Charles University in Prague

Faculty of Mathematics and Physics

DOCTORAL THESIS

RNDr. Tomáš Bartoš

Indexing Arbitrary Similarity Models

Department of Software Engineering

Supervisor of the doctoral thesis:	doc. RNDr. Tomáš Skopal Ph.D.
Study programme:	Computer Science
Specialization:	Software Systems

Prague 2014

I would like to thank my supervisor doc. RNDr. Tomáš Skopal, Ph.D. for his valuable comments, insights, and continuous assistance during my study. I am also very grateful for the endless support from my wife Stanka, my parents, and my family who encourage me and always stand by me.

Annotation

Title:
Indexing Arbitrary Similarity Models

Author:
RNDr. Tomáš Bartoš
email: bartos@ksi.mff.cuni.cz

Department:
Department of Software Engineering
Faculty of Mathematics and Physics
Charles University in Prague

Supervisor:
doc. RNDr. Tomáš Skopal, Ph.D.
email: skopal@ksi.mff.cuni.cz

Abstract:
The performance of similarity search in the unstructured databases largely depends on the employed similarity model. The properties of *metric space model* enable indexing the data with metric access methods efficiently. But for unconstrained or nonmetric similarity models typical for multimedia, medical, or scientific databases, in which metric postulates do not hold, there exists no general solution so far.

Motivated by the successful application of *Ptolemaic indexing* to the image retrieval, we introduce SIMDEX Framework which is a universal framework that is capable of revealing alternative indexing methods that will serve for efficient yet effective similarity searching for any similarity model. It explores the axiom space in order to discover novel techniques suitable for database indexing. We review all existing variants (simple `I-SIMDEX`; `GP-SIMDEX` and `PGP-SIMDEX` which both use genetic programming) and we outline how the different groups of domain researchers can benefit from them.

We also describe a real application of SIMDEX Framework to practice while building the *Smart Pivot Table* indexing method together with advanced Triangle$^+$ filtering for metric spaces empowered by *LowerBound Tightening* technique. At all cases, we provide extensive experimental evaluations of mentioned techniques.

Keywords:
indexing similarity models, metric and nonmetric access methods, symbolic regression, genetic programming, simdex

Anotace

Název práce:
Indexovanie podobnostných modelov

Autor:
RNDr. Tomáš Bartoš
email: bartos@ksi.mff.cuni.cz

Katedra:
Katedra softwarového inženýrství
Matematicko-fyzikální fakulta
Univerzita Karlova v Praze

Školitel:
doc. RNDr. Tomáš Skopal, Ph.D.
email: skopal@ksi.mff.cuni.cz

Abstrakt:
Charakteristika výkonu podobnostného vyhľadávania v neštruktúrovaných databázach záleží od použitého podobnostného modelu. Vlastnosti metrických priestorov nám umožňujú efektívne indexovať dáta pomocou tzv. *metrických prístupových metód*. Ale pre prípad nemetrických priestorov, ktoré sú typické pre multimediálne, medicínske a vedecké databázy, a v ktorých neplatia axiómy metrických priestorov, zatiaľ nepoznáme všeobecné riešenie.

Na základe úspešnej aplikácie ptolemaického modelu indexovania, predstavujeme SIMDEX Framework, univerzálny nástroj, ktorý dokáže objaviť alternatívne metódy indexácie dát za účelom efektívneho podobnostného vyhľadávania pre ľubovoľný podobnostný model. Na pozadí prehľadáva priestor platných axióm tak, aby našiel nové techniky určené pre indexovanie databáz. Preskúmame všetky existujúce varianty (prostý `I-SIMDEX`; `GP-SIMDEX` a `PGP-SIMDEX` využívajúce genetické programovanie) a zhodnotíme ich prínos a použitie v praxi pre profesionálov v rozličných doménach.

Nakoniec opíšeme konkrétnu aplikáciu SIMDEX Framework-u v praxi na vytvorenie indexu *Smart Pivot Table* s pokročilým filtrovaním pre metrické priestory (*Triangle*$^+$ *filtering*) spoločne s technikou na zlepšovanie kvality filtrovania (*LowerBound Tightening*). Vo všetkých prípadoch uvádzame aj experimentálne vyhodnotenie a porovnanie spomínaných metód.

Klíčová slova:
podobnostné vyhľadávanie a indexovanie, metrické a nemetrické prístupové metódy, symbolická regresia, genetické programovanie, simdex

Contents

Part I

Introduction and Similarity Search

Chapter 1

Introduction

Finding the right information in large or growing databases within the acceptable time frame is crucial in almost every area. For structured data (e.g., data stored to tables in relational databases with the predefined structure) and simple unstructured content types such as text documents, the efficient querying methods are known for decades. On the other hand, the currently popular *unstructured* databases such as multimedia databases, social network data, biometric, medical or scientific databases, which affects our daily lives to a considerable extent, are more difficult to search or explore due to the higher complexity of stored objects.

Therefore, we use *content-based retrieval* [1, 2], which for querying purposes converts the objects from their native formats to more appropriate forms. This approach extracts the important information and creates *object descriptors* which provide the form of object representations usable for searching models. A popular type of such a mechanism is the *similarity search* principle [3] in which, given a sample query object (e.g., an image), the database engine searches for the most similar objects (images). In fact, searching collections of a priori unstructured data entities requires a kind of aggregation that ranks the data as more or less relevant to a query – the *similarity function* which depends on the type of dataset and on the application we deal with.

During the querying, users explore the underlying database and handle the database objects in different ways which results in the range of simple to complex *similarity models*. The current trends such as Big Data [4, 5] leads us to the challenge of finding information in large-scale databases of unstructured data. We know how to model data, how to store it, and which similarity model provides the best results for specific databases when searching for the most similar objects to the given query. However, we still struggle with the speed of query evaluations and need to optimize the query efficiency with respect to the quality of results. With the increasing number of users trying to find information in growing collections of unstructured data, there is a huge pressure on the database performance, so efficient indexing techniques for similarity search are required.

For quite a long time, the database-oriented research of similarity search employed the definition of similarity restricted to the *metric space model* [6, 3, 7]. Due to the fixed properties of *identity*, *positivity*, *symmetry*, and especially the *triangle inequality*, metric similarity functions enable to index the given database for efficient querying using *metric access methods* or *metric indexes* [6, 7, 3], preventing thus from searching the whole database sequentially.

Together with the increasing complexity of data types across various domains and due to very specific user demands, recently there also appeared many *non-metric* or *unconstrained* similarity functions [8, 9, 10]. As the nonmetric similarity functions are generally not constrained by any properties that need to be satisfied (unlike the metric ones), they allow us to better model the desired concept of similarity, better address particular issues in some domains, and therefore lead to more precise retrieval. While metric models usually lack some of the anticipated features, the nonmetric models provides simply a better fit.

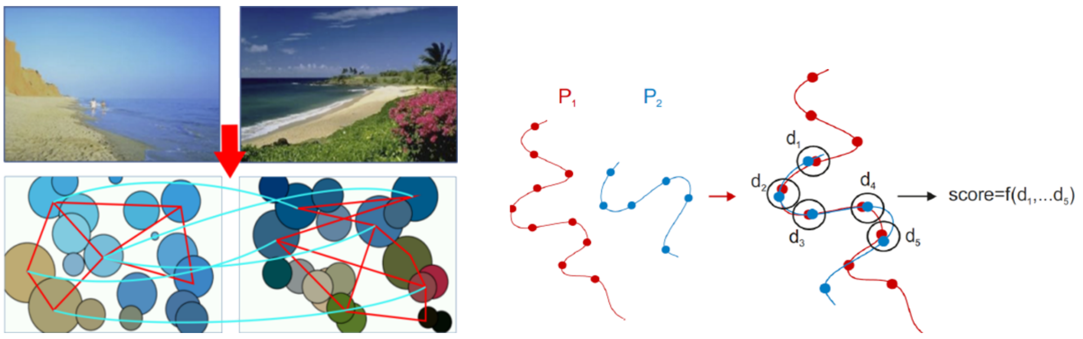
Because these functions rarely fit the strong assumptions of the metric space, there is a question how to handle distinct (generally nonmetric) similarity models in a unified way in order to achieve superior performance during query evaluations [8, 10]. If we do not address this issue, each query will finally degrade to slow sequential scanning which is applicable only to small-sized datasets. For growing datasets, long waiting time (hours or days) in order to obtain the anticipated results is simply unacceptable.

In summary, the increasing diversity of unstructured databases leads to the development of advanced indexing techniques as the metric indexing model does not fit to the general similarity models. Once the most critical metric postulate, namely the *triangle inequality*, does not hold in these models, any metric access method produces notable errors during the query evaluation. To overcome this situation and to obtain better qualitative results, we want to discover better indexing models for databases using arbitrary similarity measures. However, each database is unique in a specific way, so we do not consider manual methods and would prefer an automatic way of exploring the best indexing method. In our work, we introduce and compare several variants of the recently developed SIMDEX Framework which allows the automatized discovery of new indexing methods in a consistent way.

1.1 Motivation

Nowadays, we identify two types of research groups concerned with different aspects of similarity search – *database experts* and *domain experts*. The database experts deal with performance issues of similarity search and (mostly) do not care of particular domain applications. They just assume a similarity model that is constrained by some specific properties useful for database indexing, such as the *metric space* axioms, because these properties provide the ways of indexing the database for efficient similarity search. But these researchers typically do not investigate the applicability of their techniques to specific domains.

On the other hand, there are much larger *domain expert* communities of different kinds (e.g., computational biologists or various scientists), people who use the specialized similarity search applications and are ready to apply any method in order to get expected results quickly. These experts model similarities for specific practically oriented applications, while they do not (like to) care of any database-specific requirements such as applicable indexing techniques or performance issues to a certain extent, as this is completely outside of their expertise. As the result, the best approach for them is to use the simplest (possibly inefficient) database methods as they are easy to implement.



(a) Image similarity (b) Protein similarity

Figure 1.1: Sample similarity models

Besides the simple similarity models for which there exist suitable indexing techniques, domain experts often develop or come up with similarity models involving more sophisticated features or complex similarity functions. Such models better reflect the desired concept of similarities and lead to more effective/precise retrieval. For example, see Fig. 1.1a for a sketch of robust matching using local image features. Naturally, the more complex similarity function the domain experts come with, the lower the likelihood is that it will be a *metric* model which satisfies metric space postulates. Therefore, we need to focus generally on both *metric* and *nonmetric* similarity models.

More complex and nonmetric similarities allow to design models that cannot be formalized into a closed-form equation. They could be defined as heuristic algorithms such as a specific alignment or a transformational procedure, while the enforcement of metric axioms could be very difficult or even impossible. For example, alignment algorithms for measuring functional similarity of the protein sequences [11] or protein structures [12] (see Fig. 1.1b for sample alignments of proteins).

Therefore only the simplest similarity models comply with the objectives of both expert groups. However, in the long term and with large-scale databases, the efficiency will become a critical factor when choosing suitable indexing methods for similarity search. Because of the not really integrated research efforts, both groups (database experts and domain experts) head into trouble in the near future due to these reasons:

- **Database research** – Current efficient solutions for constrained similarity models (e.g., based on the metric space model) might not be applicable to the future state-of-the-art similarity search problems. Simply, the database technology might provide only solutions for trivial or obsolete models.
- **Research in various domains** – assumes the slow sequential search as more-or-less sufficient. As the models become very complex and/or the databases become so large-scale, an efficient database solution would be one of the most critical requirements.

Although both research communities have different perspectives, perceptions, and objectives, there is a common goal to get/extract the relevant information from the database quickly and efficiently. This ultimate goal is our main objective and provides the basis of our research.

The major challenge is to discover a complex solution in terms of general applicability that provides various domain experts with database techniques that speedup similarity search yet that do not require any database-specific intervention to the generally unconstrained similarity models.

We start with the database perspective of finding and building a general and easy-to-use tool that employ state-of-the-art techniques for general similarity models. Then we explore additional improvement possibilities for the specific domains regarding the database indexing efficiency. We will focus on cross-domain applicability of complex (yet effective and efficient) methods also to researchers outside the database area.

Before we outline the fundamental steps towards creating the algorithmic framework for flexible similarity searching applicable to general similarity models (Section 4) and describe the ground-breaking nature of the SIMDEX Framework, we shortly summarize the state of the art techniques (Section 3) and the previous attempts to unconstrained similarity search (Section 3.2).

1.2 Contributions

In our research, we revisit the existing state-of-the-art techniques for indexing unstructured databases with the aim of efficient and effective similarity searching and present the foundation for indexing general and unconstrained similarity models not necessary limited to metric space. We describe SIMDEX Framework, our recently introduced algorithmic framework for flexible indexing/similarity searching of various similarity models together with all its existing variants and provide extensive experimental evaluations that validates its feasibility.

Hence, our contribution and impact are twofold depending on the point of view. First, from the technical point of view, the most beneficial outcome is for the database research because we introduce the framework for the discovery of new indexing methods. It is applicable to any database and it will enable large-scale similarity search in many applications where content-based retrieval is the essential component, e.g., multimedia retrieval, time series databases, biometric databases, etc.

Consequently, other domain experts such as computational biologists could benefit from while creating models for practically oriented applications using database techniques. Since applications come from different domains outside the computer science, the contribution of our outcomes is truly multi-disciplinary.

We describe into details the theoretical concept of the proposed algorithmic framework SIMDEX (Section 4), apply and validate the theoretical principles by implementing three different SIMDEX variants, namely:

- `I-SIMDEX` – naive implementation of the iterative exploration (Section 5)
- `GP-SIMDEX` – guided exploration using the *genetic programming* (Section 6)
- `PGP-SIMDEX` – exploring with parallelized genetic programming (Section 7)

Moreover, with the implemented and working prototypes of all framework variants, we not only validate the theoretical concept but also provide the extensive experimental evaluations for real-world data with various similarity models.

Afterwards, we outline a specific application of SIMDEX Framework and use it to improve Pivot Table [3] query evaluation. As the result, we obtain *Smart Pivot Table* concept (Section 8) together with *Triangle*$^+$ *lowerbounding* (Section 8.2).

As an additional outcome, we introduce *LowerBound Tightening* approach (Section 8.4) which can be applied orthogonally to any of the existing SIMDEX variants as the post-processing phase in order to improve resulting lowerbounds. Nevertheless, this technique is empowered by *expression tightening algorithm* (see Section 8.4.2) and experimental evaluations validate that it provides a superior performance compared to existing methods (such as triangle lowerbounding or TriGen) also by simply tightening the standard triangle lowerbound $LB_{\triangle}$ (see Section 8.4.4).

Secondly, from the philosophical point of view, the newly discovered indexing methods (or *axioms*) contribute to the theoretic foundations of data engineering, data mining and disciplines beyond, such as computational geometry, geometric topology, and related disciplines. If a discovered axiom is general enough, it could open new horizons or research interests in many disciplines related to data engineering, similarity search, data mining, etc. and so SIMDEX Framework exhibits substantial inter-disciplinary nature.

Finally, our consolidated review leads towards universal and flexible indexing of unconstrained similarity models.

Chapter 2

Similarity Search Essentials

The principles of the *similarity search* [3] are based on the extraction of important objects features (*object descriptors*) combined with the similarity measure which we use for query evaluations. Here, the general concept is called *content-based retrieval* [1, 2] which uses the query object (could be outside of the database) and searches for most similar objects from the database.

As the first (pre-processing) step we need to obtain the mapping function φ that converts the complex data objects into the n-dimensional (typically real-valued vector) representations of object features

$$\varphi : Obj \to \mathbb{R}^n$$

Then, instead of dealing with complex objects, we work with simpler object features. If we take the multimedia objects, the most popular conversion functions transform them into global MPEG-7 visual descriptors [13] or use domain-specific features such as color, textures, and shapes for image [2]. Mostly due to the increased interest of end-users, there continually appears new representations of multimedia objects, that are suitable for specific problems such as the *feature signatures* for images [14, 15, 16] in order to allow visual image search (see Fig. 2.1 for the graphical overview of derived signatures for sample images).

We put a strong emphasis on the query performance while retrieving the objects. We prefer the *effectiveness* (qualitative and precise query results) and the *efficiency* (the speed of evaluation). While the first attribute depends mainly on the employed similarity model, we can tweak the query performance by choosing fast but approximate results over slow yet exact results.

2.1 Similarity Queries

The most popular similarity queries are the *range query* (see Figure 2.2a) that for a given query object q returns all database objects o_i that are similar to the query object to some extent which is defined by the radius r

$$\mathrm{RQ}_\delta(q, r) = \{o \in \mathcal{D} : \delta(q, o) \le r\} \tag{2.1}$$

and the kNN queries (see Fig. 2.2b) that return k most similar objects from the database:

$$k\mathrm{NN}_\delta(q) = \{R \subseteq \mathcal{D} : |R| = k, \forall x \in R, y \in \mathcal{D} - R : \delta(q, x) \le \delta(q, y)\} \tag{2.2}$$

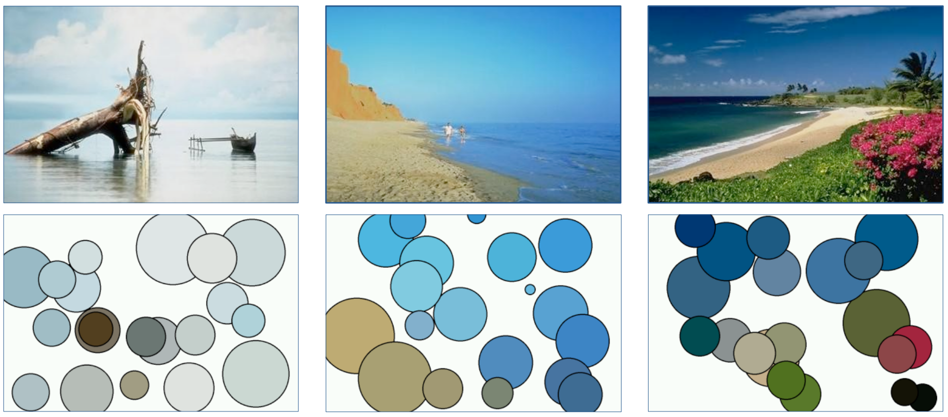


Figure 2.1: Deriving the representational object features (*signatures*) from the complex object (images) for similarity searching

While the latter similarity query specifies exactly the size of the result set to be k, in general we are not able to estimate or predict the result size for the range queries $\mathrm{RQ}_\delta(\cdot, r)$ with a radius r.

For completeness, we distinguish two approaches regarding the precision of the result set. First, there is a category of *exact similarity search* for which queries are usually slower but always give precise results. The other category of *approximate similarity search* include queries that trade off the precision for the speed of the evaluation, so they are faster but might produce errors.

2.2 Similarity Models

To properly rank the results, we need a *similarity function* that specifies the relevancy between any pair of database objects. Here, we follow the notion of similarities as previously defined [17].

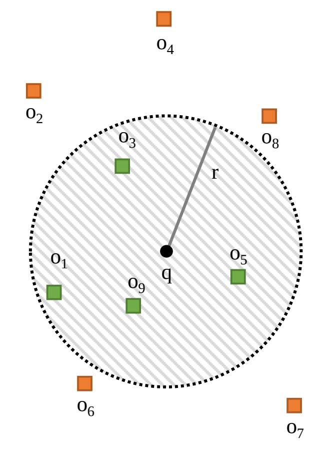
Definition 2.1 (Similarity function)**.** *With the database $\mathcal{D}$, we define the* similarity function s *on $\mathcal{D}$ as $s : \mathcal{D} \times \mathcal{D} \longrightarrow \mathbb{R}$ which for any two objects $x, y \in \mathcal{D}$ guarantees*

$$\begin{array}{cc} \textit{non-negativity} & s(x,y) \geq 0 \\ \textit{symmetry} & s(x,y) = s(y,x) \\ \textit{self-superiority} & s(x,x) \geq s(x,y) \end{array}$$

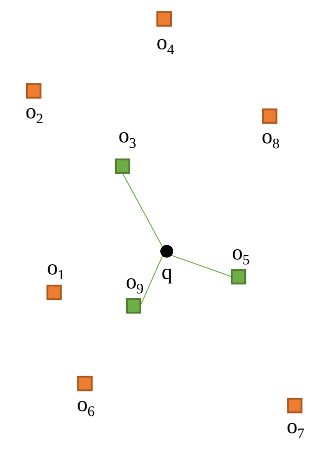
where $s(x,x) = s(x,y)$ if and only if $x = y$.

With such a definition of similarity functions, we are able to rank the database objects accordingly, as the function result $s(x, y)$ gives us the similarity score of two complex database objects x and y. The higher the value, the more similar the objects are. Usually, we assume similarity function to be normalized and to return values from the interval $\langle 0, 1\rangle$.

In some cases, we require a *dissimilarity* or *distance* function δ instead of similarity functions.



(a) Range query $\text{RQ}_\delta(q, r)$



(b) 3NN query for $k\text{NN}_\delta(q)$

Figure 2.2: Sample similarity queries

Definition 2.2 (Dissimilarity, Distance function)**.** *With the database $\mathcal{D}$, we define the* dissimilarity *or* distance function *δ on $\mathcal{D}$ as $\delta : \mathcal{D} \times \mathcal{D} \longrightarrow \mathbb{R}$ for the equivalent similarity function s if for any three objects $q, x, y \in \mathcal{D}$ it holds*

$$s(q, x) > s(q, y) \Leftrightarrow \delta(q, x) < \delta(q, y)$$

Note that distance functions assign higher scores for less similar objects and vice versa which follows the typical notion of distances (closer objects are better).

So, the problematic challenge is how to select a good mapping that converts the similarity s into the equivalent distance function δ. There exist several approaches [3, 17] how to define the dissimilarity measure for the normalized similarity functions (those with values limited to $s \in \langle 0, 1\rangle$) such as $\delta = 1 - s$, $\delta = \sqrt{1 - s}$, $\delta = -\ln s$, or $\delta = \frac{1-s}{s}$ even though these transformations might suffer from the inefficient performance due to the (bi)directional distance-to-similarity conversions [18, 19].

The opposite transformation (from distance to similarity) is more complex, as the widely adopted distance / dissimilarity measures typically do not come from the limited interval range $\langle 0, 1\rangle$. The basic solution is to normalize the distances before we use the inverted version of the previously mentioned similarity-to-distance transformation functions.

In the following text, we might often interchangeably use distances and similarities, as the increasing similarity (s) gives decreasing distance (δ) between any two objects from the database. To avoid any confusion, we will use the similarity in the context of *similarity models* (for more formal explanation see Def. 4.1), however, we will work primarily with normalized distances such that $\delta \in \langle 0, 1\rangle$.

2.2.1 Metric Space Model

For many applications, the widely known *metric space model* [3] has been considered as the best choice for indexing, mainly because of its simplicity and fixed properties. We represent the metric space as $(\mathbb{U}, \delta)$ where $\mathbb{U}$ is the set of objects with the corresponding dissimilarity function δ which for any three objects $x, y, z \in \mathbb{U}$ satisfies the metric space properties specified in Table 2.1.

identity	$\delta(x,y) = 0$	$\Leftrightarrow x = y$
positivity	$\delta(x,y) > 0$	$\Leftrightarrow x \neq y$
symmetry	$\delta(x,y) = \delta(y,x)$	
triangle inequality	$\delta(x,y) + \delta(y,z) \geq \delta(x,z)$	

Table 2.1: Metric space $(\mathbb{U}, \delta)$ postulates for any three objects $x, y, z \in \mathbb{U}$

If a distance function δ satisfies all metric properties, we denote it as a *metric distance* or simply *metric*. We assign the metric distances into two groups based on the character of returned values [3]: *discrete* or *continuous* distances.

In the following text, we depict some of the well-known and commonly used metric distances for multidimensional vectors, strings, or sets. Typically, the domains for such distances is the domain of real numbers $\mathbb{R}$.

Minkowski L_p Distances

One of the typical metric distances is the family of metric Minkowski L_p distances designed for n-dimensional real-valued vector data $\vec{u}, \vec{v} \in \mathbb{R}^n$:

$$\mathrm{L}_p(\vec{u}, \vec{v}) = \left(\sum_{i=1}^{n} |u_i - v_i|^p \right)^{1/p}, p \geq 1 \tag{2.3}$$

We provide the graphical representation of 2-dimensional space for some L_p distances in Fig. 2.3. The selection of most famous representatives [3] includes

- *Manhattan* (L_1) distance for $p = 1$

$$\mathrm{L}_1(\vec{u}, \vec{v}) = \sum_{i=1}^{n} |u_i - v_i| \tag{2.4}$$

- *Maximum* (L_∞) distance, *infinite distance*, or also *chessboard distance*

$$\mathrm{L}_\infty(\vec{u}, \vec{v}) = \max_{i=1,\ldots,n} |u_i - v_i| \tag{2.5}$$

- *Euclidean* (L_2) distance that is widely used for general purposes, e.g., for comparing various histogram representations of extracted image features:

$$\mathrm{L}_2(\vec{u}, \vec{v}) = \sqrt{\sum_{i=1}^{n} (u_i - v_i)^2} \tag{2.6}$$

For n-dimensional feature histograms derived from the original images and represented by their individual histogram bins u_i, v_i as n-dimensional vectors $\vec{u}, \vec{v}$, we are supposed to compute the distances between any two database objects $u, v \in \mathcal{D}$.

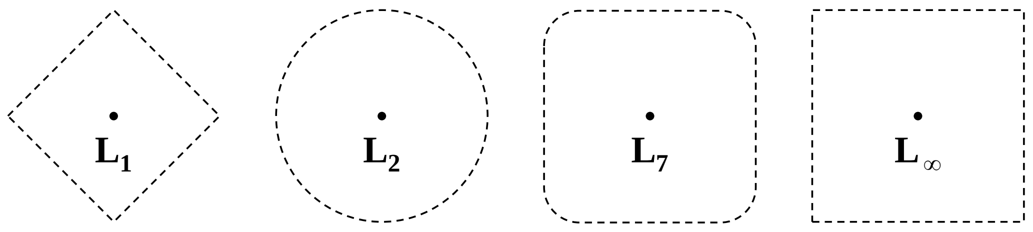


Figure 2.3: Sets of points within the same distances from the center point using various L_p distances

Quadratic Form Distance

Another example of a metric distance, the *Quadratic Form Distance* (QFD) known also as Mahalanobis distance [8, 20], is the generalized case of Euclidean (L_2) distance :

$$\delta_{\mathrm{QFD}_A}(\vec{u},\vec{v}) = \sqrt{(\vec{u}-\vec{v})A(\vec{u}-\vec{v})^T} \tag{2.7}$$

where A is $n \times n$ positive-definite matrix[1] (called the *QFD matrix* or the *similarity matrix*) and $\vec{u}^T$ is the vector transposition.

The identity matrix A reduces QFD to the ordinary Euclidean distance, while with the diagonal matrix A', we get the reduction to weighted Euclidean distance:

$$\delta_{\mathrm{QFD}_{A'}}(\vec{u},\vec{v}) = \sqrt{\sum_{i=1}^{n} w_i(u-v)^2} \tag{2.8}$$

Particularly, we use QFD as an effective way of searching for similarities in a set of color images [21, 1, 22]. In this case the QFD matrix stores correlations between individual dimensions of image descriptors, provides better applicability, and contributes to more robust similarity measuring.

For example, consider a 3-dimensional space, where the dimensions represent the number of red (R), green (G), and blue (B) pixels in an image. Because the human perception views green and blue colors as more similar than reds and blues or reds and greens, the matrix A could be set as follows:

$$A = \begin{array}{c} \\ R \\ G \\ B \end{array} \begin{array}{c} \begin{array}{ccc} R & G & B \end{array} \\ \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0.5 \\ 0 & 0.5 & 1 \end{pmatrix} \end{array}$$

A number of algorithms and image retrieval methods using QFD were developed within the QBIC project [2, 22, 1]. Other applications of QFD similarity search include 2D & 3D shapes [23, 24], protein structures [25, 26, 27], or flow cytometry [28]. In almost all the cited applications, the QFD matrix A is static (not changing from query to query), while the correlations between dimensions are defined based on a scoring function related to the particular domain.

[1]Positive-definiteness is defined for an $n \times n$ matrix A and n-dimensional vectors z as: $zAz^T > 0, \forall z \neq \mathbf{0}$, where $\mathbf{0}$ represents the zero vector [20].

Hence, the matrix A is not a query parameter and also it is not data-dependent. For RGB image histograms, the matrix A might be defined according to [22] as:

$$A_{ij} = 1 - \frac{\delta_{ij}}{\delta_{\max}}$$

where δ_{ij} is the Euclidean L_2 distance between representatives of colors i and j in the RGB color space and $\delta_{\max} = \max_{1 \leq i,j \leq n}\{\delta_{ij}\}$.

Although most of the approaches employ static QFD matrix, there appear also applications that use dynamic QFD matrix or even a kind of generalized QFD distance. A characteristic usage of a dynamic similarity matrix in QFD is shown in several papers such as [29, 30, 23].

In [29] authors propose a general method of iteratively guessing the distance function based on user preferences (i.e., *MindReader*). This concept uses QFD and tries to determine which attributes are important and to find correlations between them to satisfy the user query. The principle of finding the ideal distance function (changing the QFD matrix) is similar to relevance feedback techniques.

Signature Quadratic Form Distance

The recently proposed QFD variant, *Signature Quadratic Form Distance* (SQFD) [31, 15], enables to use feature signatures (vectors of variable dimensionality) instead of just feature histograms (vectors of fixed dimensionality). In fact, the SQFD concatenates the compared signatures represented as vector $\vec{u}, \vec{v}$ into a vector $(\vec{u}| - \vec{v})$, followed by the usual QFD computation:

$$\delta_{\mathrm{SQFD}_A}(\vec{u}, \vec{v}) = \sqrt{(\vec{u}| - \vec{v})A(\vec{u}| - \vec{v})^T} \tag{2.9}$$

This also requires a dynamic QFD matrix A that fits the particular features included in the signatures u, v. In other words, the SQFD constitutes a dynamic extension over the original QFD function, and it has proved a superior effectiveness in image classification applications based on feature signatures [15].

Edit Distance

The strings domain very often include the *edit distance* also known as *Levenshtein distance* [3], to compute the dissimilarities between two given sequences of characters. The resulting distance conforms to the minimum number of basic (single-character) edit operations that consist of *insert*, *delete*, and *substitute* that are required to convert one sequence into another.

With a string $s = s_1 s_2 \ldots s_n$, we define the operations as follows [3]:

- *insert* the character x into the string s at the position i

$$insert(s, i, x) = s_1 s_2 \cdots s_i x s_{i+1} \cdots s_n$$

- *delete* the character from the string s at the position i

$$delete(s, i) = s_1 s_2 \cdots s_{i-1} s_{i+1} \cdots s_n$$

- *replace* the character in the string s at the position i with character x

$$replace(s, i, x) = s_1 s_2 \cdots s_{i-1} x s_{i+1} \cdots s_n$$

Algorithm 1 Levenshtein(a, b)

```
Require: string $a = a_1 a_2 \cdots a_m$, string $b = b_1 b_2 \cdots b_m$
 1: $dist$ = new empty $Matrix(m, n)$
 2: for $i = 1$ to $m$ do
 3:     $dist[i, 0] = i$
 4: end for
 5: for $j = 1$ to $n$ do
 6:     $dist[0, j] = j$
 7: end for
 8: for $j = 1$ to $n$ do
 9:     for $i = 1$ to $m$ do
10:         if $a_i = b_j$ then
11:             $dist[i, j] = dist[i-1, j-1]$
12:         else
13:             $del = dist[i-1, j] + 1$    {delete}
14:             $ins = dist[i, j-1] + 1$    {insert}
15:             $repl = dist[i-1, j-1] + 1$    {replace}
16:             $dist[i, j] = \min(del, ins, repl)$
17:         end if
18:     end for
19: end for
20: return $dist[m, n]$
```

We can further apply weights for individual operations to get the generalized edit distance. Nevertheless, if the weights of insert and delete operations differ, we do not obtain a symmetric distance, therefore we do not have a metric.

