND RELATED WORK

3.1 Introduction

In real life, available knowledge is often imperfect in the sense that it represents different possible states of the external world, for which the state corresponding to the actual situation of the world is unknown. Imperfect knowledge can be of different categories ([GRZYMALA-BUSSE; HU, 2001](#)): *erroneous, incomplete, uncertain or vague*.

Incomplete data, i.e., data with missing values, occur in a wide range of research and industry domains, such as medicine, computational biology, data mining, knowledge discovery tasks, surveys, etc. One form of data incompleteness is missing attribute values, i.e., values on some attributes are unknown. In real-world datasets, missing attribute values are very common. This may arise from a number of factors, such as equipment malfunction, data collecting or recording errors, absent or redundant diagnose tests, participants failing to answer questions intentionally or unintentionally, subjects withdrawal from studies before they are completed, etc.

The main interest in missing data first appeared in the problems caused in surveys and census data ([RUBIN, 1976](#); [SCHAFER, 1997](#); [LITTLE; RUBIN, 2002](#); [RUBIN, 2004](#)). Modern scientific and engineering applications generated large amounts of digital data. These data are growing not only in the number of objects and attributes, but also in the complexity of the attributes that describe them. Thus, it is not uncommon to have missing values occurring in some of the attributes. This type of scenario, though inevitable, is unintended and uncontrolled by researchers. The fact is that, most of the existing applications assume (or require) complete data since they are not designed to perform on a partially specified data. Consequently, the performance of the algorithms that implement such applications break down with the increasing amount of missing data. This fact creates new challenges for researchers since the underlying applications need to be redesigned to support data with missing values or a special treatment of missing data should be incorporated to maintain their functionality. Some of these applications

include biometrics, medical diagnosis, financial index prediction, signal processing, pattern recognition, weather forecasting, earthquake prediction, etc. In the last two decades, the problem of missing data has attracted more interest and motivated the development of techniques for missing data treatment (OOI; GOH; TAN, 1998; AGGARWAL; PARTHASARATHY, 2001; JONSSON; WOHLIN, 2004; CANAHUATE; GIBAS; FERHATOSMANOGLU, 2006; HULSE, 2007; HIMMELSPACH; CONRAD, 2010; EIROLA *et al.*, 2013).

There are many approaches to deal with missing attribute values. One common approach is to remove all the data with missing attribute values and operate only on the complete data. However, this does not fully preserve the characteristics of the original data, especially when a large amount of attribute values is missing. Therefore, it is important to understand the background of missingness and its context, because it helps to find the best strategy to deal with the issues of retrieving data with missing values.

Notice that, our goal is to understand and provide practical techniques that allow the retrieval of data with missing values in database management systems. We are not interested in "correcting" nor inferring the missing values. There are many statistical techniques that applications can employ to estimate missing values, many of them based on "completing" the missing data, such as regression, k nearest neighbors voting and other inference tools. However, applying any of such techniques concerns the application software, not the DBMS. Therefore, our goal is to deal with missing data so that the retrieval of complete data is not hampered by incomplete data and that data with significant amount of missing values can be retrieved in order to allow the application to process them.

This chapter presents a description of the main concepts related to missing data and how they can affect the retrieval of data in a DBMS, including identifying the most common reasons of missingness and the practical issues they cause. The first section describes the theory of missing data. A definition of missing data and the main characteristics that describe them are presented. A classification of the types of missing data is also provided with examples to illustrate each type. The second section provides a review of the methods for dealing with missing data. The last section concludes the chapter.

3.2 Missing Data Theory

3.2.1 Missing Data

Missing data occur in different situations for different reasons. At the initial stage of a new database, not all the data are supplied, then we have a partially filled database. Then, new data may be introduced, updating the partially filled database. For privacy or security reasons, it is also possible that data concerning some elements of a domain or attribute are not specified in the database. Moreover, some data may be missing because it is expensive, difficult, or not ethical to obtain.

There are two forms of missing data (ALLISON, 2001): (i) Data can be missing for some but not all attributes of a given element. (ii) Data can be missing for some but not all elements of a given attribute. If data are missing for all the elements of an attribute, the attribute is said to be *latent* or *unobserved*. On the other hand, if data are missing for all the attributes of an element, then we have a *unit non-response* or *missing record*. While missing records may only be problematic in restricted situations (e.g., smaller size of the sample provides lower statistical power of the data), missing attribute values can severely affect the quality of data and the performance of tasks that operate on such data. Figure 13 illustrates an example of a table where columns represent the attributes and rows represent the data objects. In Figure 13a, all the attributes are observed for all the objects. In Figure 13b, some attribute values are missing for all the data objects, where attribute A_2 is a latent attribute and object Obj_3 is a missing record.

	A_1	A_2	A_3	A_4	A_5
Obj_1					
Obj_2					
Obj_3					
Obj_4					
Obj_5					

(a)

	A_1	A_2	A_3	A_4	A_5
Obj_1		?		?	
Obj_2	?	?			
Obj_3	?	?	?	?	?
Obj_4		?			
Obj_5		?			?

(b)

Figure 13 – (a) All attribute values are observed for all the objects. (b) Some attribute values are missing for all the objects.

In relational databases, the main concept used for modeling missing data is the NULL values (IMIELINSKI; JR., 1984). A NULL value is a place holder for an attribute whose value cannot be represented by an ordinary value. Usually, programming environments provide explicit codes for missing data that aim at indicating missing values. Yet, these codes are not standardized and different codes are used according to different situations. The lack of standardization is one of the leading causes of the problem of *disguised missing data* (PEARSON, 2006). The latter problem arises when missing values are not explicitly identified, but are coded with values that can be confused with valid values.

3.2.2 Legitimate or Illegitimate Missing Data

Legitimate missing data are unobserved data where it is appropriate to be unobserved (OSBORNE, 2012). Legitimate missingness occurs when it is not possible to get the value of an attribute because the object has no suitable value for it. For example, when filling out a survey that asks for a marital status and if married then for how long. If the respondent is not married, it is legitimate to skip the question on how long he/she has been married. Legitimate missing

attribute value is also called *not applicable* or *empty value*. *Illegitimate missing data*, or simply missing data, are an unobserved data where it is not appropriate to be unobserved. Illegitimate missingness occurs when an attribute value is missing, but it is possible to get this value in principle, because it is actually *applicable*. Illegitimate missing data are more common and appear in all types of research. For instance, because of sensor malfunction, battery levels got flatten or resolution is insufficient, data will not be recorded until the sensor problem is salved. Therefore, illegitimate missingness is mainly the one that most of researches are concerned with, because it has the potential of bias.

3.2.3 Missing Data Mechanisms

An important consideration when dealing with missing data is the missingness mechanism, i.e., the reason data are missing (OSBORNE, 2012). Several classifications of missingness have been proposed in the literature, most of them discussing the possible assumptions that can influence the strategy for dealing with missing data. Rubin (RUBIN, 1976) defined three mechanisms of missingness:

Let a variable Y be a subject of missing values, and let I be an indicator variable which takes the value 1 if Y is missing (unobserved), or 0 if Y is observed. We can write Y as $Y = (Y_{obs}, Y_{miss})$.

Missing Completely At Random - MCAR: It is the strongest assumption that is commonly made and satisfies the highest level of randomness. It holds when the probability that data are missing is independent of both observed and missing data as follows:

$$P(I = 1|Y) = P(I = 1) \quad (3.1)$$

At this level of randomness, any missing data treatment method can be applied without risk of introducing bias to the data.

Missing At Random - MAR: It is a weaker assumption but more prevalent than MCAR. It holds when the probability that data are missing is independent of missing data, but may be missing as a function of observed data as follows (see also the Definition 7 in Chapter 4):

$$P(I = 1|Y) = P(I = 1|Y_{obs}) \quad (3.2)$$

Note that MCAR is a special case of MAR, that is, if the data are MCAR they are also MAR.

Missing Not At Random - MNAR: If the data are not at least MAR, the data are said to be missing not at random. This assumption occurs when data are missing as a function of missing values as follows (see also the Definition 8 in Chapter 4):

$$P(I = 1|Y) = P(I = 1|Y_{miss}) \quad (3.3)$$

Table 1 – List of patients with their age and the result of a medical test (MAGNANI, 2004).

Patient ID	Age	Test Result			
		Complete	MCAR	MAR	MNAR
1	23	1453	1453	1453	1453
2	23	1354	NULL	NULL	1354
3	23	2134	2134	2134	NULL
4	23	2043	NULL	NULL	NULL
5	75	1324	1324	1324	1324
6	75	1324	NULL	1324	1324
7	75	2054	2054	2054	NULL
8	75	2056	NULL	2056	NULL

Table 1 (MAGNANI, 2004) illustrates a complete example that shows the different missing data mechanisms. It represents a database of patients with their age and the result of a medical test. Assuming that the test is expensive, it is performed on a few randomly selected patients. In this case, MCAR holds because missingness in the test result is independent of both missing and observed values. Notice that $Pr(\text{TestResult} = \text{Null} | \text{Age} = 23) = Pr(\text{TestResult} = \text{Null} | \text{Age} = 75)$ and $Pr(\text{TestResult} = \text{Null} | \text{TestResult}) = Pr(\text{TestResult} = \text{Null})$. Alternatively, suppose that the test is performed mainly on old people. This time, data are missing as a function of the attribute age, therefore, MAR assumption holds. In particular, $Pr(\text{TestResult} = \text{Null} | \text{Age} = 23) = 0.5$, while $Pr(\text{TestResult} = \text{Null} | \text{Age} = 75) = 0.0$. In another situation, suppose that the device used to perform the test fails to detect high values. In this case, data are missing as a function of the values of the attribute that has missing values, thus, MNAR holds. Formally, $Pr(\text{TestResult} = \text{Null} | \text{TestResult} < 2000) = 0.0$, while $Pr(\text{TestResult} = \text{Null} | \text{TestResult} > 2000) = 1.0$. Notice that without external knowledge about the state of the device used to perform the test, this data is not statistically different from the case where data are missing completely at random (MCAR), because $Pr(\text{TestResult} = \text{Null} | \text{Age} = 23) = Pr(\text{TestResult} = \text{Null} | \text{Age} = 75)$ and $Pr(\text{TestResult} = \text{Null} | \text{Age}) = Pr(\text{TestResult} = \text{Null})$. Consequently, if we do not know the underlying dependency, which is usually difficult to get, it may be very difficult to analyze the data.

3.2.4 Ignorable and Non-ignorable Missingness

Another classification of missingness is the *ignorable missingness* and *non-ignorable missingness* (SCHAFER, 1997). Ignorable missingness applies to MCAR and MAR mechanisms and non-ignorable missingness is often correlated to MNAR mechanism. If the mechanism of missingness is ignorable (i.e., the process that causes missing data can be safely ignored), then a model for missingness is not required. Nevertheless, it does not mean that the problem of missing data can be completely ignored. Instead, a special treatment is required, but without external knowledge about the reason data are missing. On the other hand, when data are MNAR, the process that controls missingness is not ignorable. Therefore, missingness needs to be modeled as part of the

missing data treatment. However, modeling missingness when data are not missing at random is a very difficult task. The reason is that an external knowledge about the process that caused missing data is required, which is in most cases impossible to acquire.

3.2.5 Consequences of MCAR, MAR and MNAR

The main consequence of MCAR data is the loss of statistical power. In fact, under MCAR assumption, a sub-sample of elements with complete data is equivalent to a simple random sample, and it is well-known that simple random sampling does not cause bias (i.e., tendency or skew). If data are MAR but not MCAR, analyzes also do not cause bias in the parameter estimation when the cause of missingness is taken into account. For example, to estimate the mean income for some population, in the sample 85% of women reported their income but only 60% of men reported their income (a violation of MCAR). The missing data fall in the MAR category. Assuming that on average men earn more than women, at first it would appear that missingness on income is related to the value of the income itself. But, if within each gender, missingness on income does not actually depend on the value of the income, the data would still be MAR. Now, if it appears that the conditional probability of missingness is related to the value of income within each gender (i.e., people who earn more are less likely to report their income), then the missing data fall in the MNAR category, and the estimated average income will be probably lower than the reality. Therefore, the analysis would yield a biased estimate of the average income for the whole population. If people with high income are in fact less likely to report their income than people with low income, an equation (a model) could presumably be able to predict missingness on the basis of income. Then, such an equation could be incorporated into a more complex model for estimating missing values. However, it is very difficult to develop a missingness model because it is application-dependent.

3.3 Missing Data Treatment

The treatment of missing data is important to assess the data quality, especially when datasets contain many missing values. There are many types of studies for which missing data are an issue. Major interest has long been focused on the problems caused in surveys and census data and many solutions have been reported in the statistics literature. In the context of the relational theory, handling incomplete databases has also been an active research field (IMIELINSKI; JR., 1984; REITER, 1984; REITER, 1986; ZIMANYI, 1992; SOLIMAN; ILYAS; BEN-DAVID, 2010). There are two categories of missing data treatment, treatment at the level of data and treatment at the level of the index structure.

3.3.1 Treatment at the Data Level

Missing data treatment at the data level aims at constructing a complete dataset from an incomplete dataset, so that traditional applications, such as similarity search, data mining, knowledge discovery, etc, can be directly applied. It is considered to be a preprocessing step for analysis and it is often associated to data cleaning and data preparation.

Any missing data solution should satisfy the following rules (PENG; LEI, 2005):

- **Estimation without bias:** The solution should not change the data distribution,
- **Correlation:** The solution should maintain the correlation among the attributes,
- **Cost:** The solution should be simple and amenable for implementation.

Missing data treatment at the data level can be divided into the following three approaches:

3.3.1.1 Case Deletion

Traditional approaches for dealing with missing data essentially include *Case Deletion* (OSBORNE, 2012). It consists of deleting cases with missing values and performing the analysis on complete cases only. This method, known as *casewise deletion* or *listwise deletion*, performs by deleting all the elements (cases) with missing data. Another variant consists of deleting the attributes with high rates of missing data, however, before deleting any attribute, it is necessary to run relevance analysis, especially on the attributes with high amounts of missing data. The main advantage of this approach is simplicity. Thus, if a missing data problem can be salvaged by discarding a small sample of the data, then the method is fairly effective. However, even in such situation, one must explore the data to make sure that the discarded data are not influential, otherwise it can lead to an important loss of relevant information.

3.3.1.2 Missing Data Imputation

Imputation approach aims at estimating missing values using relationships that can be identified in the observed values of the dataset (ALLISON, 2001). Several methods of imputation have been proposed in the literature, ranging from the most simple *Mean or Mode Substitution* (OSBORNE, 2012) to the most sophisticated *Multiple Imputation* (OSBORNE, 2012). Mean or mode substitution is one of the most frequently used method. It consists of replacing the missing data for a given attribute by the mean (quantitative attribute) or mode (qualitative attribute) of all the observed values of that attribute. This method is very optimal in terms of computational cost, but suffers from producing inaccurate imputations, especially when data are not MCAR.

Hot Deck Imputation (PEARSON, 2006) employs the estimated distribution of the missing value from the dataset to estimate a missing attribute value. It performs in two steps. In the first step, the data are partitioned into clusters. In the second step, each element with missing

values is associated to one cluster then the complete elements in the cluster are used to fill in the missing values. This can be done by calculating the mean or mode of the attribute within a cluster.

Multiple Imputation (OSBORNE, 2012) provides a more sophisticated data imputation. In fact, instead of filling in a single value for each missing value, multiple imputation procedure replaces each missing value with a set of plausible values drawn from their predictive distributions. Missing values are imputed k times to produce k complete datasets then a number of simple rules provided in (LITTLE; RUBIN, 2002) are used to produce overall estimates and standard errors that reflect missing data uncertainty. Multiple imputation method has proven to be very powerful in a variety of applications, as it achieves unbiased results and accurate estimates. However, due to its iterative nature, it is only effective after a sufficiently high number of iterations which, in the context of Machine Learning, becomes impractical as training and analyzing sophisticated models tend to be computationally expensive.

k-Nearest Neighbors Imputation is the most popular method in Machine Learning, Knowledge Discovery and Data Mining (KDD). It consists of finding among the complete elements the k -nearest neighbors of an incomplete element (JONSSON; WOHLIN, 2004). Then, the missing value of a given attribute in this element is replaced with the average value of its neighbors on the same attribute. The performance of this strategy is very limited when the proportion of missing data is high, and when only a few elements are complete. A simple improvement consists in looking also for incomplete neighbors of the element, knowing that these neighbors have observed values for the missing attribute in the element (HULSE, 2007). This method, called *Incomplete Case k-Nearest Neighbors Imputation* (ICkNNI), still fails in high-dimensional datasets with large proportions of missing data or when distances cannot be faithfully computed.