Algorithm 1 shows an effective way of computing the edit distance between two arbitrary strings a, b of lengths $len(a) = m, len(b) = n$ using the bottom-up dynamic programming technique [32]. We apply this distance to various areas in which strings encapsulate the original data objects such as DNA sequences [33].

Tree Edit Distance

A very similar approach applied to the domain of trees gives us the *tree edit distance* [3] in which we replace the basic single-character string operations with basic tree operations of *inserting*, *deleting*, or *replacing* a node. This way, we are able to measure the proximity of any two tree structures. The useful application is when comparing structures of XML documents [34, 35, 36].

Jaccard's Coefficient

The similarity between two finite sets A, B we gives *Jaccard's coefficient* [3]:

$$s_{\text{Jaccard}} = \frac{|A \cap B|}{|A \cup B|} \tag{2.10}$$

It returns the ratio between the intersection of same objects in both sets and the union of all objects from both sets. For example, we use this to compare user preferences expressed as individual sets.

Because Jaccard's Coefficient is a similarity measure, we very often use the *Jaccard distance* instead:

$$\delta_{\text{Jaccard}} = 1 - s_{\text{Jaccard}} = 1 - \frac{|A \cap B|}{|A \cup B|} \tag{2.11}$$

Hausdorff Distance

So far, we considered simple evaluations of a single distance function. In real world scenarios, there appear more complicated distances such as the *Hausdorff distance* [3, 37]. Similarly to Jaccard's Distance (see Section 2.2.1), it is defined for finite sets but it is more flexible. We do not compare objects in a binary way with $\{0, 1\}$, $\{\texttt{true}, \texttt{false}\}$, or $\{\text{same}, \text{different}\}$ return values but we provide a more granular approach.

For two sets of objects $A = \{a_i\}, B = \{b_j\}$, we define the greatest *partial distance* h over objects in A to their nearest neighbors in B as

$$h(A, B) = \max_{a \in A}\{\min_{b \in B}\{\delta_g(a, b)\}\}$$

where δ_g is the ground distance function which provides the dissimilarity between two complex objects. Then, we compute *Hausdorff distance* as

$$\delta_{\text{HD}}(A, B) = \max\{h(A, B), h(B, A)\} \tag{2.12}$$

In order for δ_{HD} to be a metric, the ground distance δ_g must be a metric.

2.2.2 Non-Metric Spaces

There are also distances that hold only some (but not all) of the metric space postulates such as

- *semi-metric* functions satisfy *identity*, *positivity*, and *symmetry* but violate the problematic *triangle inequality*;
- *pseudo-metrics* do not satisfy the *identity*;
- *quasi-metric* distances are not symmetric, so the triangle inequality must be oriented;
- and stronger *ultra-metric* distances which require the *ultrametric inequality*:

$$\delta(x, y) \leq \max\{\delta(x, z), \delta(z, y)\}$$

Generally, we will denote such distances as *nonmetric* distances; they provide the basis of *nonmetric* spaces. In the following sections, we introduce some of the famous nonmetric distances.

Fractional L_p Distances

The problem of "nonmetricity" is not limited just to the complex distances because even a slight change of a well-known metric distance could lead to a nonmetric one. For example, the family of L_p distances (see Eq. 2.3) with fractional values $p \in (0, 1)$ are generally nonmetric and are mostly used for *robust* matching of histograms [38].

Cosine Distance

As another example, we choose the semi-metric *cosine distance* which is based on the *cosine similarity measure* [39]. The *cosine measure* computes the angle between two N-dimensional vectors $\vec{x}, \vec{y}$ for which the magnitude is not relevant:

$$s_{\cos}(x, y) = \frac{\sum_{i=1}^{N} x_i y_i}{\sqrt{\sum_{i=1}^{N} x_i^2 \cdot \sum_{i=1}^{N} y_i^2}} \tag{2.13}$$

Afterwards, we define the equivalent *cosine distance* as $\delta_{\cos}(x, y) = 1 - s_{\cos}(x, y)$.

Kullback-Leibler Divergence

For comparing images, we apply *Kullback-Leibler Divergence* (KLD) [40] to the image histogram representations with N bins and get the inefficiency of coding one histogram using the other:

$$\delta_{\text{KLD}}(x, y) = \sum_{i=1}^{N} x_i \cdot \log(\frac{x_i}{y_i}) \tag{2.14}$$

Jeffrey-Divergence

For a similar purpose of image histograms matching, there exists *Jeffrey-divergence* (JD) distance [8] defined as

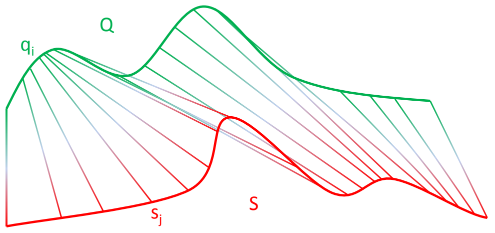
$$\delta_{\text{JD}}(x, y) = \sum_{i=1}^{N} x_i \cdot \log\left(\frac{x_i}{\frac{x_i+y_i}{2}}\right) + y_i \cdot \log\left(\frac{y_i}{\frac{x_i+y_i}{2}}\right) \tag{2.15}$$

Dynamic Time Warping

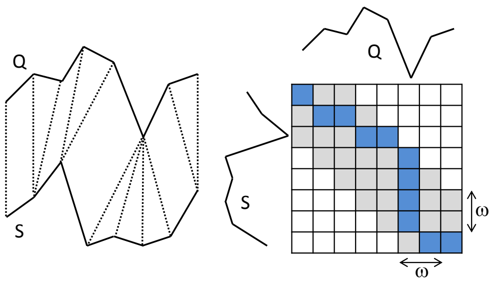
The interest in time series, especially in financial data and market trading, highlights another specific distance which violates the triangle inequality postulate, the *Dynamic Time Warping* (DTW) distance [41]. It has been proposed primarily for speech recognition processes with the main goal to minimize the dissimilarity between a speech sample and speech patterns in order to find a "perfect" match. Over time, DTW has become a popular technique for measuring the similarity between general time series.

For two time series represented as vectors $Q \in \mathbb{R}^n$ and $S \in \mathbb{R}^m$, we build the $n \times m$ cumulative matrix M in which each item $M[i, j]$ corresponds to the alignment between elements q_i and s_j (see Fig. 2.4a). The alignment is computed by the *ground distance* δ_g. We usually apply the L_2 or L_1 from the Minkowski L_p distances as the ground distances [42, 43].

Then, we examine various warping paths $W = \{w_1, \ldots, w_X\}$ which provide mappings or alignments between elements in time series Q and S. The warping path W (see Fig. 2.4b) has a limited length of $\max\{n, m\} \leq X \leq n + m + 1$ and represents a sequence of elements from the matrix M. Each element in the matrix $w_k = (i, j)_k$ must satisfy several criteria [42], namely



(a) Sample time series alignment



(b) Warping path cumulative matrix

Figure 2.4: Dynamic Time Warping distance

1. *Boundary conditions* to restrict the searching space for warping paths:

$$w_1 = M[1,1] \qquad \text{and} \qquad w_X = M[n,m]$$

2. *Monotonicity* ensures that the element pairing is monotonous according to the *time*. If we have $w_k = M[a,b]$ and $w_{k-1} = M[a',b']$, then

$$a - a' \geq 0 \qquad \text{and} \qquad b - b' \geq 0$$

3. *Continuity* assures that neighboring elements in the warping path correspond to adjacent cells in the matrix. Let $w_k = M[a,b]$ and $w_{k-1} = M[a',b']$, then

$$a - a' \leq 1 \qquad \text{and} \qquad b - b' \leq 1$$

Finally, for the ground distance δ_g, we define the DTW as a minimized warping path from the universe of all acceptable warping paths as

$$\delta_{\text{DTW}}(Q,S,\delta_g) = min_W \left\{ \sum_{k=1}^{X} \delta_g(w_k) \right\} \tag{2.16}$$

Because there is an exponential number of possible warping paths, we employ a method of dynamic programming for evaluating DTW [44].

Specific domains such as speech recognition require alignments between time series to have some additional constraints – to avoid warping paths with excessive time stretch, to avoid sequence distortions, or to discard "non-interesting" warping paths [45]. These include the *Slope constraint* to restrict the slope of warping paths or *Warping window* (denoted by the integer ω) known also as *Sakoe-Chiba band*, in which elements of the warping path must fit [44],

Finally, our previous work [43] generalizes the slightly different definitions of DTW [41, 46, 47] and defines the generalized DTW distance for an arbitrary ground distance

$$\delta_{\text{GDTW}}(Q,S,\delta_g,\omega,f) = min_W \left\{ f(\sum_{k=1}^{X} \delta_g(w_{k,\omega})) \right\} \tag{2.17}$$

where $w_{k,\omega}$ is the kth item in the working path W with the corresponding warping window constraint ω and f is a monotonic function.

Some authors [46] use $f = \sqrt{\cdot}$ because if we take the zero warping window together with the ground distance L_2, the result distance corresponds to L_2 (Euclidean) distance

$$\delta_{\text{GDTW}}(\cdot,\cdot,L_2,0,\surd) = L_2(\cdot,\cdot)$$

In all cases, notice that the magnitude of metric/nonmetric behavior is heavily determined by the selection of the ground distance function δ_g.

Other Nonmetric Distances

We already mentioned some other more complex nonmetric distances such as the various alignment algorithms for measuring functional similarity of protein sequences [11] or structures [12]. For a more precise overview of existing nonmetric spaces and their application domains, we refer the readers to the extensive survey [10].

Chapter 3

Indexing Similarity Models

The basic task concerning similarity search is to increase the performance of these queries by means of indexing. Compared to traditional relational databases where the I/O cost represents the standard measure of real-time performance, here the major component is assumed to be the number of distance computations (DCs) we use during the query evaluation for ranking the database.

The reason is because the time complexity of computing a single distance score between two (complex) objects ranges from $\mathcal{O}(n)$ to $\mathcal{O}(2^n)$ where n denotes the complexity of a single object (e.g., the length of the vector data). Hence, with the increasing complexity of distance evaluations, the total performance largely depends on the number of DCs.

Even though, in the real world scenarios we would definitely prefer measuring the real-time performance, using the number of DCs over real response times eliminates the requirements for a specific hardware used for query evaluations, gives us a better basis for comparing different indexing methods, and therefore it is more general.

However, we acknowledge that for very simple similarity models with "cheap" distance computations (e.g., $\mathcal{O}(1)$ or $\mathcal{O}(n)$) we could accomplish better efficiency if we do not use any indexing method. But this applies only to small-sized data and therefore it is only an extra-ordinary and marginal situation. So, in the following text, we will consider the query performance as the number of DCs that are required for ranking the database.

3.1 Lowerbounding Techniques

To eliminate as many DCs as possible, we leverage cheaper measures for distance value estimations – the *lowerbounds*. These expressions allow us to filter out non-interesting and irrelevant database objects o_i for the given query q. For any valid lowerbound expression LB it holds

$$LB(\delta(q, o)) \leq \delta(q, o) \tag{3.1}$$

where q is the query object, o is a database object, $\delta(q, o)$ is the distance between q and o, and LB is its estimation. The tighter/better the lowerbound is, the more objects it filters out without computing the generally expensive distance $\delta(q, o)$ and thus saves the evaluation time.

To compute the lowerbound, we often use reference objects p_i called pivots which create the basis for various indexes, e.g. *AESA* or *LAESA* known as *pivot tables* [48].

Observe that the main characteristics of the lowerbound is that its value is always smaller than the real distance (no matter of the pivot selection), so it provides true distance estimations. Any situation in which there exists objects q and o such that $LB(\delta(q, o)) > \delta(q, o)$ will discard LB from being the lowerbound.

While evaluating range or kNN queries, we use the *ball partitioning* [49] to break the database $\mathcal{D}$ into subsets using a spherical cut with respect to the query object q. Then, we build the *query ball* with a radius r which corresponds to the set of all objects o_i such that

$$\texttt{query\_ball}_\delta(q, r) = \{o_i \in \mathcal{D} \mid \delta(q, o_i) \leq r\} \tag{3.2}$$

For the range query $\mathrm{RQ}_\delta(q, r)$, this already presents the result set but for kNN queries, we can either transform the kNN search to finding the value of radius r for which the range query returns the result set of size k; or we typically maintain the distance to the kth nearest neighbor found so-far as the radius r in order to eliminate irrelevant results. Then, we use the priority queue for processing.

While using the lowerbounding mechanism, we very often follow the *filter-and-refine* principle which suggests how to process the objects in two subsequent steps. The first step of *filtering* stands for the lowerbound pruning which just eliminates the non-interesting objects yet it does not form the final result set. After discarding irrelevant objects, we can still get relatively large set of candidate results to be inspected. The second phase of *refining* the obtained results conforms to computing the distance to objects that were not filtered out and thus forming the final results by comparing the distances between the query and candidate objects.

There are two well-known lowerbounds that we use in metric / ptolemaic spaces, namely *triangle* (Section 3.1.1) and *ptolemaic* (Section 3.1.2) lowerbounds that we will describe into more details in the following sections.

3.1.1 Triangle Lowerbound

The well-known triangle (metric) lowerbound $LB_\triangle$ applies to the metric spaces, as it relies on the triangle inequality and uses a single pivot p:

$$LB_\triangle(\delta(q, o)) = |\delta(q, p) - \delta(p, o)| \leq \delta(q, o) \tag{3.3}$$

It is a cheap and effective lowerbound for metric spaces, however for nonmetric distances in which the triangle inequality is violated, it usually leads to false dismissals and only approximate results [8].

We build the region balls for query object q and database object o for which the distances $\delta(q, p)$ and $\delta(o, p)$ are (pre)computed and the triangle inequality guarantees the minimal distance $\delta(q, o)$. If the object o is outside of the query ball (given by the radius r or by the already computed distance to the kth nearest neighbor from the result set), we immediately filter out the object o (see Fig. 3.1).

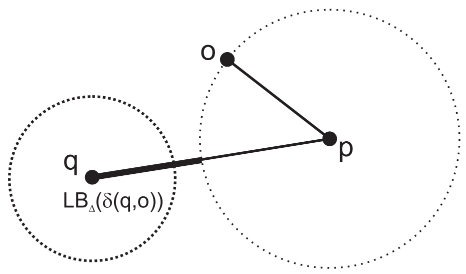


Figure 3.1: Triangle lowerbound $LB_\triangle(\delta(q,o))$ using a single pivot p

Naturally, having multiple pivots $p_j \in \mathbb{P}$, we combine the lowerbounds in order to obtain tighter or better distance approximation, as we take the maximum of all estimations:

$$LB_\triangle(\delta(q,o)) = \max_{p_j \in \mathbb{P}} |\delta(q,p_j) - \delta(p_j,o)| \leq \delta(q,o) \tag{3.4}$$

Example 1. *Assume we have two database objects o_1, o_5, the pivot p, and the distance measure δ which produces distances $\delta(o_1,p) = 1$, $\delta(o_7,p) = 7$. When we evaluate a range query $\mathrm{RQ}_\delta(q,2)$ for the query object q while $\delta(q,p) = 2$, we get the lowerbounds*

$$LB_\triangle(\delta(q,o_1)) = |2-1| = 1$$
$$LB_\triangle(\delta(q,o_7)) = |2-7| = 5$$

The latter lowerbound allows us to eliminate the object o_7 without computing the distance from the query, as the actual distance $\delta(q,o_7) \geq 5$ which is definitely outside of the ***query_ball***$_\delta(q,2)$*. However, we need to propagate the first object to the second phase of refining and compute the distance $\delta(q,o_1)$ to find out whether it lies within the actual query ball or not.*

3.1.2 Ptolemaic Lowerbound

Another specific approach uses *Ptolemy's inequality* [17, 50, 15] to construct lowerbounds. *Ptolemy's inequality* defines for any four database objects x, y, u, and v the following relation

$$\delta(x,v) \cdot \delta(y,u) \leq \delta(x,y) \cdot \delta(u,v) + \delta(x,u) \cdot \delta(y,v) \tag{3.5}$$

In order to form a lowerbound based on this relation, the distance function δ must be *Ptolemaic distance.* This applies when it holds the properties of *identity*, *positivity*, *symmetry*, and satisfies *Ptolemy's inequality.*

To construct the ptolemaic lowerbound ($\mathrm{LB}_{\mathtt{ptol}}$), we first define the candidate bound δ_C using two pivots p and s:

$$\delta_\mathrm{C}(q,o,p,s) = \frac{|\delta(q,p) \cdot \delta(o,s) - \delta(q,s) \cdot \delta(o,p)|}{\delta(p,s)} \tag{3.6}$$

For simplicity, we let $\delta_\mathrm{C}(q,o,p,s) = 0$ if $\delta(p,s) = 0$. Considering a set of pivots $\mathbb{P}$, we maximize the candidate bound δ_C over all pairs of distinct pivots which results in the final Ptolemaic lowerbound $\mathrm{LB}_{\mathtt{ptol}}$:

$$\mathrm{LB}_{\mathtt{ptol}}(\delta(q,o)) = \max_{p,s \in \mathbb{P}} \delta_\mathrm{C}(q,o,p,s) \leq \delta(q,o) \tag{3.7}$$

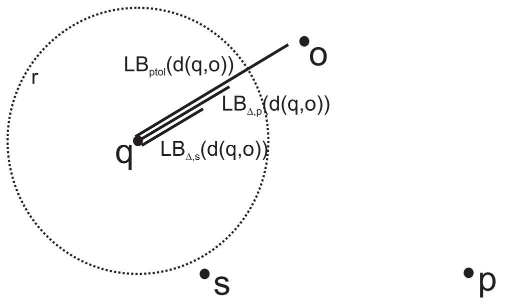


Figure 3.2: Ptolemaic lowerbound $\mathrm{LB}_{\mathtt{ptol}}(\delta(q,o))$ using two distinct pivots p, s compared to two estimations provided by triangle lowerbounds $LB_{\triangle,p}(\delta(q,o))$ and $LB_{\triangle,s}(\delta(q,o))$

Similarly as for the triangle inequality, we filter out irrelevant objects using the query ball as with $LB_{\triangle}$. Figure 3.2 shows a situation when neither the triangle lowerbound for pivot p $LB_{\triangle,p}(\delta(q,o))$ nor the triangle lowerbound for pivot s $LB_{\triangle,s}(\delta(q,o))$ discards object o. But, the ptolemaic lowerbound $\mathrm{LB}_{\mathtt{ptol}}(\delta(q,o))$ better estimates the real query-object distance and filters out object o.

The concept of ptolemaic indexing with the employed ptolemaic lowerbound $\mathrm{LB}_{\mathtt{ptol}}$ was successfully used with *signature quadratic form distance* [14, 15] that was proved to be suitable for effective matching of image signatures [14]. This lowerbound provides the basis for *ptolemaic access methods* (see Section 3.2.4) which is a solid alternative to metric access methods if they fail to work properly [51].

Example 2. *Assume we have two database objects o_1, o_5, two pivot p, s, and the distance measure δ which produces distances $\delta(o_1,p) = 1$, $\delta(o_5,p) = 5$, $\delta(o_1,s) = 10$, $\delta(o_5,s) = 5$, and $\delta(p,s) = 2$. When we evaluate a range query $\mathrm{RQ}(q,2)$ for the query object q while $\delta(q,p) = 2$ and $\delta(q,s) = 3$, we get the lowerbounds*

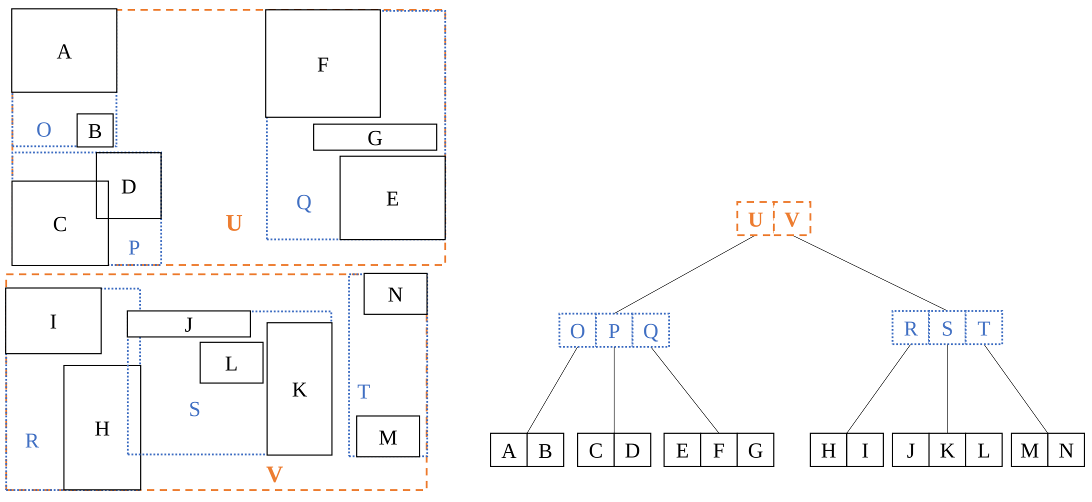
$$\mathrm{LB}_{ptol}(\delta(q,o_1)) = \frac{|2\cdot 10 - 3\cdot 1|}{2} = \frac{17}{2} = 8.5$$

$$\mathrm{LB}_{ptol}(\delta(q,o_5)) = \frac{|2\cdot 5 - 3\cdot 5|}{2} = \frac{5}{2} = 2.5$$

In this case, we can simply prune both objects, as their distances to the query object will be outside of the $\mathtt{query\_ball}_{\delta}(q,2)$. Here, the selection of good pivots is essential and affects the overall filtering.

3.2 Similarity Indexing Techniques

In addition to the modeling of domain-specific similarity search problems in similarity spaces, there are substantial efforts spent on developing indexing techniques that speed up the similarity queries in large databases. We distinguish two basic classes of database indexing methods that are used predominantly for the similarity search in *vector spaces*– the *Spatial Access Methods* (SAMs; see Section 3.2.1) and *Metric Access Methods* (MAMs; see Section 3.2.2), together with emerging *Ptolemaic Access Methods* (PtoAMs; see Section 3.2.4)



a) b)

Figure 3.3: The visualization of minimum bounding rectangles (MBRs) for the selected database objects (a) with the corresponding R-tree index structure (b)

The trend towards nonmetric or unconstrained similarity models is not a marginal experience but rather a rule however the response of the database research is inappropriate. Almost all of the general-purpose database techniques that are designed to support nonmetric similarity models map the problem to the metric ones and use MAMs or SAMs.

Older approaches directly map the data into some L_p space (see Section 3.2.2) while this mapping often suffers from an unpleasant trade-off. If the mapping preserves the similarity orderings, i.e., provides same query results as the nonmetric version, then it usually suffers from high intrinsic dimensionality [6].

Particularly, *TriGen algorithm* [52, 8] applies a system of concave functions to the nonmetric distances in order to obtain an approximately metric behavior (see Section 3.2.5).

3.2.1 Spatial Access Methods

The *spatial access methods* (SAM) [53] mostly treat the vector space independently of the distance function used for the similarity search. Hence, a SAM index is constructed using the vectorial structure of the descriptor (we use the values in individual dimensions). In particular, we consider the family of R-tree variants [54], namely X-tree [55] or VA-file [56], as representative SAMs.

The basic *R-tree* is a dynamic indexing structure based on similar principles as B-tree [57]. It is a height-balanced tree structure which does not index the object features but the *minimum bounding rectangle* (MBR) instead. In the leaf nodes, we store the object identifier together with its MBR, while in the inner (non-leaf) nodes, we save the pointers to child nodes together with the MBR that covers MBRs of all descendant nodes in the subtree. Then, the searching algorithm traverses the index tree while we are looking for MBRs that intersect the query MBR (see the sample visualization of the index structure in Fig. 3.3).

As the SAM index is not dependent on a particular distance function, a distance function could be provided right at the query time as a parameter, allowing thus flexible similarity searching. This is especially important for applications when the distance function needs to be adjusted either occasionally or regularly from query to query, for example, when user preferences have to be incorporated into the distance function.

On the other hand, the independence from the distance function is also the main drawback of SAMs. Since a SAM indexes the database objects within (rectangular) regions minimizing the volumes, surfaces and overlaps (e.g., the volume of MBRs in case of R-tree), the objects in regions do not form tight clusters with respect to the distance function that will be used for querying. In consequence, the regions could be unnecessarily large, which leads to poor filtering ability and thus slower query processing. This negative effect is even magnified with the increasing dimensionality of the space (the so-called *curse of dimensionality* [56, 6, 58]). It was many times proved that the data of the dimensionality beyond 10-20 cannot be efficiently searched by SAMs specially for uniformly distributed data and exact search, i.e. a search not allowing any false dismissals, regardless of the distance function used for queries [59, 60].

3.2.2 Metric Access Methods

The *metric access methods* (MAMs) or metric indexes [3, 6] represent a different indexing concept, treating the vector space together with the distance function as a black-box metric space. That is, only the distances between vectors can be utilized to build the index, not the particular vector coordinates.

Here the pros and cons are exactly opposite as for SAMs. Since MAMs build the index using a particular *static* distance function, they are not suitable when the distance function has to be modified after indexing (e.g., at the query time). For example, changing the QFD matrix A in the QFD distance (see Section 2.2.1) results in a different distance function than the one used for indexing. Such a change would require the reorganization of the metric index, making thus the actual index (and the distance values stored within) invalid.

To name the advantages, MAM index regions are more compact than those of SAMs, since the database objects are organized in clusters gathering objects close with respect to the distance function. In turn, MAMs are more successful in the fight with the curse of dimensionality, because the embedding (vector) dimensionality is irrelevant. Instead, the complexity of MAM indexing is determined by the distance distribution, namely, the *intrinsic dimensionality* [6, 8], which is usually smaller than the embedding dimensionality.

In particular, we name the M-Tree family [61], M-Index [62], Vantage-Point (vp-tree) [63], Pivot tables [64, 65, 6], GNAT (Geometric Near-neighbor m-ary Access Tree) [66], or SAT (Spatial Approximation Tree) [67] as representative MAMs.

In the following sections, we will introduce and describe some of the basic MAMs into more details to get an overview how they work and handle different data in a consistent way.

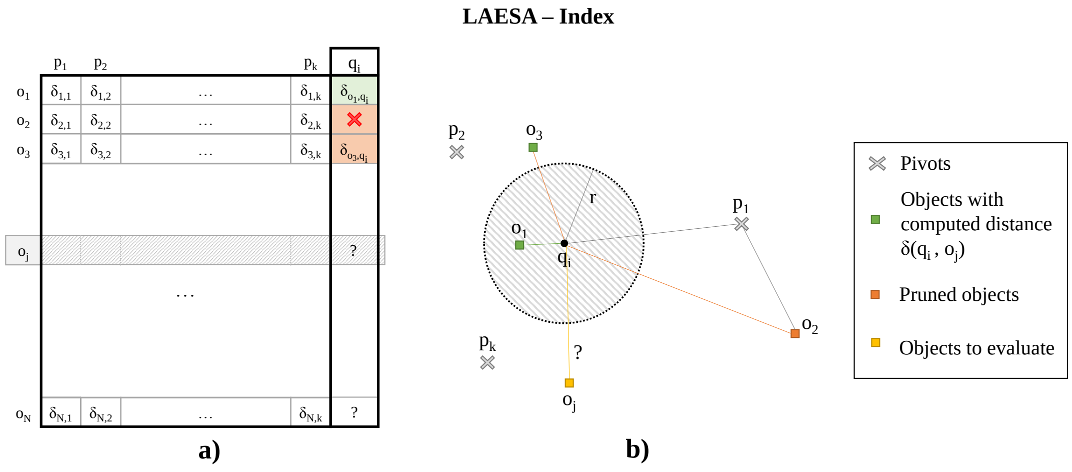

LAESA – Index
p1 p2 pk qi
o1 δ1,1 δ1,2 … δ1,k δo1,qi
o2 δ2,1 δ2,2 … δ2,k
o3 δ3,1 δ3,2 … δ3,k δo3,qi
oj ?
oN δN,1 δN,2 … δN,k ?
a)
p2 o3 r o1 qi p1 o2 pk ? oj
Pivots
Objects with computed distance δ(qi , oj)
Pruned objects
Objects to evaluate
b)

Figure 3.4: Sample pivot table (LAESA) index structure with pre-computed distances $\delta(o, p)$ between objects $o_1, \ldots, o_N$ and pivots $p_1, \ldots, p_k$ while evaluating the range query given by q_i and r (a); $\texttt{query\_ball}_\delta(q_j, r)$ with processed objects $o_1 - o_3$, the pruned object o_2, the discarded object o_3, and object o_j being evaluated (b).

Sequential File

The sequential file is a very naive MAM that is represented by a flat binary file that is built from a series of dynamic insertions by just appending the inserted objects at the end of the file. Any query involves a sequential scan over all the objects in the binary file. For a query object q and every data object o_i, the distance $\delta(q, o_i)$ must be computed (regardless of the query selectivity). Although this kind of "MAM" is not very smart, it is a baseline structure mostly for qualitative comparisons, as it does not make any mistakes in terms of *false negatives* or *false positives*. Moreover, it does have a predicted behavior and in any case, the time complexity remains the same: $\mathcal{O}(N \cdot C)$ where N denotes the number of database objects and C conforms to the distance computation complexity.

Pivot Tables

A simple but efficient solution to the similarity search in metric spaces represent methods called *Pivot tables*, such as AESA [64] or LAESA [65]. In general, we select a set of *pivot* objects $p \in \mathbb{P}, \mathbb{P} = k$ and create for every database object $o \in \mathcal{D}$ the k-dimensional vector of distances to the pivots. The distance vectors belonging to the database objects then form a limited distance matrix – the *pivot table*.

Choosing $|\mathbb{P}|$ pivots from the database sample of a size N depends on a pivot selection technique which takes D distance computations (usually $D \ll N$). The total time to construct a pivot table and insert N database objects depends on the distance computation complexity C and belongs to $\mathcal{O}(D + N \cdot C \cdot |\mathbb{P}|)$.

There are various pivot selection techniques and each method affects the overall performance during query evaluations [68]. We choose the pivot sets based on the given algorithms starting with basic random selection through incremental selections to various heuristic models [69]. But in general, we try to select pivots that are not close to each other.

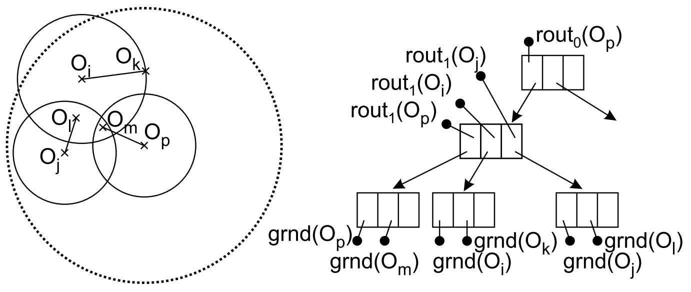


Figure 3.5: *M-tree* index structure with ball-shaped regions of routing objects

When performing a range query $\mathrm{RQ}_\delta(q, r)$ with the radius r, we determine a distance vector for the query object q the same way as for a database object, i.e. the overhead is in $\mathcal{O}(C \cdot |\mathbb{P}|)$. From the distance vector of the query q and the query radius r we create a k-dimensional hyper-cube, centered in the query and with edges of length $2r$.

Then, we process the range query $\mathrm{RQ}_\delta(q, r)$ within the pivot table and the k-dimensional vectors of database objects o_i that do not fall into the query cube are filtered out from the further processing. The database objects that cannot be filtered have to be subsequently checked by the usual sequential search.

Similarly, for kNN queries, the preprocessing remains unchanged and we maintain the priority queue of resulting objects with the object with maximum distance at the top. This corresponds to the radius r from the previous case. Whenever an object o_i falls into the query cube, we compute the distance $\delta(q, o_i)$, update the result set and the radius for subsequent evaluations.