Imputation-based approaches have two main disadvantages:

1. Additional information about missing data mechanism is often required to choose the most appropriate technique for missing data treatment. Obtaining this information is not always straightforward as it requires sophisticated techniques for data analysis.
2. In many situations, some dimensions have missing values because they are not important for certain objects. In this case, the data are not considered missing, instead, they are irrelevant for such objects and any imputation approach would become inappropriate.

3.3.1.3 Model-based Imputation

This approach consists of modeling the distribution of data by means of a procedure called *Maximum Likelihood* (ALLISON, 2001). Maximum likelihood uses the *Expectation Maximization* algorithm to estimate parameters of the probability density function of an incomplete sample. Provided a sample (set of elements) with a likelihood that it has been drawn by a given distribu-

tion, the expectation maximization algorithm tries to find the parameters of the distribution that would have more likely produced the observed sample.

3.3.2 Treatment at the Index Level

The object of missing data treatment at the index level is to incorporate into the underlying indexing technique a special method for missing data that do not require to fill in the missing values. This treatment is specialized for every application, therefore, it is important to take into account the nature of the tasks that are indented to be performed on the data.

The missing data problem is as important for similarity search as it is for Statistics, Data Mining and Machine Learning. Similarity search often uses an access method to support data retrieval over high-dimensional datasets. Traditionally, this is accomplished by using multidimensional or distance-based indexing techniques. Unfortunately, these access methods do not perform well when datasets are incomplete (OOI; GOH; TAN, 1998; CANAHUATE; GIBAS; FERHATOSMANOGLU, 2006), as the performance of the underlying indexing techniques can severely deteriorate to a point where even an exhaustive search in the database can yield better performance.

Although, similarity search in high-dimensional datasets has been extensively investigated in the literature; only few techniques have been proposed to support similarity queries over incomplete datasets. These techniques aim to efficiently index datasets with missing attribute values and perform similarity queries based on the available data rather than estimating missing values.

3.3.2.1 Conventional Method

The basic approach to extend multidimensional access methods to support similarity queries over incomplete datasets is to use an indicator that identifies the missing values, and employ a traditional indexing technique, like R-tree (GUTTMAN, 1984), to index the dataset. One of the issues of this approach is that all the objects with missing values on a given dimension will be projected into the same value of the hyper-region, causing skew in the indexed space. And, if the proportion of missing values is large, it will result in a highly skewed data space. In addition, when a query that involves k attributes is issued, the query is decomposed into 2^k sub-queries. For example, given a query that involves 3 attributes (1,2,3), all the following objects (1,2,3), (? ,2,3), (1, ? ,3), (1,2, ?), (? , ? ,3), (? ,2, ?), (1, ? , ?), (? , ? , ?) are a potential response to the query. This is achieved by processing sub-queries with all the combinations of missing and observed values in the attributes. This strategy is very simple and easy for implementation. Its main drawback is that the searching space grows exponentially with the growing dimensions of the data, causing poor query performance.

3.3.2.2 BR-tree

The Bit-string augmented R-tree (OOI; GOH; TAN, 1998), called BR-tree, provides a mapping function that assigns distinct indicators to the missing values in order to avoid skew in the indexed space. Thus, all the objects with missing values are randomly scattered in the indexed space. However, to perform a query that involves k attributes, it is necessary to be transformed to 2^k sub-queries, corresponding to all the possible combinations of missing and observed values, to find the matching objects. As a result, this technique suffers from a poor query performance since it requires performing an exponential number of sub-queries.

3.3.2.3 MOSAIC

Multiple One-dimensional Single-Attribute Indexes (OOI; GOH; TAN, 1998), called MOSAIC, is a set of B^+ -trees where every dimension is indexed independently. Similarly to BR-tree, missing data are mapped to distinct values and objects are randomly scattered in the indexed space. Given a query that involves k attributes, for each dimension, there are two possible combinations for the objects that can match the query (i.e., observed or missing) and the result is the concatenation of the k sub-results, producing $2k$ potential responses. The main advantage of this technique is that it requires only $2k$ sub-queries to perform a query with k attributes. However, it is not appropriate for high-dimensional data since every dimension requires an independent index structure.

3.3.2.4 Extended Bitmaps

The Extended Bitmaps indexing technique (CANAHUATE; GIBAS; FERHATOSMANOGLU, 2006) was proposed to index and search incomplete databases when missingness is not ignorable. For Bitmaps indexes, each attribute acquires a bitmap (i.e., a column-wise representation of each position in the bit-string) for the attribute values. With the Extended Bitmaps, an extra bitmap is used for each attribute to denote missing values of the objects on that attribute. Table 2 (CANAHUATE; GIBAS; FERHATOSMANOGLU, 2006) shows an example of a Bitmap index, where a bit $B_{i,j}[s] = 1$ if the object s has a value j for the attribute A_i . $B_{i,0}$ is the indicator of missing values, where $B_{i,0}[s] = 1$ if the object s has a NULL value for the attribute A_i . In Table 3, the value of a Bitmap $B_{i,j}$ indicates all the objects that have a value j for the attribute A_i . With this approach, queries are performed using union and intersection operations over the Bitmaps, allowing to choose the objects with missing values to be included in the query response.

3.3.2.5 Extended VA-Files

The Extended VA-Files indexing technique (CANAHUATE; GIBAS; FERHATOSMANOGLU, 2006) is similar to the Extended Bitmaps. However, VA-Files employ bit-strings to encode attribute values. For each attribute A_i in the database, bit-string of b_i bits are used to represent 2^{b_i} bins that enclose the entire attribute domain. The Extended VA-Files use an extra bit-string

Table 2 – Bitmaps encoding with missing data (CANAHUATE; GIBAS; FERHATOSMANOGLU, 2006).

Object	Value	$B_{i,0}$	$B_{i,1}$	$B_{i,2}$	$B_{i,3}$	$B_{i,4}$	$B_{i,5}$
1	5	0	0	0	0	0	1
2	2	0	0	1	0	0	0
3	3	0	0	0	1	0	0
4	NULL	1	0	0	0	0	0
5	4	0	0	0	0	1	0
6	5	0	0	0	0	0	1
7	1	0	1	0	0	0	0
8	3	0	0	0	1	0	0
9	NULL	1	0	0	0	0	0
10	2	0	0	1	0	0	0

Table 3 – Bitmaps index (CANAHUATE; GIBAS; FERHATOSMANOGLU, 2006).

Bitmap Vector	Value
$B_{i,0}$	0001000010
$B_{i,1}$	0000001000
$B_{i,2}$	0100000001
$B_{i,3}$	0010000100
$B_{i,4}$	0000100000
$B_{i,5}$	1000010000

Table 4 – Database with missing values and VA-Files representation (CANAHUATE; GIBAS; FERHATOSMANOGLU, 2006).

Object	Value	VA-Files Representation
1	5	11
2	1	01
3	3	10
4	NULL	00

of 0's for each attribute A_i to indicate the objects that have missing values on that attribute (see the last column in the Table 4). Similarly to the Extended Bitmaps, queries are performed on a basis that allows to choose the objects with missing values to be included in the query response.

Table 5 summarizes the techniques for missing data treatment at the index level and their characteristics. Compared to the BR-tree and MOSAIC, the benefit of the Extended Bitmaps and Extended VA-Files is that they are not hierarchical and exhibit a linear performance with respect to the dimensionality of the dataset. Also, the fact that each dimension is indexed independently and searched separately, there is no need to transform the query into an exponential number of sub-queries. However, the possibility that some dimensions could exhibit a high selectivity cannot be exploited to restrict the search in the remaining ones. Therefore, despite the deterioration of the hierarchical multidimensional indexes due to missingness, these techniques perform better on complete data. This fact motivates the development of better solutions for the hierarchical multidimensional access methods that allow to support incomplete datasets, while preserving

Table 5 – Summary of the techniques for missing data treatment at the index level.

Aspect \ Method		Conventional	BR-tree	MOSAIC	Bitmaps	VA-files
Number of Sub-queries	1				X	X
	2k			X		
	2^k	X	X			
Scalability					X	X
Data Skew		X				
Linear Performance					X	X
Hierarchical Multidimensional		X	X			

their efficiency and effectiveness.

Metric access methods are more appropriate to support similarity search over non-dimensional and high-dimensional datasets. However, when the concept of similarity search is modeled using only a distance function, the problem of missing data increases. Specifically, the problem of similarity search is to find the object that is the nearest to a single object under some distance measure from the dataset. But, when objects have missing values, the distances are undefined. Previous works aiming to directly estimate the distances between objects with missing values were presented in (DIXON, 1979)(CLEARY; TRIGG, 1995)(HIMMELSPACH; CONRAD, 2010)(EIROLA *et al.*, 2013). These techniques were developed for different purposes, such as, feature selection and data clustering. To the best of our knowledge, among the methods published in the literature to handle missing data for similarity search, no one was designed for metric access methods to support similarity queries over high-dimensional incomplete datasets.

3.4 Final Considerations

In this chapter we presented an overview of the concepts related to missing data that are the basis for the development of this Doctoral dissertation. We provided a definition of missing data and classifications of missingness based on the degree of randomness and based on the ignorability of missing data processes. We also provided a classification of the approaches for missing data treatment and we analyzed some of the well-known methods found in the literature. Most of the methods perform at the data level and aim at constructing complete datasets. Some of them are simple and amenable for implementation but are potentially biased, like Mean or Mode Substitution. Some others, such as Multiple Imputation and Expectation Maximization, are very powerful and accurate, but very expensive to apply in real-world applications. Missing data treatment should not always consider estimating, predicting, or recovering missing values. In many cases, this is not even appropriate because missingness is legitimate. Therefore, it is

important to develop techniques that can be incorporated into the underlying access methods to support data with missing values.

THEORY OF MISSING DATA IN METRIC SPACES

4.1 Introduction

Advanced database applications often deal with imperfect data, i.e., erroneous or incomplete. Such applications usually involve large datasets of both non-dimensional and high-dimensional data objects, therefore, there is always a possibility of having erroneous or missing data. In particular, as a time series is a sequence of real-valued measurements continuously recorded by a sensor over a finite interval of time, failing to record a single value may invalidate the whole time series, so they are especially sensitive to missing data. Recent advances in mobile computing and sensor technologies have permitted to collect tremendous amounts of time series data. There is a variety of application domains where time series analysis is particularly useful, such as signal processing, pattern recognition, statistics, mathematical finance, weather forecasting and climate change monitoring, earthquake prediction, among others. However, due to transient interruptions in sensor readings caused by incorrect hardware design, improper calibration, lack of resolution or by low battery levels, some parts of the time series may remain unrecorded or be wrongly evaluated.

Important operations over both non-dimensional and high-dimensional data include the similarity search. It involves finding objects that are similar to a given query object based on some similarity measure. Metric access methods are known to be more suitable to store and retrieve sets of non-dimensional and high-dimensional data. There are a variety of proposals for metric access methods, however, nearly all of them are designed to operate on complete datasets.

Multimedia data usually require preprocessing procedures to extract relevant features in the form of feature vectors, which are used in the place of the raw data to perform similarity queries. In some cases, missing attribute values occur in the feature vectors during the feature extraction process. For example, when extracting the gray-level histogram from a set of satellite

Figure 14 – Image of the moon and its gray-level histogram: (a) the histogram is complete, (b) the histogram is incomplete.

images, the brighter values of the histogram may be missing due to sensor saturation with very high energy levels. Figure 14 shows an example of a dark image with a bright moon in the center and its associated gray-level histogram. In Figure 14a, the histogram is complete but in Figure 14b, the values of the histogram corresponding to the bright part of the moon are missing due to failure of the sensor to record data at high energy levels. Such data cannot be indexed in a metric space. However, this problem can considerably reduce the size of the dataset and, consequently, affect the query performance. Therefore, a special treatment for missing data is required to enable the indexing process.

On the other hand, missing attribute values can occur in the raw data during the collecting process, and two situations arise: (i) The raw data are directly indexed and the indexing process fails to operate on such data, because the distances are undefined. (ii) The raw data require a feature extraction procedure before the indexing process, but missing values fade out in the feature vectors during the feature extraction process. An example that illustrates the second situation concerns the time series data. When recording time series, some objects may lack values on certain attributes, but after they are submitted to a transform, such as Fourier or Wavelets, the missing values fade out in the resulting transforms (i.e., feature vectors), but turn into energy loss in the overall time series. Figure 15 shows an example of a time series submitted to the Discrete Wavelet Transform (DWT) (MALLAT, 1989) for feature extraction and the reconstructed signal from the transform using the Inverse Discrete Wavelet Transform (IDWT) (MALLAT, 1989). In

Figure 15 – NDVI time series submitted to the Discrete Wavelet Transform and its reconstruction: (a) the signal is complete, (b) the signal has 10% missing values.

Figure 15a where the signal is complete, the reconstruction process achieves a nearly perfect signal which is very close to the original one. However, in Figure 15b where the original signal contain 10% of missing values, the reconstructed signal is complete, but looks different from the original one. Note that, the resulting transform (i.e., feature vector) indexed in the place of the original signal is complete, however, it does not sufficiently represent the original signal. This is due to loss of energy during the feature extraction process caused by missing values. This fact enables metric access methods to index objects with missing values, because the distances now are well defined. Nevertheless, any potential of skew caused by missingness in the raw data remains in the feature vectors, which can reduce the representative power of these latter and affect the ability to correctly identify the objects. Consequently, the performance of the metric access methods can seriously deteriorate, leading the similarity queries to retrieve non-relevant objects and missing the relevant ones.

In this chapter we focus on the issues of indexing and searching datasets with missing attribute values in metric spaces and we provide a framework to study and evaluate the impact of the different mechanisms of missingness on metric access methods. The main contributions include the following:

1. Discussing the problems involved with metric access methods when applied on incomplete datasets, including illustrative examples.

2. Formalizing the problem of missing data in metric spaces and demonstrating that missing data not at random can yield skew in the metric spaces.
3. Developing a model of missingness depicting the behavior of missing data mechanisms in metric spaces.
4. Conducting a set of experiments to evaluate the performance of metric access methods when applied to incomplete datasets, and examine the effects of each mechanism of missingness, in order to reinforce our theoretical basis.

The structure of this chapter is as follows. Section 4.2 presents a formal model of missing data that provides an understanding of the effect of bias caused by missingness in metric spaces. Section 4.3 provides a set of experiments to evaluate the effect of missingness on the distance distribution in the metric spaces, evaluating both MAR and MNAR mechanisms. Section 4.4 summarizes the results.

4.2 Formal Model for Missing Data in Metric Spaces

Large datasets with missing attribute values are becoming more common in many applications. In these applications, similarity search over data with missing attribute values introduces new challenges that are fundamentally different from those of similarity search over complete data. Specifically, metric access methods were developed to speed up similarity queries and improve the performance of the similarity search over high-dimensional data and metric data. The major research problems are how to minimize the number of distance computations and how to minimize the I/O cost (disk accesses). Numerous successful algorithms for indexing and retrieving large and complex datasets were proposed in the literature. However, these algorithms are impractical when data have missing values. This limitation was previously discussed in Chapter 1.

4.2.1 Bias of Parameter Estimation

Despite the fact that missing values do not always implicate a loss of information (such as when missingness is legitimate), missing attribute values can represent a serious problem, especially when data are not missing at random. In fact, MNAR data produce biased estimates of the parameters, such as the variance and the mean of the attributes with missing values.

Definition 6. (Bias of the Parameter Estimation) Bias, also called skew, of a parameter estimation α , is the gap between the expected estimation of α and its real measurement. It occurs when α is underestimated or overestimated within a certain range ϵ .

Table 6 shows an example of a biased parameter caused by MNAR data. It represents a database of employees with their age and their salaries. Assuming that people who are over

50 did not report their salaries; MAR assumption holds because missingness on the attribute salary depends on the values of the attribute age. Now, suppose that, for a specific reason, people who earn \$1000 or less (MNAR (1)) or people who earn \$8000 or more (MNAR (2)) did not report their salaries. In this case, data are MNAR because missingness depends on the values of the attribute salary where missing data occurred. If we observe the average of the attribute salary, for the complete data, the average is estimated to \$5000. For MAR data, the average salary is \$5017, which is close to the average salary of the complete data. For MNAR(1) and MNAR(2) data, the average salary is \$7100 and \$3017, respectively. Looking at the results, the average salary of MNAR(1) is overestimated with an error of 0.42, and the average salary of MNAR(2) is underestimated with an error of 0.40. When parameter estimates are overestimated or underestimated, the difference between the estimated value and the real value of the parameter is called bias (skew). Note that, unlike MNAR data, MAR data do not cause skew in the parameter estimates.

Table 6 – List of employees with their age and their salaries with MAR and MNAR data.

ID	Age	Salary (\$)			
		Complete	MAR	MNAR(1)	MNAR(2)
1	32	8000	8000	8000	NULL
2	35	6000	6000	6000	6000
3	20	700	700	NULL	700
4	53	5200	NULL	5200	5200
5	58	8900	NULL	8900	NULL
6	36	10000	10000	10000	NULL
7	39	900	900	NULL	900
8	27	4500	4500	4500	4500
9	51	800	NULL	NULL	800
Avg	38	5000	5017	7100	3017
Error	-	-	0.0034	0.42	0.40

4.2.2 Formal Model for Missing Data

In the following, we present a formal model for missing data in metric spaces, based on the mechanisms of missingness, aiming at describing the behavior of missing data indexed in a metric space. Under this model, we formulate the problem of missingness with different mechanisms of missing values. Without loss of generality, we employ the Euclidean distance as a distance function.

Let S be a dataset of cardinality n and dimensionality m , generated by a random variable $X = (x_1, x_2, \dots, x_m)$, with a probability density function f . The dataset S is incomplete if there exist some attributes that have missing values. We denote the original data by $X = (X_{obs}, X_{miss})$, where X_{obs} are the fully observed attributes and X_{miss} are the attributes with missing values.

Let I be an indicator variable that identifies missing values. I is defined with a set of random variables with a joint probability distribution g . In statistical literature, the distribution of I corresponds to the mechanism of missingness.

Definition 7. (Missing At Random - MAR) Missing data are said to be MAR if the distribution of missingness depends on X_{obs} but does not depend on X_{miss} :

$$g(I|X_{obs}, X_{miss}) = g(I|X_{obs}) \quad (4.1)$$

Definition 8. (Missing Not At Random - MNAR) Missing data are said to be MNAR if the distribution of missingness depends on X_{miss} :

$$g(I|X_{obs}, X_{miss}) = g(I|X_{miss}) \quad (4.2)$$

Let $(\mathbb{S}, L_2)$ be a metric space and $S \subset \mathbb{S}$ an incomplete dataset associated with the Euclidean metric L_2 . We denote the observed attributes by X_{obs} , and the attributes with missing values by X_{miss} . Thus, we propose the following lemma.

Lema 1. For a metric data associated with the Euclidean distance, missingness exhibits the same phenomenon described in the statistical literature. That is, the MAR mechanism does not introduce skew in the metric space for a certain amount of missing data, whereas the MNAR mechanism causes skew in the metric space.

Proof: Under this assumption, we search for an evidence of skew in the distribution of data in the metric space. For this purpose, we analyze the distance parameter among the objects, because it determines the data distribution in the metric space.

To provide evidence that Lemma 1 holds in real data, we show that MNAR mechanism has a direct effect of skew on the metric space. Thereafter, we report interesting practical evaluations of missing data responsible for skew in the experimental study to support our theoretical assumption. Because a metric space is organized around representative objects and their distances to the rest of the objects in the space, analysis of the pairwise distances is fundamental to reveal important features about the data distribution in the space. So, let rep be a representative object taken randomly from the metric space. The following equation represents the average of the squared distances between rep and the rest of the objects in the bounding region of the subtree centered by rep :

$$\frac{1}{n'} \sum_{i=0}^{n'-1} L_2^2(rep, s_i) = \frac{1}{n'} \sum_{i=0}^{n'-1} \sum_{j=0}^{m-1} (rep[j] - s_i[j])^2 \quad (4.3)$$

where $0 < n' \leq n$ and $m > 0$, $n', m \in \mathbb{Z}$ are the number of objects in the subtree and the dimensionality of the dataset, respectively. Note that we take the squared distances to remove the root in the Euclidean distance in order to facilitate further developments without affecting the results.

After developing the right hand side of the Equation 4.3 we obtain:

$$\frac{1}{m} \sum_{i=0}^{m-1} L_2^2(Rep, s_i) = \sum_{j=0}^{n-1} (Rep[j])^2 - \frac{2}{m} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (Rep[j] \times s_i[j]) + \underbrace{\sum_{j=0}^{n-1} \left(\frac{1}{m} \sum_{i=0}^{m-1} (s_i[j])^2 \right)}_{\alpha} \quad (4.4)$$

The under-bracketed part of the Equation 4.4 represents the summed averages of the attributes with squared values. In other terms, it is the sum of parameters α that estimate the average of every attribute (actually, squared attribute) in (X_{obs}, X_{miss}) , where $\alpha = \frac{1}{n'} \sum_{i=0}^{n'-1} (s_i[j])^2$. If we assume that the subtree contains objects with missing values and that MNAR data occurred in at least one of the attributes of X_{miss} , then the corresponding parameter α will be skewed (i.e., underestimated or overestimated) and, subsequently, the overall equation that involves α will also be skewed.

The importance of this result lies in the fact that Equation 4.4 depends on a parameter α that can be a subject to skew because it involves components of X_{miss} . Rubin (RUBIN, 1976) claims that MAR mechanism does not yield to skew in the parameter estimations and that MNAR mechanism does. So, if we assume that X_{miss} are MAR, then α is not skewed and, consequently, the overall outcome of the Equation 4.4 will not be skewed. Contrarily, if X_{miss} are assumed to be MNAR, then α is skewed and the results of the Equation 4.4 will be skewed. However, skew in the Equation 4.4 implies skew in the distance distribution of the data in the metric space. Notice that, Equation 4.3 makes no assumption over the distance function L_2 , so it is valid for any metric space. An important observation is that, although, the MAR mechanism does not cause skew in the metric space, it does not mean that we can obtain the same results as if there were no missing data.

4.3 Experimental Analysis

In this section we describe the experiments performed to evaluate the effects of missingness on the distance distribution of data objects, considering both MAR and MNAR mechanisms. We aimed at answering the following questions:

- Does missingness have the effect of skew on metric spaces as described in the statistics?
- If yes, what are the forms of skew in metric spaces?
- Do the distinct mechanisms of missingness affect the performance of a MAM differently?
- How sensitive are MAMs to missing data in terms of effectiveness and efficiency?

4.3.1 Experimental Set-up

4.3.1.1 Data Description

To conduct the experiments, we used real and synthetic datasets. The real dataset consists of 500,000 NDVI¹ time series extracted from satellite images, corresponding to the state of São Paulo. The time series possess 108 measurements monthly recorded, ranging from April 2001 to March 2010. Thus, the NDVI dataset has the cardinality $n = 500,000$ and the dimensionality $m = 108$.

The synthetic dataset contains 10,000 time series of weather forecast obtained from *Embrapa Informática* and *INPE*, available in the *AgroDataMine* Server of the GBdI group². The data represent a simulation of three climate variables corresponding to pressure, minimum temperature, and maximum temperature, monthly recorded in a range of latitudes/longitudes covering the area of Brazil and standing from January 2012 to July 2015. This gives a total of 128 measurements for each time series. Thus, the WeathFor dataset has the cardinality $n = 10,000$ and the dimensionality $m = 128$.

4.3.1.2 Data Preparation

The datasets described in Section 4.3.1.1 do not have no missing values. We generate the missing data in a controlled way to produce data with MAR mechanism and data with MNAR mechanism, each data with different amounts of missing values. We employ two procedures to produce datasets with missing attribute values based on the available data.

Procedure for MAR data

Given a complete m -dimensional dataset S , we generated the incomplete datasets by removing some values from a set of specific attributes (OOI; GOH; TAN, 1998). We allow up to k attributes to have missing values, where $k < m$. Therefore, any data element in S can have at least 0 and at most k missing values. Equation 4.5 shows the proportion of data objects $F(i)$ that will have i missing values, $i = 0, 1, \dots, k$. $F(i)$ is defined as follows:

$$F(i) = \left(\frac{k-i+1}{k+1} \right)^2 - \left(\frac{k-i}{k+1} \right)^2 \quad (4.5)$$

Table 7 shows an example of the distribution of missing data when $k = 5$; where k is the number of attributes allowed to have missing values. Notice that $F(i)$ is inversely proportional to i , that is, a larger number of data objects will have a smaller number of missing attribute values and vice versa. Moreover, $F(i)$ should satisfy the property $\sum_{i=0}^k F(i) = 1$.

¹ Normalized Difference Vegetation Index (NDVI) indicates the soil vegetative vigor represented in the pixels of the images and it is strongly correlated with biomass.

² <http://www.gbdi.icmc.usp.br/>

Table 7 – Distribution of missing data for $k = 5$.

Number of missing values i	Proportion of data $F(i)$
0	11/36
1	9/36
2	7/36
3	5/36
4	3/36
5	1/36

After the proportions of missing data are established, we proceed to the selection of the k attributes that will have missing values in each element s_i from the original dataset S . This step depends on the application and it is very important because it determines the distribution of missing data that will secure the MAR assumption. Useful aspects that can be employed to choose the attributes that will have missing values for time series data are the time component and the correlation among the attributes. For example, considering a collection of daily temperature time series collected over a period, intuitively, the missing values are more likely to occur in a continuous sequence of time points. The reason is that, if a sensor unexpectedly fails to record a data at time point t , it is more likely to fail at time points $t + 1$, $t + 2$, etc, as long as the failure continues, until the sensor is fixed or replaced. Based on this example, we randomly select one attribute in the time series and drop the next k attributes.

Procedure for MNAR data

We recall that the MNAR mechanism holds when the likelihood of missing data depends on the missing values. This happens, for instance, when a sensor fails to record certain values due to lack of resolution. In such cases, missing values are usually very high or very low. Based on this example, we set an upper and a lower threshold for the attributes, and we remove all the attribute values that are above the upper threshold or below the lower thresholds. Note that, for the attribute domains that are different, we set appropriate thresholds for each attribute.

Tables 8 and 9 show the number of attributes with missing data and the real amount of missing values for the incomplete time series datasets created for the experiments. NDVI_x is the real dataset followed by the percentage of missing values, and WeathFor_x is the synthetic dataset followed by the percentage of missing values.

4.3.1.3 Methodology

The experiments are conducted executing the following steps (see also the diagram in Figure 16):

1. **Feature extration:** Before indexing the time series, it is required to use a transformation method that maps them to a lower dimensional feature space. For this intent, we used the Discrete Wavelet Transform (DWT) as a feature extraction procedure. Also, we

Table 8 – MAR datasets with the number of attributes with missing values and the percentage of missing values.

Name	N° attributes	% missing data
NDVI_2	7	2.025
NDVI_5	17	5.1
NDVI_10	33	10.03
NDVI_15	49	14.97
NDVI_20	65	19.91
NDVI_25	82	25.15
WeathFor_2	8	1.97
WeathFor_5	20	5.08
WeathFor_10	39	10
WeathFor_15	58	14.91
WeathFor_20	78	20.05
WeathFor_25	97	24.90

Table 9 – MNAR datasets with the percentage of missing values.

Name	% missing data
NDVI_20	19.77
WeathFor_2	2.55
WeathFor_5	4.96
WeathFor_10	10.23
WeathFor_12	12.15
WeathFor_16	15.78
WeathFor_18	18

implemented the classical Pyramid Algorithm (MALLAT, 1989) based on convolutions with filters for DWT to compute the wavelet coefficients.

2. **Indexing:** For each transform obtained at the preprocessing step, we selected the first 20 coefficients to represent the corresponding time series. We employed the Euclidean metric to compute the distances and build a metric index for every dataset, based on the Slim-tree (TRAINA *et al.*, 2002) metric access method available in the Arboretum framework³, to conduct the experiments. The metric trees were built using the *minOccup* and *MSTSplit* policies, which are considered as the most optimal configurations by the authors.
3. **Performing similarity queries:** Assuming that query objects do not have missing values, we selected randomly 500 query objects from the subset of complete objects for each dataset and we processed k -NN_q and R_q queries. In order to compare the results of the similarity queries obtained from datasets with different amounts of missing values, we selected the same query objects for all the datasets. The number k of nearest neighbors was set to 50 for the NDVI datasets (0.01% the size of the original dataset), and it was set to 10 for the WeathFor datasets (0.1% the size of the original dataset). The covering radius of

³ <http://www.gbdi.icmc.usp.br/old/arboretum>

Figure 16 – Diagram of the experiments performed to evaluate the effects of missingness on metric access methods.

the range queries is the distance between the query object s_q and its k^{th} nearest neighbor. To evaluate the performance of the access method when applied on incomplete datasets, we measured the precision and recall. Precision denotes the ratio of the number of relevant objects retrieved to the total number of irrelevant and relevant objects retrieved. Recall denotes the ratio of the number of relevant objects retrieved to the total number of relevant objects available. Since there is no certainty that objects with missing values are relevant, we consider only complete objects as relevant.

$$Precision = \frac{|\text{Relevant complete objects retrieved}|}{|\text{Total objects retrieved}|}, Recall = \frac{|\text{Relevant complete objects retrieved}|}{|\text{Total objects relevant}|} \quad (4.6)$$

We employed the results obtained from complete datasets as a reference to estimate precision and recall of the queries when processed on incomplete datasets. During the experiments, we generated the graphs of precision and recall, and we measured the average number of disk accesses, the average number of distance computations and the total time required to process all the queries for every dataset. The tests were performed on a machine with a Pentium D 3.4GHz processor and 2Gb of memory RAM.

4.3.2 Performance Results

This section reports and discusses the results obtained from the experiments, considering effectiveness parameters (precision and recall) and efficiency parameters (disk accesses, distance

computation and time execution).

4.3.2.1 Effectiveness Parameters

Figures 17, 18 and 19 present the average precision and recall for NDVI datasets with MAR and MNAR assumptions, WeathFor datasets with MAR assumption and WeathFor datasets with MNAR assumption, respectively. The last measurements in the graphs of the NDVI datasets correspond to the NDVI dataset with MNAR assumption.

Figure 17 – Precision and recall for NDVI datasets with MAR and MNAR assumptions: (a) k -NN_q query, (b) Range query.

The graphs of precision and recall show that the precision and recall decrease rapidly with the increasing amount of missing values. Under MAR assumption, for NDVI_2 and WeathFor_2, with only 2% of missing data, k -NN_q query achieves less than 30% of precision and recall (see Figures 17a and 18a), whereas R_q query performs better in terms of precision, but achieves only 30% of recall (see Figures 17b and 18b). For NDVI_25 and WeathFor_25, both k -NN_q and R_q queries achieve only 10% of recall, with a better precision rate for R_q query (see Figure 18b). Under MNAR assumption, k -NN_q query achieves its worse case with NDVI_20 (see the last measurement in Figures 17a and 17b), while it performs better with WeathFor datasets (see Figure 19). Note that, a drop of the precision and recall measurements implies that a significant amount of non-relevant objects have been retrieved, while the relevant objects are missing from the query response. We can also observe that, for k -NN_q query, the measurements of precision and recall are identical. This is due to the fact that the total number of relevant objects is equal to the total number of retrieved objects, which is set to k . This is not the case for R_q query, because the number of retrieved objects depends on the number of objects that are in the covering radius.