Figure 3.4 depicts a sample instance of the query evaluation using the triangle lowerbound (see Section 3.1.1) in metric spaces. The first part (a) reveals the way of storing the index, while the latter part (b) displays the $\texttt{query\_ball}_\delta(q, r)$ with sample range query evaluation steps. While we are able to immediately add some objects (e.g., o_1) to the result set, others might be easily discarded with the distance estimations using pivots. Then, we check the rest of objects sequentially and prune objects that are not in the query ball, such as the object o_3.

M-tree

The *M-tree* [61] is a dynamic index structure that provides a good performance in the secondary memory (i.e., in database environments). M-tree is a hierarchical index, where we select some of the data objects and mark them as centers of ball-shaped regions (local pivots), while we partition the remaining objects among the regions in order to build up a balanced and compact hierarchy of data regions.

We end up with two types of nodes – the *routing* (internal) objects we use for navigating within the index structure and the *leaf* (ground) objects that store indexed database objects (see Fig. 3.5). The structure of any internal *routing* object o_r consists of

- pointer to the root of the object's *covering tree* $T(o_r)$ denoted as $ptr(T(o_r))$
- *covering radius* $r(o_r) > 0$ which encapsulates all objects in the covering tree within this distance: $\forall o_i \in T(o_r) : \delta(o_i, o_r) \leq r(o_r)$
- *distance* to object's parent node $\mathcal{P}ar(o_r)$: $\delta(o_r, \mathcal{P}ar(o_r))$

This structure forms the following *routing* entry:

$$rout(o_r) = [o_r, ptr(T(o_r)), r(o_r), \delta(o_r, \mathcal{P}ar(o_r))]$$

For the *leaf* nodes o_l which are in some cases equivalently considered as *ground entries* (*grnd*), the situation is less complex. Here, we do not need any covering radius; the pointer either contains the object or includes the identifier (*id*) which references the object directly to the separated data file. This simplifies the *leaf* entry to:

$$leaf(o_l) = [o_l, id(o_l), \delta(o_l, \mathcal{P}ar(o_l))]$$

In its original version, the M-tree is built by dynamic insertions in the same way as B-tree [70]. First, we have to find a suitable leaf for the newly inserted object, which takes $\mathcal{O}(\log(m))$ time where m is the number of objects stored in the tree. This time is guaranteed as the M-tree is a paged, dynamic and balanced index structure.

Next, the insertion into a leaf could cause an overfill of the node, which results in splitting bottom-up along the path to the root based on the *split policy*. If we split a node n_i, we create a new internal node n_j and divide all routing objects into disjoint sets that are stored within these two nodes. The only criterion we need to ensure is that the covering radius is computed as the maximal distance between the new node n_j and its direct descendants. Simply said, the time complexity of the dynamic insertion in the M-tree is analogous to the insertion complexity in B-trees, hence leading to $\mathcal{O}(m \log(m))$.

The similarity queries are implemented by traversing the tree, starting at the root level. We recursively access the internal nodes that cannot be directly excluded until we reach the leaf nodes. During the traversal, we can prune whole sub-trees without computing any new distances based on the adjusted form of the triangle inequality (see Eq. 3.3) [61]. In general, we process such M-tree nodes whose regions are overlapped by the query ball.

Since its introduction, there appeared several variants and modifications of M-tree index resulting in a whole family of indexing methods based on M-tree, such as M^2-tree [71], M$^+$-tree [72], M^3-tree [73], PM-tree [74], or NM-tree [75].

Mapping Approaches

Another way to ensure metric searching is to directly map the data into some low-dimensional vector space and apply the existing SAMs to index the transformed data. Several techniques use mappings or embeddings such as FastMap [76], BoostMap [77], MetricMap [78], or SparseMap with Lipschitz embeddings [79].

Formally, we represent the mapping as a function $F : S \to \mathbb{R}^k$ which converts the input metric space $(\mathbb{U}, \delta)$ into k-dimensional vector space $(\mathbb{R}^k, \delta^*)$ with the new (and cheaper) distance measure $\delta^* : \mathbb{R}^k \times \mathbb{R}^k \to \mathbb{R}^+$.

The mapping F might be *contractive*

$$\delta^*(F(o_i), F(o_j)) \leq \delta(o_i, o_j) \tag{3.8}$$

which enables us to filter out some irrelevant objects just by using δ^* and for remaining objects, we refine the search by computing the original distance δ.

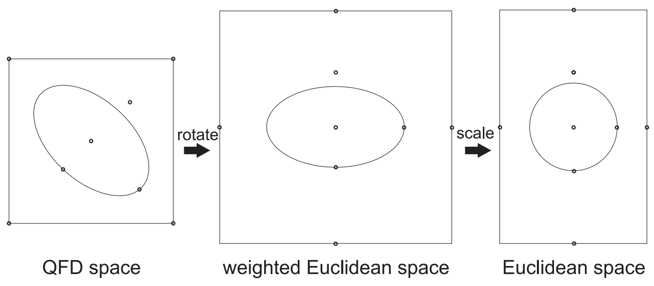


Figure 3.6: The idea of QMap – homeomorphic transformation of a QFD space into the Euclidean space.

We previously mentioned a sample contractive mapping that is employed in pivot table index, namely LAESA (see Section 3.2.2). There, we use k pivots to embed the metric space into $(\mathbb{R}^k, L_\infty)$. The mapping F transforms each object o_i into the vector data $[\delta(o_i,p_1),\delta(o_i,p_2)\ldots\delta(o_i,p_k)]$ and each range query $RQ_\delta(q,r)$ into $RQ_{\delta^*}(F(q),r)$.

The reason why we are able to apply embeddings for (finite) metric spaces $(\mathbb{U},\delta)$ is that we can find a function F which maps all objects $o_i \in \mathbb{U}$ to the k-dimensional vector space (k is sufficiently large) such that we approximately preserve the distance values between the objects when using a distance function δ^* in the k-dimensional space: $\delta^*(F(o_i),F(o_j)) \approx \delta(o_i,o_j)$ [80].

If the metric space $(\mathbb{U},\delta)$ is however infinite, we approach this by selecting a finite set of objects $S \subset \mathbb{U}$ for which we find a suitable mapping. This mapping often suffers from an unpleasant trade-off. If the mapping preserves the similarity orderings, i.e., provides query results exactly the same as the nonmetric version, then it usually suffers from high intrinsic dimensionality [6].

Nevertheless, not all mapping methods guarantee the exact similarity search on the embedded objects and thus result in false dismissals and approximate searching [80]. In this case, the mapping is only approximate and the search is fast but simultaneously introduces a retrieval error. In multimedia retrieval, the trade-off provides satisfactory results. In many other situations, however, any loss in retrieval precision is unacceptable. For example, in medical or biometric applications every percent of precision counts heavily.

One of the concrete examples is the mapping of QFD space into L_2 space with QMap approach [20] that utilizes the process of decomposing the QFD matrix A into matrices B and B^T by so called *Cholesky decomposition* or Cholesky factorization [81] as outlined in Fig. 3.6. We depict the theoretically proved superior performance with corresponding time complexity analysis which we depict in the consolidated view in Table 3.1 (for more details, see the original work [20]).

The basic idea of mapping objects to vector spaces provides a solid alternative to lowerbounding because the distances between the embedded objects approximates the actual distances. If this holds, we can run queries on the embedded objects and use the original distance measure only for the refinement. Here, we expect that computing the distances on embedded objects is more efficient, so we actually speedup the query evaluation by embedding (refer to Appendix A.1 for QMap evaluations on real-world data, as shown in Fig. A.2).

Method (model)	Action	Time complexity	Better
Sequential file (QFD) Sequential file (QMap)	indexing	$\mathcal{O}(mn)$ $\mathcal{O}(mn^2)$	QFD
Pivot tables (QFD) Pivot tables (QMap)	indexing	$\mathcal{O}(cn^2 + mn(pn))$ $\mathcal{O}(cn + mn(p + n))$	QMap
M-tree (QFD) M-tree (QMap)	indexing	$\mathcal{O}(mn^2 \log(m))$ $\mathcal{O}(mn^2 + mn \log(m)))$	QMap
Sequential file (QFD) Sequential file (QMap)	querying	$\mathcal{O}(mn^2)$ $\mathcal{O}(mn)$	QMap
Pivot tables (QFD) Pivot tables (QMap)	querying	$\mathcal{O}(n(pn) + mp + xn^2)$ $\mathcal{O}(n(p + n) + mp + xn)$	QMap
M-tree (QFD) M-tree (QMap)	querying	$\mathcal{O}(xn^2)$ $\mathcal{O}(n^2 + xn)$	QMap

Table 3.1: Indexing and Time Complexity Comparison for QMap mapping applied to the database of size m that contains n-dimensional vectors with p pivots (we require c distance computations to select those) and with x as the number of non-filtered objects for which we need to evaluate the original QFD.

M-Index

The introduction of *M-Index* structure is motivated by the efficiency issues the metric-based indexes face [82, 62]. It combines the existing metric approaches for pruning and filtering the metric space. With a fixed costs of building the index and efficient exact and approximate similarity search, M-Index provides the best-of-breed solution and a superior alternative to other MAMs [62].

Its efficiency comes from the static set of reference points (pivots), from the reliance on storing the data with B$^+$-trees or in a distributed storage, and the potential for distributing the index if required.

First, it takes the concept of iDistance [83] built for efficient kNN query processing in high-dimensional metric spaces in order to partition the data in clusters. More formally, iDistance partitions the given sample $S \subseteq \mathcal{D}$ into n clusters and for each cluster $C_i, i \in \{0, 1, \ldots, n-1\}$, it selects the global reference point (pivot) p_i. Because any object $o \in S$ belongs to exactly one cluster C_i, we assign the object a numeric key according to the distance to the cluster's reference point p_i which is at the same time its closest pivot:

$$iDist(o) = \delta(o, p_i) + i \cdot c \tag{3.9}$$

where c is a sufficiently large constant that separates individual clusters.

The partitioning process corresponds to mapping the objects from the universe $\mathbb{U}$ to a fixed m-dimensional space represented by the set of global pivots $p_1, \ldots, p_m$. Note that, like other mapping approaches, also iDistance provides a lossy transformation and two different objects o_i, o_j might be mapped to a single (and identical) numeric value $x = iDist(o_i) = iDist(o_j)$, as we map all objects from a single cluster C_i to the interval $\langle i \cdot c, (i + 1) \cdot c\rangle$.

The originally proposed iDistance approach serves only vector spaces, however the M-Index removes this restriction and generalizes this concept to general metric spaces. M-Index picks a set of m pivots $p_1, \ldots, p_m$ beforehand and then applies Voronoi-like partitioning which finally partitions the space into m partitions, *clusters* in this context. It also eliminates the separation constant c as it assumes normalized distances for which it suffices to use $c = 1$.

Additionally, authors suggest *multi-level variant* that is more scalable for the increasing amounts of data. For this purpose, we apply recursive partitioning of clusters using *distance permutations* [84] or referred as *pivot permutations* [85, 86].

Definition 3.1 (Pivot permutation)**.** *For the fixed set of pivots $\mathbb{P} = \{p_1, p_2, \ldots, p_m\}$ and the database object $o \in \mathcal{D}$, we define the* permutation function

$$(\cdot)_o : \{1, 2, \ldots, m\} \to \{1, 2, \ldots, m\}$$

such that for $\forall i, j \in \{1, 2, \ldots, m\}$ we have

$$\begin{aligned}(i)_o \leq (j)_o \leftrightarrow \delta(o, p_{(i)_o}) &\leq \delta(o, p_{(j)_o}) \text{ or} \\ \delta(o, p_{(i)_o}) &= \delta(o, p_{(j)_o}) \text{ and } i < j\end{aligned}$$

The resulting sequence of pivots $p_{(1)_o}, p_{(2)_o}, \ldots, p_{(m)_o}$ is ordered based on the distances between pivots and the database object o and forms the pivot permutation.

Having $0 \leq L \leq m$ levels, we partition the space into $m \cdot (m-1) \cdot \ldots \cdot (m-L+1)$ clusters by recursively applying Voronoi partitioning. As the first step, we assign objects $o \in \mathcal{D}$ to their closest pivot p_i representing the cluster C_i, so we get $(0)_o = i$ for $\forall o \in C_i$. Then, we subsequently partition each cluster C_i into $n - 1$ clusters with the same approach but using only $n - 1$ pivots $\{p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_m\}$. As the result, we create clusters $C_{i,j}$ where j denotes the second closest pivot to object $o \in C_{i,j}$ which gives us $(1)_o = j$. Then, we proceed similarly further.

We have the mapping $key_L : \mathcal{D} \to \mathbb{R}$ such that for any object $o \in \mathcal{D}$ we get

$$key_L(o) = \delta(o, p_{(0)_o}) + \sum_{i=0}^{L-1} (i)_o \cdot m^{(L-1-i)}$$

where $(\cdot)_o$ is a permutation of indexes:

$$\delta(o, p_{(0)_o}) \leq \delta(o, p_{(1)_o}) \leq \ldots \leq \delta(o, p_{(m-1)_o})$$

The integral part of the obtained key value $key_L(o)$ represents the assigned cluster from Voronoi partitioning (cluster $C_{i_0,\ldots,i_{L-1}}$ is identified by the number $i_0 i_1 \ldots i_{L-1}$ in a numeral system with base m), while the fractional part provides the distance between the object o and its closest pivot $p_{(0)_o}$. We depict such a partitioning in Fig. 3.7.

Authors also introduce M-Index with dynamic levels which dynamically increase the number of levels only for large clusters in order to increase the search efficiency. For the description of this modification, we refer to papers [82, 62].

Similarity searching with M-Index starts by computing the distances between the query object q and global pivots $p_i \in \mathbb{P}$ while sorting them accordingly in order to build the pivot permutation $(\cdot)_q$. Then, we traverse the *cluster tree*

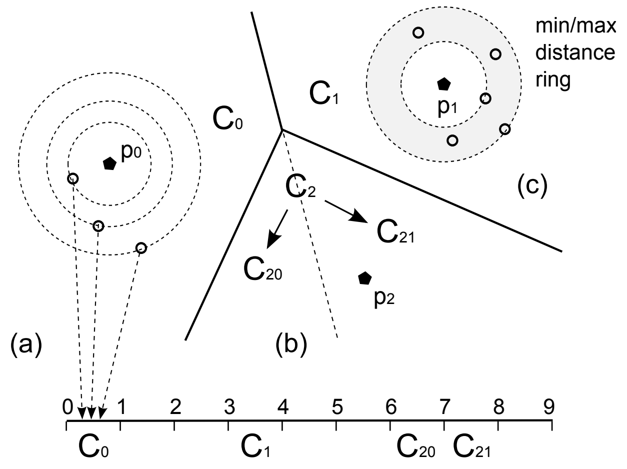


Figure 3.7: Assigning the key mapping values $key_L(o)$ for $\forall o \in \mathcal{D}$ (a), together with the recursive partitioning of objects into clusters (b); and minimal/maximal distances to the closest pivot (c) used for Ptolemaic M-Index (see Section 3.2.4).

using the *breadth-first search* (BFS) [87] and try to prune whole branches using the metric *double-pivot distance constraint* [3] which allows to skip clusters C_i if

$$\delta(q, p_i) - \delta(q, p_j) > 2r \tag{3.10}$$

where $p_j \in \mathbb{P}$ is any pivot and r represents the radius of the range query $\mathrm{RQ}_\delta(q, r)$. To maximize the left-hand side of the inequality, we use $p_j = p_{(0)_q}$ and further apply this rule L-times for each level.

For remaining cluster tree nodes, we apply the *range-pivot distance constraint* [3] which skips the cluster $C_{i,*}$ if any of the following holds

$$\delta(q, p_i) + r < r_{\min}$$
$$\delta(q, p_i) - r > r_{\max}$$

where $r_{\min}$ and $r_{\max}$ is minimum/maximum distance in the leaf cluster $C_{i,*}$

$$r_{\min} = \min\{\delta(o, p_i) \mid o \in C_{i,*}\}$$
$$r_{\max} = \max\{\delta(o, p_i) \mid o \in C_{i,*}\}$$

We get these values by storing the minimal ($key_{\min}$) and maximal ($key_{\max}$) keys in all clusters, as they are the fractional part of the keys: $r_{\max} = frac(key_{\max})$ and $r_{\min} = frac(key_{\min})$.

Next, if the cluster $C_{i,*}$ is not pruned by neither of the previous methods, we take the adequately shifted *key* interval

$$\langle \delta(q, p_i) - r, \delta(q, p_i) + r \rangle$$

to be searched which applies the *object-pivot distance constraint* [3] combined with iDistance filtering [83].

Finally, we apply the standard metric filtering with the triangle lowerbound (see Section 3.1.1) to avoid as many distance computations as possible, i.e. by employing the *pivot filtering* [3]. After all these steps, the remaining non-filtered objects o comprise the final result set.

Although we consider and describe the evaluation of range queries $\mathrm{RQ}_\delta(q, r)$, the kNN queries are quite similar and authors provide three strategies for their execution [82, 62].

According to the evaluation and comparison, authors claim that the superior performance of M-Index on the tested datasets outperforms the selected state-of-the-art metric indexing methods in terms of I/O, computational costs, and response times for similarity queries. They provide not only exact but also approximate search strategy for M-Index.

3.2.3 Indexability Indicators of Metric Access Methods

The metric postulates give the basis for various MAMs, they do not guarantee the efficiency of the proposed indexes due to the ubiquity in high-dimensional spaces [6]. For example, the performance of vector-data indexing methods is to a high extent affected by the growing dimensions, very often as much as the exponential dependency. To estimate the efficiency of a selected MAM in metric spaces, we describe two existing indicators, that provide good estimations of the data indexability of the input similarity model: *Intrinsic Dimensionality* [6] and *Ball-Overlap Factor* [52]. These approaches allow us to qualitatively compare individual datasets and reveal the problematic issue called the *curse of dimensionality.*

Intrinsic Dimensionality

We use the *Intrinsic Dimensionality* (IDim) to indicate the efficiency limits of any MAM index. It describes the underlying data and gives some meaningful insight about object clusters or how close to each other the objects in average are. We compute the IDim value ρ using the *distance distribution histogram* [66] for pair-wise distances of any two database objects. With a database $\mathcal{D}$ and a metric distance δ, we compute the mean μ and the variance σ^2 of the distance distribution as

$$\mu = \frac{1}{|\mathcal{D}|^2} \sum_{o_i \in \mathcal{D}} \sum_{o_j \in \mathcal{D}} \delta(o_i, o_j) \tag{3.11}$$

$$\sigma^2 = \frac{1}{|\mathcal{D}|^2} \sum_{o_i \in \mathcal{D}} \sum_{o_j \in \mathcal{D}} (\mu - \delta(o_i, o_j))^2 \tag{3.12}$$

Then, we define the IDim value ρ as

$$\rho(\mathcal{D}, \delta) = \frac{\mu^2}{2\sigma^2} \tag{3.13}$$

In Fig. 3.8 we see the distance distribution histograms with various IDim values. The low values ρ cover datasets with tight object clusters far from the others. This gives a good perception for pruning because the query ball is supposed to overlap only a small number of clusters that we will need to examine. On the other hand, if all objects are equally distant and the clusters overlap each other, which gives higher values of ρ, it is difficult to distinguish the individual objects/clusters in general.

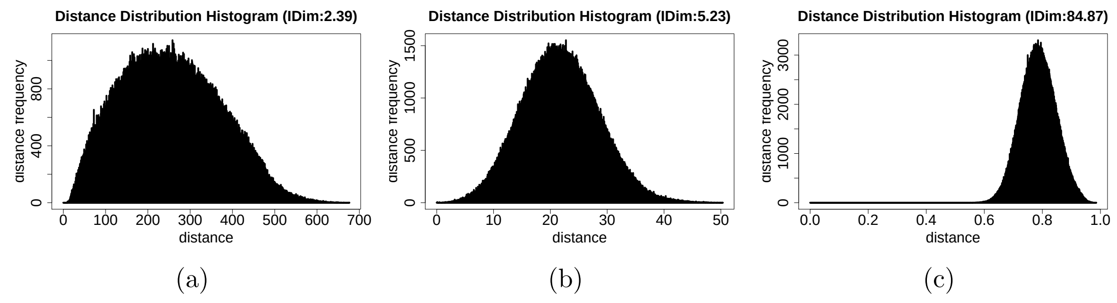


Figure 3.8: Distance distribution histograms for sample datasets with low IDim values of $\rho = 2.39$ (a), $\rho = 5.23$ (b) and a high IDim value of $\rho = 84.87$ (c)

As the result, we need to evaluate all clusters which decreases the query performance no matter of the employed indexing scheme and subsequently degrades to sequential scanning. This illustrates the problem with high intrinsic dimensionality as the generalized case of the *curse of dimensionality* [56].

Ball-Overlap Factor

As the promising alternative to IDim which gives only indirect prediction of indexing efficiency, there appeared another indicator of indexing, the *Ball-Overlap Factor* (BOF) that uncovers the relationships between regions of data clusters [52, 8]. For the database $\mathcal{D}$ with the distance δ, we define the *ball region* for the object o_i with the radius r_k that conforms to the distance to its k-th nearest neighbor as

$$\texttt{ball}_k(o_i) = (o_i, r_k) = (o_i, \delta(o_i, k\mathrm{NN}(o_i)))$$

Two ball regions $\texttt{ball}_k(o_i)$, $\texttt{ball}_k(o_j)$ overlap in the geometric meaning if

$$\delta(o_i, k\mathrm{NN}(o_i)) + \delta(o_j, k\mathrm{NN}(o_j)) \geq \delta(o_i, o_j)$$

To determine whether two ball regions overlap, we use operators *sgn* and $\overline{\cap}$:

$$sgn(|\texttt{ball}_k(o_i) \quad \overline{\cap} \quad \texttt{ball}_k(o_j)|) = \begin{cases} 1 & \text{ball regions overlap} \\ 0 & \text{otherwise} \end{cases}$$

Then, we define the BOF as

$$\mathrm{BOF}_k(\mathcal{D}, \delta) = \frac{2}{|\mathcal{D}| \cdot (|\mathcal{D}| - 1)} \sum_{\forall o_i, o_j \in \mathcal{D}, i > j} sgn(|\texttt{ball}_k(o_i) \quad \overline{\cap} \quad \texttt{ball}_k(o_j)|) \quad (3.14)$$

The resulting value contains the ratio of overlapping ball regions where each ball region contains at least $k+1$ objects including the database object itself. We can view these ball regions as indexing regions. Observe that similarly as IDim, also BOF returns index-independent measure of the underlying data indicating its indexing efficiency.

3.2.4 Ptolemaic Access Methods

While metric access methods use the triangle inequality axiom (see Table 2.1) as the basic property, there has been recently proposed the class of *Ptolemaic Access Methods* (PtoAMs) which employ the ptolemaic inequality (see Section 3.1.2) as the basic stone for indexing [51, 50]. In general, authors adapt the selected metric indexes to the ptolemaic indexing and create *Ptolemaic Pivot tables*, *Ptolemaic PM-Tree*, and *Ptolemaic M-Index*. Moreover, a recent study shows how to efficiently handle the computationally expensive signature quadratic form distance [14, 15] by these indexes. In the following text, we shortly summarizes the principles of the proposed Ptolemaic Access Methods.

Ptolemaic Pivot Tables

The proposed *Ptolemaic Pivot Table* (PtoPT) indexing structure is capable of filtering and pruning objects by either triangle inequality and/or ptolemaic inequality which provides an additional filtering power. For the latter method, we will use *pivot permutations* [84, 88] to provide good pivot pairs for any database object o (for more details see Def. 3.1).

With the formed pivot permutations, we have to employ the way of using it with a specific *heuristic* [15, 51]. Except for the naive method, authors propose the *unbalanced* and *balanced* heuristics which provides the best results.

To work properly, PtoPT maintains two components:

- the *pivot file* which stores the set of pivots $\mathbb{P}$ with the *pivot distance matrix* of $|\mathbb{P}| \times |\mathbb{P}|$ pairwise distances between any two pivots $p, s \in \mathbb{P}$

- the *index file* that encapsulates the distances between database objects $o_i \in \mathcal{D}$ and pivots $p \in \mathbb{P}$. It also saves the pivot permutation for the object o_i based on the proposed heuristics. Each entry contains:

  $$[o, (\cdot)_o, \delta(o, p_{(1)_o}), \delta(o, p_{(2)_o}) \ldots \delta(o, p_{(|\mathbb{P}|)_o})]$$

  where o is the database object, $(\cdot)_o$ is the pivot permutation sequence for the object o, and $\delta(o, p_{(i)_o})$ is the distance between the object o and $p_{(i)_o}$ which is the i-th pivot from the pivot permutation sequence $(\cdot)_o$.

  Note that the pivot permutation is a special sorting of pivots typically based on the computed distances $\delta(o, p_i)$ from the object o and given by the heuristics method. Therefore, it is generally different for each object o.

With additional information stored in the PtoPT index structure, we evaluate any similarity query as follows. We traverse the stored database objects sequentially and apply triangle / ptolemaic / both filtering options for each object o_i with the enabled early termination flag. This means, that the first filtering options which prunes the object o_i, stops the subsequent computations and we proceed with the next object o_{i+1}. For ptolemaic filtering, we limit the maximum number of pivot pairs to be examined by the value κ resulting in the total complexity of $\mathcal{O}(\kappa)$. If we set $\kappa = |\mathbb{P}|$, we will achieve the same time complexity as with metric indexes, i.e. $\mathcal{O}(|\mathbb{P}|)$.

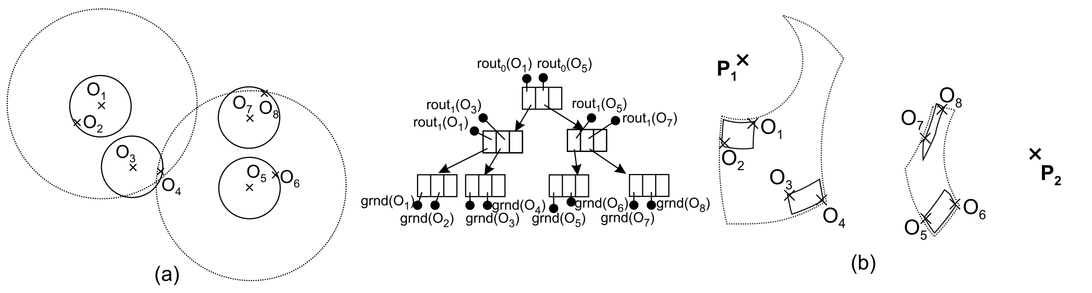


Figure 3.9: Comparing *M-tree* ball-shaped regions (a) and *PM-tree* regions with additional pruning by ring regions using two pivots p_1 and p_2 (b)

Ptolemaic PM-tree

Before we describe the ptolemaic approach to the pivoted M-Tree (PM-Tree), we outline the original PM-Tree structure and its fundamentals. The PM-Tree index combines the principles of pivot tables and M-Trees into a single index that is dynamic, persistent, paged, and extensible [89, 90, 74]. It enhances the M-tree index with extra information given by a static set of global pivots $p_i \in \mathbb{P}$. These pivots allow us to cut off the hierarchy of M-Tree ball-shaped metric regions with a set of rings. This way, the regions become more compact and provides better pruning than rigid M-tree ball-shaped regions (see the comparison in Fig. 3.9).

To achieve this performance, we modify and enhance the routing and leaf entries of M-tree (see Section 3.2.2). We adjust the routing entry of PM-tree to

$$rout_{\texttt{PM}}(o_r) = [o_r, ptr(T(o_r)), r(o,r), \delta(o_r, \mathcal{P}ar(o_r)), \mathrm{HR}]$$

where HR is an additional attribute which stands for the array of $k_{hr} \leq |\mathbb{P}|$ intervals. The t-th interval item HR_{p_t} represents the smallest interval $\mathrm{HR}_{p_t} = \langle \mathrm{HR}^{\min}_{p_t}, \mathrm{HR}^{\max}_{p_t} \rangle$ that covers distances between the pivot p_t and each of the database objects o_j stored in the leaf nodes of the subtree $T(o_r)$ where

$$\mathrm{HR}^{\min}_{p_t} = \min_{o_j \in T(o_r)} \delta(o_j, p_t)$$

$$\mathrm{HR}^{\max}_{p_t} = \max_{o_j \in T(o_r)} \delta(o_j, p_t)$$

The combination of the interval HR_{p_t} with the pivot p_t creates the (hyper)ring region (p_t, HR_{p_t}) which is defined as the ball region (p_t, $\mathrm{HR}^{\max}_{p_t}$) reduced by the corresponding "hole" (p_t, $\mathrm{HR}^{\min}_{p_t}$) as stated in [51].

Adequately, we amend the leaf (ground) entry of PM-Tree to be

$$leaf_{\texttt{PM}}(o_l) = [o_l, id(o_l), \delta(o_l, \mathcal{P}ar(o_l)), \mathrm{PD}]$$

where PD is an array of $k_{pd} \leq |\mathbb{P}|$ pivot distances such that $\mathrm{PD}_{p_t} = \delta(o_l, p_t)$.

The main advantage comes with similarity queries when we map the query object q to the $k = |\mathbb{P}|$-dimensional pivot space. In this case, the $\texttt{query\_ball}_\delta(q, r_q)$ is represented as the hyper-cube:

$$\langle \delta(q,p_1) - r_q, \delta(q,p_1) + r_q \rangle \times \ldots \times \langle \delta(q,p_k) - r_q, \delta(q,p_k) + r_q \rangle$$

which we use for checking the overlaps with minimum bounding rectangles of the routing and leaf nodes produced by the global pivots $p_i \in \mathbb{P}$. If the MBRs do not overlap, we prune and filter out the whole sub-tree.

The modification of the existing PM-Tree to become *Ptolemaic PM-Tree* (PtoPM-Tree) includes a new filtering method based on the ptolemaic lowerbounds (see Section 3.1.2). Again, as with ptolemaic pivot tables, we can use both (triangle and ptolemaic) lowerbounds in synergy. The index structure holds

- the *pivot file* which stores the set of pivots $\mathbb{P}$ with the *pivot distance matrix* of $|\mathbb{P}| \times |\mathbb{P}|$ pairwise distances between any two pivots $p, s \in \mathbb{P}$
- the *index file* that conforms to PM-Tree index structure. We modify the leaf nodes to include the pivot permutation:

$$leaf_{\texttt{PtoPM}}(o_l) = [o_l, id(o_l), \delta(o_l, \mathcal{P}ar(o_l)), \mathrm{PD}, (\cdot)_{o_l}]$$

While evaluating similarity queries, we use the array of pivots PD which employ the naive heuristics only. Two additional filtering options are discussed further [51]:

- *Extra pivot for ptolemaic filtering* – using the parent node as the dynamic but local pivot for filtering leave nodes
- *Ptolemaic shell filtering* – using intersections of hyper-rings (*shells*) stored in routing nodes

The residual parts of the original PM-Tree index structure remain unchanged and work as described previously.

Ptolemaic M-Index

The extended version of the original M-Index metric index (see Section 3.2.2) for ptolemaic filtering, the *Ptolemaic M-Index* (PtoM-Index), is a minor modification of its basis which contains

- the *pivot file* which stores the set of pivots $\mathbb{P}$ with the *pivot distance matrix* of $|\mathbb{P}| \times |\mathbb{P}|$ pairwise distances between any two pivots $p, s \in \mathbb{P}$
- the *index file* that conforms to M-Index structure while storing *Ptolemaic Pivot Table* instead of the metric variant

The process of building the PtoM-Index is identical except for the fact that we build and maintain the ptolemaic pivot table instead of a regular pivot table. The querying still considers the metric filtering as the initial step to determine the relevant *buckets* from the cluster tree and prune the non-interesting ones. Then, we also apply the ptolemaic indexing to filter out additional irrelevant objects. The rest of objects is scanned sequentially, as usual for pivot tables.

Applicability of Ptolemaic Access Methods

The extensive evaluation of individual ptolemaic access methods [51] reveals their superiority over standard metric access methods if the distance measure is ptolemaic metric. There are factors that highly affect the overall performance such as the pivot selection methods, heuristic methods for best pivot pairs, or the number of candidate pairs to be considered. However, for other distances, it seems that the MAMs will perform better in general. It still remains an open question whether there exist any ptolemaic distance which is non-metric.

3.2.5 TriGen – Framework for Nonmetric Searching

While the previous methods require the validity of metric space or ptolemaic space postulates, there are lots of distance measures that do not conform to them and violate one or more of the properties. If we limit the distance δ to symmetric and non-negative distances, we are able to transform it to the metric distance algorithmically using the *TriGen* algorithm [52, 8].

TriGen provides an automated way of finding the optimal solution for exact or approximate indexing and similarity searching for nonmetric spaces in which the triangle inequality is violated. Its flexibility allows to search for solutions that provide exact but slower or approximate but faster retrieval using the input distances as the black-box functions. Because it is relatively trivial, yet complete solution for nonmetric distances, we take it as the standard base for indexing nonmetric similarity models. Before we sketch the idea of TriGen algorithm, we provide assumptions for applying it correctly.

TriGen Fundamentals

As the basis for the algorithm, we will use *similarity-preserving* (SP) modification, *SP-modifier*, and *similarity ordering* [8].

Definition 3.2 (SP-modifier)**.** *For the normalized distance $\delta : \mathcal{D} \times \mathcal{D} \rightarrow \langle 0, 1\rangle$, we define the* similarity-preserving modification *of δ as the transformation δ^f*

$$\delta^f(o_i, o_j) = f(\delta(o_i, o_j)) \tag{3.15}$$

where $f : \langle 0, 1\rangle \rightarrow \langle 0, 1\rangle$ is a strictly increasing function called similarity-preserving *(SP) modifier for which $f(0) = 0$.*

Definition 3.3 (Similarity Ordering)**.** *We define thesimilarity ordering $SO_\delta : \mathbb{U} \rightarrow 2^{\mathbb{U}\times\mathbb{U}}$ as the function that orders objects by their distances to the query object q:*

$$\forall o_i, o_j, q \in \mathbb{U} : \quad \langle o_i, o_j\rangle \in SO_\delta(q) \leftrightarrow \delta(q, o_i) < \delta(q, o_j) \tag{3.16}$$

It has been proved that any SP-modification of the distance δ gives us the same similarity ordering SO [8]. This results in the fact that while performing the sequential scan, we can use either δ or δ^f for pruning, as they produce the same similarity ordering.

There is a special group of metric-preserving SP-modifiers, e.g., concave SP-modifiers. If we have a strictly concave SP-modifier, we call it *triangle-generating modifier* (TG-modifier). TG-modifiers provide the basis for the TriGen, as it always exists TG-modifier f for which the SP-modification δ^f is a metric.

For completeness, we introduce also *triangle-violating* (TV) modifiers (e.g., convex SP-modifiers) which provide the backward sequence – while TG-modifiers turn semi-metric into metric, TV-modifiers transform metric into semi-metric.

Definition 3.4 (Triangular triplet, Distance triplet)**.** *We take the* triangular triplet *or simply* triplet *as the 3-tuple $[a, b, c]$ of real numbers $a, b, c \in \mathbb{R}^+_0$ such that $a+b \geq c$, $a+c \geq b$, and $b+c \geq a$. With a metric distance δ, we define* distance triplets *for any three database objects $o_i, o_j, o_k \in \mathcal{D}$ as $[\delta(o_i, o_j), \delta(o_j, o_k), \delta(o_i, o_k)]$.*

We use the triangular triplets for measuring the retrieval error of (nonmetric) distances in terms of the number of violated triangle inequalities, *T-error* or the *triangle-error.*

Definition 3.5 (T-error)**.** *We define* T-error *as the ratio between the number of distance triplets that violate the triangle inequality and are non-triangular ($m^\times$) and the number of sampled triplets (m) from the database sample $S \subseteq \mathcal{D}$:*

$$\epsilon_{\delta,S} = \frac{m^\times}{m} \tag{3.17}$$

Because computing the T-error value for the whole database $\mathcal{D}$ and thus evaluating all $m = \binom{\mathcal{D}}{3}$ triplets is very extensive and not applicable in most cases, we employ sampling approaches and examine only a sample of the database $S \subseteq \mathcal{D}$ [52, 8]. The sampling approaches minimizes the difference between $\epsilon_{\delta,S}$ and $\epsilon_{\delta,\mathcal{D}}$ while keeping the total triplets count m as low as possible.

We will use this measurement over the triangle-violation error [8] even though it depends on both – the distance δ and the selected data sample S. The main reason is, that it better corresponds to the expected behavior of the optimal T-Modifier (TV- or TG-modifier) with the rest of the data.

As we will describe later, for automation purposes of the TriGen algorithm, we put an extra parameter to the T-modifiers with a concavity/convexity weight w and mark it as the *T-base.*

Definition 3.6 (T-base)**.** *The function $g : \langle 0,1\rangle \times \mathbb{R} \rightarrow \mathbb{R}_0^+$ is a* base of T-modifiers *or* T-base *with the* concativity–convexity weight $w \in \mathbb{R}$ *if*

1. $g(x,0) = x$
2. $g(x,w), w > 0$ *is a TG-modifier*
3. $g(x,w), w < 0$ *is a TV-modifier*
4. $w_1, w_2 > 0$ *and* $w_1 > w_2$*, then* $g(x,w_1) > g(x,w_2)$ *for* $\forall x \in (0,1)$
5. $w_1, w_2 < 0$ *and* $w_1 > w_2$*, then* $g(x,w_1) < g(x,w_2)$ *for* $\forall x \in (0,1)$
6. *g is continuous:* $\lim_{w_1 \rightarrow w_2} g(x,w_1) = g(x,w_2)$ *for* $\forall x \in \langle 0,1\rangle$

The concativity–convexity weight w serves as the parameter for continuously adjusting the T-modifier to increase/decrease the resulting T-error ratio. The increasing value of $w > 0$ gives more concave function g which lowers T-error of $\delta^{g(w)}$, while decreasing the value of $w < 0$ produces the convex function g which raises T-error value of $\delta^{g(w)}$. As an example, authors suggest two T-bases [8]:

- simple *Fractional-power* (FP-base) pictured in Fig. 3.10a defined as:

$$\mathrm{FP}(x,w) = \begin{cases} x^{\frac{1}{1+w}} & \text{for} \quad w > 0 \\ \\ x^{1-w} & \text{for} \quad w \leq 0 \end{cases}$$

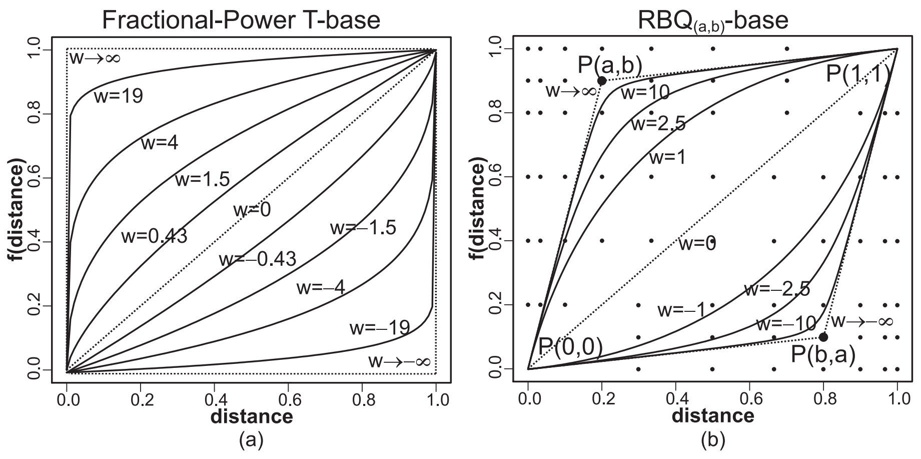


Figure 3.10: *Fractional-power* T-base (FP-base) (a) and *Rational Bézier-Quadratic* T-base (RBQ-base) (b) values for distances $\delta \in \langle 0, 1\rangle$

- and more sophisticated *Rational Bézier-Quadratic* (RBQ-base[1]) depicted in Fig. 3.10b which we define as

$$\text{RBQ}_{(a,b)}(x, w) = \begin{cases} rbq(x, w, a, b) & \text{for} \quad w > 0 \\ \\ rbq(x, -w, a, b) & \text{for} \quad w \leq 0 \end{cases}$$

Figure 3.10 shows how a specific T-base modifies the original distance values within the interval $\langle 0, 1\rangle$. Observe the behavior for positive values $w > 0$ which *inflate* the distances and for negative values $w < 0$ that decrease the distances.

TriGen Algorithm

Having all the prerequisites, we describe the overall process of finding the optimal T-modifier for the given distance δ with respect to the given database sample S. Together with these two inputs, we provide also T-error *threshold ratio* θ which represents the acceptable number of violated triangle inequalities.

First, we sample a predefined number of *triplets* for measuring the T-error of the candidate T-modifiers, and create the T-bases. The algorithm selects the T-modifier and tunes the weight parameter w in order to find the optimal value. We start by setting the right direction either for TG-modifiers ($w = 1$) for nonmetric (triangle inequality-violating) distances, or for TV-modifiers ($w = -1$) for metric distances. To do this, we compute the initial T-error for the given model.

Then, we perform a fixed number of iterations to find the target w value. In each iteration, we either increase or decrease the value of w based on the current T-error compared to the threshold value θ. To help us with bounding the w value, we use previously computed lower- (w_{LB}) and upper- (w_{UB}) bounds of the parameter w. We use these bounds regularly for adjusting the next w value by either halving or doubling the interval in order to accelerate the computation.

Finally, we compute the *indexability* of the final T-modifier which stands for computing the *intrinsic dimensionality* [6], or the *ball-overlap factor* [52] using the previously sampled distance triplets (see Section 3.2.3). We depict the generalized TriGen process with all individual steps in Algorithm 2.

[1]For a detailed description of *rbq* function, we refer readers to [8].

Algorithm 2 TriGen (S, δ, θ)

```
Require: database sample $S$, (semi-)metric distance $\delta$, T-error threshold $\theta$
1: $T \leftarrow$ SampleTriplets$(S, \delta)$
2: $\mathcal{F} \leftarrow$ CreateT-Bases()
3: for all $f \in \mathcal{F}$ do
4:   $w = 0$; $w^* = \infty$
5:   $error \leftarrow$ ComputeT-Error$(f, w, T)$
6:   if $error < \theta$ then
7:     $w_{\texttt{LB}} = -\infty; w_{\texttt{UB}} = 0; w = -1$
8:   else
9:     $w_{\texttt{LB}} = 0; w_{\texttt{UB}} = \infty; w = 1$
10:  end if
11:  for ($i = 0$; $i <$ MaxIterations; $i$++) do
12:    $error \leftarrow$ ComputeT-Error$(f, w, T)$
13:    if $error \leq \theta$ then
14:      $w^* = w$; $w_{\texttt{UB}} = w$
15:    else
16:      $w_{\texttt{LB}} = w$;
17:    end if
18:    if $w_{\texttt{LB}} = -\infty$ or $w_{\texttt{UB}} = \infty$ then
19:      $w = 2 \cdot w$
20:    else
21:      $w = (w_{\texttt{LB}} + w_{\texttt{UB}})/2$
22:    end if
23:  end for
24:  if $w^* \neq \infty$ then
25:    $idx$ = ComputeIndexability$(f, w^*, T)$;
26:    if $idx >$ maxIndexability then
27:      $f_{\texttt{best}} = f; w_{\texttt{best}} = w^*$
28:      maxIndexability = $idx$;
29:    end if
30:  end if
31: end for
32: return T-Modifier$(f_{\texttt{best}}, w_{\texttt{best}})$ {return the best T-base $f_{\texttt{best}}$ }
```

TriGen Time Complexity

We acknowledge that TriGen seems perfect for converting semi-metric distances to metric ones, even though we need to study its time complexity to reveal its potential for real-world applications. The total time complexity of TriGen is

$$\mathcal{O}(|S|^2 \cdot \mathcal{O}(\delta) + \text{MaxIterations} \cdot |\mathcal{F}| \cdot m)$$

where $|S|^2$ is required for building the distance matrix for the sample $S \subseteq \mathcal{D}$, $\mathcal{O}(\delta)$ is the time complexity of evaluating a single distance value between any two objects, *MaxIterations* parameter defines the number of iterations when searching for the appropriate T-modifier, $|\mathcal{F}|$ is the size of all tested T-bases, and m is the number of sampled triplets.

With TriGen, we are able to handle efficiently some of the nonmetric similarity models in a unified way. The biggest advantage of TriGen is that its conversion from semi-metric to metric distances allows us to consider any suitable metric access method for indexing of the transformed models.

3.2.6 NM-Tree

The introduction of TriGen has provided a great motivation for further research of nonmetric spaces. With a clear goal of finding the optimal value w for the parametrized T-modifier, we can handle any semi-metric space and subsequently use any suitable MAMs for indexing. However, this has proved to be a limiting factor for various cases due to the strong requirement of computing w value and applying it before indexing. For approximated queries, we need to known prior to the indexing the levels of user-defined error tolerance and the query precision.

To overcome this challenge and to provide higher degree of the flexibility during the processing of similarity queries, the *nonmetric tree* (NM-tree) has been proposed [75]. It combines M-Tree index with TriGen algorithm in order to provide flexible approximate (non)metric search. It allows users to specify the precision at the query time which gives a benefit of tuning the trade-off between the efficiency and the precision for each performed query.

Regarding the structure of NM-tree index, if follows the original structure of M-Tree, with a slightly modified insertion algorithm. We assume the input distance δ to be semi-metric (see Section 2.2.2) that violates the triangle inequality property (see Table 2.1). We turn this distance δ into a metric δ^{f_M} with a T-modifier f_M for the threshold value $\theta = 0$ using standard TriGen algorithm (see Section 3.2.5). With zero T-error, we store the modified distances under the metric $\delta^{f_M}(\cdot,\cdot)$ to the underlying M-tree structure.

To function properly, we need to obtain *inversely symmetric* T-bases f_i such that for $\forall w \in \mathbb{R}$ and for $\forall x \in \langle 0, 1\rangle$

$$f_i(f_i(x, w), -w) = x$$

Following the proposed notation, we mark these inversed functions as $f_i^{-1}(\cdot, w)$. As authors state, both previously introduced T-bases (FP-base and RBQ-base; see Def. 3.6) hold this condition [75].

The main difference from the M-Tree is the algorithm for query evaluation. The *exact* search is simple as we use the standard M-Tree algorithm for traversing the tree-based index structure. We only update the value of the *query radius* r before evaluating any range query $\mathrm{RQ}_\delta(q, r)$ and use the value $f_M(r)$ instead. After the execution, if we want to return also the distances to the query object q, we need to reverse the distances of returned objects $\delta^{f_M}(q, o_i)$ to $f_M^{-1}(\delta^{f_M}(q, o_i))$.

The *approximate* search with the accepted error threshold of $\theta > 0$ for which we compute adequate T-modifier f^θ is more complex as we need to modify all distances $\delta^{f_M}(\cdot,\cdot)$ in two ways. We apply the inversed T-modifier f_M^{-1} followed by the required T-modifier f^θ which results in:

$$f^\theta(f_M^{-1}(\delta^{f_M}(\cdot,\cdot)))$$

Besides the direct distances between objects (and their parents), the index structure of NM-tree includes also radii which consists of aggregation.

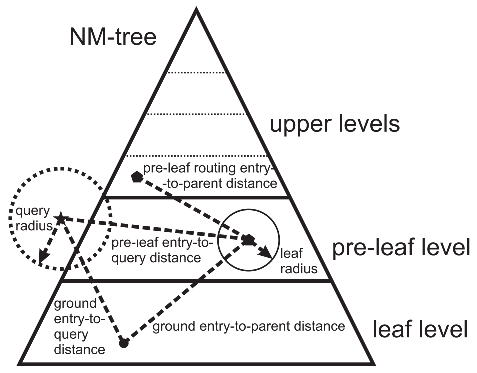

NM-tree
upper levels
pre-leaf routing entry--to-parent distance
query radius
pre-leaf entry-to-query distance
leaf radius
pre-leaf level
ground entry-to-query distance
ground entry-to-parent distance
leaf level

Figure 3.11: Dynamic modification of distances in NM-tree at the query time on *leaf* and *pre-leaf* levels

Except for the two lowest levels (*leaf* and *pre-leaf* level), we cannot apply the distance modifications due to the inherited direct distances produced as the consequence of node splitting (see Fig. 3.11). So we perform exact metric search at higher levels and the approximation on the two lowest levels.

The experiments show that the NM-tree performs as fast as multiple M-Tree indexes for specific approximation levels, overall with greater flexibility options.

3.2.7 Fuzzy Similarity

Another specific idea of *fuzzy indexing* [18, 91, 19] reveals a method that combines the similarity search domain with the *fuzzy logic*. In their work, authors focus on pure similarity models which conform to the prerequisites of the fuzzy logic and so they diminish the necessity of bi-directionally converting similarities to distances and vice versa which might provide a lossy transformation. Here, the loss of information within the conversion could lead to the perfect behavior in distance-model, however the backward transformation to similarities inherently conduces to increased error rates [19].

To overcome issues that nonmetric similarities bring, authors replace the standard metric model with the *fuzzy similarity* and *fuzzy logic model*. Instead of transforming or modifying the non-metric spaces (distances/data), we use a suitable function or fuzzy operators which is one of the first attempts to apply fuzzy concepts to similarity indexing. Previously, authors proposed a utilization of fuzzy sets to reason about distance measures but only to user-perceived real-world distances [92]. Also, much more interest has been on the indexing in fuzzy databases [93, 94]. However, the authors deal with the fuzzy data stored in a database, while this approach uses crisp data interpreted with fuzzy operators.

Fuzzy Logic Model Introduction

The concept of using *fuzzy logic* for indexing non-metric data [18] leads to the data indexing with *fuzzy operators*. Instead of modifying the distances or data, we tune the "+" operator and thus replace the traditional triangle inequality condition in metric filtering by so called parameterized *T-norm* operators [95].

Definition 3.7 (T-norm, Fuzzy T-norm). *A parameterized triangular norm (T-norm) T_λ is a binary operation $T_\lambda : \langle 0,1\rangle \times \langle 0,1\rangle \to \langle 0,1\rangle$, such that for a fixed λ value and $\forall x,y,z \in \langle 0,1\rangle$, the following conditions are satisfied:*

(T1)	$T_\lambda(x,y) = T_\lambda(y,x)$	commutativity
(T2)	$T_\lambda(T_\lambda(x,y),z) = T_\lambda(x, T_\lambda(y,z))$	associativity
(T3)	$T_\lambda(x,y) \leq T_\lambda(x,z)$ whenever $y \leq z$	monotonicity
(T4)	$T_\lambda(x,1) = x$	boundaries

In all cases, we work with similarities, so it is natural to limit the boundaries of any T-norm to the interval $\langle 0,1\rangle$. Moreover, the similarity between objects $s(o_i,o_j)$ represents the degree of the fuzzy predicate "o_i and o_j are the same". The aim of T-norms is to provide an alternative to the triangle inequality (see Table 2.1) which is forthwith replaced by the *triangle transitivity* or simply *T-transitivity.*

Definition 3.8 (T-transitivity). *With a similarity function s that returns the value of a similarity between any two objects, we define the* T-transitivity *as*

$$s(o,q) \geq T^{\wedge}_{\lambda}(s(o,p), s(p,q)) \tag{3.18}$$

where $o,p,q \in \mathcal{D}$ are objects from the database $\mathcal{D}$ and $T^{\wedge}_{\lambda}$ is a T-norm conjunction. We denote s as T-transitive *if the* T-transitivity *holds for it.*

There is an obvious duality between the triangle inequality and the *T-transitivity* for any three database objects $o,p,q \in \mathcal{D}$ with respect to the corresponding (dual) similarity (s) and distance (δ) functions

$$\delta(q,o) \leq \delta(q,p) + \delta(p,o) \tag{3.19}$$
$$s(q,o) \geq T^{\wedge}_{\lambda}(s(q,p), s(p,o)) \tag{3.20}$$

This duality maps the inequality operator "$\leq$" from the inequality (3.19) to "$\geq$" in the inequality (3.20), and similarly the operator "$+$" to operator "$T^{\wedge}_{\lambda}$". While the $+$ operator is fixed in the metric spaces, in the fuzzy model, we have a flexible and parameterizable conjunction $T^{\wedge}_{\lambda}$.

Fuzzy Similarity Searching

The similarity searching within the introduced fuzzy logic model uses the same *object-pivot distance constraints* [3] in terms of lower/upper bounds for distances. However, we need to use the *residuation* from the fuzzy logic [95], namely the fuzzy implication $T^{\to}_{\lambda}$ residual to the conjunction $T^{\wedge}_{\lambda}$ for any three real values $x,y,z \in \langle 0,1\rangle$:

$$T^{\wedge}_{\lambda}(x,y) \leq z \quad \to \quad T^{\to}_{\lambda}(x,z) \geq y$$

If we have a T-transitive similarity function s according to the Def. 3.8, use objects q (query), p (pivot), and o (object) from the database $\mathcal{D}$, and replace x,y,z for $x = s(q,p)$, $y = s(q,o)$, and $z = s(p,o)$, we will get

$$T^{\to}_{\lambda}(s(q,p), s(p,o)) \geq s(q,o)$$

Algorithm 3 LTA – Lambda Tuning Algorithm (t, θ, S)

```
Require: T-norm $T$, T-error threshold $\theta$, database sample $S$
 1: $\lambda_{\mathtt{best}} = -1$; $\lambda_{\min} = 0$; $\lambda_{\max} = T$.getLambdaMax()
 2: $\lambda = (\lambda_{\min} + \lambda_{\max})/2$
 3: $triplets \leftarrow$ SampleTriplets$(S, \delta)$
 4: for ($i = 0$; $i <$ MaxIterations; $i$++) do
 5:    $error$ = ComputeT-TransitivityError$(T, \lambda, triplets)$;   {violating triplets}
 6:    if $error \leq \theta$ then {worsen the operator by increasing $\lambda$}
 7:       $\lambda_{\mathtt{best}} = \lambda_{\min} = \lambda$;
 8:    else {improve the operator by decreasing $\lambda$}
 9:       $\lambda_{\max} = \lambda$;
10:    end if
11:    $\lambda = (\lambda_{\min} + \lambda_{\max})/2$;
12: end for
13: return $\lambda_{\mathtt{best}}$; {Return the best $\lambda$ found for the T-norm $T$}
```

Then, we can use the following lower- and upper- bounds for the estimation of the similarity between the query q and any object o.

$$T^{\wedge}_{\lambda}(s(q,p), s(p,o)) \leq s(q,o)$$
$$s(q,o) \leq \min\{T^{\rightarrow}_{\lambda}(s(q,p), s(p,o)), T^{\rightarrow}_{\lambda}(s(p,o), s(q,p))\}$$

This gives us a method of evaluating similarity queries using the parametrized T-norms. The only challenge is to find the optimal λ value for a given T-norm T that will behave efficiently for the database $\mathcal{D}$ with a similarity function s.

Lambda Tuning Algorithm

Finding the optimal value of λ parameter for a parametrized T-norm operator T is the most crucial point when using fuzzy logic for data indexing. Inspired by the TriGen algorithm [8] described in Section 3.2.5, there emerged the *Lambda Tuning Algorithm* (LTA) [19] which aims to address this challenge.

Initially, authors normalize all T-norms T to accept $\lambda \in \langle 0, \infty)$ in which $\lambda = 0$ is the most flexible (with the minimal error) and $\lambda = \infty$ is the most strict (with the maximal error). The LTA returns the most suitable λ parameter for the given fuzzy T-norm operator T, error threshold θ, and the given database sample S. It works as follows.

First, we sample the triplets from the database sample S to be used for the tuning mechanism and initialize the minimum/maximum values of λ parameter, as they might differ for various families of T-norms.

Then, in each iteration we compute the error ratio (the number of violating triplets) for the sample S and the current λ value of the T-norm T. While in TriGen we compute triplets violating the triangle inequality (T-error ratio, see Def. 3.5), here we count triplets for which the T-transitivity does not hold (see Def. 3.8, especially Eq. 3.18). According to the resulting error value, we modify the λ parameter and continue with the next iteration. At all times, we maintain the current interval of suitable and applicable λ values $\langle \lambda_{\min}, \lambda_{\max} \rangle$. We return the best $\lambda_{\mathtt{best}}$ for the given T-norm T as depicted in Algorithm 3.

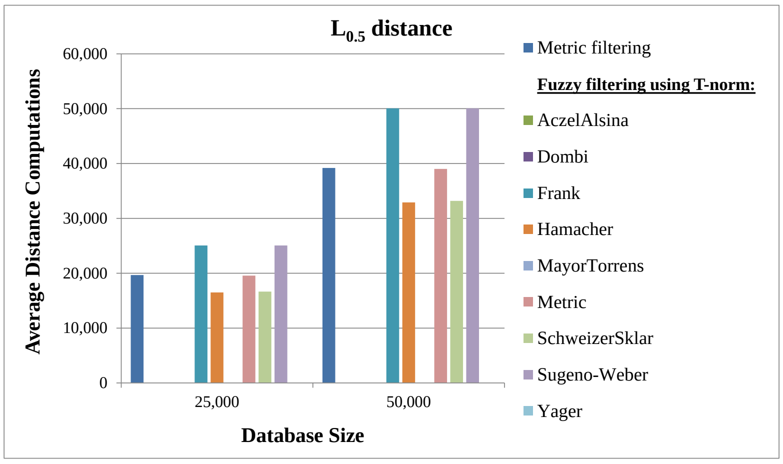


Figure 3.12: Average Distance Computations of Fuzzy T-Norms

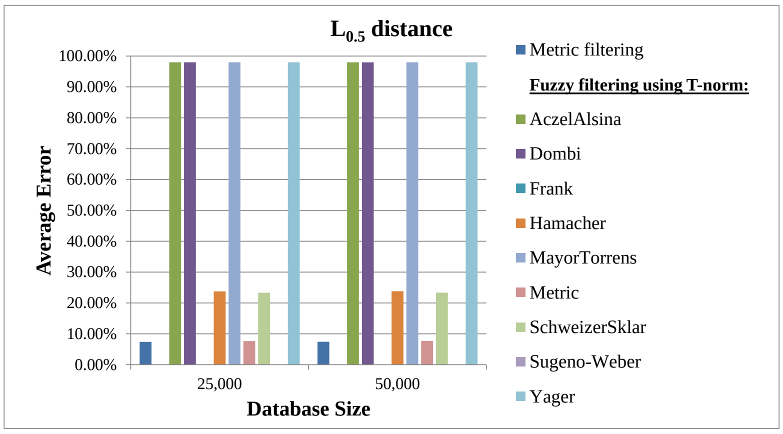


Figure 3.13: Average Error of Fuzzy T-Norms

Evaluation of Lambda Tuning Algorithm

The authors of the proposed LTA algorithm verified its validity while studying the behavior of nine different fuzzy families of T-norms defined in [95], namely *AczelAczina, Dombi, Frank, Hamacher, MayorTorrens, Metric, SugenoWeber, SchweizerSklar,* and *Yager*. The major focus is put on upper bound limits and how they adapt to the error threshold θ changes which provides the efficiency/effectiveness of selected fuzzy T-norms.

For the purpose of experimental evaluations, they use the *CoPhIR* database [96] with up to 50,000 images represented by the normalized vectors with 512 dimensions. They apply the fractional L_p functions (see Eq. 2.3) with $p = 0.5$ (semi-metrics), and the distance-to-similarity conversion $s(q, o) = \frac{1}{1+\delta(q,o)}$.

Fuzzy Pivot Tables

For experimental evaluation, we replace the conventional Pivot table filtering [3, 48] by the *fuzzy filtering* in which the *distance lowerbound* is substituted for the corresponding *similarity upperbound* obtained by the specific T-norm T_λ using their duality (see Section 3.2.7).

Then, we examine the *efficiency* (the average number of distance computations required to evaluate the similarity query; see Fig. 3.12) and the *effectiveness* (the precision of the result together with the average error; see Fig. 3.13). We measure the error rate as the difference between the result sets obtained by the sequential scanning (SEQ) and by the pivot table for the given query. We take into account the presence of actual identifiers of the objects $id(o)$ within the result set, not their distances to the query. We evaluate the results on range queries using 50 randomly selected objects with maximum error rate of 0.001%.

The closer look reveals that even though the fuzzy *T-norms* supposed to be flexible and more adaptable compared to the classic non-metric searching, the best results showed only a small speed up of the query evaluation for the price of higher error rate (e.g. *Hamacher*, *SchweizerSklar*).

This means that the proposed usage of T-norms is not applicable to the examined data. The reason is the discrepancy between the λ parameter of T-norm T evaluated (found) for the given error threshold ratio θ and the (considerable) error rate that T_λ produces when applied for fuzzy filtering of similarity queries.

Moreover, there continuously occur bi-directional distance-to-similarity conversions which are required for the fuzzy filtering to be working in pivot tables. This is a major drawback for any distance measure and might degrade the overall performance and efficiency. But as authors notice, this is still only an unconfirmed speculation which needs to be examined into more details by applying fuzzy T-norms to pure similarity models. Then, we can definitely determine whether indexing similarity models with fuzzy logic brings any advantages which again remains as the future work.

3.2.8 Specific and Alternative Techniques

Besides the mainstream SAMs, MAMs, PtoAMs or the specific fuzzy similarity approach, sometimes there also appears completely different methods that try to handle (non)metric data efficiently. The majority of these approaches focus on domain-specific tasks such as the non-metric similarity searching in the area of tandem mass spectra [37] or in the area of protein databases [12]. There are many of such solutions primarily targeted to a specific domain which employ a customized version of an existing indexing method. This is however their main advantage (specifically tuned for a particular problem) and disadvantage (they could not be generally applied) at the same time.

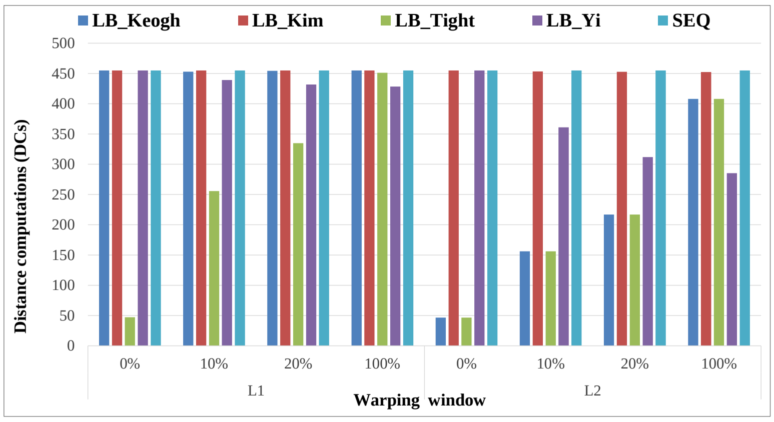
Indexing Time Series with Dynamic Time Warping Distance

One example is the research of time series together with DTW distance (see Section 2.2.2). The simplest idea of speeding up expensive DTW distance computations is to use the *early abandoning* method. Given a radius r_Q of a query that is being evaluated[2], we can stop the DTW computation if we know that the final distance will be greater than r_Q [97]. This eliminates further computations of the DTW matrix, discards inappropriate objects early enough, and provides not approximate but exact results. In fact, the early abandoning makes DTW a lowerbounding function to itself.