Under MAR mechanism, the precision of R_q query is better than that of k -NN_q query, because, as expected, there are less false hits in the query responses. However, it does not mean

Figure 18 – Precision and recall for WeathFor datasets with MAR assumption: (a) k-NN_q query, (b) Range query. Figure 19 – Precision and recall for WeathFor datasets with MNAR assumption: (a) k-NN_q query, (b) Range query.

that R_q query performs better than $k\text{-NN}_q$ query. The reason is that, the associated recall that should reflect the completeness of the query answer is relatively poor. In fact, when non-relevant objects are retrieved, the query response can be filtered to keep the most relevant objects only. But when relevant objects are missing, it becomes difficult to recover them and, thus, recall turns out to be more problematic than precision.

The results obtained so far can lead us to several conclusions. Regarding the $k\text{-NN}_q$ queries, given a query object s_q , the k nearest neighbors retrieved from the complete dataset S are totally different from those retrieved from the incomplete dataset S' . Alternatively, for R_q queries, given a query object s_q and a covering radius r , the objects that are in the covering radius from s_q in the complete dataset S are different from those that are in its covering radius in the incomplete dataset S' . Intuitively, when objects have missing values, the pairwise distances tend to be uncertain as there is no warranty that close objects are close in the complete data space.

So, changing neighborhood when the query objects have missing values is natural, however, when the query objects do not have missing values, changing the neighborhood indicates that the distribution of data in the complete data space is different from the distribution of data in the incomplete data space. The reason is that, missing data tend to change the distribution of the objects when indexed in the metric space. What we mean by change in the distribution is that objects move randomly in the space while keeping the overall sparseness (i.e., volume) of the space. In addition, choosing representatives among the objects with missing values causes distortion and inconsistency in the data structure, leading to inaccurate query response and poor query performance. These statements apply mainly on the MAR data because the performance of similarity queries degenerate severely with small amounts of missing data.

4.3.2.2 Efficiency Parameters

Figures 20, 21 and 22 show the average number of disk accesses, the average number of distance calculations and the total time execution for the NDVI datasets with MAR and MNAR assumptions, for the WeathFor datasets with MAR assumption and for the WeathFor datasets with MNAR assumption, respectively. The last measurements in the graphs of the NDVI datasets correspond to the NDVI dataset with MNAR assumption.

Figure 20 – Query performance for NDVI datasets with MAR and MNAR assumptions: (a) Average disk accesses, (b) Average distance calculations, (c) Total execution time.

When observing the graphs, it can be seen that, with MAR assumption, for both NDVI and WeathFor datasets, all the efficiency parameters required during the query processing decrease with the increasing amount of missing data, whereas with MNAR assumption, the queries exhibit higher values for all the parameters, in particular k -NN $_q$ queries. Typically, R_q queries are more efficient than k -NN $_q$ queries because a limiting radius is provided which allows to prune more subtrees and speed up the searching process.

The performance deterioration of k -NN $_q$ queries under MNAR assumption (i.e., increasing of time execution, disk accesses and distance calculations) is an evidence that skew occurred in the metric spaces. Literally, the skew is often caused by large dimensionality (either from

Figure 21 – Query performance for WeathFor datasets with MAR assumption: (a) Average disk accesses, (b) Average distance calculations, (c) Total execution time.

Figure 22 – Query performance for WeathFor datasets with MNAR assumption: (a) Average disk accesses, (b) Average distance calculations, (c) Total execution time.

the embedded space or from the equivalent fractal dimension of metric space), where objects tend to spread over broader spaces and the distance differences among pairs of objects tend to homogenize, which makes the nearest neighbor meaningless. This problem, known as the "dimensionality curse", is more prevalent in high-dimensional datasets. When datasets have missing attribute values, we do not deal with high-dimensionalities, instead, we deal with lack of data. In fact, missing attribute values tend to reduce (i.e., underestimate) the distances among pairs of objects, preventing these latter to spread in the space. This phenomenon causes distance concentration in the clusters (i.e., regions) where missingness occurred, and when searching for nearest neighbors, the k^{th} nearest neighbor is too close that a small increase in the radius will include many others. Note that distance concentration favors the node overlap in the metric spaces and makes the searching process more expensive in terms of time search, number of disk accesses and number of distance calculations. But, when data are MNAR, objects tend to keep their position in the clusters where missing data occurred instead of moving randomly in the indexed space. This fact explains the better rates of precision and recall achieved by *k-NN_q* and *R_q* queries under MNAR assumption.

4.3.2.3 Distance Concentration

Figure 23 – Probability Density Function of the distances for WeathFor datasets with MAR assumption: (a) Complete data, (b) 2% missing data, (c) 5% missing data, (d) 10% missing data.

To better understand the phenomenon of distance concentration, we generated the graphs of the probability density function of the pairwise distances among the objects, in order to analyze their behavior with regard to missingness mechanisms in both data and transform domains. Figures 23 and 24 show the probability density function of the distances for WeathFor datasets with MAR and MNAR mechanisms, respectively. In the data domain, the graph of the distance concentration has almost the same shape, height, and width for all MAR and MNAR datasets with different portions of missing data. The peak in the graph indicates a distance concentration that already exists in the complete datasets. However, this peak gets higher with MNAR datasets and nearly doubles with 10% of missing data. This already shows that MNAR data are prone to distance concentration.

In the transform domain, the graph of distance concentration becomes gradually wider and low-lying for both MAR and MNAR datasets. Interestingly, we can observe a development of many peaks that become increasingly numerous and homogeneous, spanning over a wider range of distance values. These peaks result from the distance concentration caused by missingness in the datasets. For MAR data, the distance concentration is very moderate as the peaks are not abundant, but for MNAR data, the distance concentration is more prevalent since the peaks are more abundant and spread in a much broader space. This data distribution is directly mapped to

Figure 24 – Probability Density Function of the distances for WeathFor datasets with MNAR assumption: (a) Complete data, (b) 2% missing data, (c) 5% missing data, (d) 10% missing data.

the metric space where data objects are indexed.

The conclusion that is enforced by the graphs about missingness is that, when data are MAR, objects move randomly in the space while keeping the overall sparseness of the space (see Figure 25b). This introduces changes to the data distribution and also causes distortion and inconsistency in the indexed space, especially when the representatives that are used to build the metric space have missing values. Metric access methods are sensitive to distortion caused by missingness. The evidence is shown in the results of the effectiveness parameters where the accuracy of the similarity queries is relatively poor.

When data are MNAR, objects tend to keep their position in the clusters where missing data occurred. However, these latter move closer to each other inside the clusters where missing data occurred, making close objects become closer (in this case the distances are underestimated) and leading to distance concentration. Meanwhile, objects from cluster to another cluster become more distant (in this case the distances are overestimated), allowing to clusters to spread over a much broader space (see Figure 25c). Metric access methods are sensitive to distance concentration. The evidence is shown in the results of the efficiency parameters where k -NN_q queries degenerate completely even when the datasets have only 2% of missing values.

Although, MAR data do not cause skew in the data space, this does not exclude the possibility of having distance concentration in some parts of the space. Alternatively, MNAR data

Figure 25 – Example of data distribution in an internal node of an index structure: (a) Complete data space, (b) missing values with MAR process occurred in some clusters causing distortion in the space, (c) missing values with MNAR process occurred in some clusters causing skew in the space.

are capable to cause severe skew in the data space, but are also capable to introduce distortion to the metric space, leading to inaccurate query response and poor query performance.

4.4 Final Considerations

In this chapter, we discussed the issues involved when indexing and searching datasets with missing attribute values. We first presented a formal model for missing data that provides an understanding of the effects of distortion and skew caused by missingness in metric spaces and, based on the theoretical framework established by the statisticians, we demonstrated that missing data have a direct effect on the metric spaces. We also provided a set of experiments to evaluate the effect of missingness on the distance distribution in the metric spaces, considering both MAR and MNAR mechanisms. The general conclusions of our results are the following:

- MAR data introduce changes to data distribution and cause distortion and inconsistency in the indexed space;
- For MNAR data, objects tend to keep their position in the clusters where missing data occurred while moving closer, making close objects become closer, leading to distance concentration, and distant objects become more distant, making the space become much broader;
- MAR data cause distortion in the metric space and affect the effectiveness parameters of the query performance;

- MNAR data cause skew in the metric space and affect the efficiency parameters of the query performance;
- MAR data may cause distance concentration in some parts of the space but do not affect the efficiency parameters of the query performance;
- MNAR data may cause distortion in the metric space and affect the effectiveness parameters of the query performance;

Degradation of the precision and recall is not an indicator of skew in the metric space, because it is natural to obtain different results when queries are performed on different datasets. However, degradation of the efficiency measurements is not natural unless there is a reason that led to the degradation of the metric access method. We identified that the reason is the effect of distance concentration caused by MNAR processes. These results let us to confirm our assumption about missing data in metric spaces and they are useful to guide the development of two types of solutions for handling missing data:

- Make the metric access methods able to index incomplete datasets when missing attribute values occur, in order to perform similarity queries on the available data.
- Reduce the distortion and skew caused by missing data in metric spaces, in order to improve the query performance and speed up the searching process.

We highlight that our results are the first to provide a basis to support the development of indexing techniques that take into account the effects caused by missing data and really allow bypassing them in the index structure.

HOLLOW-TREE

5.1 Introduction

Although there are many algorithms for indexing and searching large and complex datasets, none of them is designed to handle incomplete datasets. Developing algorithms to answer similarity queries when data have missing values is very challenging, as the traditional techniques are not straightforwardly adaptable to handle missingness. The problem stems from the fact that when objects have missing attribute values, the pairwise distances are undefined and thus, these objects are dismissed from the index structure, leading to important loss of information. But, when missing attribute values are ignored, missingness can yield distortion and skew in the resulting index structure. Recall that distortion in the index structure leads to inaccurate query response by retrieving non-relevant objects and missing relevant ones, while skew degenerates the underlying MAM, causing a significant overlap between the nodes and making the searching process very expensive.

This chapter proposes the *Hollow-tree* MAM, a new access method built over the Slim-tree platform for indexing and querying data with missing attribute values. The techniques presented are able to overcome the limitations of the metric access methods when applied on incomplete datasets. The first technique includes a set of indexing policies that allow to index objects with missing values and guarantee that no object is missing from the index structure. The second technique employs the fractal dimension of the dataset and the local density of the query region to achieve an effective and efficient query answer, considering also objects with missing values as a potential response. In principle, our techniques are independent from the underlying metric access method and can be applied to a wide range of MAMs, allowing the extension of many metric access methods. However, in this work we extended just the Slim-tree MAM and focused our evaluation on it. To our knowledge, this work is the first to address the problem of missing data in metric access methods. The rest of this chapter is organized as follows: Section 5.2. introduces the new *Hollow-tree* MAM for similarity search over incomplete

datasets. Section 5.2.1 describes the proposed indexing technique and section 5.2.2 presents the proposed extensions to Range and k - NN_q queries for data with missing values. Section 5.3 summarizes the main contributions of this chapter.

5.2 The Hollow-tree Access Method

In this section we propose two novel techniques specifically for metric access methods able to support indexing and retrieval over datasets with missing attribute values. Contrarily to traditional indexing techniques, our technique is able to build consistent indexes by preventing objects with missing values from causing distortions in the index structure. It also guarantees that no object is missing from the index. The second technique employs the fractal dimension of the dataset and the local density of the query region to search by similarity and filter data in the query response. Also, it takes advantage of the indexing technique to improve the query performance. Following, we assume that:

- Missingness is ignorable and we consider two cases of missing attribute values: (1) The attribute values exist but for unspecified reason are unknown. (2) The attribute values do not exist because they are irrelevant to the corresponding objects.
- Objects can have missing values in one or more attribute but not all the attributes.
- Query objects do not have missing attribute values.
- For metric data, where attribute values are not specified, we assume that missing attribute values are handled by a distance function that admits missing attribute values. Multidimensional data with missing attribute values are treated as a special case.

For the sake of generality, and following the universal assumption made by the overwhelming majority of the existing relational database management systems, we use the NULL value to denote both unknown and irrelevant attribute values.

5.2.1 Indexing Technique

Metric access methods enables employing a recursive processing to build the index structure. Each object from the dataset is read once and compared recursively with the representatives of the nodes at each level of the tree, following a particular indexing policy, to choose a potential subtree to store the object. Each object is inserted in the subtree that satisfies the required conditions. Our indexing proposal requires to read the dataset in two steps. In the first step, complete objects are loaded and an initial metric tree is straightforwardly constructed using a traditional distance metric. In the second step, the distance metric is forced to ignore NULL values and consider only observed attributes. Thereafter, objects with NULL values are inserted into

Figure 26 – Building a metric tree in two steps following the new indexing technique.

the already built tree without interfering in the index structure. Figure 26 shows the different steps for building the metric tree. The aim of this strategy is to prevent data with NULL values from being chosen as representatives and, thus, avoiding to introduce substantial distortion in the internal structure of the index.

Notice that building a metric tree in two steps is particularly important. In fact, if the first few objects that are indexed have NULL values, we will be forcing the metric access method to choose the first representatives among objects with NULL values. Since data with NULL values are processed separately during the indexing procedure, it is important to distinguish them from the rest of the data. Therefore, when building the index, the objects are tagged to indicate if they have NULL attribute values or not. Slim-tree MAM supports two types of nodes: index and leaf nodes. Index nodes hold representatives and pointers to the next levels of the tree, and leaf nodes hold the features and pointers to the original data objects. Since the search keys are stored in the leaf nodes, each entry in the leaf node includes an attribute "missing" that operates as the missing indicator, as follows:

$$missing = \begin{cases} true & \text{if the object has NULL values,} \\ false & \text{otherwise.} \end{cases}$$

The original structure of a Slim-tree leaf node is

$$leaf_node[array\ of\ <Oid_i, d(s_i, s_{rep}), s_i >],$$

where Oid_i is the identifier of the object s_i and $d(s_i, s_{rep})$ is the distance between the object s_i and the representative s_{rep} of the node. With the new scheme, the structure of the leaf node is

$$leaf_node[array\ of\ <Oid_i, d(s_i, s_{rep}), missing, s_i>],$$

where *missing* is the indicator variable of the object s_i .

Note that objects in the index nodes do not require an indicator variable because these nodes hold only representatives that are not allowed to have NULL values. This is guaranteed by the fact that the tree is built from the bottom to the root and the choice of the representatives is made at the leaf level. Notice that for other metric access methods, as for example for the DBM-tree (VIEIRA *et al.*, 2010), the index structure may support only one type of node and the representatives are stored along with the objects. In this case, the indicator variable would be necessary at every level of the tree structure.

Following, we define new policies that will provide a particular treatment for data with NULL values and help to establish consistency in the index structure. In a metric access method, there are two main components essential to make crucial decisions when organizing data in a tree structure: (1) the partitioning technique that allows to divide the space into subspaces, and (2) the choosing subtree technique that allows to choose the most adequate node or subtree where to store a new object. The Slim-tree MAM provides the *SplitNode* and the *ChooseSubtree* procedures for space partitioning and choose subtree, respectively. For each procedure, several policies are defined, and the way they are extended to allow handling missing data are described following.

5.2.1.1 SplitNode Policies

When a node overflows, *SplitNode* procedure is called to create a new node at the same level and distribute the objects among the two nodes. New representatives are selected and the parent node that keeps the information about the representatives is updated. When the node to be split contains both complete objects and objects with NULL values, representatives are chosen among the complete objects only. For this purpose, the policies of *SplitNode* are redefined to satisfy the new requirements.

1. **randomSplitMiss** - The *randomSplitMiss* algorithm allows choosing randomly two representatives that do not contain NULL values. Algorithm 5 illustrates the *randomSplitMiss* policy.

2. **minMaxSplitMiss** - The *minMaxSplitMiss* algorithm tests all the possible pairs of complete objects and selects the pair that minimizes the covering radii of the resulting nodes. Algorithm 6 illustrates the *minMaxSplitMiss* policy.