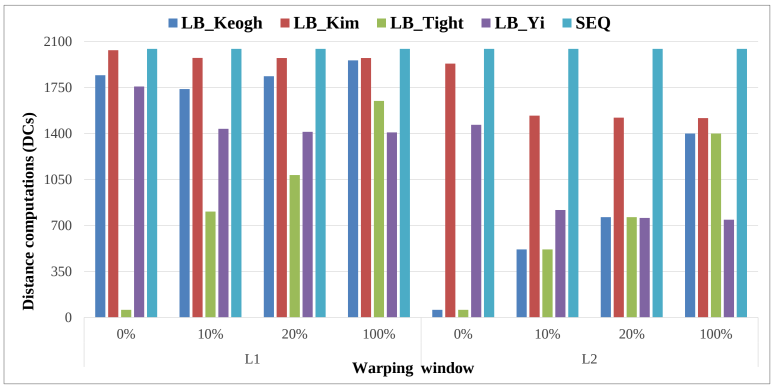
[2]Either the fixed radius of a range query, or the current radius of a kNN query.

Furthermore, we study additional specific time series lowerbound approaches named according to their authors as proposed in [46]:

1. `LB_Yi` computes the lowerbound of the distance between two time series Q, S depending on the arrangements of their ranges $\langle \min(Q), \max(Q) \rangle$ and $\langle \min(S), \max(S) \rangle$ [47].
2. `LB_Kim` provides guaranteed lowerbounds when the ground distance δ_g is any L_p distance function (see Section 2.2.1) which, in our terms, applies to any $\mathrm{DTW}(\cdot, \cdot, L_p)$. The mechanism is based on extracting and using 4-tuple vectors consisting of (1) first, (2) last, (3) smallest (min), and (4) largest (max) elements of two time series Q, S [98]. To compute the lowerbound value, authors consider the maximum value of all absolute values of differences
$$\texttt{LB\_Kim}(Q, S) = \max(|q_1 - s_1|, |q_{\text{last}} - s_{\text{last}}|, |q_{\min} - s_{\min}|, |q_{\max} - s_{\max}|)$$
It is a simple observation, that for long time series this typically does not give a very tight lowerbound. On the other hand, the method works with time series with different lengths.
3. `LB_Keogh` is considered to be the best lowerbounding method for constrained DTW distance and fixed-length time series [46]. With the warping window constraint ω, this method encapsulates any time series S of length n into two additionally computed time series U_S and L_S (upper and lower part):
$$u_i = max(s_{i-\omega}, \ldots, s_{i+\omega}), \quad l_i = min(s_{i-\omega}, \ldots, s_{i+\omega})$$
Given a query object Q together with time series U_S, L_S that correspond to Keogh's envelope for an input time series S (all with the same length n), the proposed lower bound `LB_Keogh` is defined as
$$\texttt{LB\_Keogh}(Q, S) = \sqrt{\sum_{i=1}^{n} \begin{cases} (q_i - u_i)^2 & \text{if } q_i > u_i \\ (q_i - l_i)^2 & \text{if } q_i < l_i \\ 0 & \text{otherwise} \end{cases}}$$
Several experiments verified claims that this lowerbound is the tightest one compared to other approaches [46]. The same research group led by Eamonn Keogh makes an enormous effort in speeding DTW-based similarity search. In famous papers [46, 42] introducing `LB_Keogh`, authors consider additional dimensionality reduction of the time series using piecewise aggregate approximation and indexing of reduced envelopes by R-tree, together with high-level indexing solutions such as iSAX framework [99, 100].
4. `LB_Tight` as the generalized method of `LB_Keogh` applicable to a wider range of ground distance functions in generalized $\mathrm{GDTW}(\cdot, \cdot, \delta_g, \omega, f)$ [43]. Given a query object Q together with time series U_S, L_S that correspond to Keogh's envelope for an input time series S (all with the same length n), using ω as the warping window parameter, the proposed lowerbound `LB_Tight` for $\mathrm{GDTW}(Q, S, \delta_g, \omega, f)$ is defined as
$$\texttt{LB\_Tight}(Q, S, \delta_g, f) = f\left(\sum_{i=1}^{n} \begin{cases} \delta_g(q_i, u_i) & \text{if } q_i > u_i \\ \delta_g(q_i, l_i) & \text{if } q_i < l_i \\ 0 & \text{otherwise} \end{cases}\right)$$



(a) *50words* dataset



(b) All fixed-length datasets

Figure 3.14: Average number of Distance Computations for DTW lowerbounds

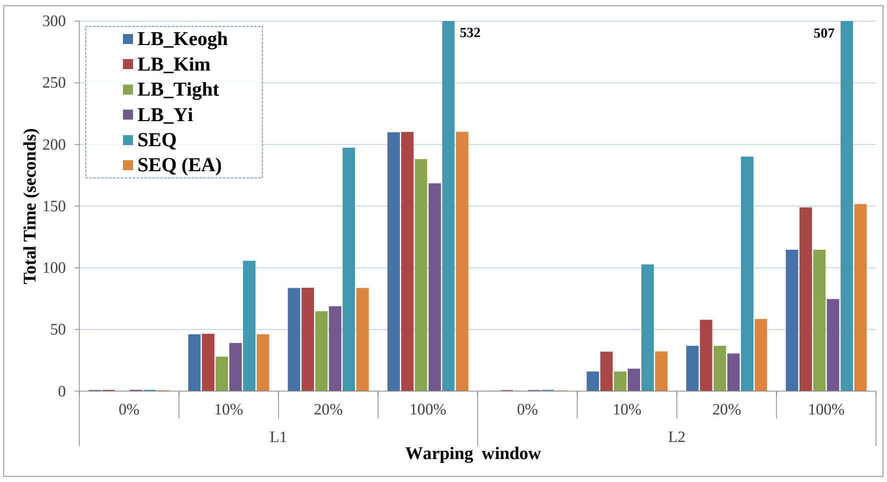
With the introduction of LB-compliant distance function, we can easily prove that `LB_Tight` is a true lowerbound – for any two time series Q and S of the same length n, for any value of the warping window ω such that $(j-\omega) \leq i \leq (j+\omega)$, and for any ground distance δ that is LB-compliant, the following inequality holds[3]:

$$\texttt{LB\_Tight}(Q, S, \delta, f) \leq GDTW(Q, S, \delta, \omega, f)$$

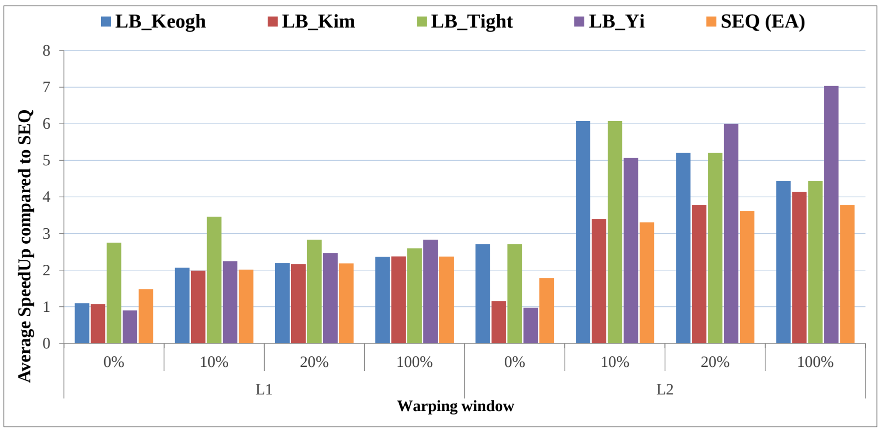
As was later discovered [43], not all lowerbounds are applicable to generalized DTW (GDTW; see Eq. 2.17). This study shows also specific counter-examples for which the methods does not provide the guaranteed lowerbounds. Authors also provide experimental evaluations and comparison of the described lowerbounding methods on real-world-data known as *UCR Time Series* [101] collected by Keogh.

As the results were similar for most datasets, we explicitly show *50words* dataset (Fig. 3.14a) followed by average results over all fixed-length datasets (Fig. 3.14b). The obtained test results confirm that `LB_Tight` outperforms all other methods for ground distance L_1 in terms of the number of DCs. So it is the tightest lowerbound that achieves up to two orders of magnitude speedup over competitors. For ground distance L_2, `LB_Keogh`, which equals to $\texttt{LB\_Tight}(\cdot, \cdot, L_2)$, remains the dominant lowerbounding mechanism.

[3]For more details and the proof of this theorem, we refer readers to the work which introduces this lowerbound [43]



(a) Average query time



(b) Average Speed-up vs. SEQ scanning

Figure 3.15: Average query time and average speed-up compared to sequential scanning for DTW lowerbounds

We show average query response times (Fig. 3.15a) and the average speed-up of the query efficiency in terms of the compared to the sequential scanning (Fig. 3.15b) for the fixed-length databases. For the comparison of variable-length time series (e.g., DNA sequences of genes of *Listeria monocytogenes*[4] using the technique suggested in [100]), we refer to the original study [43].

Efficient Index-Free Similarity Search

From time to time, authors also come with novel indexing methods like the idea of index-free *D-file* [102, 103] suitable for data mining, streaming data, and other domains that produce large amounts of data. Authors remove the expensive operation required for building the index and apply the sequential scanning enhanced with *D-cache*. It tracks the previously evaluated distances in the main memory and thus provide the basis for computing future lowerbounds which results in faster query processing.

[4]For details see *Metric Spaces Library* at `http://sisap.org/`

Approximate Search

So far, we have been dealing mostly with exact searching in (non)metric spaces. There are also other approximate searching techniques that usually trade-off the precision for the speed of query evaluations. These methods include but are not limited to approximate $(1 + \epsilon)$ kNN searching with M-trees using the *relative distance error* $\epsilon \geq 0$ [104], contractive mapping mechanisms when we omit or discard the second step of refining candidates with the original distance measure, clustering methods in which we access a limited number of nearest clusters [105, 106], or the probabilistic method such as the probabilistic LAESA index [107].

3.3 Indexing Challenges

No matter whether we apply exact or approximate similarity search, we always face some challenges when it comes to indexing the data. If we knew some specific properties of the input similarity model we deal with, it would be easier to propose the *lowerbounding method* for indexing and querying customized for the given scenario such as the cases of metric triangle lowerbounding (see Section 3.1.1) or ptolemaic lowerbounds (see Section 3.1.2). Whether we will be able to achieve this depends purely on the complexity of the given similarity model. Without any a priori constraints, selecting the best indexing method for a general similarity model which assumes only a black-box similarity/distance function with a former unknown universe of object descriptors is a challenging task. Here, the only valuable information we could use is the computed distance between any two arbitrary objects.

Having this in mind, we want to derive more information and insight about the data only through the *distance matrix* – a matrix which on ith row and jth column contains the distance $\delta(o_i, o_j)$ between two objects o_i and o_j that belong to the database $\mathcal{D}$:

$$M_{\delta,\mathcal{D}}[i, j] = \delta(o_i, o_j), \quad o_i, o_j \in \mathcal{D}$$

This means that no previous knowledge of the similarity model is required to provide a good indexing technique for any given data.

If we obtain any results from such a matrix (positive or negative) it would give us some information about the whole domain $\mathcal{D}$. However, sometimes even evaluating $\mathcal{D} \times \mathcal{D}$ distance computations is infeasible, especially for large data volumes. So, we apply a specific way of data sampling to build a smaller yet usable sample $S \subseteq \mathcal{D}$ in terms that it holds the same properties as the overall dataset $\mathcal{D}$. This approach of sampling data has been already proven by TriGen algorithm (see Section 3.2.5) and employed also by LTA algorithm (see Section 3.2.7).

Although we can find ways to sample good data in order to build a suitable database sample S, we still do not know what the next steps will and should be. How to process such data to get valuable insights? What to search for in order to obtain efficient yet effective indexing model? How do we know that the acquired results are applicable? To continue further, we need to go beyond these questions and ask ourselves whether there exist an automated process of discovery selected (analytical) properties for given data samples.

3.3.1 Can We Automate the *Axiom* Exploration?

The ideas of ptolemaic indexing (see Section 3.1.2) and ptolemaic access methods (see Section 3.2.4) show that finding novel techniques suitable for database indexing could be a solution to speeding up (exact) similarity search in other way than mapping the problem to the metric space model (see Section 3.2.2) or transforming the nonmetric distances to metric ones (see Section 3.2.5). On the other hand, doing so *manually* would be even harder than forcing a *domain expert* to implant the metric axioms into her/his generally nonmetric similarity model. Note that even proving the "simple" triangle inequality property in the "simple" Euclidean distance is quite a complicated task, let alone proving the complex Ptolemy's inequality for a nonmetric distance implemented by a complex heuristic algorithm.

Hence, the challenging question that remains open here is whether we are able to automate the whole process of *axiom exploration*[5] for specific databases and similarity models in which the newly discovered *axioms* will provide the fundamental parts for novel indexing methods and structures.

[5]In this case, we use the term *axiom* to denote an important property that holds in the given similarity model for the given database. For more details, see the proper and more formal Def. 4.6.

Part II

SIMDEX Framework

Chapter 4

Flexible Indexing Framework

To address the challenge of exploring suitable indexing techniques for specific databases, we introduce and outline SIMDEX Framework concept [108]. We can describe SIMDEX as a *universal framework that is capable of revealing alternative indexing methods that will serve for efficient yet effective similarity searching for any similarity model.* This framework offers a complex solution that provides various domain experts with database techniques that speedup similarity search yet do not require any database-specific intervention to existing similarity models.

As there continuously appear new variants of SIMDEX Framework [108, 109, 110] since its first release, we first generalize the whole concept, describe the existing variants in more details, and consolidate the SIMDEX portfolio.

In our work, we follow the initial ideas of developing the general and flexible algorithmic framework for automatic exploration of *axiom spaces.* We focus on a precise description of all existing variants (Sections 5, 6, and 7) while highlighting their pros or cons, and provide the comparison of individual SIMDEX variants between each other (Section 9). We append the experimental evaluations on real-world datasets (exploration and indexing tasks) and show how SIMDEX results behave with respect to other existing methods.

The main objective of our research is to provide the feasibility study of SIMDEX Framework – validate its functionality, compare the variants between each other, and extensively examine all variants on various databases. Finally, we describe a concrete application of SIMDEX Framework for building Smart Pivot Table (Section 8).

4.1 General Overview

Before we formally define SIMDEX Framework, we describe the fundamental yet very simple idea of how the framework really works. As the main input, we consider a particular similarity space described by

1. a *black-box distance function* δ and

2. a *database sample* $S \subseteq \mathcal{D}$

The "mining field" for the framework is the distance matrix $M_{\delta,S}$ that we obtain by computing the pair-wise distances between all 2-tuples of objects from the database sample S using the black-box distance function δ.

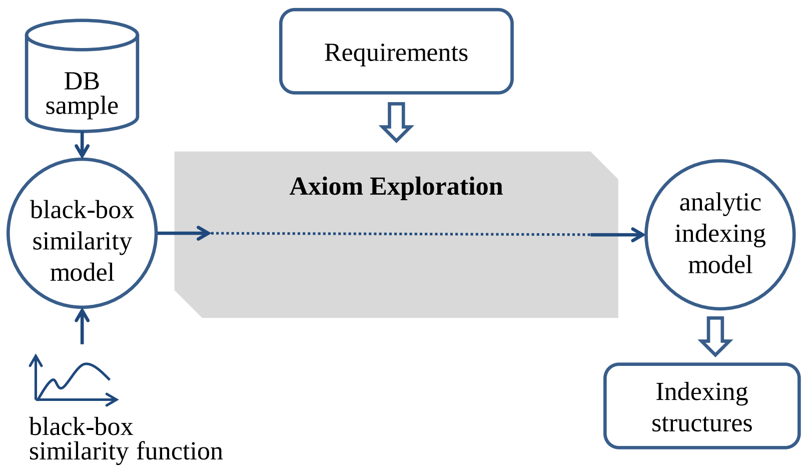


Figure 4.1: General SIMDEX Framework high-level schema

Then, based on the chosen variant, we select a set of mathematical expressions in specific forms to be tested within the distance matrix $M_{\delta,S}$. If all evaluations of a single expression pass the test, we have discovered a new *axiom* valid in the similarity model and applicable as a new indexing method.

Using this simple idea, we are able to algorithmically explore axiom spaces specified in a syntactic way – that is, we are not using a single canonized form and a tuning parameter as *TriGen* [8], Lambda Tuning Algorithm in fuzzy similarity approach [19], or other mapping approaches do (see Section 3.2.2). Consequently, the triangle inequality (Eq. 3.3) and Ptolemy's inequality (Eq. 3.6) could be rediscovered as two instances in the axiom universe.

The resulting set of analytical properties (axioms) will be obtained in specific forms which we name as the *analytic indexing models*. These models can be implanted in various suitable indexing structures, e.g., using the pivot table (AESA or LAESA [6]) in the same way as ptolemaic indexing was implemented [15]. Thus, we immediately use the output indexing models for database indexing.

Note that this process gives only empirical testing of the expressions using multiple interpretations of object-to-object distances $\delta(o_i, o_j)$ from the distance matrix $M_{\delta,S}$ which leads just to the empirical evidence that an expression holds for the given model. Nevertheless, this cannot be replaced by an analytical resolution that absolutely confirms the evidence, although for our case, a large number of positive tests could be treated as a sufficient confirmation.

Figure 4.1 depicts the high-level schema of the general SIMDEX Framework which is common and followed by all its existing variants. It shows individual steps towards the common goal, how they are connected, and how the framework works. The arrows outline the basic data flow. Besides the described items (DB sample S, black-box similarity function δ, analytic indexing models, indexing structures), the "Requirements" item let us set further conditions that the resulting axiom must hold in order to be considered as the resulting indexing technique. These restrictions include but are not limited to the *axiom complexity*, *axiom form*, the *number of variables within the axiom*, *mandatory objects* or *variables within the axiom*, etc.

4.2 Breaking the Metric Paradigm

The idea of SIMDEX Framework is based on the ground-breaking fundamental research at least within the scope of data engineering and database systems. To the best of our knowledge, there has never been proposed such a complex framework for mining properties from similarity data as in this case. Although there have been many proposals introducing alternative perspectives on data indexing and mining, they were always based on a single mathematical model (such as the metric space model, ptolemaic indexing, multidimensional scaling, neural networks, etc.). In SIMDEX Framework, we do not a priori select a particular mathematical foundation, as we will analytically discover the foundation itself.

Hence, the impact of the proposed framework is two-fold. First, from the technical point of view, the discovery of new axioms will enable large-scale similarity search in many practical applications where the content-based retrieval is the essential component, e.g., multimedia retrieval, biometric databases, time series databases, etc. Since applications come from different domains outside the computer science, the contribution of these outcomes is truly multi-disciplinary.

On the other hand, from the philosophical point of view, "newly discovered" mathematical expressions (or axioms) will contribute to the theoretic foundations of data engineering, data mining, and disciplines beyond such as computational geometry, geometric topology, and related disciplines. If a discovered axiom is general enough, it could open new horizons or research interests in many disciplines related to data engineering, similarity search, data mining, etc. and so the framework exhibits substantial inter-disciplinary nature.

Therefore, we set the main goal of our research to find alternative methods for indexing specific or unusual data. We get inspired by the previous research [52, 8, 50, 51] that showed that it is possible to find other models that for some databases might increase the efficiency and provide better results than the metric space model. We also develop the potential of our recent work [108] in which we introduce SIMDEX Framework and suggest to explore the universe of expressions (so called axioms) with the proposed SIMDEX Framework in order to reveal new and unknown indexing possibilities.

We set the boundaries of our research by defining the *ideal outcome* – data-specific (or similarity model-specific) indexing method (or a specific lowerbound inequality) that provides efficient yet effective similarity searching based on the inter-object distances in the data space given by the input similarity model.

4.3 Framework Methodology

In this section we formalize and describe into more details the methodology of the SIMDEX Framework. We outline general requirements together with specific options applicable to the implementation of the framework. We give the step-by-step tutorial of how to create and use SIMDEX in different environments as the methodology is generally applicable regardless of the selected programming language or platform because it forms the theoretical basis and foundation. Furthermore, SIMDEX might be modified, customized, or extended for specific purposes if necessary, as we later depict by future variants (`I-SIMDEX`, `GP-SIMDEX`, or `PGP-SIMDEX`).

Symbol	**Description**
$\mathcal{D}$	database $\mathcal{D}$
$S \subseteq \mathcal{D}$	database sample S
$\delta(o_i, o_j)$	distance between database objects o_i and o_j ($\delta : \mathcal{D} \times \mathcal{D} \to \mathbb{R}$)
$M_{\delta,S}$	distance matrix for a sample S and distance δ where $M_{ij} = \delta(o_i, o_j)$ for each $o_i, o_j \in S$
$\mathcal{M}$	the universe of distance matrices $M_{\delta,S}$
LB_i	lowerbound expression LB_i
R_i, R_i^*, $R^\circledast$	relation R_i, candidate relation R_i^*, compound relation $R^\circledast$
$\mathcal{R}$, $\mathcal{R}_\mathcal{V}$, $\mathcal{R}_\mathcal{T}$	all ($\mathcal{R}$), validated ($\mathcal{R}_\mathcal{V}$), and successfully tested ($\mathcal{R}_\mathcal{T}$) relations
A_i, $\mathcal{A}$	axiom A_i, set of all axioms $\mathcal{A}$ valid for a given model

Table 4.1: SIMDEX Framework basic notation

As we mentioned in the previous section, the input for the framework consists of a database sample $S \subseteq \mathcal{D}$, $|S| = n$ that is a subset of the given database $\mathcal{D}$, and a black-box distance function δ. We want to obtain a set of *axioms* $A = \{A_i\} \subseteq \mathcal{A}$ that is a subset of all axioms $\mathcal{A}$ valid in the given model. The proper definition of the term *axiom* will follow shortly (see Def. 4.6).

There are two extreme cases – in the worst case, the resulting set will be empty (no axiom has been revealed and $|A| = 0$); in the ideal situation, we will discover all axioms ($A = \mathcal{A}$). Therefore, we want to develop or select such exploration method that will maximize the size of the resulting set of axioms $|A|$.

In order to proceed with the methodology and the framework structure, we need to properly define all commonly used terms such as a *lowerbound*, a *relation*, or an *axiom*. We summarize all terms and symbols that we use in the text in Table 4.1.

Definition 4.1 (Similarity model)**.** *The distance matrix $M_{\delta,S}$ or the combination of the distance function δ with the database sample S define the* similarity model.

Definition 4.2 (Lowerbound)**.** *In our context, any valid mathematical expression might be considered as a* lowerbound LB.

Definition 4.3 (Relation, Candidate relation)**.** *A* relation R_i *(or a* candidate relation R_i^**) is a mathematical inequality in a standardized form:*

$$R_i \equiv \left[\delta(q, o) \geq LB_i\right] \tag{4.1}$$

where $\delta(q, o)$ is the real distance between the query object q and a database object o, and LB_i is a lowerbound.

Definition 4.4 (Relation cardinality)**.** *The cardinality* **card** *of the relation R_i equals the number k of distinct variables used within the relation:* $\texttt{card}(R_i) = k$.

Definition 4.5 (Evaluation function). *The* evaluation function $\texttt{eval}_k$ *is defined for any relation R_i with the cardinality $\texttt{card}(R_i) = k$ and for any selection of k objects from the database sample S. It returns* **true** *if the relation holds for the given k-tuple and* **false** *otherwise:*

$$\texttt{eval}_k : R_i \times S^k \longrightarrow \{\texttt{true}, \texttt{false}\} \tag{4.2}$$

where R_i is a relation and $S^k = \overbrace{S \times S \times \ldots \times S}^{k}$ stands for k-tuple of different objects from S.

Definition 4.6 (Axiom). *If we have a relation R_i with cardinality $\texttt{card}(R_i) = k$ and $\texttt{eval}_k(R_i, S^k)$ returns* **true** *for any k-tuple objects from the database sample S, we say that R_i is an* axiom *A_i:*

$$R_i \text{ is axiom } A_i \iff \forall t \in S^k : \texttt{eval}(R_i, t) = \texttt{true} \tag{4.3}$$

More precisely, A_i is an *empirical* axiom for the given similarity model as it holds within the database sample S only and probably in $\mathcal{D}$. In order for A_i to become a "real" axiom, it must be theoretically proved which is out-of-scope for our work. Nevertheless, we will primarily focus on finding axioms.

Definition 4.7 (Axiom space). Axiom space $\mathcal{A}$ *is the set of all axioms A_i valid for the given similarity model: $\mathcal{A} = \{A_i\}$.*

Now and then, it might be useful to obtain relations that are not always true, but are valid for majority of k-tuples, to obtain approximate values. For this purpose, we define the probability values P_i (for relations R_i).

Definition 4.8 (Relation probability value). *The* relation probability value P_i *for the given relation R_i (with the cardinality $\texttt{card}(R_i) = k$) determines the ratio of positive occurrences of R_i to all results for the set of k-tuple objects from the sample S:*

$$P_i = Prob_{t \in S^k}\Big(\texttt{eval}_k(R_i, t) = \texttt{true}\Big) = \frac{|\texttt{eval}_k(R_i, t) \text{ returns } \texttt{true}|}{|\texttt{eval}_k(R_i, t) \text{ returns any value}|} \tag{4.4}$$

The resulting value of P_i depends purely on the number $|S^k|$ of tuples t we use for the evaluation of the relation R_i.

These terms introduce the formal background for various SIMDEX variants that we will individually describe and compare in the following text. Besides the general overview and common features, they all differ at least in the way the *exploration phase* is implemented – the way the we generate and test relation candidates. We focus on the complete description of the exploration phases and always highlight both the advantages and the disadvantages.

Beforehand, we would like to depict the principles that are in common for any exploration process no matter which variant we select.

$\langle$ Terminals $\rangle$	$\rightarrow$	$0, 1, 2, 3, 4, 5, 6, 7, 8, 9, var_{pqo}$	
$\langle var_{pqo} \rangle$	$\rightarrow$	$var_p \mid q \mid o$	
$\langle var_p \rangle$	$\rightarrow$	$p_{\texttt{MaxPivots}}$	
$\langle$ MaxPivots $\rangle$	$\rightarrow$	$[0-4]$	(5 pivots at most)
$\langle$ Distance $\rangle$	$\rightarrow$	$\delta(var_p, var_{pqo})$	
$\langle E \rangle$	$\rightarrow$	$E+T \mid E-T \mid T$	(expressions)
$\langle T \rangle$	$\rightarrow$	$T*F \mid \frac{T}{F} \mid F$	(terms)
$\langle F \rangle$	$\rightarrow$	Number $\mid$ Distance $\mid$ $\texttt{abs}(E)$ $\mid -F$	(factors)
$\langle$ Number $\rangle$	$\rightarrow$	$([0-9])+$	

Table 4.2: SIMDEX Mathematical Expression Grammar

4.4 Exploration Process

The axiom exploration phase is the most critical part of the whole framework. It defines how we search for the most promising candidates (deterministic or random-based methods of generating and validating the set of candidates), how we evaluate each single expression, and how we determine the success of tested candidates. Although each SIMDEX variant defines its own exploration method, some principles remain common, namely the *evaluation* and *fitness* functions. We describe the common fundamentals in the following sections.

4.4.1 Grammar for Mathematical Expressions

The initial step in the exploration is to create or load a *grammar definition* G. Although we can use any grammar, in SIMDEX Framework we apply the grammar theory specifically modified for the purpose of generating *lowerbounds*. This means to define the grammar that generates good *candidate relations* (R_i^*). The language defined by the grammar contains:

1. a reasonable number of *terminals*, such as

   (a) *object descriptor variables* ($q, o, p_1, p_2 \ldots$) and *constants* (c_i) where some of them are fixed and act as global reference points (such as pivots p_j); while others could stand for any object

   (b) *functions* f_j modifying the whole expressions or particular distances. Here we allow specific functions such as the the triangle generating (TG) and triangle violating (TV) modifiers from TriGen algorithm [8], however we have not included those yet.

   (c) *standard arithmetic operators* $(+, *, -, /)$, *numeric constants*, etc.

2. a limited number of their *combinations*

As the grammar is concerned, we refer to *regular languages* or *L3/Type-3* grammar in Chomsky hierarchy [111] which is sufficient for this case. These languages are basically handled by a finite state automaton. For most cases, we will use the grammar as depicted in Table 4.2.

Using the grammar, we are able to guarantee that each *candidate relation* R_i^* is in the standardized form as outlined in Eq. 4.1. What remains as the main concern is to verify that evaluating the lowerbound value in *candidate relation* R_i^* which is in $\mathcal{O}_{LB}$ is computationally more efficient compared to the direct distance computation between two objects $\mathcal{O}_\delta$: $\mathcal{O}_{LB} \ll \mathcal{O}_\delta$. Otherwise, the estimation of the distance $\delta(\cdot,\cdot)$ with the lowerbound LB will not speedup the querying.

Also, other existing *lowerbounds* indicate that it is good to use some reference objects p_i (called pivots), for which the distances $\delta(o, p_j)$ might be pre-computed and $\delta(q, p_j)$ is computed just once before the query evaluation starts. In this case, we expect the existence of combinations $\delta(q, p_j)$ and $\delta(p_j, o)$ within the generated *lowerbound*. However, the expanded *lowerbound* form cannot at any case include or reference $\delta(q, o)$, as we try to estimate this distance. Therefore, we prohibit recursive references within the lowerbound.

4.4.2 Evaluation Functions

Having the set of candidates ready for the inspection, we evaluate each mathematical expression individually using the *evaluation function*. It determines how good the expression is in terms of correct estimation of real distances (formally see Def. 4.5). There are multiple options for choosing the appropriate evaluation function such as

1. *sampling random k-tuples*
   With the fixed number N of tests to be performed, we randomly select N different k-tuples for each candidate relation R_i^*. If we set a maximum $k_{\max}$ such that $\forall k : k \leq k_{\max}$, we can determine the $k_{\max}$-tuples once before the evaluation.

2. *testing the whole data sample*
   With the fixed size $N \times N$ of the input distance matrix $M_{\delta,S}$, we test all distinct k-tuples for each candidate relation R_i^*. Even for small values of k, this approach is resource-demanding as we need to evaluate $\binom{N}{k}$ tuples.

3. *LAESA-like evaluation*
   Here, we simulate the query evaluation as it is present in LAESA index [65, 48] by choosing a small set of queries $Q \subset S$ and pivots (reference objects) $\mathbb{P} \subset S, \mathbb{P} \cap Q = \emptyset$. If it holds $\forall k : k \leq |\mathbb{P}|$, we evaluate each candidate relation R_i^* with respect to every object $o \in S, o \notin Q, o \notin \mathbb{P}$ and a respective query $q \in Q$.

Also, we would suggest to consider one more method which combines the proposed approaches and it uses a *multi-level* evaluation. In this case, each level L defines the maximum number of randomly selected k-tuples to be evaluated $\max_L$. At the next level $L+1$, the maximum number $\max_{L+1}$ is always considerably higher than the previous level ($\max_{L+1} \gg \max_L$), while the final/top level corresponds to all k-tuples. The candidate relation R_i^* is taken to the next level $(L+1)$ only if it passes the evaluation at the current level L. This way, only the best candidate relations will be extensionally tested.

In all cases, the value k represents the cardinality of the candidate relation R_i^*: $\texttt{card}(R_i^*) = k$ (see Def. 4.4).

Note that the result of any evaluation function just shows how the candidate relation R_i^* performs on the tested database sample S. Therefore we can only estimate its performance on the rest of the database $\mathcal{D}$ because it does not guarantee the same results as with the sample data. A good relation minimizes or removes such a difference in the query performance.

4.4.3 Fitness Functions

Occasionally, we will have to figure out the next steps of the exploration towards better results. This usually means to discard weakest and promote/enhance most promising candidate relations. For this purpose, we use the *fitness functions* that allow us to designate the future of a candidate based on its actual test results. To do so, we engage several properties to build the adequate metrics. Among others, we consider properties such as

- *success ratio* - the ratio of successful tests compared to the total number of tests (see Def. 4.8)
- *lowerbound tightness* - how good the estimation is based on the average difference between the lowerbound and the real distance values
- *cardinality* of the candidate relation

There are couple more options we might consider in the future work such as to include expression tree height / width or complexity of the expression among the others. Including these properties might lead to better fitness.

4.5 Framework Summary

In the previous sections, we summarized the theoretical foundation for SIMDEX Framework and describe the shared basis that we will be referencing to. Now, we are ready to continue with the introduction of the first real implementation of SIMDEX concept – the Iterative SIMDEX.