3. **MSTSplitMiss** - In the *MSTSplitMiss* algorithm, complete objects and objects with NULL values are first separated, then the *Minimal Spanning Tree* algorithm is performed on the

Algorithm 5: randomSplitMiss

Data: Node A to be split
Result: Nodes A_1 and A_2 with their representative objects rep_1 and rep_2 , respectively

```

1 Choose randomly  $rep_1$  from  $A$ ;
2 Choose randomly  $rep_2$  from  $A$ ;
3 while  $rep_1$  has NULL values do
4   | Choose randomly  $rep_1$ ;
5 end
6 while  $rep_2$  has NULL values or  $rep_1 = rep_2$  do
7   | Choose randomly  $rep_2$ ;
8 end
9 Set the new representatives  $rep_1$  for node  $A_1$  and  $rep_2$  for node  $A_2$ ;
10 for each object  $s_i$  of the node  $A$  do
11   | if  $d(s_i, rep_1) \neq 0$  and  $d(s_i, rep_1) < d(s_i, rep_2)$  then
12     | Add  $s_i$  to node  $A_1$ ;
13   | else
14     | Add  $s_i$  to node  $A_2$ ;
15   | end
16   | end
17 end
18 Return  $(A_1, rep_1), (A_2, rep_2)$ ;

```

complete part, then the rest of the objects are distributed between the resulting nodes. Algorithm 7 illustrates the *MSTSplitMiss* policy.

In order to guarantee consistent results with the *SplitNode* procedure, a minimum number of complete objects in the node to be split is set to 4. That is, nodes that have less than 4 complete objects cannot be split again. The reason is that, 4 complete objects are necessary to generate two consistent nodes; two objects are set as the representatives and two others delimit the covering radii of the resulting nodes. In fact, data with NULL values tend to shrink the indexed space, especially when many dimensions are missing. So, to minimize the distance concentration around the representatives in the nodes, at least one complete object is kept (in addition to the representative) to guarantee a reasonable covering radius for the node. Figure 27 illustrates an example of node splitting using the *MSTSplitMiss* algorithm with a minimum number of complete objects. If every qualified node has less than 4 complete objects, the best qualified one, with at least two complete objects, is selected and the same radius is maintained for the resulting nodes. However, if all the qualified nodes have only one complete object, which corresponds to the representative, the best qualified node is selected and its representative is duplicated and promoted as the representative of the resulting nodes. Thereafter, the rest of the objects are equally distributed between the two nodes. This can increase the node overlap in the tree (see Figure 28), but it certainly guarantees consistency in the index structure. Although, this extreme seldom situation occurs, it did not occur in any of the evaluations performed for all the experiments reported in this work.

Algorithm 6: minMaxSplitMiss

Data: Node A to be split
Result: Nodes A_1 and A_2 with their representative objects rep_1 and rep_2 , respectively

```

1 for each object  $s_i$  of the node  $A$  do
2   if object  $s_i$  has no NULL values then
3     for each object  $s_j \neq s_i$  of the node  $A$  do
4       if object  $s_j$  has no NULL values then
5         Set  $s_i$  for  $rep_1$  and set  $s_j$  for  $rep_2$ ;
6         Create temporary nodes  $A'_1$  and  $A'_2$  with  $rep_1$  and  $rep_2$ , respectively;
7         Distribute the objects of the node  $A$  between  $A'_1$  and  $A'_2$ ;
8         Save the covering radii of the resulting nodes;
9       end
10    end
11  end
12 end
13 Select the pair  $(rep_1, rep_2)$  that minimizes the covering radii of the resulting nodes;
14 Set the new representatives  $rep_1$  for node  $A_1$  and  $rep_2$  for node  $A_2$ ;
15 for each object  $s_i$  of the node  $A$  do
16   if  $d(s_i, rep_1) \neq 0$  and  $d(s_i, rep_1) < d(s_i, rep_2)$  then
17     Add  $s_i$  to node  $A_1$ ;
18   else
19     Add  $s_i$  to node  $A_2$ ;
20   end
21 end
22 end
23 Return  $(A_1, rep_1), (A_2, rep_2)$ ;

```

5.2.1.2 ChooseSubtree Policies

ChooseSubtree procedure helps to choose the node that is most suitable to hold the object when several nodes are qualified. However, when objects with NULL values are indexed, more constraints must be considered :

1. If the qualified nodes have no free space to store the new object, one of the nodes that has at least 4 complete objects is selected to guarantee the conditions of node split.
2. If all the qualified nodes have less than 4 complete objects, the qualified node that has more complete objects is selected.
3. If the distance between the object and the representative of the qualified node is zero, the node is disqualified and the object is inserted into a node where the distance is not zero.

Note that zero value distances occur when the values of all the common attributes between the object and the representative are equal. Usually, this happens when the most discriminative attributes have missing values. Note that, the most discriminative values are those that are more

Algorithm 7: MSTSplitMiss

Data: Node A to be split
Result: Nodes A_1 and A_2 with their representative objects rep_1 and rep_2 , respectively

```

1 Create a temporary empty node  $A'$ ;
2 for each object  $s_i$  of the node  $A$  do
3   if object  $s_i$  has no NULL values then
4     | Add object  $s_i$  to  $A'$ ;
5   end
6 end
7 Perform the MST algorithm on  $A'$ ;
8 Set the new representatives  $rep_1$  for node  $A_1$  and  $rep_2$  for node  $A_2$ ;
9 for each object  $s_i$  of the node  $A$  do
10  if  $d(s_i, rep_1) \neq 0$  and  $d(s_i, rep_1) < d(s_i, rep_2)$  then
11    | Add  $s_i$  to node  $A_1$ ;
12    else
13      | Add  $s_i$  to node  $A_2$ ;
14    end
15  end
16 end
17 Return  $(A_1, rep_1), (A_2, rep_2)$ ;
```

”different” from the others and, thus, they are usually those in the border of the space. Figure 29 shows an example of equal gray-level histograms resulting from different images with missing values. In Figure 29a, the histogram includes different levels of intensity corresponding to both bright and dark parts of the moon image. In Figure 29b, the bright part of the image corresponding to the moon is missing and, consequently, many attributes of the gray-level histogram are missing. Notice that, without additional information about the state of the image in Figure 29b, the moon image in Figure 29b and the dark image in Figure 29c are numerically equal. The reason is that, the most discriminative attributes corresponding to the bright part of the moon, that would make a difference between the image in Figure 29b and the image in Figure 29c, are missing and, thus, the distance between the corresponding histograms is zero.

5.2.2 Similarity Queries

Having all the available data, we can now perform similarity queries. One relevant question that arises when dealing with missing data is “how similar are objects that are partially specified?” The answer to this question is very broad and depends on the application domain and the user’s interpretation of missingness. Our intent here is not to interpret the similarity between objects with missing values, but to make the data fully available and then let the user decide how to include the data with missing values in his/her analysis. With our indexing technique, it is now feasible to search for complete data as well as for data with NULL values. For this purpose, we redefine the traditional Range query and k -Nearest Neighbor query to support data objects with NULL values. Both Range and k -NN_q queries can be performed over the indexed dataset and each

Figure 27 – Example of a node split using MSTSplitMiss algorithm : (a) Node to be split, (b) build MST tree, (c) split the node.

Figure 28 – Example of a node split with duplicated representative.

of the queries now return two lists; a list of complete objects and a list of objects with NULL values.

5.2.2.1 The RM_q Query

The Range query, which we call $RM_q(s_q, r)$, is similar to the traditional query. It receives in parameter a query center s_q and a query radius r and returns the objects that are in the covering radius. The result is divided into two lists: The first list includes the complete objects and the second list holds the objects with NULL values (see Algorithm 8).

5.2.2.2 The k -NNFM $_q$ Query

The k -Nearest Neighbor query is more sensitive to missingness. The reason is that, closest objects are not always the most similar to the query center. So, when searching for nearest neighbors,

Figure 29 – Example of gray-level histogram resulting from: (a) A complete image of the moon, (b) an image of the moon with missing values, (c) a dark image.

Algorithm 8: $RM_q(s_q, r)$

Data: Query center s_q and query radius r

Result: A list L_1 of complete objects and a list L_2 of objects with NULL values

```

1 for each object  $s_i$  that is in the covering radius  $r$  from  $s_q$  do
2   if object  $s_i$  has NULL values then
3     Add object  $s_i$  to  $L_2$  ;
4   else
5     Add object  $s_i$  to  $L_1$ ;
6   end
7 end
8 end
9 Return  $L_1, L_2$ ;

```

the first elements are more likely to be objects with NULL values, because the distances tend to be smaller. The problem can step further when only objects with NULL values are returned in the query response, because they dominate the query region (see Figure 30.a). This can cause a significant inconvenience when relevant objects are not included in the query answer.

Our proposition consists of dividing the query response into two ordered lists, a list of complete objects and a list of objects with NULL values, then take advantage of the fractal dimension $\mathcal{D}$ of the dataset to retrieve the objects of each list. Knowing that $\mathcal{D}$ is invariant to the size of the dataset, it is computed from the subset of complete objects only. The reason is that,

Figure 30 – Example of a 10-Nearest Neighbour query. (a) Traditional $k\text{-}NN_q$, (b) $k\text{-}NNFM_q$.

objects with NULL values can distort the results. Provided that data are missing at random, we can safely consider the subset of complete objects as a random sample and its fractal dimension is the same as the fractal dimension of the entire dataset. This property allows to estimate $\mathcal{D}$ for any dataset with a reasonable amount of complete objects.

Now consider a $k\text{-}NN_q(s_q, k)$ query with k objects to be retrieved. In an ideal situation where all the objects in the dataset are perfectly specified, the query returns the closest k objects to the query center. Alternatively, considering a $k\text{-}NNF_q(s_q, k)$ query; a final radius r_f can be estimated using the fractal dimension $\mathcal{D}$ of the dataset (see Equation 2.4), and a $k\text{AndRange}(s_q, k, r_f)$ algorithm is processed using the final radius r_f . If the desirable number k of objects is not reached, that is, a quantity $k' < k$ is retrieved, a larger radius r'_f is estimated using the local density of the objects around the query object and $k\text{RingRange}(s_q, k - k', r_f, r'_f)$ algorithm is processed to retrieve the remaining objects. The process is repeated until k is reached or the new local radius is greater than the diameter of the dataset. Assuming that the dataset includes objects with NULL values, if the query is centered where the density of objects with NULL values is important, $k\text{-}NN_q(s_q, k)$ query may lead to inaccurate results. The fact is that more objects with NULL values are likely to be retrieved. Hence, our proposition is to estimate the cardinality of the query response first and set it as the desirable number of objects to be retrieved, then perform a $k\text{-}NNF_q(s_q, k)$ query until this number is reached. The new variant of $k\text{-}NNF_q(s_q, k)$, which we call $k\text{-}NNFM_q(s_q, k, p)$, takes a new parameter p that indicates the proportion of missing data in the dataset and performs the four following steps (see also Algorithm 9):

1. Estimate a limiting radius r_f where the k nearest neighbors are located, using the fractal dimension $\mathcal{D}$;
2. Estimate the desirable number k_{com} of complete objects to be retrieved;
3. Perform a $kAndRange(s_q, k, r)$ query using the final radius r_f ;
4. Refine the searching procedure using a local radius r'_f until at least k_{com} and at most k complete objects are retrieved.

We define two desirable numbers of objects to be retrieved: A number k_{com} of complete objects, which corresponds to the cardinality of the first list in the query response, and a number k_{null} of objects with NULL values, which corresponds to the cardinality of the second list in the query response. The number of complete objects k_{com} is more important since we can have the certainty that they are relevant to the query object. However, objects with NULL values are likely to be relevant, but there is no guarantee for that. Therefore, when performing a $k-NNFM_q(s_q, k, p)$ query, the major intent is to reach the desirable number of complete objects.

The desirable number of objects to be retrieved depends on the proportion of missing data in the dataset. Consider a metric space $(\mathbb{S}, d)$, a query center s_q and a number k of its neighbors within a distance r . If we take the same metric space with a proportion p ($0 < p < 1$) of objects with NULL values, the query center s_q will have $k_{com} = (1 - p) * k$ neighbors as complete objects and $k_{null} = p * k$ neighbors with NULL values, within the same distance r . This is guaranteed by the assumption that data are missing at random. Note that, k_{com} and k_{null} are just an approximation since the exact values depend on the local density around the query center. Therefore, if k_{com} is underestimated (that is, there are more complete objects in the query region than expected), $k-NNFM_q(s_q, k, p)$ will retrieve at most k complete objects. In this case, the number of retrieved objects with NULL values may be smaller than the desirable k_{null} . On the other hand, if k_{com} is overestimated (that is, there are fewer complete objects in the query region than expected), the local radius r'_f will be extended until at least k_{com} complete objects are retrieved or r'_f is greater than the diameter of the dataset. In this case, the number of retrieved objects with NULL values may be greater than k_{null} . Nevertheless, if the number of retrieved objects with NULL values exceeds k_{null} or the number of retrieved complete objects exceeds k , the results will be pruned. Figure 31 illustrates the desirable number of objects to be retrieved, given a proportion p of objects with NULL values in the dataset.

Figure 31 – Desirable number of complete objects and objects with NULL values to be retrieved.

Algorithm 9: $k\text{-NNFM}_q(s_q, k, p)$

Data: Query center s_q , an entire number k and the proportion p of objects with NULL values

Result: A list L_1 of k_{com} complete objects ($k_{com} \leq k$) and a list L_2 of k_{null} objects with NULL values

- 1 Obtain N as the number of elements in the dataset;
 - 2 Obtain R as the diameter of the dataset indexed;
 - 3 Estimate the final radius r_f ;
 - 4 Compute the desirable numbers k_{com} and k_{null} ;
 - 5 Clear lists L_1 and L_2 ;
 - 6 Process $k\text{AndRange}(s_q, k, r_f)$, store the answer in L_1 and L_2 , and set k' as k_{com} ;
 - 7 **while** $k' < k$ and $r_f < R$ **do**
 - 8 Estimate the local final radius r'_f ;
 - 9 Execute $k\text{RingRange}(s_q, k - k', r_f, r'_f)$, store the answer in L_1 and L_2 , and set k' as k_{com} ; Set $r_f = r'_f$
 - 10 **end**
 - 11 Return L_1, L_2 ;
-

5.3 Final Considerations

This chapter presented the new *Hollow-tree* access method that we developed as part of this Ph.D. work. The main contributions of the *Hollow-tree* access method are:

1. **Indexing technique for data with missing values:** The new indexing technique allows to identify objects with missing attribute values and process them separately during the indexing procedure. The metric index is built in two steps to prevent objects with NULL

values from being chosen as representatives and, substantially, introduce distortion in the internal structure of the index.

2. **SplitNode policies for data with missing values:** The new *SplitNode* policies provide a particular treatment for data with NULL values and help to establish consistency in the index structure. When nodes are split, representatives are always chosen among the complete objects, and if a node does not contain enough complete objects, the representative of the node is duplicated and promoted as a representative of the new nodes.
3. **ChooseSubtree policies for data with missing values:** The new *ChooseSubtree* policies help to choose the node that is most suitable to hold the object when several nodes are qualified. If the qualified nodes have no free space to store the new object, one of the nodes that has at least 4 complete objects is selected to guarantee the conditions of node split.
4. **Similarity queries for data with missing values:** Similarity queries perform on the new index structure and return two lists of response; a list of complete objects and a list of objects with NULL attribute values.
5. **RM_q query:** It receives in the input parameters a query center s_q and a query radius r , and returns the objects that are in the covering radius. The retrieved complete objects and objects with NULL values are returned in two different lists.
6. **k -NNFM_q query:** In a response to distance concentration caused by missingness, k -NNFM_q query estimates a desirable number of complete objects to be retrieved, based on the proportion of missing data, then uses the fractal dimension of the data space to estimate a final radius where the relevant objects are located. If the desirable number of objects is not reached, the searching process is refined using a new radius based on the local density of the query region. The main object of this technique is to improve the accuracy of the query response when the query region is dominated by data with NULL attribute values.

Our indexing technique allows to build consistent metric indexes and reduce the distortion caused by missing data. It also enables to insert new objects after building the metric index if the underlying access method supports new insertions. The presented techniques allow to explore properly all the available data and they are independent of the underlying access method. Therefore, they can be applied to a broad range of MAMs, allowing to extend the family of metric access methods.