Chapter 5

Iterative SIMDEX

Initially, we implemented the axiom discovery trivially as the incremental exploration of axiom spaces [108]. We will refer to this method as *Iterative SIMDEX* or `I-SIMDEX` and in this chapter, we provide the detailed overview of this technique.

5.1 I-SIMDEX Overview

In order to implement the incremental exploration process, we divide technically the exploration phase into several *stages* that cover specific functionalities

- **(S1) Grammar Definition**
  We follow the general grammar definition G (for details see Section 4.4.1) for generating valid SIMDEX *lowerbounds* that form *candidate relations* in the standardized forms.

- **(S2) Expression Generation**
  Because the grammar-based generation of expressions leads to an infinite universe, we need to optimize the set of tested lowerbound inequalities in order to discover the most promising candidates first.

- **(S3) Expression Testing**
  Once a *candidate relation* is generated, we test it with selected evaluation function within the distance matrix $M_{\delta,S}$. Only such *candidate relations* R_i^* pass, for which their probability value P_i in the given model is higher than the required threshold probability value $T : P_i \geq T$.

- **(S4) Expression Reduction**[1]
  We propose the optional step that investigates and applies the heuristic techniques of expression combinations and pruning.

The original concept defines two more stages. First, it expects the stage *(S5) Indexing Structures* that validates the resulting set within the indexing structure. In this case, we want to implant the outputs directly to the indexing phase to validate the theoretical results. Secondly, we assume *(S6) Parallelization* in order to speedup the overall testing. This is easily achievable with multi-threaded evaluations or multi-CPU testing and does not need a special attention. The complete overview of stages is depicted in Fig. 5.1.

[1]This stage was not implemented in the initial `I-SIMDEX` prototype

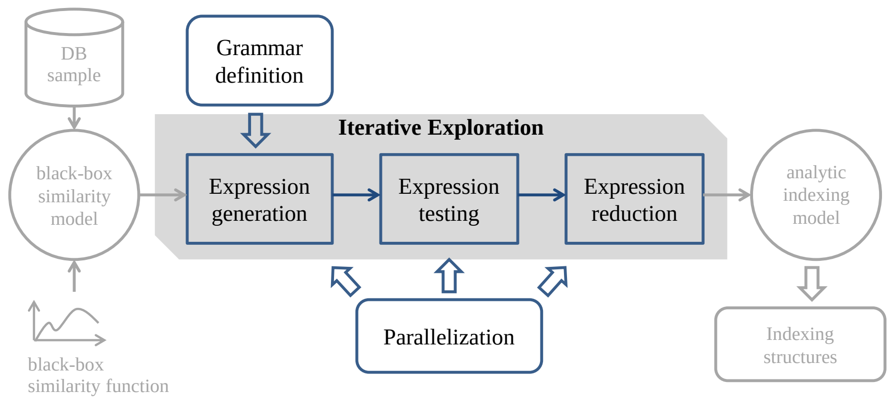


Figure 5.1: Iterative SIMDEX schema

Definition 5.1 (Iterative SIMDEX)**.** *The* Iterative SIMDEX *is defined as a tuple* $\texttt{I-SIMDEX}(G, C, T)$ *where G is a grammar definition, C is a set of conditions for generating lowerbound expressions, and T is a threshold probability value. Then, for a universe $\mathcal{M}$ of distance matrices $M_{\delta,S}$, the* Iterative SIMDEX *acts as a function:*

$$\texttt{I-SIMDEX}(G, C, T) : \mathcal{M} \longrightarrow \{(\mathcal{R}, \mathbb{R}_{\langle 0,1\rangle})\} \tag{5.1}$$

where $\mathcal{R}$ is a set of all relations that could be defined by lowerbounds generated by the grammar G and $\mathbb{R}_{\langle 0,1\rangle}$ covers real numbers within the interval $\langle 0, 1\rangle$.

The result consists of a set of pairs (R_i, P_i) where each relation from the resulting set $R_i \in \mathcal{R_X}$ is valid for the sample S with the corresponding probability P_i $(\forall i : P_i \in \langle 0, 1\rangle, P_i \geq T)$. The set of all resulting relations $\mathcal{R_X}$ is a subset of all valid relations $\mathcal{R}$: $\mathcal{R_X} = \{R_i\} \subseteq \mathcal{R}$.

If the threshold probability value $T = 1$, we can omit all probabilities P_i and simplify the framework definition to

$$\texttt{I-SIMDEX}(G, C) : M \longrightarrow \{\mathcal{A}\} \tag{5.2}$$

where $\mathcal{A}$ is a set of all axioms valid for the given model. Then, the result will be a set of axioms $\mathcal{R_X} = \{A_i\} \subseteq \mathcal{A}$.

In the following sections, we describe the important stages (S1–S4) into details and show how they together form the exploration phase.

5.1.1 Grammar Definition (S1)

Regarding the grammar for generating valid candidates, in `I-SIMDEX` we follow all the previously mentioned principles and apply the general grammar introduced in Section 4.4.1.

5.1.2 Expression Generation (S2)

With the defined grammar G, we can start generating expressions that will form *relation candidates* R_i^*. Even if we limit the language and expansion recursion, grammar-based generation is an exponential problem. Therefore our main objective is to guide the exploration to the most promising *candidate relations* first.

Then, we are able to discard inappropriate *relation* classes at early stage, so that early termination of the exploration will end up with nonempty result set.

For this purpose, we use two options for limiting the set of tested *relations*. First, there is a finite set of conditions $C = \{c_1, c_2, \dots, c_m\}$ for generating the *candidate relations* where each c_i stands for a specific condition (e.g., discarding *lowerbound* expressions $\frac{x}{x}, -x$).

Second, we introduce a technique called *fingerprints* which enables to eliminate multiple forms of the same mathematical expression. For each expression (or *relation*), it returns the string representation (*signature*) which allows us to divide *relations* into equivalent classes. For example, *lowerbounds* $\delta(p_3, p_4)$ and $\delta(p_6, p_5)$, where p_i is a variable, contribute to the fingerprint with the same value $[\delta(var_1, var_2)]$ and are equivalent for testing. So, we test only one of them as the second one gives the same results.

To apply such conditions, we define a corresponding validating function that decides whether the given *relation* is suitable for further testing and refinement (returns `true`) or whether it will be discarded from further testing (returns `false`). This function relies on the set of fingerprints for previously considered *relations*.

Definition 5.2 (Validation function). *Suppose $\mathcal{F} = \{F_i\}$ is the set of existing fingerprints in which F_i is the fingerprint for a previously tested* relation *R_i, C represents the set of conditions, and $\mathcal{R}^*_G$ conforms to the set of all* candidate relations *generated from the grammar G. Then, we set the* validation function *to*

$$\texttt{validate}_{C,\mathcal{F}} : \mathcal{R}^*_G \longrightarrow \{\texttt{true}, \texttt{false}\} \tag{5.3}$$

Note that the set of fingerprints $\mathcal{F}$ is initialized with an empty set and dynamically changes during the evaluation. Thus the result of the validation function for any input *candidate relation* depends on the current state of $\mathcal{F}$.

The best situation is if the validating function $\texttt{validate}_{C,\mathcal{F}}$ reduces the set $\mathcal{R}^*_G$ to a large extent and eliminates all non-*axioms* because we try to maximize the number of *axioms* in the resulting set.

We further mark the output set of successfully validated *candidate relations* as $\mathcal{R}_\mathcal{V} = \{R^*_i | R^*_i \in \mathcal{R}^*{}_G : \texttt{validate}_{C,\mathcal{F}}(R^*_i) = \texttt{true}\}$. We want to maximize the intersection $(\mathcal{R}_\mathcal{V} \cap \mathcal{A})$ while minimizing the set size $|\mathcal{R}_\mathcal{V}|$, so the number of candidates to be tested does not exceed too much all the *relations* / *axioms* valid in the given model.

5.1.3 Expression Testing (S3)

In this stage, we test all validated *candidate relations* $R_i \in \mathcal{R}_\mathcal{V}$ from the previous step within the input distance matrix $M_{\delta,S}$ using one of the evaluation functions depicted in Section 4.4.2. For clarity, we mark the successfully validated *candidate relations* (or simply *relations*) as R_i as opposed to all generated *candidate relations* R^*_i.

Suppose the cardinality of the *relation* R_i is $\texttt{card}(R_i) = k$ and the size of the database sample is $|S| = N$, we need to test the given expression for up to $\binom{N}{k}$ k-tuples. We have to ensure the integrity and the consistency, so the selection of distance values for variables must be consistent and equally named variables get the same values.

Definition 5.3 (Relation probability). *We define the* relation probability *testing function as*

$$\texttt{relation\_probability} : \mathcal{R}_{\mathcal{V}} \times \mathcal{M} \longrightarrow \mathbb{R}_{\langle 0,1 \rangle}$$

where $\mathcal{R}_{\mathcal{V}}$ is a set of validated relations *to be tested, $\mathcal{M}$ is the universe of distance matrices $M_{\delta,S}$. The resulting real number determines the* relation probability value P_i *with which the* relation *holds in the specified model (see Def. 4.8).*

If the probability value P_i is greater than the threshold probability value T ($P_i \geq T$), the relation R_i is *valid* and we add it to the result set of successfully tested relations $\mathcal{R}_{\mathcal{T}}$. If $P_i = 1$, then R_i is an axiom A_i (see Def. 4.6).

In the two stages (S2 and S3) where we generate and test the list of *candidate relations*, we employ combinatorial grammar coverage [112] and modern techniques from the constraint logic programming [113]. These approaches allow us to reduce the exponential size of the hierarchy using the standard methods such as expansion, inference, or pruning.

5.1.4 Expression Reduction (S4)

The last (but not yet implemented) step is to reduce the resulting *relations* from the previous stages in order to refine the final result. The goal is to retain only the best *relations / axioms.* However, at this moment we do not provide a clear description of how we would like to achieve this because we skip this part and push it towards future enhancements.

We propose the blueprint of how to combine multiple valid *relations* $R_i \in \mathcal{R}_{\mathcal{T}}$ into a *compound relation* $R^{\circledast}$. This should bring the tightest possible *lowerbound* values for reasonable costs even though we get a more complex *relation.*

Definition 5.4 (Compound relation). *The* compound relation $R^{\circledast}$ *is defined as*

$$R^{\circledast} \equiv \left[\delta(q, o) \geq \texttt{MaxLB}\{LB_i\} \right]$$

where $\{LB_i\}$ is a non-empty set of lowerbounds *while each LB_i corresponds to a valid* relation R_i*, and the function* `MaxLB` *selectively returns the* lowerbound LB_j *which for the given set of variables gives the maximal numeric value.*

Moreover, `MaxLB` function might select a different *lowerbound* LB_j for different selections of variable values which improves its filtering power yet at the same time increases the complexity.

5.2 I-SIMDEX Algorithm

Having all the stages ready, we depict the main algorithm of `I-SIMDEX` prototype in Algorithm 4 which navigates us through the whole computation. We expect the grammar G for generating expressions as one of the parameters (see Def. 5.1).

Afterwards, we generate *candidate relations* R_i^* and validate each R_i^* using the validation function $\texttt{validate}_{C,\mathcal{F}}$ (Def. 5.2) to identify duplicates during the execution. We iteratively build the set of fingerprints $\mathcal{F}$ for validated *relations* for further usage. Then, we compute the *relation probability value* P_i (Def. 4.8) that decides whether R_i^* passes the test ($P_i \geq T$) and will be in the resulting set or not. At last, the set of remaining *candidate relations* conforms the output.

Algorithm 4 Iterative SIMDEX (G, C, T, S, δ)

```
Require: Grammar definition $G$, validation conditions $C$, threshold probability
    value $T$, database sample $S$, distance function $\delta$
 1: $\mathcal{F} \leftarrow$ new empty set   {initialize the set of valid fingerprints}
 2: $M_{\delta,S} \leftarrow$ new distance matrix $(\delta, S)$
 3: $lowerbounds \leftarrow$ ExpressionGeneration(G, C)
 4: $relations \leftarrow$ BuildCandidateRelations($lowerbounds$)
 5: for all $R_i^*$ in $relations$ do {test all candidate relations}
 6:    if validate$_{C,\mathcal{F}}(R_i^*)$ equals false then
 7:       $relations$.Remove($R_i^*$)   {validity check fails}
 8:       continue   {skip further testing of the relation $R_i$}
 9:    end if
10:    $fp \leftarrow$ CreateFingerPrint($R_i^*$)
11:    $\mathcal{F}$.Add($fp$)   {store the fingerprint of $R_i^*$ for future validations}
12:    if relation_probability$(R_i^*, M_{\delta,S}) < T$ then
13:       $relations$.Remove($R_i^*$)   {probability test fails}
14:    end if
15: end for
16: return $relations$   {remaining relations compose the result set}
```

5.3 I-SIMDEX Evaluation

To validate the potential of `I-SIMDEX`, we developed a prototype that covers stages S1–S3 and S6 and employs iterative exploration of the axiom space [108]. First, we describe the output from the exploration (Section 5.3.1) and then verify the results with indexing tasks (Section 5.3.2). We evaluate following datasets:

- **Corel Image Features** [114] with non-metric L_p distance ($p = 0.5$)
- **CoPhIR** [96] – for simplicity, we also use L_p distance ($p = 0.5$)
- **Movie ratings**[2] – we take the movie ratings as sets and employ Jaccard coefficients (see Section 2.2.1) to model similarities between them
- **Listeria** [115] using *Levenshtein (edit) distance* (see Section 2.2.1)
- **Spectometry** [37] with parameterized Hausdorff distance (see Section 2.2.1)

5.3.1 I-SIMDEX Exploration Evaluation

The previously depicted Algorithm 4 gives an overview of the whole expression generating / validating / testing process. All steps are straightforward and correspond to stages S1 through S3 as we present in previous sections.

We take five different datasets, each comprised of a distance matrix computed for 20 random database objects. We normalize each input distance matrix, so that the values fit to interval $\langle 0, 1\rangle$ and the individual results can be easily compared between each other. For the grammar G, we use *standard arithmetic operators* $+, *, -, /$ together with three pivots.

[2]http://www.grouplens.org/node/73

Dataset	Expression	MIN	MAX	AVG
Corel	triangle inequality	0.0034	0.9983	0.3764
	$\lvert\delta(q,p)\cdot\delta(o,p)\cdot(\delta(o,p)-\delta(q,p))\rvert$	0.1059	0.9991	0.5020
	$(\delta(q,p)-\delta(o,p))^2$	0.1352	0.9999	0.5054
	$\lvert(\delta(q,p_1)-\delta(o,p_1))\cdot$ $\cdot(\delta(q,p_1)-\delta(o,p_2))\rvert$	0.0420	0.9999	0.5161
CoPhIR	triangle inequality	0.0021	0.9736	0.2696
	$(\delta(q,p)-\delta(o,p))^2$	0.0718	0.9979	0.3808
	$\lvert(\delta(q,p_1)-\delta(o,p_1))\cdot$ $\cdot(\delta(q,p_2)-\delta(o,p_2))\rvert$	0.0845	0.9969	0.3935
Ratings	triangle inequality	0.6067	1.0	0.9037
	$\frac{1}{2\cdot\delta(o,p)}$	0.0119	0.5	0.4254
	$\frac{(\delta(q,p_1)+\delta(o,p_1))}{\delta(p_1,p_2)}\cdot\delta(q,p_1)$	0.0103	0.5845	0.4254
Listeria	triangle inequality	0	0.9559	0.1388
	$\delta(p_1,p_2)\cdot\frac{1}{\delta(p_1,p_2)+\delta(o,p_2)}$	0.0075	0.9994	0.2393
	$\delta(q,p_1)^2\cdot\frac{1}{\delta(o,p_2)\cdot\delta(q,p_2)}$	0.0008	0.9985	0.2401
	$(\delta(q,p_1)+\delta(o,p_1))\frac{\delta(q,p_1)}{\delta(p_1,p_2)}$	0.0032	0.9970	0.2555
Spectometry	triangle inequality	0.1823	0.93	0.7329
	$\delta(o,p)-\delta(o,p)^2$	0.0009	0.8758	0.6638
	$\lvert(\delta(q,p_1)\cdot\delta(o,p_2))-\delta(q,p_2)^2\rvert$	0.0148	0.9399	0.7054

Table 5.1: `I-SIMDEX` exploration initial evaluation results

We test 25,000 relations and for each relation we examine all possible variable assignments from the datasets‘ objects. This process is very time-consuming, so we have to pick a very small size of the distance matrices. During the tests, we look for lowerbound tightness – `min` / `max` / `avg` difference between the real distance value compared to the lowerbound value.

We exhibit the results in Table 5.1 and for better transparency, we present only the expanded LB non-terminals of standardized forms $\delta(q,o) \geq LB$ (see Def. 4.3) and omit the repeating left-hand sides, $\delta(q,o)$, of the resulting relations.

The output from these experiments shows that after we explore several expressions for different datasets, we are able to find axioms that might be directly used for indexing purposes. However, we still need to evaluate the quality of the obtained expressions compared to existing models based on triangle / ptolemaic inequalities (see Section 3.1). Then, we can detect which axioms are applicable and could become efficient variants for real-world situations.

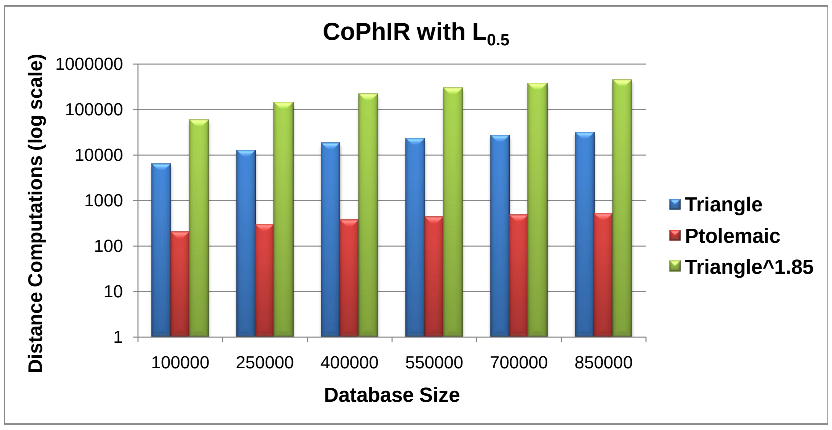


Figure 5.2: CoPhIR - Distance computations (log scale)

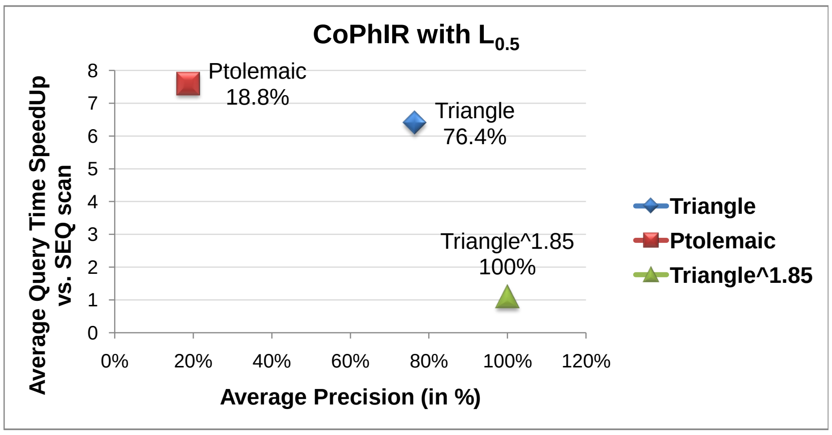


Figure 5.3: CoPhIR - Avg speedup vs. avg precision

5.3.2 I-SIMDEX Indexing Tasks

After the naive implementation of the framework's exploration process, we combine the results with the real-world datasets focusing on nonmetric similarity models in which metric postulates used for indexing and querying produced notable errors. This step moves our theoretical concept little bit further and as a proof we present convincing results.

Observe that the best resulting axioms (Fig. 5.3) in this case are *slightly different* than the theoretical outputs from the exploration. The reason for this is the fact that evaluating a real indexing task is quite different than examining all potential combinations of dataset objects.

Index Test Settings

For this task, we pick two of the previously tested datasets, namely CoPhIR [96] dataset with nonmetric $L_{0.5}$ distance and color histograms from *Corel Image Features* [114] dataset but with nonmetric Jeffrey Divergence distance measure [10].

Using a sample database, we verify the outcomes (resulting axioms) on indexing processes with Pivot Table [3] while studying (1) the precision compared to results of the sequential scan (SEQ), (2) the number of distance computations $\delta(\cdot, \cdot)$ as the basic efficiency measure, and (3) the average speedup.

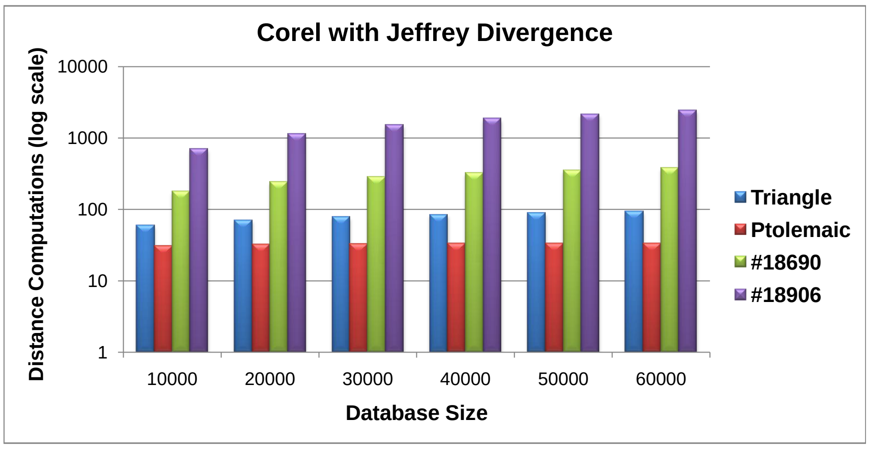


Figure 5.4: Corel - Distance computations (log scale)

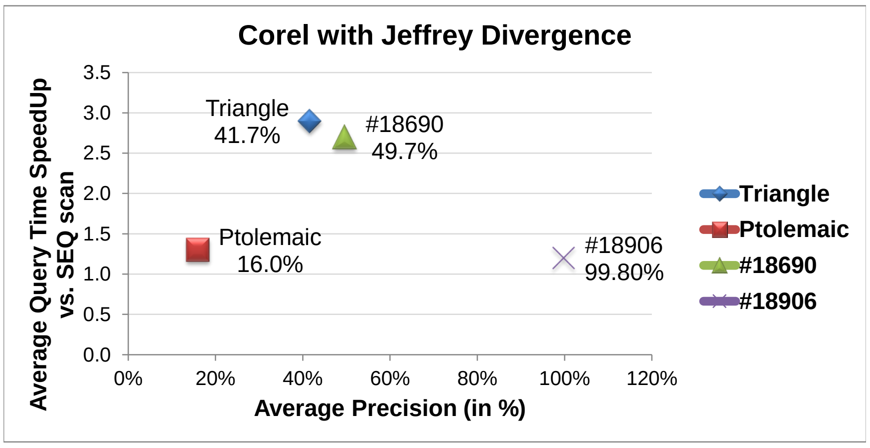


Figure 5.5: Corel - Avg speedup vs. avg precision

Note that we do not study whether the similarity model is valid and whether the resulting items make any sense. We suppose that the SEQ scan always returns correct results and take them as the ground truth.

CoPhIR results

The best result for CoPhIR was the relation

$$\delta(q,o) \geq Triangle^{1.85}(\delta,q,p,o) = |\delta(q,p) - \delta(p,o)|^{1.85}$$

which does not dominate in number of DCs (Fig. 5.2) but it clearly produces no errors (Fig. 5.3) together with 1.1× speedup vs. SEQ scan. This brings a good indexing alternative for queries where precision is crucial.

Corel Image Features results

For Corel, we found the following relations

$$\begin{array}{ll} \#18690 & \delta(q,o) \geq Triangle^{2}(\delta,q,p,o) = |\delta(q,p) - \delta(o,p)|^{2} \\ \#18906 & \delta(q,o) \geq (\delta(q,p_1) - \delta(o,p_1)) \cdot (\delta(q,p_2) - \delta(o,p_2)) \end{array}$$

While the squared triangle inequality (#18690) is only slightly more precise than the triangle lowerbound $LB_{\triangle}$ (Fig. 5.5), we achieve an enormous success with the next expression (#18906) – 99.8% precision together with 1.2× speedup compared to the sequential scan. Although $LB_{\triangle}$ still dominates in the number of DCs (Fig. 5.4), it produces notable error rates (up to 59%).

5.4 I-SIMDEX Challenges

The preliminary experimental evaluations verify the viability of the straightforward exploration approach, however they also reveal challenges we need to address in order to improve the performance of SIMDEX Framework:

1. The basic concept of generating expressions iteratively covers all *candidate relations* (which is the advantage), however, the discovery of a more complex axiom valid in the given space could take enormous time to be revealed.
2. Despite using various enhancements such as *fingerprints* or the validation function $\texttt{validate}_{C,\mathcal{F}}$, we struggle with testing only unique *lowerbounds* and skipping the various forms of the similar ones, as there exist infinite forms of a single math expression
3. In this context, there is always a trade-off between testing larger samples (for more precise results) or testing more candidates. Testing the whole sample S does not have to be always appropriate and we could select only some interesting objects from the sample.
4. To validate that resulting axioms could be used for indexing purposes, we need to run a separate indexing process on the data outside the sample which is correct but time-consuming. This feedback is however important.

In conclusion, the initial implementation of `I-SIMDEX` provides the basis for all future SIMDEX variants, as the proposed algorithmic framework is (after some tweaking) capable of exploring the analytical properties, i.e. axioms.

Chapter 6

GP-SIMDEX

Inspired by previous relatively successful results and driven by the investigation of other possibilities to further enhance the performance, we come up with the next variant of the SIMDEX Framework named `GP-SIMDEX` [116, 109]. It connects the existing theoretical concept together with *genetic programming* algorithms to enrich SIMDEX with the real and applicable context and to gain a powerful tool for axiom exploration. The great potential comes from creating multiple populations of candidate relations based on the feedback from the previous evaluation, so we modify the candidate relations to improve their efficiency accordingly.

In our work, we decide to apply the principles of genetic algorithms [117, 118] to the exploration phase and we introduce the modified version of the *symbolic regression* algorithm [117, 119] customized for discovering inequality relations, i.e. *inequality symbolic regression* [109]. According to the experimental evaluation, this approach helps `GP-SIMDEX` to find appropriate results that in most cases outperform the best-known indexing methods.

Here, the motivation comes from the success of *genetic programming* (GP) in various domains [117, 118], as GP proves to be useful for finding relatively good results from a very large domain within the limited time frame which is exactly our case. Also, the recently introduced Eureqa tool [120] demonstrates the viability of such a solution in a very similar domain.

6.1 Genetic Programming

One of the first approaches that apply the (purely biological) evolutionary process to the artificial systems [121] reveals the potential of using *genetic algorithms* (GA) also to other domains. In general, GA transforms a set of objects into a new population using genetic operations.

Genetic programming (GP) is a specific extension of the traditional evolutionary genetic algorithm which was developed for the computer programs [117]. GP is a unified and domain-independent technique which searches a relatively large space of possible computer programs for such a program that provides the *best fit* to the problem, as it either solves, or approximately solves the problem.

The concept of GP is not new and has been studied for several years since one of the first inspiring books was published [117]. In general, GP applies evolutionary patterns to a particular problem to achieve a specific goal using operations such as *selection*, *crossover*, or *mutation* [122].

Another advantage is that GP is studied and applied widely to lots of different areas and there exists multiple variations how to perform each operation in order to obtain the next population [123]. Therefore we can pick the method that will be mostly related and suitable to mathematical expressions.

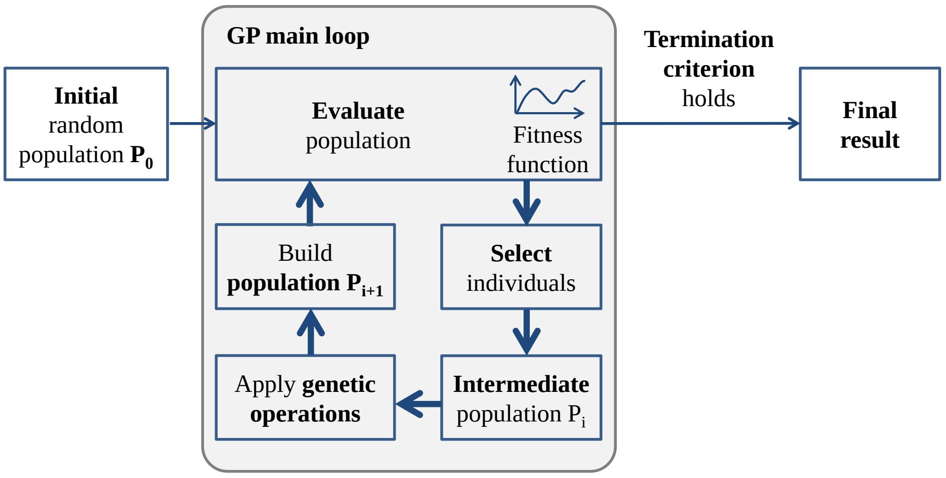
6.1.1 Instance of GP Problem

Before we build the GP instance for a specific problem, we need to follow the recommended preparatory steps [118]. This means to specify

1. the *set of terminals* such as independent variables of the problem, zero-argument functions, and random constants
2. the *set of primitive functions* for our particular problem. Typically, we use the standard arithmetic operations such as *addition* (+), *subtraction* (−), *multiplication* (∗), and *division* (/) together with problem-specific custom functions such as `cos`, `exp`, `min`, `max`... A good choice of function set leads to faster detection of good results.
3. the *fitness function* for explicitly / implicitly computing the fitness values for individuals within the population, i.e., how good or bad an individual performs. While the previous two points define the search space, the fitness function imply the main goal we want to achieve. Here, we will be using higher fitness values for individuals that provide better solutions.
4. certain *parameters* that control the program execution. These include the *population size*, the *probabilities* of how often the genetic operations occur, the *maximum size* of the result, and other program-specific details.
5. the *termination criterion* which specifies when the algorithm finishes. This is either a maximum number of populations or when a particular condition holds (e.g., the fitness value is greater than a given threshold).
6. and the method for *identifying results* of the run. This returns the single best-so-far individual as the result.

Then, within a single GP execution (the *run*) we create multiple *populations*[1] (sets of possible solutions) while each *population* contains a predefined number of *individuals* (individual solutions to the problem). Starting with randomly generated *initial population*, we build the subsequent populations by applying the genetic operations to the individuals in the current population. We randomly *select* one or more individuals (depending on the operation) and perform the genetic operation, while the output is automatically added to the next population. Here, the *selection process* usually leverages the given *fitness function* which determines the viability of each individual, so only best individuals are selected as arguments to the genetic operations or propagated to further populations. We continue with building and evaluating further populations until we satisfy the *termination criterion*. Afterwards, we stop the execution and return the *best-so-far individual* as the final result.

[1]In this context, *populations* are also known as *generations*.



GP main loop
Initial random population $\mathbf{P_0}$
Evaluate population
Fitness function
Termination criterion holds
Final result
Build population $\mathbf{P_{i+1}}$
Select individuals
Apply genetic operations
Intermediate population P_i

Figure 6.1: Genetic programming – main loop for general execution

For clarity, we depict in Fig. 6.1 the general overview of the execution for any GP problem with focus on building subsequent populations within the GP main loop. Although we outline the general evaluation rules for GP problems, we study the GP execution main loop more closely to find out the principles of inner steps – how to *select* individuals to create the *intermediate* populations, which *genetic operations* to use, and how to build the subsequent population.

6.1.2 Selection Process

Starting with the *selection* process, there exist several *selection methods* to be employed within the GP execution in order to select individuals for genetic operations. We illustrate three most famous categories of selection methods, just to have sense how they determine which individuals will be reproduced to further populations or will participate in the genetic operations.

- *Fitness-proportionate selection* [121] is the best-known and most popular category of selection methods which select the individuals based on their computed or normalized fitness values. Suppose the fitness value of i-th individual is f_i within the p-th population, then the probability that this individual will be propagated to the next population $p + 1$ is

$$\frac{f_i}{\sum_{j=1}^{N} f_j}$$

  where N is the size of the population. It is like repeatedly spinning the roulette wheel that contains (multiple) instances of individuals proportionally to their fitness values – also known as *stochastic sampling with replacement* [123]. Similarly, *stochastic universal sampling* [123] applies a single spin of the roulette wheel with N equally spaced pointers. The underlying values are distributed randomly but proportionally as the pie graph.

- *Tournament selection* [124, 117, 118] is the most straightforward approach. We randomly select a set of individuals (typically two) and propagate the one with the higher fitness value.

- Compared to previous categories, techniques that employ *Rank selection* [125, 124] do not consider the numerical values but take into account their ranking positions amongst all individuals. This way, we partially reduce the dominating individuals with high fitness values.

The selection process is random and if we enable *re-selection* of individuals, the best individuals could be duplicated several times in the next population.

6.1.3 Genetic Operations

With the randomly selecting the individuals, we create the *intermediate* population. It includes individuals selected for genetic operations and the size of this population might be larger than regular populations. We process this population and apply the *genetic operations*, each with the assigned probability value of whether or how often the operation occurs. In general, genetic operations in a GP program correspond to genetic operations on chromosomes in terms of Darwin evolution theory even though they are customized for GP programs. They simply allow us to generate new individuals based on the existing ones in order to search further the large space of potential results. The list of standard genetic operations [118, 126] involves:

- *Reproduction* (with probability $\mathcal{P}_R$) picks the individual that will be copied without any further modifications to the next population.

- *Mutation* (with probability $\mathcal{P}_M$) chooses a random position within the individual's representation and replaces the item at that position with a new random value. If we represent individuals as fixed-length strings, this modifies the randomly selected position of the string to a different value.

- *Crossover* (with probability $\mathcal{P}_C$) also known as *Recombination* randomly selects two individuals from the intermediate population and picks a random *split point* in the representation of each individual. Then, we recombine the individual parts and create two new offsprings. Having individuals A and B with string representations $A = a_1a_2 \dots a_n$ and $B = b_1b_2 \dots b_m$, with split points i and j respectively, we work with four elements $A_1 = a_1 \dots a_i$, $A_2 = a_{i+1} \dots a_n$, $B_1 = b_1 \dots b_j$, and $B_2 = b_{j+1} \dots b_m$. We generate offsprings as $O_1 = A_1B_2$ and $O_2 = B_1A_2$ (see the visualization in Fig. 6.2).

For completeness, there also exist other domain-specific genetic operations such as customized *Alter-architecture* operations [127] for classifying protein segments [128].

We build the next population by applying the operations on selected individuals from the intermediate population while maintaining the population size, so the next one does have the same size N. Then, we return back to the beginning of the main loop and iterate further. After the termination criterion is satisfied, we return the best-so-far individual and finish.

Note that the described algorithms use random data for many operations such as building the initial population, or selecting the individuals for genetic operations. Therefore, the algorithm is from its nature *non-deterministic* and we do not have to get the same results after every execution. So, we generally repeat the GP algorithm execution several times to obtain acceptable results.

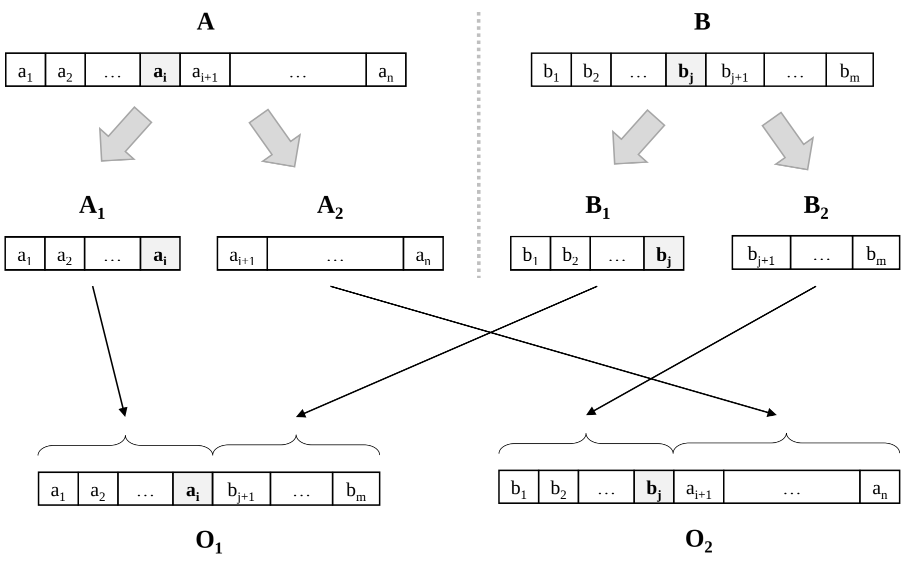

A
B
a_1 a_2 … a_i a_{i+1} … a_n
b_1 b_2 … b_j b_{j+1} … b_m
A_1
A_2
B_1
B_2
O_1
O_2

Figure 6.2: *Crossover* operation for individuals A and B with string representations $A = a_1 a_2 \dots a_n$ and $B = b_1 b_2 \dots b_m$, random split points i and j respectively marked with gray background color, producing new offspring O_1 and O_2.

6.2 Symbolic Regression

In our case, we are dealing with numeric data together with mathematical expressions which leads us to the existing GP problem with known solutions – the algorithms of *symbolic regression* [117, 119]. *Symbolic regression* (SR) or *function identification*, is an example of error-driven evolutionary algorithms which were introduced to solve computationally intensive problems for which simpler algorithms or random optimizations do not work. Because we work with an instance of the GP problem, all GP principles apply here, so we highlight only differences.

We acknowledge that GP is not the only way we solve SR problems, although there exist various successful applications in several domains such as finance [129], developing stock selection models [130], robotics [131], industrial processing [132, 133], or the area of designing integrated circuits [134].

However, there also exist deterministic algorithms such as Fast Function Extraction (FFX) [135] or Prioritized Grammar Enumeration (PGE) [136]. The first one applies machine learning techniques (path-wise regularized learning), while the second one deals with non-linear regression and uses abstract parameters during the computations. As stated in the author's work, FFX suffers from the lack of coefficients or parameters in the basis functions which restricts its abilities to some extent. Also, individuals in summations are with limited complexities [136]. Yet, in our case, we will deal prevalently with GP-based symbolic regression.

6.2.1 Symbolic Regression Overview

In symbolic regression, we deal with independent and dependent variables with a fixed set of constant numerical values. The main goal is to find the *perfect* or *best fit* – a perfect model which provides a linear combination of the independent variables and numeric constants with corresponding coefficients that minimizes the measured error rate.

If we take the candidate relation R_i^* as the model, we can see the parallelism with SIMDEX Framework. To follow the mentioned preparatory steps, we define

1. the *set of terminals* T as the set that includes the applicable independent variables from appropriate domain $x_i \in X$ and constant numeric values in a reasonable small interval range $\mathcal{I}$ of real values such as $\langle -10.0, +10.0 \rangle$

2. the *set of primitive functions* $F = \{+, -, *, /, \mathtt{min}, \mathtt{max}, \mathtt{sin}, \mathtt{cos}, \mathtt{pow}^2\}$

3. the *fitness measure* as the inverted *mean absolute error*. Because we typically consider multiple evaluations of a single individual, each producing a numeric value, we would like to compare the result value (v) with the expected value (v^*). We assume that individuals for which the result is closer to the expected value would behave better and provide better fit. So, we compute the mean absolute error (MAE) as the sum of differences between the result value and expected value: $\text{MAE}_i = \sum_j |v_j - v_j^*|$. Then the fitness value for an individual i is

$$f_i = \frac{1}{\text{MAE}_i} = \frac{1}{\sum_j |v_j - v_j^*|}$$

   To avoid potential divisions by zero, we explicitly define $f_i = \infty$ if $\text{MAE}_i = 0$ which is exactly the perfect fit, so in this case, we can immediately terminate the execution. Otherwise, the closer the value of MAE_i is to 0, the better the individual performs.

4. certain *parameters* that control the run such as probabilities of genetic operations.

5. the *termination criterion* as the maximum number of iterations or when the MAE_i value of individual i is smaller than the given threshold value t: $\sum_j |v_j - v_j^*| < t$.

6. the method for *identifying results* which outputs the best-so-far individual.

The genetic operations consider expression trees instead of strings but the principles remain unchanged. *Mutation* selects and modify a random node in the expression tree; *Crossover* selects random nodes in two expression trees and swap the subtrees defined by these nodes between each other; and *Reproduction* is without modifications. The new *Alter-Architecture* operation picks and alters a node, e.g., changing the constant value of the node or switching the current function to another primitive function from the set F.

During the execution, we maintain a set of M tuples of numeric values, for which we would like to find the equation that perfectly describes these tuples. Having the set of primitive functions F, we build the initial population with random expressions. Then, we perform the main GP loop (Fig 6.1) until we find an acceptable solution.

[2]We denote `pow` as the *power function*: $\mathtt{pow}(x, n) = x^n$

6.2.2 Existing Solutions for Symbolic Regression

There already exists an implemented solution based on the concepts of symbolic regression which is the software tool Eureqa [120] that revealed and "reinvented" some interesting analytical laws mainly for physics. In another paper [137], the same authors provide a method for solving implicit equations with SR, however the main reason why we cannot apply this approach immediately to our case, is that their main focus is on 2-dimensional vector space. For general feature representations of complex database objects, this is far too limiting.

So, the idea that comes to the mind is to reuse Eureqa tool also for the purpose of SIMDEX framework and incorporates it within the exploration phase. However, there are two issues that prevent us from doing so straightforwardly. The first one is that based on the input values, Eureqa tries to find the equations which in our case means relations $\delta(q, o) = LB_i$. This provides a very strong requirement because we speak about a perfect distance estimation that is in most cases unreal to find. So, to be able to use a tool similar to Eureqa, we will need to modify the process to consider also inequalities.

Secondly, there is a problem with the error rate measurement. Generally, the symbolic regression takes the mean absolute error (MAE) which is totally unacceptable for us. It computes the absolute error and do not distinguish whether the expected value is smaller or bigger than the computed value. But during query evaluations and object filtering, we always require that the lowerbound is smaller or equal to the real distance between the query q and the database object o. Any occurrence $\delta(q, o) < LB_i$ leads to incorrect filtering, as we generate a false negative, and thus we cannot consider LB_i as the proper lowerbound. To overcome this, we need to adjust the error measure.

6.3 Inequality Symbolic Regression (ISR)

As the answer to the two major limitations of the standard symbolic regression approach, namely (1) validating only equations and (2) providing absolute error measurements, we introduce a customized version of symbolic regression designed for inequalities - *inequality symbolic regression* [109]. It solves these domain-specific limitations and provides a basis for the exploration phase implantable to `GP-SIMDEX` which stands for GP-driven SIMDEX Framework.

ISR replaces the mean absolute error with a specific error measurement that incorporates checking for the *lowerbound correctness.* To quickly determine and dismiss inappropriate candidate relations, we require all evaluations to comply with $\delta(q, o) \geq LB_i$. Furthermore, we also do not consider and skip candidate relations for which $LB \leq 0$, as their power to filter out any objects during the query evaluation is minimal or none.

The proposed error measure takes the average difference between the expected value which is the real distance $\delta(q, o)$ and the computed value of LB_i for all evaluations of the relation $R_i \equiv \left[\delta(q, o) \geq LB_i\right]$. This means to evaluate multiple k-tuples t while each one comprises of k objects from the sample S ($t \in S^k$) and $k = \texttt{card}(R_i)$ is the cardinality of the relation R_i which corresponds to the number of different object variables persistent in LB_i (see Def. 4.4).

We do this for all tuples t in the set of tuples T that the relation R_i is tested with and get the average error:

$$\texttt{ISR\_AvgError}(R_i) = \begin{cases} \frac{1}{|T|}\sum_{t \in T}\Big(\delta(q,o) - LB_i(t)\Big) & \text{if} \quad \forall t \in T : \delta(q,o) \geq LB_i(t) \\ \\ \infty & otherwise \end{cases} \tag{6.1}$$

where $LB_i(t)$ corresponds to the lowerbound value computed based on the given tuple t. To follow the lowerbound correctness criterion, any negative result $\delta(q,o) < LB_i$ immediately informs us that the relation R_i must be dismissed from the result set, as it provides an incorrect distance estimation.

It is quite clear that in all cases we would like to minimize the error that the given relation R_i produces in order to obtain a good lowerbound with a perfect distance approximation.

In summary, the lowerbound correctness criterion together with the error measurement `ISR_AvgError` allow us to search for relations in the form of inequalities (see Def. 4.3) and define `GP-SIMDEX`.

6.4 GP-SIMDEX Architecture

The innovative nature of `I-SIMDEX` together with promising results from Eureqa tool result in `GP-SIMDEX` [109] which combines and enhances these two methods. We completely rely on *inequality symbolic regression* (Section 6.3) during the exploration phase while `GP-SIMDEX` still conforms to the rules of genetic programming. Using GP-based method within the axiom exploration requires several customizations of individual framework stages. Figure 6.3 outlines the general overview of `GP-SIMDEX` architecture.

Regarding inputs and outputs (for more details see Sections 4.1 and 4.3), `GP-SIMDEX` follows the requirements of SIMDEX Framework and differentiates only in the way the new axioms are discovered. For the exploration purposes we use the corresponding distance matrix obtained by computing the pair-wise distances between objects in the database sample.

The first modification is that the Grammar definition (see Section 5.1.1) is replaced by the *Terminal set* and the *Function set*, so it better corresponds to the genetic programming principles (see Sections 6.1 and 6.3). As the *Terminal set*, we choose real-valued and integer numeric constants from the interval $\langle -5, 5\rangle$ together with distance expressions $\delta(\cdot,\cdot)$ between random pairs of six independent database objects (query q, object o, and pivots $p_1 - p_4$).

The *Function set* consists of unary functions (`abs`, `sqr`[3]) and binary operations (`min`, `max`, `+`, `-`, $*$, `/`). We customize and modify both sets as they are included in general ISR settings.

[3]$sqr(x) = x^2$

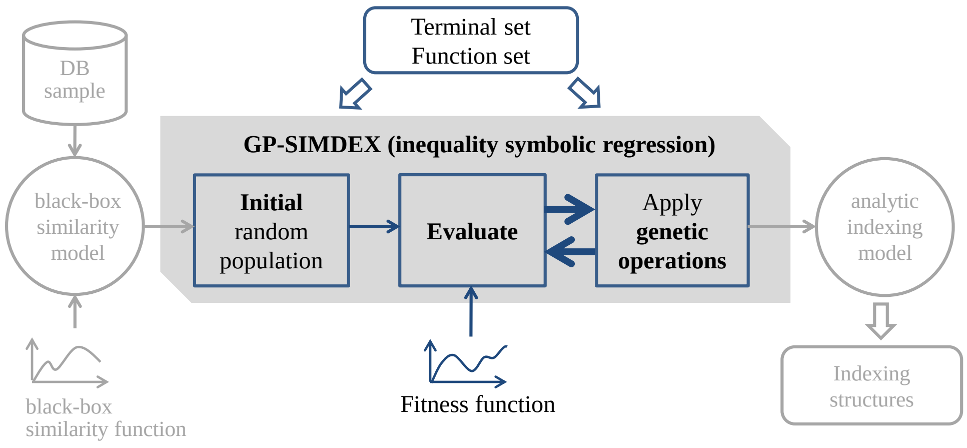


Figure 6.3: General overview of `GP-SIMDEX` implementation

6.4.1 GP-SIMDEX Exploration Process

We start with the initial population of randomly generated lowerbounds LB_i with limited complexity that produce corresponding candidate relations R_i^*. Then, we iterate through the GP main loop until the termination criterion holds while population evolves with each iteration of the program.

Whenever we compose a new population, we evaluate it with the selected *evaluation function* (see Section 4.4.2) within the input distance matrix $M_{\delta,S}$. While `I-SIMDEX` considers only a single evaluation method, in `GP-SIMDEX` we test two different options:

1. to evaluate all different k-tuples for each candidate relation R_i^* where k denotes its cardinality: $k = \texttt{card}(R_i^*)$
2. to evaluate a fixed number $|T|$ of randomly selected k-tuples

Although the first method validates the whole sample S and is dominant, we prefer the second approach as it suits better in this case, The whole `GP-SIMDEX` relies on the randomness and we feel that taking a fixed (but not complete) number of tuples from the sample S equals to taking a smaller-sized sample S.

At all times, we measure the success of evaluating a single candidate relation R_i^* with the relation probability value P_i (see Def. 4.8). For the optimization purposes, we also track the success iteratively and provide *early abandoning* when there is no potential that the candidate relation R_i^* will provide acceptable results (e.g., P_i would not reach the required probability threshold T) or when *lowerbound correctness* criterion does not hold.

After evaluating the population, we compute fitness values with the specific *fitness function*. We compute the generic fitness value for a relation R_i as:

$$f_i = \begin{cases} \dfrac{P_i}{\texttt{ISR\_AvgError}} & & \textit{otherwise} \\ 0 & \text{if} & \texttt{ISR\_AvgError} = \infty \end{cases} \tag{6.2}$$

where P_i denotes the percentage of successful tuple evaluations, and `ISR_AvgError` is the average error (see Eq. 6.1). Observe that we try to maximize P_i while minimizing `ISR_AvgError` at the same time which guarantees higher fitness values for better candidate relations and follows our interest.

Genetic Operations

Having selected right individuals based on their fitness values, we perform the slightly customized genetic operations with given probabilities on expression trees built for the lowerbounds LB_i:

- *Reproduction* (with probability $\mathcal{P}_R$)
- *Mutation* (with probability $\mathcal{P}_M$) that picks a random node in the expression tree of an individual (except for the root) and replaces the node's subtree with a new random expression tree
- *Crossover* (with probability $\mathcal{P}_C$) that randomly selects a node in each expression tree of two given individuals (except for the root) and mutually swap their subtrees.
- *Alter-Architecture* (with probability $\mathcal{P}_A$) that selects a random node in the expression tree of an individual and changes the operator in the unary/binary function or replaces the current terminal (numeric constant or distance) with another random terminal

The offspring of these operations provide us the next population which is ready for the next iteration of evaluation and selection processes. At all times, we maintain a set of top k individuals[4] with the highest fitness values which represent the best-so-far results. When the termination criterion holds, this set contains k best individuals that we further validate with the indexing/querying tests. This extension allows us to output multiple items as the final result.

6.4.2 GP-SIMDEX Algorithm

After the initialization, we build the initial population of randomly generated candidate relations R_i^* with the limited complexity based on the given criteria (terminal set, function set, population size). Then, we iterate through subsequent populations until the termination criterion T is satisfied.

For each population, we evaluate the candidate relations, compute fitness values for all individuals, select individuals for genetic operations, and apply those. We include all settings necessary to perform these steps in the algorithm's input parameter ISR. At the end, we return the set of best-so-far individuals for the current run, as depicted in Algorithm 5.

During the execution, we maintain the current population (*population* variable) and the set of best-so-far individuals (*topIndividuals*) throughout the whole execution, as the set might evolve with each population and the best results do not have to reside in the last population that we evaluate.

[4]For efficiency purposes the set is implemented as the fixed-size heap with the minimal fitness value in the root which guarantees the optimal insertion of additional best results

Algorithm 5 GP-SIMDEX (ISR, T, S, δ)

```
Require: Inequality Symbolic Regression properties $ISR$, termination criterion
    $T$, database sample $S$, distance function $\delta$
 1: $M_{\delta,S}$ ← new distance matrix ($\delta, S$)
 2: $population$ ← RandomPopulation(ISR.Terminals, ISR.Functions, ISR.Size)
 3: $topIndividuals$ ← new empty fixed-size set(ISR.TopIndividualsCount)
 4: while not T.IsSatisfied($ISR$, $population$, $topIndividuals$) do
 5:     Evaluate($population$, $M_{\delta,S}$)   {evaluate candidate relations}
 6:     ComputeFitnessValues($population$, ISR.FitnessFunction)
 7:     $topIndividuals$.Add ($population$.GetBestIndividuals())   {best-so-far}
 8:     $intermediate$ ← SelectIndividuals($population$, ISR)
 9:     $population$ ← ApplyGeneticOperations($intermediate$, ISR)
10: end while
11: return $topIndividuals$   {return the set of best-so-far individuals}
```

6.5 GP-SIMDEX Evaluation

To verify the usability of the previously introduced concept of GP-SIMDEX in practice, we extensively test it by applying the implemented prototype to the real-world datasets. Although we focus our research primarily on similarity models in which indexing and querying with metric postulates produce notable errors, we employ similarity models which involve nonmetric distances and also show how GP-SIMDEX behaves in metric spaces.

During the experiments, we traced several parameters in order to find out how vital GP-SIMDEX is, namely

- **Fitness evolution** – how the fitness value of the best-so-far individual changes over time
- **Average Precision** – during the query evaluation with a chosen lower-bounding method, we study the precision averaged over all queries. We understand the precision as the *success rate* which is the number of items in the query result identical to items in the result from the sequential scan divided by the total number of items in the query result:

$$\texttt{SuccessRatio} = \frac{|\texttt{Result}_{LB} \cap \texttt{Result}_{SEQ}|}{|\texttt{Result}_{SEQ}|} \tag{6.3}$$

- **Average DC ratio** – we take the average number of distance computations compared to the DCs produced by the SEQ scanning. The reason why we select the average number of distance computations is that it is a relatively good measurement for query efficiency not dependent on the hardware used.

Inequality Symbolic Regression Settings

Before we present the results, we will provide background information about framework settings we use to get the results that we will present.

The function set consists of the unary functions (`abs`, `sqr`) and the binary functions (`min`, `max`, `+`, `-`, `*`, `/`).

As the terminal set, we use distances between random pairs of six independent database objects (query q, object o, and pivots $p_1 - p_4$) and random numeric constants from the interval $\langle -5, 5 \rangle$.

As the evaluation function, we apply the *random sampling* of k-tuples with a fixed number of 8,000 tuples for each k which corresponds to the number of independent variables in a relation, the relation cardinality (see Def. 4.4). We enable the lowerbound correctness testing (see Section 6.3).

Regarding ISR settings, we use the population size of 800 individuals with a limited complexity (the maximal depth of all initial expression trees is six levels) that evolve in 1,000 generations using the standard tournament selection with re-selection enabled (this provides the best results compared to others). The genetic operations occur with probabilities: $\mathcal{P}_M = 5\%$, $\mathcal{P}_R = 10\%$, $\mathcal{P}_A = 10\%$, and $\mathcal{P}_C = 75\%$.

We generate the initial population iteratively by randomly building the expression trees with respect to the terminal and function sets together with limited complexity settings. We do not put any further restrictions on creating the candidate relations.

Because we are interested in tight lowerbounds (which we expect to be most powerful in the distance estimations and object filtering), we put a strong focus on the value of the average difference (see `ISR_AvgError` in Eq. 6.1). Therefore, we define the fitness function as

$$fitness = \frac{\texttt{SuccessRatio}}{5 \cdot \texttt{ISR\_AvgError}} \tag{6.4}$$

For completeness, we accept only lowerbound candidates with 100% success ratio and use a maximal number of 100 threads for ISR evaluation.

Datasets

Regarding the datasets, we test more than 10 different similarity models with various data. However, for presenting results, we pick two representational datasets that provide the most promising results. More precisely, we depict results for

- *PolygonSet* – a synthetic dataset of 250,000 polygons in 2D space, each consisting of 5 to 15 vertices measured with Hausdorff distance using L_2 as the ground distance (see Section 2.2.1)
- color histograms from *Corel Image Features* [114] with nonmetric Jeffrey Divergence distance measure [10]. While the first dataset is representational for metric similarity models, this one belongs to nonmetric ones.

For the experimental evaluations, we create a distance matrix for each dataset by computing pair-wise distances between 1,000 randomly selected objects from the database.

6.5.1 GP-SIMDEX Fitness Evolution

Our first task is to track how the candidates evolve during the time in subsequent generations. For this purpose, we log and audit the fitness values of the best-so-far candidate relations. This provides us a good feedback whether the evolution is converging to a better result, or whether it is just randomly testing new relations.

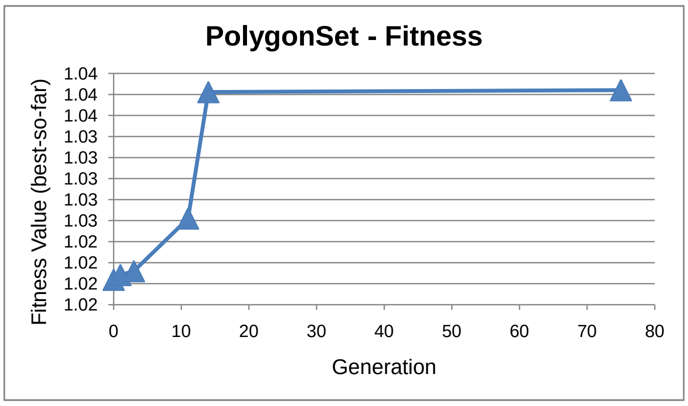


Figure 6.4: PolygonSet – best fitness value evolution

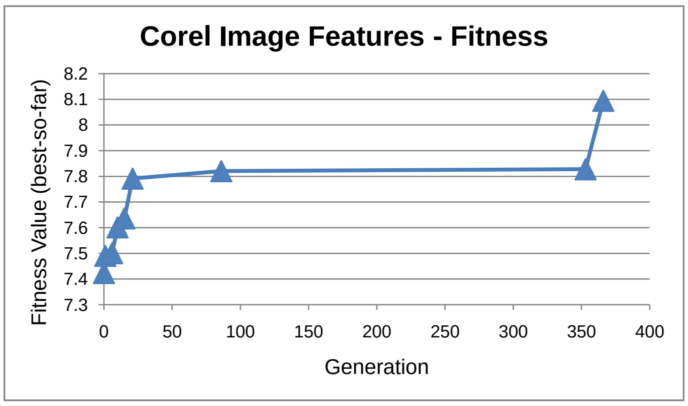


Figure 6.5: Corel Image Features – best fitness value evolution

PolygonSet Fitness Evolution

In Fig. 6.4, we study the fitness evolution for the dataset *PolygonSet*. Although it might not be clear at the first sight, we notice that in the 14th generation `GP-SIMDEX` rediscovers (as the candidate #6) the triangle lowerbound $LB_\triangle$ (see Eq. 3.3) which is the best-known indexing method for general metric spaces.

However, in the 75th generation, it slightly improves the fitness value by finding the candidate #5:

$$\delta(q,o) \geq \left| \min\left\{2.3192, \frac{-4.9608}{\delta(o,p_1)^2}\right\} - (\delta(o,p_2) - \delta(p_2,q)) \right|$$

Within the top 10 individuals, we find also interesting candidate #10 with a similar fitness value as $LB_\triangle$:

$$\delta(q,o) \geq |\min\{\delta(p_2,p_1) \cdot 3.2683, \delta(o,p_1) - \delta(p_1,q)\}|$$

Corel Image Features Fitness Evolution

For the nonmetric similarity model represented by *Corel Image Features*, we depict the fitness evolution in Fig. 6.5. We highlight two generations (21 and 366) in which the huge improvements occur.

The responsibility for these improvements hold the following candidates, namely the candidates #49 and #50

$$\#49: \qquad \delta(q,o) \geq \min\left\{\frac{\delta(p_1,q)\cdot(\delta(o,p_2)-\delta(p_2,q))^2}{2.0993}, \delta(o,p_3)\right\}$$

$$\#50: \qquad \delta(q,o) \geq \min\left\{\left|\frac{\delta(o,p_1)-\delta(p_1,q)}{max(4.3194,\delta(p_2,q))}\right|, \delta(q,p_1)\right\}$$

In following sections, we will explain how the discovered candidate relations act in the indexing tasks.

6.5.2 GP-SIMDEX Indexing Tasks

To verify the outcomes (resulting lowerbounds), we run several indexing tests using Pivot Table indexing method [3] with 10 pivots. We employ the lowerbounds in the same way as ptolemaic indexing has been applied [15].

To evaluate the effectiveness of the lowerbounds, we compare the newly found lowerbound expressions with traditional triangle (see Section 3.1.1) and ptolemaic (see Section 3.1.2) lowerbounds.

We focus our observation on (1) the number of distance computations of the lowerbound method, (2) the number of distance computations of the sequential scan, (3) the error rate, and (4) the average precision (refer to Section 6.5).

We perform 100 randomly chosen kNN queries with $k = 10$ and average the results over two database sizes (10,000 and 20,000) for *Corel Image Features* and over three database sizes (100,000; 150,000; and 200,000) for *PolygonSet*.

Corel Image Features Indexing

Because *Corel Image Features* dataset with Jeffrey divergence distance is nonmetric, we also take into account the *TriGenFP* lowerbound which is the triangle lowerbounding modified by TriGen algorithm [8]. We use 100,000 triplets in 24 iterations to find the best Fractional Power (FP) modifier, and we get FP with the weight $w = 0.802037$; for further details about TriGen algorithm and its settings see [52, 8] or Section 3.2.5.

The results depicted in Fig. 6.6 are interesting – the similarity model is clearly nonmetric (triangle lowerbound's precision of 48%) and the ptolemaic lowerbound (with the precision of tiny 24%) does not work neither.

On the other hand, we discover the lowerbound #50 which gives 99.8% precision rate using less than 35% of SEQ scan‘s DCs. We acknowledge that this is not a perfect fit, however we obtain the lowerbound from a relatively small database sample which indicates a pretty good result. The other lowerbound #49 acts in this case as the best solution. It gives 100% precision and at the same time it needs only 23.77% of DCs compared to SEQ scanning.

For completeness, we also provide a lowerbound expression which returns 100% success during the exploration phase, however it performs poorly while indexing. This false alarm causes the expression #47:

$$\min\{2.4454 + \min\{\delta(p,o), 0.841284 + \delta(q,p)\}, 0.0407\}$$

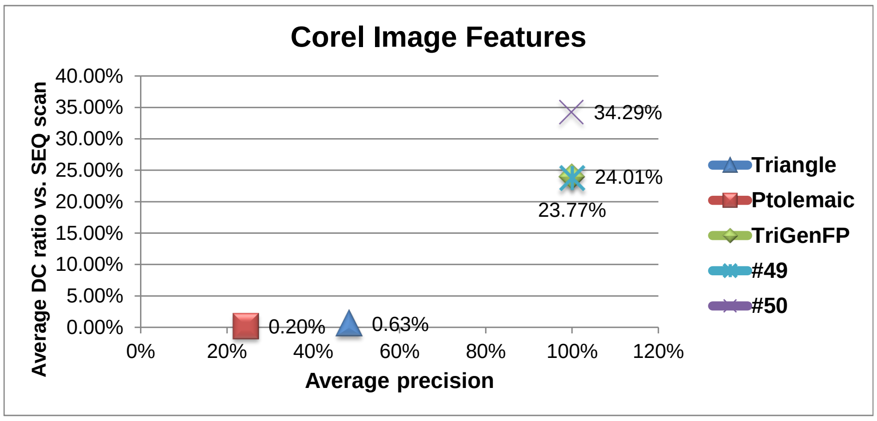


Figure 6.6: Corel Image Features – Average DC ratio vs. Average precision

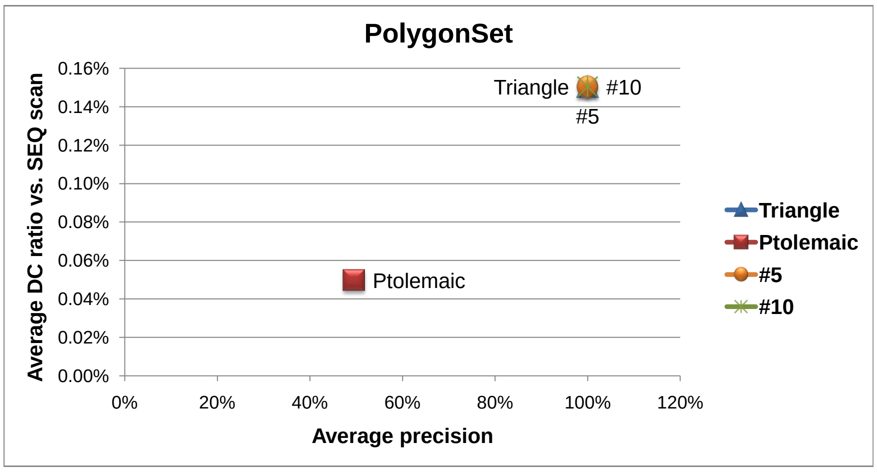


Figure 6.7: PolygonSet – Average DC ratio vs. Average precision

Studying Fig. 6.6, we see that this lowerbound returns only 80.75% precision using more than 81% SEQ scan‘s DCs.