HOLLOW-TREE EVALUATION

6.1 Introduction

This chapter presents the experimental evaluation of the proposed access method *Hollow-tree*. We implemented a functional prototype of the *Hollow-tree* and we conducted numerous experiments on different datasets to assess its performance. We also compared our proposed method with the conventional Slim-tree access method to evaluate the gain of performance achieved by the new techniques.

6.2 Experimental Set-up

To conduct our experiments, we used a variety of real and synthetic datasets. Assuming that data are missing at random, we generated from each dataset new sets with different amounts of missing values, ranging from 5% to 40%. The percentage of missing data is given by the number of objects with NULL values. We assumed that objects are allowed to have NULL values on one or more attributes, but not all the attributes. The datasets are the following :

- **Weather** - a real set of 14350 time series that represent three climate variables corresponding to maximum temperature, minimum temperature and precipitation, recorded during 365 days, which gives a total of 1095 measurements for each time series. The data were recorded by 287 different weather stations of INMET¹ from 1960 to 2010, covering various regions of Brazil. The data were provided by the *Agro-meteorological Monitoring System - Agritempo*².
- **Sierpinski** - a synthetic set of 9842 points from the Sierpinski triangle, projected in a two-dimensional space.

¹ <http://www.inmet.gov.br/portal/>

² www.agritempo.gov.br

Table 10 – Datasets used in the experiments and their characteristics.

Dataset	Num. obj.	Num. attri.	Num. attr. with NULL values	Frac. dim.
Weather	14350	1095	365	2.40
Sierpinski	9842	2	1	1.55
Sphere	19901	3	1	1.96
Square	10000	4	1	2.07
Gaussian2	5000	2	1	1.75
Gaussian32	1024	32	3	2.07
Gaussian64	1024	64	3	2.07

Table 11 – Datasets with missing attribute values.

Dataset	5%	10%	15%	20%	30%	40%
Weather	Weath_5	Weath_10	Weath_15	Weath_20	Weath_30	Weath_40
Sierpinski	Sierp_5	Sierp_10	Sierp_15	Sierp_20	Sierp_30	Sierp_40
Sphere	Sph_5	Sph_10	Sph_15	Sph_20	Sph_30	Sph_40
Square	Squ_5	Squ_10	Squ_15	Squ_20	Squ_30	Squ_40
Gaussian2	Gaus2_5	Gaus2_10	Gaus2_15	Gaus2_20	Gaus2_30	Gaus2_40
Gaussian32	Gaus32_5	Gaus32_10	Gaus32_15	Gaus32_20	Gaus32_30	Gaus32_40
Gaussian64	Gaus64_5	Gaus64_10	Gaus64_15	Gaus64_20	Gaus64_30	Gaus64_40

- **Sphere** - a synthetic set of 19901 points from the surface of a sphere, projected in a three-dimensional space.
- **Square** - a synthetic set of 10000 points from the borders of a square, projected in a four-dimensional space.
- **Gaussian2** - a synthetic set of 5000 points from a Gaussian distribution, projected in a two-dimensional space.
- **Gaussian32** - a synthetic set of 1024 points from a Gaussian distribution, projected in a 32-dimensional space.
- **Gaussian64** - a synthetic set of 1024 points from a Gaussian distribution, projected in a 64-dimensional space.

Table 10 summarizes the datasets and their characteristics, and Table 11 includes the datasets with missing attribute values. Weath_x, Sierp_x, Sph_x, Squ_x, Gaus2_x, Gaus32_x and Gaus64_x are the datasets described above, followed by the percentage of objects with NULL values. The real dataset Weather was submitted to the Discrete Wavelet Transform to map the time series to a lower dimensional feature space.

All the algorithms were implemented in C++ using the GNU gcc compiler. We applied the new techniques to the Slim-tree MAM (TRAINA *et al.*, 2002), available in the Arboretum framework³, and implemented a new variant called *Hollow-tree* MAM. To evaluate the perfor-

³ <http://www.gbdi.icmc.usp.br/old/arboretum>

Figure 32 – Diagram of the experiments performed to evaluate the performance of the *Hollow-tree* metric access method.

mance of our techniques, we compared the results obtained from the *Hollow-tree* MAM with the results obtained from the Slim-tree MAM. For each dataset, including the complete ones, we built a metric tree based on the Slim-tree MAM, and a metric tree using the *Hollow-tree* MAM. The metric trees were built using the *minOccup* and *MSTSplit* policies, which are considered as the most optimal configurations by the authors (TRAINA *et al.*, 2002). The distances were computed using the *Euclidean distance*, which was modified to ignore NULL values and operate only on observed values. For every dataset, we randomly selected 500 query objects from the subset of complete objects and we processed the original k -NN_q query and the proposed k -NNFM_q and RM_q queries on the resulting *Hollow-tree* and Slim-tree indexes. The searching radius of the Range queries was obtained using the Equation 2.5, where $k = 50$. The number k of nearest neighbors was set to different values, 50, 100, 200 and 500, in order to highlight the performance gain of k -NNFM_q query over k -NN_q query. Figure 32 illustrates the diagram of the experiments performed to evaluate the performance of the *Hollow-tree* MAM.

To compute the fractal dimension of the datasets, we used the *box counting* algorithm presented in (TRAINA *et al.*, 2010). Figure 33 shows the Sierpinski datasets with different amounts of missing values and their respective fractal dimensions. Looking at the plots, it can be seen that the fitting lines are almost parallel to each other, which justifies employing the same value of the fractal dimension for all the datasets.

Figure 33 – Fractal dimension of the Sierpinski datasets with different amounts of missing data.

6.3 Performance Metrics

6.3.1 Effectiveness Parameters

We used the precision and recall parameters to evaluate the effectiveness of our techniques. The precision and recall for Range queries are defined as follows:

$$\begin{aligned} \text{Precision} &= |\text{Relevant complete objects retrieved}| / |\text{Total objects retrieved}| \\ \text{Recall} &= |\text{Relevant complete objects retrieved}| / |\text{Total relevant objects}| \end{aligned} \quad (6.1)$$

The precision and recall for k -nearest neighbor queries are defined as follows:

$$\begin{aligned} \text{Precision} &= |\text{Relevant complete objects retrieved}| / |\text{Total complete objects retrieved}| \\ \text{Recall} &= |\text{Relevant complete objects retrieved}| / |\text{Total relevant objects}| \end{aligned} \quad (6.2)$$

To obtain the measurements of precision and recall, a comparison of the results obtained from the complete datasets with the results obtained from the incomplete datasets is particularly useful. In fact, it can reveal how much performance are lost by the Slim-tree MAM with respect to the ideal situation where all the data are complete, and how much performance are preserved by the *Hollow-tree* MAM even with a high quantities of missing data. Notice that, precision and recall are measured using the complete objects retrieved only. The reason is that, there is no certainty that objects with NULL values are relevant.

6.3.2 Efficiency Parameters

To evaluate the efficiency parameters we measured the average number of disk accesses, the average number of distance calculations and the total time required to process the queries. We also estimated the average searching radius of the k - NN_q and k - $NNFM_q$ queries for further analysis.

6.4 Experimental Results

6.4.1 Effectiveness Assessment for k - NN_q and k - $NNFM_q$ Queries

Figures 34 and 35 show the average precision and recall for k - NN_q and k - $NNFM_q$ queries over the Weather dataset. The results show that k - NN_q and k - $NNFM_q$ queries achieve better performance when processed over *Hollow-tree* indexes than when using Slim-tree indexes, with different values of k .

Figures 36 and 37 show the average precision and recall for 50- NN_q and 50- $NNFM_q$ queries for the synthetic datasets. For most of the datasets, 50- NN_q and 50- $NNFM_q$ queries perform better when processed over *Hollow-tree* indexes than when processed over Slim-tree indexes. This is still valid with larger number of nearest neighbors (i.e., when $k \in \{100, 200, 500\}$), as it is shown in Figures 38 and 39, 40 and 41, 42 and 43, respectively. Notice that, the performance of k - NN_q and k - $NNFM_q$ queries are very similar when performed on Slim-tree indexes, and that both queries suffer considerably with the increasing amount of missing data. With Weath_20 and Sierp_20 datasets, 50- NN_q and 50- $NNFM_q$ queries achieve precision and recall smaller than 60%, and with Weath_40 and Sierp_40 datasets, they achieve only 40%. On the other hand, with Gaus32_40 and Gaus64_40 datasets, 50- NN_q and 50- $NNFM_q$ queries achieve precision and recall greater than 90%. However, when k increases, k - NN_q and k - $NNFM_q$ queries show lower performance for most of the datasets. For instance, with Gaus2_40, Gaus32_40 and Gaus64_40, 200- NN_q and 200- $NNFM_q$ queries achieved a recall smaller than 60%, while 500- NN_q and 500- $NNFM_q$ queries achieve only 40%. Thus, the precision rate remains considerably high, but precision only is not an indicator of a good performance.

At first, it would appear that the proportion of recall that is not achieved corresponds to the proportion of objects with NULL values that could be relevant if they had no NULL values. However, if we look at the performance of k - NN_q or k - $NNFM_q$ queries, when performed on *Hollow-tree* indexes, it becomes clear that these queries actually degenerate when performed on Slim-tree indexes. The evidence is that, both k - NN_q and k - $NNFM_q$ queries can achieve a higher performance with *Hollow-tree* indexes. More precisely, a high rate of recall achieved by both k - NN_q and k - $NNFM_q$ queries with *Hollow-tree* indexes means that the relevant complete objects to the query exist, but the queries were not able to reach them inside the Slim-tree indexes, which is due to the distortions caused by missingness in the index structures. Regarding the *Hollow-tree*

Figure 34 – Precision for k -NN $_q$ and k -NNFM $_q$ queries processed over *Hollow-tree* and *Slim-tree* indexes for Weather dataset: (a) $k=50$, (b) $k=100$, (c) $k=200$, (d) $k=500$.

indexes, with Sierp_40, Sph_40 and Squ_40 datasets, all the queries with the variations of k achieve more than 90% of precision and recall. This fact reveals that, contrarily to Slim-tree indexes, the internal structure of the *Hollow-tree* indexes is not distorted, which confirms the consistency and, consequently, the effectiveness of the new indexing technique.

Figure 35 – Recall for k -NN $_q$ and k -NNFM $_q$ queries processed over *Hollow-tree* and *Slim-tree* indexes for Weather dataset: (a) $k=50$, (b) $k=100$, (c) $k=200$, (d) $k=500$.

Figure 36 – Precision for 50-NN_q and 50-NNFM_q queries processed over *Hollow-tree* and *Slim-tree* indexes for synthetic datasets.

Figure 37 – Recall for 50-NN_q and 50-NNFM_q queries processed over *Hollow-tree* and *Slim-tree* indexes for synthetic datasets.

Figure 38 – Precision for 100-NN_q and 100-NNFM_q queries processed over *Hollow-tree* and *Slim-tree* indexes for synthetic datasets.

Figure 39 – Recall for 100-NN_q and 100-NNFM_q queries processed over *Hollow-tree* and *Slim-tree* indexes for synthetic datasets.

Figure 40 – Precision for 200- NN_q and 200- $NNFM_q$ queries processed over *Hollow-tree* and *Slim-tree* indexes for synthetic datasets.

Figure 41 – Recall for 200- NN_q and 200- $NNFM_q$ queries processed over *Hollow-tree* and *Slim-tree* indexes for synthetic datasets.

Figure 42 – Precision for 500- NN_q and 500- $NNFM_q$ queries processed over *Hollow-tree* and *Slim-tree* indexes for synthetic datasets.

Figure 43 – Recall for 500- NN_q and 500- $NNFM_q$ queries processed over *Hollow-tree* and *Slim-tree* indexes for synthetic datasets.

When comparing $k\text{-NN}_q$ and $k\text{-NNFM}_q$ queries performed over *Hollow-tree* indexes, as can be shown in Figures 36, 38, 40 and 42, it can be seen that $k\text{-NN}_q$ query slightly outperforms $k\text{-NNFM}_q$ query in terms of precision, with all the variations of k . This occurs when the desirable number of complete objects is overestimated and, consequently, $k\text{-NNFM}_q$ query retrieves more complete objects. On the other hand, if we observe Figures 35, 37, 39, 41 and 43, we can notice that $k\text{-NNFM}_q$ query outperforms $k\text{-NN}_q$ query significantly in terms of recall. This means that $k\text{-NNFM}_q$ query is more capable to find the relevant complete objects, which confirms that the final radius estimated for $k\text{-NNFM}_q$ query is very accurate.

6.4.2 Effectiveness Assessment for RM_q Query

Figures 44, 45 and 46 present the query performance of the proposed RM_q query. The results show clearly that the query performed over *Hollow-tree* indexes outperforms the query performed over *Slim-tree* indexes. For *Weath_40* dataset, RM_q query achieves more than 80% of precision and recall when processed using *Hollow-tree* indexes, while it reaches the worst case (only 50%) with *Slim-tree* indexes. With *Sierpinski*, *Sphere*, *Gaussian32* and *Gaussian64* datasets, RM_q query achieves nearly 100% of recall with the new indexing techniques, while it reaches less than 70% of recall with the traditional techniques. The accuracy of the query response confirms the consistency of the index structure obtained by the new indexing techniques.

Figure 44 – Precision and recall for RM_q query processed over *Hollow-tree* and *Slim-tree* indexes for Weather dataset: (a) precision, (b) recall.

Figure 45 – Precision for RM_q query processed over *Hollow-tree* and *Slim-tree* indexes for synthetic datasets. Figure 46 – Recall for RM_q query processed over *Hollow-tree* and *Slim-tree* indexes for synthetic datasets.

6.4.3 Efficiency Assessment for k - NN_q and k - $NNFM_q$ Queries

Here, we evaluate the quality of the k - NN_q and k - $NNFM_q$ queries. Figures 47 and 48 show the average searching radius, the total time execution, the average number of disk accesses and the average number of distance calculations for k - NN_q and k - $NNFM_q$ queries, where $k = 50$ and $k = 200$, respectively, for Weather dataset. Figures 49, 51, 53, 55 and Figures 50, 52, 54, 56 present the same parameters for k - NN_q and k - $NNFM_q$ queries, where $k = 50$ and $k = 200$, respectively, for the synthetic datasets.

Figure 47 – Efficiency parameters for 50- NN_q and 50- $NNFM_q$ queries for Weather dataset: (a) searching radius, (b) total time execution, (c) average number of disk accesses, (d) average number of distance calculations.

As can be seen in those figures, the searching radius of k - $NNFM_q$ query is greater than that of k - NN_q query, independently from the indexing techniques. In fact, k - NN_q query performs locally in the query region and suffers from the distance concentration caused by missingness even within the *Hollow-tree* indexes. Therefore, the distance between the query objects and the k^{th} nearest neighbor (i.e., the searching radius) remains relatively small. On the other hand, k - $NNFM_q$ query estimates a radius which goes beyond the local query region to search for

relevant complete objects until the desirable number is reached. The final radius is close to the ideal radius, and it allows a k -NNFM_q query to perform accurately even over datasets with high proportions of missing data. The total time execution, number of disk accesses and number of distance calculations of k -NNFM_q query are comparable to those of k -NN_q query. Also, the values of these parameters for incomplete datasets are comparable to their corresponding values obtained for complete datasets. For k -NNFM_q query, these parameters depend on the number of times k RingRange (recall the Definition 5 in Chapter 2) query is called to achieve the desirable number of complete objects. The results show that k RingRange query is called only a few times, which confirms the accuracy of the estimated searching radius. This fact indicates that the improvement of the k -NNFM_q query, regarding the effectiveness, is not achieved at the expense of its efficiency.

Figure 48 – Efficiency parameters for 200-NN_q and 200-NNFM_q queries for Weather dataset: (a) searching radius, (b) total time execution, (c) average number of disk accesses, (d) average number of distance calculations.

Figure 49 – Average searching radius for 50-NN_q and 50-NNFM_q queries for synthetic datasets. Figure 50 – Average searching radius 200-NN_q and 200-NNFM_q queries for synthetic datasets.

Figure 51 – Total execution time for 50-NN_q and 50-NNFM_q queries for synthetic datasets. Figure 52 – Total execution time for 200-NN_q and 200-NNFM_q queries for synthetic datasets.

Figure 53 – Average number of disk accesses for 50- NN_q and 50- $NNFM_q$ queries for synthetic datasets. Figure 54 – Average number of disk accesses for 200- NN_q and 200- $NNFM_q$ queries for synthetic datasets.

Figure 55 – Average number of distance calculations for 50-NN_q and 50-NNFM_q queries for synthetic datasets. Figure 56 – Average number of distance calculations for 200-NN_q and 200-NNFM_q queries for synthetic datasets.

6.5 Final Considerations

In this chapter we evaluated the performance of the new *Hollow-tree* MAM when applied to datasets with missing values. We compared the proposed $k\text{-NNFM}_q$ query with $k\text{-NN}_q$ query when processed using *Hollow-tree* indexes and when processed using Slim-tree indexes. When processed using Slim-tree indexes, both $k\text{-NN}_q$ and $k\text{-NNFM}_q$ queries suffer from distortions caused by missing data in the index structure during the indexing process. When processed using *Hollow-tree* indexes, $k\text{-NN}_q$ query still suffers from distance concentration caused by missingness. However, $k\text{-NNFM}_q$ query performs much better, achieving more than 90% of precision and recall. The results confirm that the ideal radius estimated for $k\text{-NNFM}_q$ query is very accurate. In addition, regarding the efficiency parameters, the results indicate that $k\text{-NNFM}_q$ query performs efficiently and that the improvement of the effectiveness parameters is not achieved at the expense of the efficiency parameters. We also compared the proposed RM_q query when processed using *Hollow-tree* indexes and when processed using Slim-tree indexes. The results showed that, when processed using *Hollow-tree* indexes, RM_q query performs much better and achieves nearly 100% recall, which confirms that the consistency is maintained in the internal structure of the improved indexes, and that the similarity queries exhibit better performance using the new indexing technique.

When employed together, the techniques developed in this work are capable to properly explore all the available data, even in the presence of high amounts of missing attribute values. One key aspect of those techniques is that they are independent of the underlying metric access method and, thus, they can be applied to a broad range of MAMs, allowing to extend the class of metric access methods to support incomplete datasets. To the best of our knowledge, this is the first solution proposed to overcome the limitations of the metric access methods when applied to datasets with missing attribute values.

CONCLUSION

Advanced database applications must deal with tremendous amounts of complex datasets. The way they are created and processed often lead to voids in the elements or in the attributes that describe them. Due to their complexity and high-dimensionality, similarity search became a vital task to store and search such data. Many access methods have been proposed in the literature, however, none of them are designed to perform on datasets with missing values. Initial approaches for missing data treatment were based on including indicator variables to identify missing values, originally developed for multidimensional access methods to support incomplete datasets. Unfortunately, these methods suffer from poor query performance, which is often worse than the sequential scan when the number of missing values is significantly large. Metric access methods are more suitable to process large and complex datasets. However, when the concept of similarity search is modeled by means of a distance function, the problem of missing data takes a new magnitude and metric access methods are not easily adaptable to perform on incomplete datasets. Therefore, special treatments for missing data are necessary to make the underlying metric access methods able to operate on incomplete datasets, while preserving their functionalities and performances. To our knowledge, this is the first work that discussed and addressed the problem of missing data when datasets with missing values are processed by MAMs, and addresses it by providing solutions to improve metric access methods, so they become able to support incomplete datasets.

7.1 Main Contributions

7.1.1 *Background of Missing Data*

In our first contribution, we presented a comprehensive background of missing data and the essential concepts that constitute the basis for the development of this Doctoral dissertation. We provided a classification of the methods proposed in the literature for missing data treatment and

discussed the advantages and disadvantages of each class. Based on this analysis, we set up the goals that motivated this project.

7.1.2 Formal Model for Missing Data in Metric Spaces

Since this work is the first attempt to deal with missing data in metric spaces, the initial investigation was conducted to provide a formal definition of missing data and describe the theory of missingness in metric spaces. We established a formal model for missing data based on the theoretical framework defined by the statisticians, and we demonstrated that missingness can have a direct effect on data distribution in metric spaces. As data structures for metric spaces are organized around representative elements and their distances to the rest of the elements in the space, analysis of the pairwise distances led us to discover important features about data distribution in the spaces. In the theoretical demonstration we provided an evidence that, if missing data at random do not cause skew in the parameter estimation, then missingness does not cause skew in the metric space, and if missing data not at random cause skew in the parameter estimation, then missingness can cause skew in the metric space.

7.1.3 Empirical Analysis for Missing Data in Metric Spaces

In order to support our theoretical basis about missingness in metric spaces, we conducted a set of experiments, based on real and synthetic datasets, to evaluate the effects of missingness on the distance distribution, when data are indexed in a metric space, considering MAR and MNAR mechanisms. The experimental results revealed that, with the MAR mechanism, objects move randomly in the space, changing neighborhood and introducing changes in the distribution of data while maintaining the degree of sparseness of the objects and the overall volume of the space. Since distances between objects with missing values are uncertain, changes in the distribution cause distortion and inconsistency in the indexed space, especially when representatives have missing values. With the MNAR mechanism, objects tend to remain in the clusters where missing data occurred and thus, preserve their neighborhood. However, MNAR data can only underestimate or overestimate the pairwise distances between the objects. Hence, objects move closer to each other in the clusters where missing data occurred, causing locally distance concentration, while moving further from other clusters, allowing to clusters to spread over a much broader space.

Metric access methods are sensitive to missingness and with just 2% of missing data they already yield poor query performance. The experimental results showed that the distortion caused by MAR mechanism affects the query performance in terms of precision and recall, leading to inaccurate query response. On the other hand, the distance concentration (i.e., skew) caused by MNAR mechanism degenerates the MAMs, as they cannot partition the space properly, increasing the node overlap. Recall that when nodes overlap, the number of paths traversed when a query is issued increases, which increases the number of distances computed and the number

of disk accesses. Therefore, all the benefits of the metric access methods will not be availed. Although, MAR data do not cause skew in the data space, this does not exclude the possibility of having distance concentration in some parts of the metric space. Alternatively, MNAR data are capable to cause severe skew in the data space, but are also capable to introduce distortion to the metric space, leading to inaccurate query response and poor query performance.

7.1.4 Indexing Technique for Datasets with Missing Values

After establishing a framework about missing data in metric spaces, we proposed a new technique for indexing incomplete datasets. While traditional indexing techniques fail to process elements with missing values, our technique is capable to handle datasets with large amounts of missing values, always warranting that no element is dismissed. The key aspect of this technique is that it effectively produces indexes with a consistent structure, and prevents elements with missing values from causing distortion in the metric space. To this intent, starting with an initial dataset, the complete elements are indexed first, building a consistent index, then the elements with missing values are inserted without introducing distortion to the initial index structure.

Preventing elements with missing values from causing distortion requires that they cannot be chosen as representatives. So, when they are indexed, an indicator variable is associated to every element to identify those who have missing values. Therefore, when a node is created, the representative is chosen among the complete elements only, and when a node overflows, the node is split into two nodes and the new representatives are selected among the complete elements.

After a consistent index is built, complete elements and elements with missing values can be safely inserted in the index without the risk of causing distortion. The new defined policies of *SplitNode* and *ChooseSubtree* procedures guarantee the requirements of the indexing technique. The policies support every type of missing data, including MAR and MNAR data. Besides being capable to process all the available data, they can significantly improve the index structure and reduce the node overlap caused by distance concentration.

7.1.5 Similarity Queries for Datasets with Missing Values

With the indexing technique, it is now feasible to search for complete data as well as for data with missing values. The k -Nearest Neighbor query and Range query were redefined to allow processing data elements with missing values, and both queries can be performed over the indexed dataset. Both queries return two lists: (i) a list of complete elements and (ii) a list of elements with missing values. The Range query for handling missing values, which we named $RM_q(s_q, r)$, is similar to the traditional query. It receives in parameter a query center s_q and a query radius r , and returns the objects that are in the covering radius. The result is divided into two lists: (i) The first list includes the complete elements that are certainly relevant and (ii) the second list holds the elements with missing values that might be relevant as well.

The k -Nearest Neighbor query is more sensitive to missingness. The reason is that closest elements are likely to be elements with missing values because the distances tend to be smaller. Therefore, since distances between elements with missing values are uncertain, there is no guarantee that the indexed nearest neighbor is the most similar to the query element. Based on this fact, we proposed a new variant of k -NN_q query, which we named k -NNFM_q, that employs the fractal dimension of the dataset to estimate a limiting radius where the k nearest neighbors should be probably located if all the stored elements were complete, then perform a k -NN_q query with the limiting radius. The local density around the query object is used to refine the searching process by estimating a more accurate limiting radius when the k nearest neighbors are not reached. This technique had already been used to speed up k -NN_q queries, however, our main reasoning here is to improve the accuracy of the query response when the query region is dominated by data with missing attribute values. In fact, with the estimated radius, a k -NNFM_q query is able to search beyond the local neighborhood that is dominated by elements with missing values, and retrieve the relevant elements. A desirable number of complete elements to be retrieved is used to refine the searching process, until at least this number is reached. The fractal dimension of the dataset is computed from the subset of complete elements only. Assuming that the mechanism of missingness is MAR, the subset of complete elements can be safely considered as a random sample and its fractal dimension is the same as the fractal dimension of the entire dataset. This technique applies for all the datasets with different amounts of missing data when missingness is random. The result of the k -NNFM_q query is returned in two lists: (i) The first list includes the complete elements that are certainly relevant and (ii) the second list holds the elements with missing values that might be also relevant.

7.1.6 Hollow-tree Access Method

We applied the proposed techniques to the Slim-tree access method and we obtained a new variant, called *Hollow-tree*, to accommodate data with missing attribute values. We implemented a functional prototype of the *Hollow-tree* and we conducted numerous experiments on real and synthetic datasets to assess its performance. We also compared our proposed method with the conventional Slim-tree access method to evaluate the gain of performance achieved by the new techniques. The experimental study revealed that, when the dataset has a sizable amount of complete objects, k -NNFM_q query, executed on *Hollow-tree* indexes, outperforms the traditional k -NN_q query by a significant margin. Moreover, RM_q query achieves nearly 100% of recall when processed on *Hollow-tree* indexes. Also the indicators of efficiency obtained from incomplete datasets are comparable to those achieved on complete datasets. A deterioration of the performance of the k -NNFM_q and RM_q queries can only be expected when applied on the traditional Slim-tree indexes. Therefore, k -NNFM_q and RM_q queries, executed over *Hollow-tree* indexes, can achieve a remarkable performance even with a significant amount of missing data.

7.1.7 Extended Family of Metric Access Methods

The proposed techniques for indexing and querying data with missing attribute values provide a support for the underlying access method to efficiently and effectively perform on incomplete datasets with MAR processes. Our solution is general and can be adopted by a broad range of metric access methods, allowing to extend the class of MAMs.

7.2 Future Work

There are several research topics that could further extend the research developed in this work, as described in the following:

- **Indexing and querying MNAR data:** Although the *Hollow-tree* is able to index datasets with missing values with any missingness mechanism, the way to query data with MNAR values may require evaluating not only the amount of missing data, but also the causes that led to missing values. As a future work, we intend to investigate new techniques to index and search incomplete datasets when attribute values are not missing at random.
- **Evaluating different data domains:** Missingness processes occur in different situations for different reasons. The reason data are missing depends mainly on the underlying data type and the application domain. Thus, it is conceivable that the mechanism of missingness is directly related to the data format chosen by the application experts to represent the information. Therefore, a more specific analysis of the data types employed to represent the basic information and how missing values can occur at such representation may provide clues on how the missingness in its turn affects the data distribution. As a future work, we propose to evaluate and conduct experiments on different types of data, such as, strings and graphs, using different distance functions.

In this work we evaluated data represented as time series and we assumed in the proposed approach that the data are represented in multidimensional vector spaces, where the dimensions are fixed. Thus, further research need to be conducted to index and retrieve time series and other types of data, where the dimensionality of the data space is not defined, and it is in fact a pure metric data space.

- **Considering diversity in similarity queries:** Several applications require that the similarity queries retrieve not only objects that are similar to the query center, but also the objects that are dissimilar among themselves. In fact, similarity queries with diversity aim at providing answers that cover regions of the data space that are beyond the regions that contain only almost identical objects. An interesting situation to explore the similarity queries with diversity when data have missing values is to provide a way to explore distinct regions of the space as well as a way to avoid local data retrieval from the regions where

missing data occurred. Therefore, a new type of query that considers similarity, diversity and missing values is worth to be studied.

- **Exploring missingness in other algebraic operators:** Similarity search is the basis to process queries on complex data, and they may affect every algebraic operator that constitutes the Relational Model. In this work, we evaluated the effect of missing data applied to a single comparison condition in each query. Thus, in addition to the select operator, the join and grouping operators are also important when similarity is used as a comparison criterion. The research reported in this work was focused on the select operator over missing data, but it would be interesting to evaluate how the developed concepts can be extended to other algebraic operators as well. In particular, we propose that the grouping operator can be extended, so analysis operations over large datasets with missing data may be conducted in a way that is almost insensitive to missing values in the data.

7.3 Publications Generated in this Ph.D. Work

This Ph.D. work generated two main papers. One of them, entitled "Analysis of Missing Data in Metric Spaces", corresponds to reference (BRINIS; TRAINA; TRAINA, 2014). It is a full paper published in the Journal of Information and Data Management (JIDM), and was presented in the Simpósio Brasileiro de Banco de Dados (SBBD'2014), one of the leading conferences in the database research area in Brazil. The second paper, entitled "*Hollow-tree* MAM: a metric access method for data with missing attribute values", corresponds to reference (BRINIS; TRAINA; TRAINA, 2016). An evaluation version of the paper was submitted to the Journal of Information Systems (IS), one of the top quality journals in the database research area, and is currently under evaluation.

BIBLIOGRAPHY

AGGARWAL, C.; PARTHASARATHY, S. Mining Massively Incomplete Data Sets by Conceptual Reconstruction. In: **International Conference on Knowledge Discovery and Data Mining**. San Francisco, CA, USA: [s.n.], 2001. p. 227–232. Cited on page 58.

ALLISON, P. D. **Missing Data**. London, UK: SAGE Publications, 2001. ISBN 9780761916727. Cited 3 times on pages 59, 63, and 64.

BAEZA-YATES, R.; CUNTO, W.; MANBER, U.; WU, S. Proximity Matching Using Fixed-queries Trees. In: **5th Annual Symposium on Combinatorial Pattern Matching**. Asilomar, CA, USA: [s.n.], 1994. p. 198–212. Cited on page 44.

BAYER, R.; MCCREIGHT, E. M. Organization and Maintenance of Large Ordered Indexes. **Acta Informatica**, v. 1, n. 3, p. 173–189, 1973. Cited on page 41.

BELUSSI, A.; FALOUTSOS, C. Estimating the Selectivity of Spatial Queries Using the Correlation Fractal Dimension. In: **21th International Conference on Very Large Data Bases**. Zurich, Switzerland: [s.n.], 1995. p. 299–310. Cited on page 49.

BENTLEY, J. L. Multidimensional Binary Search Trees Used for Associative Searching. **Communications of the ACM**, v. 18, n. 9, p. 509–517, 1975. Cited 2 times on pages 29 and 42.

_____. Multidimensional Binary Search Trees in Database Applications. **IEEE Transactions on Software Engineering**, v. 5, n. 4, p. 333–340, 1979. Cited 2 times on pages 29 and 42.

BERCHTOLD, S.; BOHM, C.; BRAUNMULLER, B.; KEIM, D. A.; KRIEGEL, H. Fast Parallel Similarity Search in Multimedia Databases. In: **ACM SIGMOD International Conference on Management of Data**. Tucson, Arizona, USA: [s.n.], 1997. p. 1–12. Cited on page 49.

BERCHTOLD, S.; BÖHM, C.; KRIEGEL, H.-P. The Pyramid-technique: towards breaking the curse of dimensionality. In: **ACM SIGMOD international conference on Management of data**. Seattle, Washington, USA: [s.n.], 1998. p. 142–153. Cited on page 49.

BöHM, C.; KRIEGEL, H. Dynamically Optimizing High-Dimensional Index Structures. In: **7th International Conference on Extending Database Technology**. Konstanz, Germany: [s.n.], 2000. p. 36–50. Cited on page 49.

BOZKAYA, T.; ÖZSOYOGLU, M. Indexing Large Metric Spaces for Similarity Search Queries. **ACM TODS**, v. 24, n. 3, p. 361–404, 1999. Cited on page 44.

BOZKAYA, T.; ÖZSOYOGLU, Z. Distance-based Indexing for High-dimensional Metric Spaces. In: **ACM SIGMOD International Conference on Management of Data**. Tucson, Arizona, USA: [s.n.], 1997. p. 357–368. Cited on page 44.

BRIN, S. Near Neighbor Search in Large Metric Spaces. In: **21th International Conference on Very Large Data Bases**. Zurich, Switzerland: [s.n.], 1995. p. 574–584. Cited on page 44.

BRINIS, S.; TRAINA, A. J. M.; TRAINA, C. Analyzing Missing Data in Metric Spaces. **Journal of Information and Data Management**, v. 5, n. 3, p. 224–237, 2014. Cited on page 130.

BRINIS, S.; TRAINA, C.; TRAINA, A. J. M. **Hollow-tree MAM: a metric access method for data with missing attribute values**. 2016. Cited on page 130.

BRUNO, O. M.; PLOTZE de R. O.; FALVO, M.; CASTRO, M. de. Fractal Dimension Applied to Plant Identification. **Information Sciences**, v. 178, n. 12, p. 2722–2733, 2008. Cited on page 49.

BURKHARD, W. A.; KELLER, R. M. Some Approaches to Best-match File Searching. **Communications of the ACM**, v. 16, n. 4, p. 230–236, 1973. Cited on page 43.

CANAHUATE, G.; GIBAS, M.; FERHATOSMANOGLU, H. Indexing Incomplete Databases. In: **International Conference on Advances in Databases**. Munich, Germany: [s.n.], 2006. p. 884–901. Cited 5 times on pages 34, 58, 65, 66, and 67.

CARÉLO, C. C. M.; VENTURINI, I. R. P.; CIFERRI, R. R.; TRAINA, A. J. M.; TRAINA, C.; CIFERRI, C. D. de A. Slicing the Metric Space to Provide Quick Indexing of Complex Data in the Main Memory. **Information Systems**, v. 36, n. 1, p. 79–98, 2011. Cited on page 45.

CHAUDHURI, B. B.; SARKAR, N. Texture Segmentation Using Fractal Dimension. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 17, n. 1, p. 72–77, 1995. Cited on page 49.

CIACCIA, P.; PATELLA, M.; ZEZULA, P. M-tree : An Efficient Access Method for Similarity Search in Metric Spaces. In: **23rd International Conference on Very Large Data Bases**. San Francisco, CA, USA: [s.n.], 1997. p. 426–435. Cited on page 44.

CLEARY, J. G.; TRIGG, L. E. K^* : An Instance-based Learner Using and Entropic Distance Measure. In: **Machine Learning, Proceedings of the Twelfth International Conference on Machine Learning**. Tahoe City, California, USA: [s.n.], 1995. p. 108–114. Cited on page 68.

DEHNE, F. K. H. A.; NOLTEMEIER, H. Voronoi Trees and Clustering Problems. **Information Systems**, v. 12, n. 2, p. 171–175, 1987. Cited on page 44.

DIXON, J. K. Pattern Recognition with Partly Missing Data. **IEEE Transactions on Systems, Man, and Cybernetics**, v. 9, n. 10, p. 617–621, 1979. Cited on page 68.

EIROLA, E.; DOQUIRE, G.; VERLEYSEN, M.; LANDASSE, A. Distance Estimation in Numerical Data Sets with Missing Values. **Information Sciences**, v. 240, p. 115–128, 2013. Cited 2 times on pages 58 and 68.

FALOUTSOS, C.; BARBER, R.; FLICKNER, M.; HAFNER, J.; NIBLACK, W.; PETKOVIC, D.; EQUITZ, W. Efficient and Effective Querying by Image Content. **Journal of Intelligent Information Systems**, v. 3, n. 3-4, p. 231–262, 1994. Cited on page 39.

FALOUTSOS, C.; KAMEL, I. Beyond Uniformity and Independence: Analysis of R-trees Using the Concept of Fractal Dimension. In: **Thirteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems**. New York, NY, USA: [s.n.], 1994. p. 4–13. Cited on page 49.

FALOUTSOS, C.; SEEGER, B.; TRAINA, A. J. M.; TRAINA, C. Spatial Join Selectivity Using Power Laws. In: **ACM SIGMOD international conference on Management of data**. New York, NY, USA: [s.n.], 2000. p. 177–188. Cited 2 times on pages 49 and 50.

FILHO, R. F. S.; TRAINA, A. J. M.; TRAINA, C.; FALOUTSOS, C. Similarity Search Without Tears: The OMNI family of all-purpose access methods. In: **IEEE International Conference on Data Engineering**. Heidelberg, Germany: [s.n.], 2001. p. 623–630. Cited on page 45.

GAEDE, V.; GÜNTHER, O. Multidimensional access methods. **ACM Computing Surveys**, v. 30, n. 2, p. 170–231, 1998. Cited 2 times on pages 29 and 42.

GRZYMALA-BUSSE, J. W.; HU, M. A Comparison of Several Approaches to Missing Attribute Values in Data Mining. In: **2nd International Conference on Rough Sets and Current Trends in Computing**. London, UK: [s.n.], 2001. p. 378–385. Cited on page 57.

GUTTMAN, A. R-trees: a dynamic index structure for spatial searching. In: **ACM SIGMOD International Conference on Management of Data**. New York, NY, USA: [s.n.], 1984. p. 47–57. Cited 3 times on pages 29, 42, and 65.

HAFNER, J.; SAWHNEY, H. S.; EQUITZ, W.; FLICKNER, M.; NIBLACK, W. Efficient Color Histogram Indexing for Quadratic Form Distance Functions. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 17, n. 7, p. 729–736, 1995. Cited on page 39.

HIMMELSPACH, L.; CONRAD, S. Clustering approaches for data with missing values: Comparison and evaluation. In: **Fifth IEEE International Conference on Digital Information Management**. Lakehead University, Thunder Bay, Canada: [s.n.], 2010. p. 19–28. Cited 2 times on pages 58 and 68.

HJALTASON, G. R.; SAMET, H. Index-driven Similarity Search in Metric Spaces. **ACM Transactions on Database Systems**, v. 28, n. 4, p. 517–580, 2003. Cited 2 times on pages 29 and 43.

HULSE, J. V. Incomplete-Case Nearest Neighbor Imputation in Software Measurement Data. In: **Proceedings of the IEEE International Conference on Information Reuse and Integration**. Las Vegas, NV, USA: [s.n.], 2007. p. 630–637. Cited 2 times on pages 58 and 64.

IDA, Y. S. T. Image segmentation and contour detection using fractal coding. **IEEE Transactions on Circuits System Video Technology**, v. 8, n. 8, p. 968–977, 1998. Cited on page 49.

IMIELINSKI, T.; JR., W. L. Incomplete Information in Relational Databases. **Journal of the ACM**, v. 31, n. 4, p. 761–791, 1984. Cited 2 times on pages 59 and 62.

JONSSON, P.; WOHLIN, C. An Evaluation of k-Nearest Neighbour Imputation Using Likert Data. In: **Procedings of the 10th IEEE International Softwar Metrics Symposium**. Chicago, Illinois, USA: [s.n.], 2004. p. 108–118. Cited 2 times on pages 58 and 64.

KALANTARI, I.; MCDONALD, G. A Data Structure and An Algorithm for the Nearest Point Problem. **IEEE Transactions on Software Engineering**, v. 9, n. 5, p. 631–634, 1983. Cited on page 44.

KAMEL, I.; FALOUTSOS, C. Hilbert R-tree: An Improved R-tree using Fractals. In: **20th International Conference on Very Large Data Bases**. Santiago de Chile, Chile: [s.n.], 1994. p. 500–509. Cited on page 49.

LEVENSHTEIN, V. I. Binary Codes Capable of Correcting Spurious Insertions and Deletions of Ones. **Problems of Information Transmission**, v. 1, p. 8–17, 1965. Cited on page 39.

LITTLE, R. J. A.; RUBIN, D. B. **Statistical Analysis with Missing Data**. Hoboken: Wiley, 2002. ISBN 1118625862. Cited 3 times on pages 28, 57, and 64.

MAGNANI, M. **Techniques for Dealing with Missing Data in Knowledge Discovery Tasks**. Bologna, ITALY, 2004. 10 p. Cited on page 61.

MALLAT, S. G. A Theory for Multiresolution Signal Decomposition: the wavelet representation. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 11, n. 7, p. 674–693, 1989. Cited 2 times on pages 72 and 80.

MANDELBROT, B. B. **The Fractal Geometry of Nature**. San Francisco, CA, USA: W. H. Freeman and Company, 1983. ISBN 9780716711865. Cited on page 48.

NEIL, G.; CURTIS, K. Shape Recognition Using Fractal Dimension. **Pattern Recognition** 30, v. 30, n. 12, p. 1957–1969, 1997. Cited on page 49.

NOLTEMEIER, H.; VERBARG, K.; ZIRKELBACH, C. A Data Structure for Representing and Efficient Querying Large Scenes of Geometric Objects: MB* Trees. In: **Geometric Modeling**. Dagstuhl, Germany: [s.n.], 1992. p. 211–226. Cited on page 44.

_____. Monotonous Bisector* Trees: a tool for efficient partitioning of complex scenes of geometric objects. In: **Data Structures and Efficient Algorithms**. [S.l.: s.n.], 1992. p. 186–203. Cited on page 44.

OOI, B. C.; GOH, C. H.; TAN, K.-L. Fast High-Dimensional Data Search in Incomplete Databases. In: **International Conference on Very Large Data Bases**. New York, NY, USA: [s.n.], 1998. p. 357–367. Cited 5 times on pages 34, 58, 65, 66, and 78.

OSBORNE, J. W. Dealing with Missing or Incomplete Data: Debunking the Myth of Emptiness. In: **Best Practices in Data Cleaning: A Complete Guide to Everything You Need to Do Before and After Collecting Your Data**. [S.l.]: Sage Publications, Inc, 2012. p. 105–138. Cited 5 times on pages 27, 59, 60, 63, and 64.

PAGEL, B.-U.; KORN, F.; FALOUTSOS, C. Deflating the Dimensionality Curse Using Multiple Fractal Dimensions. **IEEE Transactions on Knowledge and Data Engineering**, v. 13, n. 3, p. 589–598, 2000. Cited on page 49.

PAPADOPOULOS, A.; MANOLOPOULOS, Y. Performance of Nearest Neighbor Queries in R-trees. In: **6th International Conference on Database Theory**. Delphi, Greece: [s.n.], 1997. p. 394–408. Cited on page 49.

PEARSON, R. K. The Problem of Disguised Missing Data. **ACM SIGKDD Explorations Newsletter**, v. 8, n. 1, p. 83–92, 2006. Cited 2 times on pages 59 and 63.

PENG, L.; LEI, L. A Review of Missing Data Treatment Methods. **Intelligent Information Management Systems and Technologies**, v. 1, n. 3, p. 412–419, 2005. Cited on page 63.

PENTLAND, A. P. Fractal-Based Description of Natural Scenes. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 6, n. 6, p. 661–674, 1984. Cited on page 48.

REITER, R. Towards a Logical Reconstruction of Relational Database Theory. In: **On Conceptual Modelling**. Berlin, Germany: [s.n.], 1984. p. 198–238. Cited on page 62.

_____. A Sound and Sometimes Complete Query Evaluation Algorithm for Relational Databases with Null Values. **Journal of the ACM**, v. 33, n. 2, p. 349–370, 1986. Cited on page 62.

RUBIN, D. B. Inference and Missing Data. **Biometrika**, v. 63, n. 3, p. 581–592, 1976. Cited 4 times on pages 28, 57, 60, and 77.

_____. **Multiple Imputation for Nonresponse in Surveys**. Hoboken: Wiley, 2004. ISBN 0-471-65574-0. Cited 2 times on pages 28 and 57.

SAMET, H. The Quad-tree and Related Hierarchical Data Structures. **ACM Computing Surveys**, v. 16, n. 2, p. 187–260, 1984. Cited 2 times on pages 29 and 42.

_____. **Foundations of Multidimensional and Metric Data Structures**. San Francisco, CA, USA: Morgan Kaufmann, 2006. ISBN 9780123694461. Cited 3 times on pages 29, 42, and 43.

SCHAFER, J. L. **Analysis of Incomplete Multivariate Data**. London, UK: Chapman and Hall, 1997. ISBN 0-412-04061-1. Cited 3 times on pages 28, 57, and 61.

SCHROEDER, M. **Fractals, Chaos, Power Laws**. New York, USA: W. H. Freeman and Company, 1991. ISBN 9780486472041. Cited on page 49.

SEIDL, T.; KRIEGL, H.-P. Efficient User-adaptable Similarity Search in Large Multimedia Databases. In: **The VLDB Journal**. San Francisco, CA, USA: [s.n.], 1997. p. 506–515. Cited on page 39.

SOLIMAN, M. a.; ILYAS, I. F.; BEN-DAVID, S. Supporting Ranking Queries on Uncertain and Incomplete Data. **The VLDB Journal**, v. 19, n. 4, p. 477–501, 2010. Cited on page 62.

TRAINA, C.; TRAINA, A. J. M.; FALOUTSOS, C. Distance Exponent: a New Concept for Selectivity Estimation in Metric Trees. In: **Technology IEEE International Conference on Data Engineering (ICDE)**. San Diego - CA: [s.n.], 2000. p. 195. Cited on page 49.

TRAINA, C.; TRAINA, A. J. M.; FALOUTSOS, C.; SEEGER, B. Fast Indexing and Visualization of Metric Data Sets using Slim-Trees. **IEEE Transactions on Knowledge and Data Engineering**, v. 14, n. 2, p. 244–260, 2002. Cited 6 times on pages 34, 45, 48, 80, 106, and 107.

TRAINA, C.; TRAINA, A. J. M.; WU, L.; FALOUTSOS, C. Fast Feature Selection Using Fractal Dimension. **Journal of Information and Data Management**, v. 1, p. 3–16, 2010. Cited 2 times on pages 49 and 107.

UHLMANN, J. K. Satisfying General Proximity/Similarity Queries with Metric Trees. **Information Processing Letters**, v. 40, n. 10, p. 175–179, 1991. Cited 3 times on pages 42, 43, and 44.

VENTURINI, I. R. P.; TRAINA, A. J. M.; TRAINA, C. The MM-tree: A memory-based metric tree without overlap between nodes. In: **East-European Conference on Advances in Databases and Information Systems**. Varna, Bulgaria: [s.n.], 2007. p. 157–171. Cited on page 45.

VIEIRA, M. R.; JR., C. T.; CHINO, F. J. T.; TRAINA, A. J. M. DBM-Tree: A Dynamic Metric Access Method Sensitive to Local Density Data. **Journal of Information and Data Management**, v. 1, p. 111–128, 2010. Cited 2 times on pages 45 and 94.

VIEIRA, M. R.; TRAINA, C.; TRAINA, A. J. M.; ARANTES, A.; FALOUTSOS, C. Boosting k-Nearest Neighbor Queries Estimating Suitable Query Radii. In: **19th International Conference on Scientific and Statistical Database Management, SSDBM**. Los Alamitos, CA, USA: [s.n.], 2007. p. 10. Cited 2 times on pages [49](#) and [52](#).

YIANILOS, P. N. Data Structures and Algorithms for Nearest Neighbor Search in General Metric Spaces. In: **ACM-SIAM Symposium on Discrete Algorithms**. Austin, USA: [s.n.], 1993. p. 311–321. Cited on page [44](#).

ZIMANYI, E. **Incomplete and Uncertain Information in Relational Databases**. 280 p. Phd Thesis (PhD Thesis) — Université Libre de Bruxelles, 1992. Cited on page [62](#).