To properly evaluate the results, we compare our results with TriGenFP lowerbounding. TriGen algorithm works here, as this method gives 100% precision while using 24.01% of sequential DCs. This shows the validity of TriGen and that properly configured GP-SIMDEX is capable of finding better results.

The comparison of results from the exploration phase (expression #50 is better than #49 in terms of fitness value evaluations) and indexing test results (lowerbound #49 performs better than #50; inefficiency of #47) shows that the currently used fitness function is not ideal. We will need to make further enhancements to fitness function to disallow the occurrence of such cases.

PolygonSet Indexing

Because we use L_2 distance [3] as the ground distance for Hausdorff distance, the resulting similarity space is metric. We observe this in Fig. 6.7 which shows how the precision relates to the average DC ratio. The triangle lowerbound $LB_\triangle$ provides 100% precision while decreasing the number of DCs. We admit that the Ptolemaic method is faster but it provides a low precision of 49%.

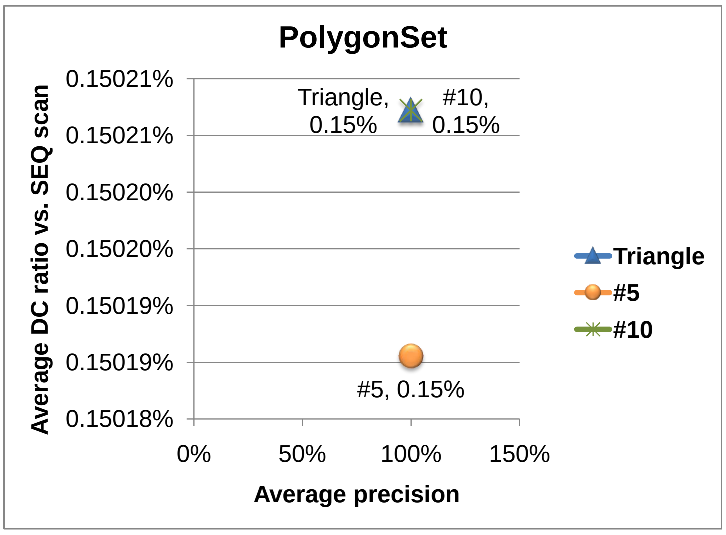


Figure 6.8: PolygonSet – Detailed Average DC ratio vs. Average precision

The newly discovered lowerbound #10 performs in our environment with the same results as the Triangle lowerbound. The other lowerbounding method #5 shows slightly better performance than the previous two techniques but the improvements are hardly noticeable (see Fig. 6.8).

Due to these representational results and our previous experience, we give the research community the open question whether there is a way of finding any better indexing method for metric spaces than the triangle lowerbound $LB_{\triangle}$. This seems to be difficult especially for any metric L_p (Minkowski) distances [3].

6.6 Comparing GP-SIMDEX vs. I-SIMDEX

The feasibility study reveals that `GP-SIMDEX` does provide advantages over the more trivial `I-SIMDEX`. Note that it is very difficult to compare these approaches between each other in other way than by comparing the final outputs or the real-time we need to reach such results.

Nevertheless, `I-SIMDEX` gives good, acceptable, and deterministic results within predictable time frames. Because we explore the axiom space iteratively, this time frame might be relatively big. Also, after reaching a good solution, it is almost impossible to improve it and direct the next steps towards better results, as there exist no feedback for the future exploration direction – it is a priori fixed. Here, `GP-SIMDEX` drives the exploration based on the feedback from the previous population and always changes the directions towards possibly better results.

A good example is the comparison of the final results depicted[5] in Fig. 6.9 (`I-SIMDEX` results) and Fig. 6.10 (`GP-SIMDEX` results) for the Corel dataset [114] with nonmetric Jeffrey Divergence distance measure [10]. `I-SIMDEX` provides almost ideal candidate, however it misses 100% precision and there is no way to improve this result to become a perfect fit. Regarding the number of DCs, the output from `GP-SIMDEX` requires 24% of SEQ DCs (which conforms up to 4× speed-up), while `I-SIMDEX` results returns only up to 1.2× speed-up.

[5]For clarity, we append these figures once again – see the original Fig. 5.5 and Fig. 6.6.

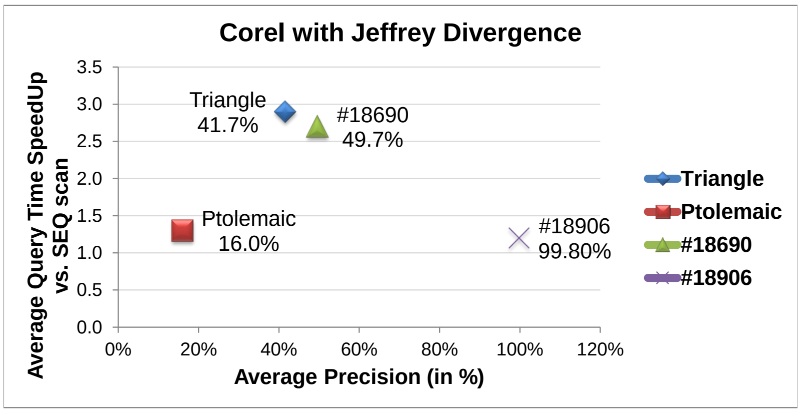


Figure 6.9: Corel Image Features – I-SIMDEX results as depicted in Fig. 5.5

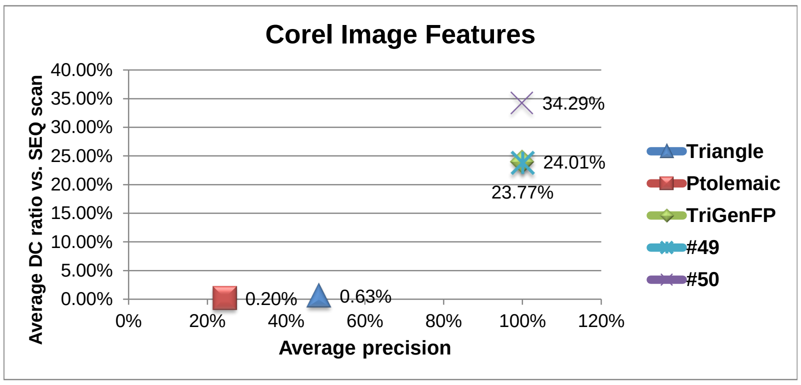


Figure 6.10: Corel Image Features – GP-SIMDEX results as depicted in Fig. 6.6

Also the multiple re-evaluations of the same mathematical expression occur far less often in GP-SIMDEX than in I-SIMDEX, as genetic operations modify the candidate relations.

However, there are several drawbacks such as the nondeterministic behavior which results in unpredictable outputs. Therefore we need to execute multiple runs of the GP-SIMDEX in order to get acceptable (and stable) results. Also, the increased complexity of the implementation and huge dependency on the random generator might be the reasons for not using GP-SIMDEX.

Overall, in the ideal world, we would mix both approaches – GP-SIMDEX for main exploration purposes and I-SIMDEX for occasional re-generations / enhancements of existing populations.

6.7 GP-SIMDEX Challenges

In our work, we validate the proposed theoretical concept of GP-SIMDEX and verify its usability as the universal framework for finding suitable indexing methods for a given database. We attach results which conform that GP-SIMDEX is capable of exploring any similarity space based only on the given data and a similarity measure in order to provide insight to the model and reveal alternative indexing methods. The discovered indexing methods for various datasets in most cases outperform the best-known indexing methods.

There are couple of challenges that we need to address in order to enable `GP-SIMDEX` for wider usage such as

1. increase the *efficiency* by applying a higher degree of parallelization using *island-population model* [138, 139]. This will allow us to either evaluate more populations and find better results, or inflate the total size of the terminal/function set to be used.

2. utilize *architecture scaling* to tens/hundreds of computers or CPUs using modern Map-Reduce [4, 140] techniques in order to boost the total performance.

3. use third-party software tools such as Maxima system[6] to *simplify* all tested lowerbounds (or mathematical expressions in general) and group them to equivalent classes. We expect that this will improve the whole performance as we will disable repeated testing of different forms of the same mathematical expression.

4. find out how to efficiently *explore equivalent classes* of candidate relations and thus provide the feedback for genetic operations, for further evolution, or for the next populations.

To address some of these issues and to improve the overall performance, we push the proposed `GP-SIMDEX` into the parallelized environment and in the following chapter, we introduce its parallelized version, `PGP-SIMDEX`.

[6]http://maxima.sourceforge.net/

Chapter 7

Parallel GP-SIMDEX

The genetic operations in the real world apply to thousands or millions of individuals at the same time, so it seems appropriate and reasonable to apply the parallelism to the GP problems as well. The next SIMDEX variant is based on this idea and also as the immediate answer to the previously introduced challenges of single-threaded `GP-SIMDEX` executions. As the consequence and as the further step in the research of SIMDEX Framework, we develop another variant that utilizes `GP-SIMDEX` in the parallel processing environment [110]. We named it appropriately `PGP-SIMDEX` which refers for Parallel `GP-SIMDEX`.

In this section, we introduce `PGP-SIMDEX` and focus on parallel enhancements of simple `GP-SIMDEX` [109] in order to reveal how `PGP-SIMDEX` applies genetic programming for discovering suitable lowerbounds in multi-threaded environment while it further improves the qualitative results by employing nontrivial parallel genetic programming methods.

The main benefit comes from the parallel implementation of the proposed (single-threaded) `GP-SIMDEX` which results in intelligent Parallel Genetic Programming (PGP) exploration method using well-known *island-population model* [138, 139] together with the recently introduced *map-reduce* approach [4, 140]. The combination of both approaches shows how to address the challenges of `GP-SIMDEX` (see Section 6.7).

Another outcome is the usability and immediate applicability of `PGP-SIMDEX` to multi-CPU environment or a cluster of computers. Besides the fact that we will be no longer dependent on the outputs from a single machine, this also adds huge computational power that we can benefit from. Before we outline `PGP-SIMDEX` algorithm, we describe the underlying principles into more details.

7.1 Island-Population Model

It is known for quite some time that distributed genetic algorithms usually outperforms serial executions of canonical genetic algorithms on many different problems [141, 142]. This however applies to GP with no modifications. More precisely, if we apply the parallelism to the known GP problem, we expect better results than the sum of results from the separated executions [143]. Often, PGP leads to the superior performance even though we implement the solution on a single multi-threaded processor. We do not necessarily have to implement the solution in a distributed environment with multiple processors.

There exist several approaches of how to implement parallelism for GP problems [142, 141, 143] but one of the most famous methods is the *island-population* model. In this case, we divide the execution of the given GP problem to several (almost) independent machines or nodes (so called *islands*).

Each island receives the same execution parameters, however it generates its own initial population and maintains all the subsequent populations. Therefore the program execution at the given island seems to be independent and there is no difference compared to multiple serial executions running at the same time. However, at periodic time-frames (the *migration intervals*), the island exchanges a small portion of its current population (with the fixed *migration size*) with another island within the process of *migration*. This way, the migration allows islands to share and propagate their genetic material [141]. For any island, this migration is a two-step process of (a) sending the best-so-far results to some non-specific island, and (b) receiving the a set of best results from another island.

The main outcome of such a model is that we are able to explore differences in several distinctly developing (sub)populations and reveal the true genetic diversity during the execution. There are specific rules how the genetic material is transferred between islands, so we mix the global differences with local ones.

Another advantage is that during the exploration, we usually receive higher quality of the discovered solution found and we are able to better navigate in the search space [144]. This results from the independent behavior of each island mixed with sharing the information through the migrations.

Note that we consider the parallelism here in the context of executing and evaluating a single GP problem. This is different from the situation in which we take into account the parallelism while executing the *evaluation function* (Def. 4.5).

7.2 Map-Reduce

With the huge expansion of data and the requirements to process very large datasets, we need a new model of how to process such data in a distributed environment. One of the successful attempts that provides a way of doing so is the *Map-Reduce* programming model [4, 140]. Authors provide the model that hides all implementation details (parallelization, fault-tolerance, or distributing data across various nodes) inspired by the primitives from the functional languages. With this abstraction layer, we apply this model to any problem that we solve in a distributed environment or within a cluster of multiple machines.

The model provides the interface that allows us to automatically parallelize and distribute the computations on large clusters. Here, users just specify two main functions: *map* and *reduce*. In the original paper, the input for the computation is a set of key/value pairs which we process with *map* function. For each pair, we generate a set of intermediate key/value pairs and run the process. Afterwards, the *reduce* function groups and handles all intermediate values with the same key and merges these values while trying to reduce the size of the final set. Such a technique allows easy and highly-scalable distributed computing.

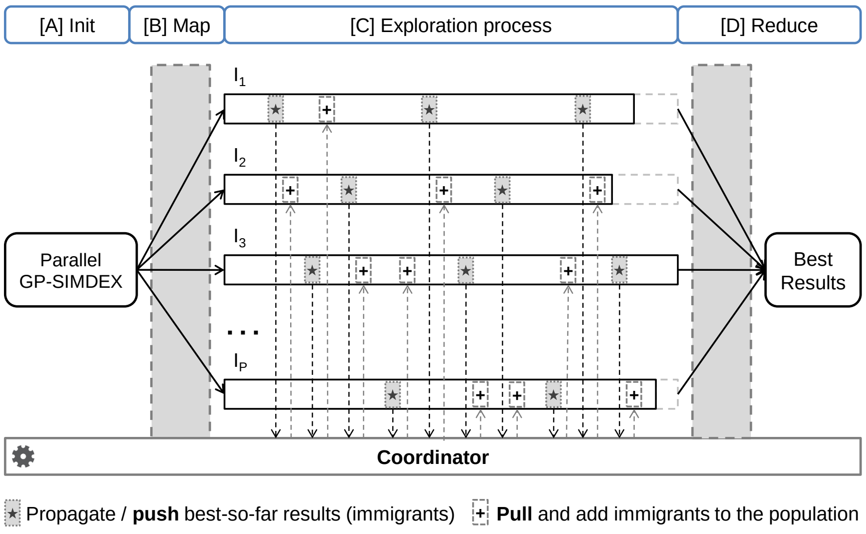

[A] Init
[B] Map
[C] Exploration process
[D] Reduce
I1
I2
I3
...
IP
Parallel GP-SIMDEX
Best Results
Coordinator
★ Propagate / **push** best-so-far results (immigrants)
+ **Pull** and add immigrants to the population

Figure 7.1: PGP-SIMDEX – High-level overview of the parallel approach

7.3 PGP-SIMDEX Architecture

The next variant of Parallel GP-SIMDEX (or shortly PGP-SIMDEX) leverages more sophisticated axiom exploration than running multiple independent executions of GP-SIMDEX at the same time [145]. It follows two previously described principles of the *island-population* model for efficient execution of parallel GP-based algorithms, and the *map-reduce* technique for distributing GP-based computations for processing huge amounts of data within the distributed environment. The first method assures that the context of each processing *island* is consistent.

As the result, we are able to parallelize the computations to a large extent and execute PGP-SIMDEX algorithm at multiple computing nodes at the same time while we are leveraging the advanced evaluation method. The general architecture of PGP-SIMDEX is very similar to the one depicted in Fig. 6.3 for GP-SIMDEX. However, it is adjusted to the distributed environment, so there are multiple instances running simultaneously.

The redistribution of immigrants between running instances is completely random, so one island might receive more immigrants than the others. For better comprehension, Fig. 7.1 outlines a sample visualization of this process. The total required computation power equals to $P + 1$ nodes where we use P nodes for the exploration plus one extra node for the *Coordinator*, as will be shown in the following text. We will divide the execution into several major steps, i.e. *phases*, and we sequentially execute these phases to obtain final results.

7.3.1 Phase A - Initialize

During the initialization phase, we build GP-SIMDEX instances for a given number P of processing nodes. Although the input parameters equal, all instances will build their own (random) initial populations of expressions and maintain the execution in their own way. If we ensure different seeds for random generators [146] in GP-SIMDEX instances, the evaluations will not be identical and will produce different results.

Because we run the exploration process on several nodes (*islands*) simultaneously, we need a special (non-computing) node that coordinates the whole exploration process from the beginning to the end. In our work, we suggest using a *Coordinator* that manages the execution and provides also the administrative role, i.e. knows about the number of running nodes, their states, guarantees the communication between nodes, manages waiting immigrants, etc. Also, this extra node consolidates the partial results and returns the final results.

7.3.2 Phase B - Map

Having all P instances ready for the execution, we select the number of computing nodes in the distributed environment that corresponds to the number of instances. We assign each computing node a single `GP-SIMDEX` instance together with the reference to the *Coordinator* and thus build a single *island*[1]. Finally, we start the exploration process on each *island*.

7.3.3 Phase C - Exploration Process

During the exploration, every island executes the `PGP-SIMDEX` algorithm (see Section 7.4) which is a modified version of `GP-SIMDEX` (see Algorithm 5). In short, the principles of GP-based axiom exploration remains unchanged – we repeatedly apply genetic operations with given probabilities to build further populations until the termination criterion holds. The only difference is that occasionally the running `PGP-SIMDEX` instance propagates the best-so-far results (denoted as *immigrants*) from its island to another instance.

Because islands do not interact between each other directly (as they do not know about each other), the whole communication is accomplished through the *Coordinator* node which randomly selects the target island that will receive the *immigrants*. When the target island next time checks for available immigrants, the *Coordinator* delivers them and they are appended to the next population. This way, they provide "fresh food" to drive the exploration efficiency. This approach is recommended to obtain better results more quickly [138, 139].

The optimal way of implementing such a solution to exchange information is to use *push* and *pull* mechanisms. The *push* action propagates the immigrants from the source island I_s to the *Coordinator* which randomly selects the target island I_t. Moreover, the *Coordinator* maintains a separate *waiting list* of immigrants to be delivered to each island – $waitlist[I]$. Each island I_t now and then asks the *Coordinator* whether there are some immigrants prepared in the waiting list to be delivered. If the waiting list $waitlist[I_t]$ is empty, nothing happens. Otherwise, the island I_t requests and *pulls* all immigrants from the waiting list and adds them to the next population to be evaluated. After they are successfully delivered to the target island I_t, the *Coordinator* removes them from the waiting list $waitlist[I_t]$.

The optimality of *push* and *pull* methods comes from the fact that we do not violate the consistency of any islands. The reason is that the initiator of *push* requests is the source island I_s, the initiator of *pull* requests is the target island I_t, and the synchronization of waiting lists using basic locking primitives is the job of the *Coordinator*.

[1]We will use the terms *nodes* and *islands* interchangeably

Algorithm 6 PGP-SIMDEX (ISR, T, S, δ, P)

```
Require: Inequality symbolic regression properties $ISR$, termination criterion
    $T$, database sample $S$, distance function $\delta$, number of parallel instances $P$
 1: $islands \leftarrow$ new array of PGP-SIMDEX instances
 2: $coordinator \leftarrow$ new PGPCoordinator()
 3: for $i = 1$ to $P$ do {$P$ denotes the level of parallelism}
 4:     $islands$[i] $\leftarrow$ new PGP-SIMDEX Instance($coordinator$, $ISR$, $T$,$S$, $\delta$)
 5:     $islands$[i].SetRandomSeed()
 6: end for
 7: $coordinator$.DistributeIslands( $islands$ )   {[B] Map}
 8: $coordinator$.ExecuteIslandsExploration( $islands$ )   {[C] Exploration}
 9: $coordinator$.WaitForIslandsToFinish( $islands$ )   {[C] Background process}
10: return $coordinator$.GetBestResults( $islands$ );   {[D] Consolidate results}
```

7.3.4 Phase D - Reduce

The very last step after all processing nodes finished the execution is to consolidate partial results and select the best results using the *reduce* technique. Each island produces and returns a limited number of best-so-far results.

Similarly to `GP-SIMDEX` execution, we use fixed-size heaps for each input similarity model within the consolidation process (see Section 6.4). In the heap, we sort the results based on their final fitness values while maintaining the result with the lowest fitness value at the top of the heap. With the fixed heap size of k items, we produce the final top-k results for each input similarity model.

As this is the job of the *Coordinator*, we incrementally build the final result based on partial results whenever an island finishes the execution and provides its best-so-far results. After the last island terminates, we already have the final results ready.

7.4 PGP-SIMDEX Algorithm

In previous sections, we outlined the individual stages (so called phases) of the algorithm, each responsible for a specific action. Here, we describe the overall execution and show how these phases interact with each other.

We divide `PGP-SIMDEX` process into two algorithms as it is more complex than the previous variants. Algorithm 6 outlines the high-level overview of how we perform the individual phases (A–D). This algorithm is easy to read, self-explainable, and relates to the previously described phases (see Section 7.3).

We outline the algorithm for `PGP-SIMDEX` Instance as the modified version of `GP-SIMDEX` algorithm, in Algorithm 7. Compared to `GP-SIMDEX`, it takes an additional parameter which provides the reference to the *Coordinator*. At the beginning, it checks for available immigrants (if required), pull them, and append them to the current population before it is evaluated. After a population is processed, the algorithm occasionally propagates / pushes the best-so-far results to another instance, again through the *Coordinator*. Here, we can see that the *Coordinator* plays an important role during the whole execution.

Algorithm 7 PGP-SIMDEX Instance ($C, ISR, T, M_{\delta,S}$)

```
Require: Coordinator $C$, Inequality Symbolic Regression properties $ISR$, termination criterion $T$, database sample $S$, distance function $\delta$
1: $M_{\delta,S} \leftarrow$ new distance matrix $(\delta, S)$
2: $population \leftarrow$ RandomPopulation(ISR.Terminals, ISR.Functions, ISR.Size)
3: $topIndividuals \leftarrow$ new empty fixed-size set(ISR.TopIndividualsCount)
4: while not T.IsSatisfied($ISR$, $population$, $topIndividuals$) do
5:    if ISR.CheckForImmigrants then {regularly pull immigrants}
6:       $immigrants = C$.PullImmigrants($M_{\delta,S}$)   {retrieve immigrants}
7:       $population$.Append($immigrants$)   {add incoming immigrants if any}
8:    end if
9:    Evaluate($population$, $M_{\delta,S}$)   {evaluate candidate relations}
10:   ComputeFitnessValues($population$, ISR.FitnessFunction)
11:   $topIndividuals$.Add ($population$.GetBestIndividuals())   {best-so-far}
12:   $intermediate \leftarrow$ SelectIndividuals($population$, ISR) {apply the selection}
13:   $population \leftarrow$ ApplyGeneticOperations($intermediate$, ISR)
14:   if ISR.PushImmigrants then {regularly push immigrants}
15:      $C$.PushImmigrants($topIndividuals$)   {propagate immigrants}
16:   end if
17: end while
18: return $topIndividuals$   {return the set of best-so-far individuals}
```

It is the matter of ISR settings to decide whether for the current population the pull or push request occur or not. Usually, the pull requests arise more often (usually at the beginning of each iteration) than the push requests. The reason for this is that in the distributed environment the target node might receive multiple immigrants from several different nodes during a single population evaluation. However, the best-so-far individuals do not change quite so often, so the "push" operation occurs less often.

7.5 Comparing PGP-SIMDEX vs. GP-SIMDEX

To examine our concept and to compare it with `GP-SIMDEX`, we generate eight datasets of 11,000 random objects (represented as points in $4D$ space) and we apply several nonmetric similarity models mostly with nonzero triangle inequality errors. We select L_p distances to demonstrate the variability of similarity models (see *Triangle Error* in Table 7.1).

For the exploration, we use database samples consisting of 1,000 objects and sample 10,000 k-tuples for evaluating each candidate relation. We run all experiments multiple times and provide only the best results.

`GP-SIMDEX` starts with the initial population that includes 1,000 candidate relations which evolve in 1,000 generations. On the other hand, `PGP-SIMDEX` uses 10 separated islands and only 100 generations, which results in comparable settings. For both cases, we deal with approximately 1,000,000 candidate relations evaluated in total.

#	Distance	Triangle Error	Best Fitness		PGP Improvement	Total PGP immigrants
			GP	PGP		
1	Jeffrey Divergence	14.0 %	38.08	38.40	0.86 %	7,198
2	$L_{0.2}$	7.6 %	23.82	24.44	2.61 %	9,452
3	$L_{0.4}$	5.5 %	22.53	23.13	2.65 %	6,984
4	$L_{0.6}$	3.8 %	21.18	22.09	4.32 %	8,150
5	$L_{0.8}$	2.5 %	20.75	21.19	2.15 %	7,671
6	L_1	0.3 %	19.94	20.39	2.25 %	7,278
7	L_2	0.0 %	16.82	17.40	3.46 %	7,343
8	L_∞	0.3 %	12.27	12.50	1.85 %	6,923

Table 7.1: Improvements of `PGP-SIMDEX` over `GP-SIMDEX`

During the exploration phase, we study

- the comparison of *best fitness values* to find out the overall improvement. Here, `PGP-SIMDEX` slightly outperforms `GP-SIMDEX`.
- the *number of immigrants* – how many additional expressions "helped" `PGP-SIMDEX` to achieve better results
- how the combination of these metrics relates to the global improvement on the performance (see Table 7.1).

With initial evaluations and obtained results, we are able to demonstrate the applicability of the parallel processing and its power to improve results. However, currently we obtain only small efficiency increase (up to 4%) which is relatively disappointing. To overcome this, we need to either find the relation between the improvements and ideal `PGP-SIMDEX` settings, or to apply the experimental evaluation at larger scale (multi-CPU farm). Only afterwards, we will be able to conclude the effectiveness of the proposed solution.

7.6 PGP-SIMDEX Summary

We perceive the proposed `PGP-SIMDEX` as the shift towards an intelligent indexing method, so called *Smart Pivot Table* [110], applicable to any similarity model. Although we address some of the `GP-SIMDEX` challenges, there are still issues we need to overcome such as the non-deterministic behavior or the increased complexity of the implementation. These tasks remain as the next steps of our research.

Having several SIMDEX variants which demonstrate the feasibility of the theoretical framework, we would like to apply it to the practice by building a smart indexing scheme usable for arbitrary similarity models. In the following chapter, we describe this application into more details.

Chapter 8

Smart Pivot Table Index

While the previous sections and individual SIMDEX variants show the steps towards building a general *intelligent* indexing scheme with the data-driven discovery of suitable lowerbounds, here, we provide a concrete example of such a technique with our recently introduced *Smart Pivot Table* index [110].

Our objective is to apply the previous results in a consolidated manner and to introduce the technique of *Triangle*$^+$ *lowerbounding* and *filtering* (see Section 8.2). The latter method inspired us to such extent that we propose a novel technique of *LowerBound Tightening* (see Section 8.4) that relies upon improving filtering capabilities of lowerbounds and is orthogonal (thus applicable) to all existing SIMDEX variants. This provides the final outcome of our research.

8.1 Smart Pivot Table Concept

Smart Pivot Table is based on the pivot table index (see Section 3.2.2) such as AESA [64] or LAESA [65, 3], but we extend the standard functionality of *metric* (see Section 3.2.2) or *ptolemaic* (see Section 3.2.4) lowerbounding by deriving the lowerbounding mechanism that will fit for the data combined with arbitrary (but fixed) similarity measures. For the discovery of appropriate lowerbounds, we use any variant of SIMDEX Framework.

The concept of the *intelligent indexing* is based on the *double-pivot constraint* [3] and it follows the approach introduced with ptolemaic filtering [15, 51]. We can accomplish this principle easily due to the enforced form of all resulting lowerbounds that we get as outputs from SIMDEX Framework (see Def. 4.3). During the query evaluation, we compute the lowerbound value `LB_Value` that estimates the real value of the distance $\delta(q, o)$ between the query object q and any database object o as

$$\delta(q,o) \geq \texttt{LB\_Value} = \max_{s_N \in \mathbb{P}} \left\{ LB(q, o, s_N) \right\} \tag{8.1}$$

where s_N is a sequence of N pivots from the pivot set $\mathbb{P}$ such that

$$s_N = p'_1, p'_2, \ldots, p'_N \quad \forall i,j = 1,\ldots,N : p'_i, p'_j \in \mathbb{P}, \quad \forall i \neq j : p'_i \neq p'_j$$

We override the standard operator of inclusion $\in$ which for our case $s_N \in \mathbb{P}$ gives:

$$s_N \in \mathbb{P} \quad \leftrightarrow \quad N \leq |\mathbb{P}| \quad \text{and} \quad \forall p'_i \in s_N : p'_i \in \mathbb{P}$$

As the result, this approach allows us to use any valid lowerbound expression LB to filter out database objects o_i without computing the generally expensive distance $\delta(q, o_i)$. This applies to objects for which the lowerbound value is greater than the given radius or the distance to the k-th nearest neighbor discovered so far, i.e. `LB_Value` $> maxRadius$. Together with *early abandoning* principle (to stop computing the lowerbound value if it currently exceeds the $maxRadius$), we obtain an efficient pruning mechanism.

We can view the sequence s_N as an ordered set of N pivots which assigns each pivot variable in the lowerbound expression LB the object of the corresponding pivot from $\mathbb{P}$ before we evaluate the LB. Note that it is easy to get the ordered list of all variable names from the given LB expression (e.g., by traversing the corresponding tree-like structure of LB). Here, the resulting order of variables is not relevant, however, the cardinality of the sequence s_N depends only on the number of independent variables N in the lowerbound expression LB.

Furthermore, we can limit the number of sequences s_N to be evaluated either to all N-tuples which gives $\binom{|\mathbb{P}|}{N}$ tuples in total, or to a smaller constant $C < \binom{|\mathbb{P}|}{N}$. To clearly distinguish between these two cases, we will explicitly mark the limited number C of lowerbound evaluations as $\texttt{LB\_Value}_C$.

The obvious challenge here is to choose the best pivot selection method [68] to be applied and to define the best order of evaluating sequences s_N while computing the lowerbound value.

Also, we are able to combine several lowerbounds to obtain `LB_Value` using the same principles as proposed for `I-SIMDEX` (see Section 5.1.4) or for ptolemaic pivot tables (see Section 3.2.4) such as the combination of metric & custom LB, ptolemaic & custom LB, etc., as we depict in *Triangle$^+$ filtering* (see Section8.2).

The described approach gets the suitable lowerbounds from the "external" engine (SIMDEX Framework) using only the data stored within the pivot table index. It discovers the indexing method for the indexed data – this is feasible as we always modify the filtering options, not the stored data. In general, we incorporate the axiom discovery techniques directly into pivot table index. Also, the combination with other methods such as those of LAESA and TriGen [8], or LAESA and fuzzy approaches [91, 19] might bring additional benefits.

8.2 Triangle$^+$ Lowerbounding

In metric spaces, the basic property of the triangle inequality (see Table 2.1) enables us to create effective and efficient triangle lowerbound $LB_\triangle$ (Eq. 3.3). However, for some similarity models, this lowerbound is not as tight as we would expect which has been demonstrated for Signature Quadratic Form Distance (see Section 3.1.2). In that case, the combination with the ptolemaic lowerbound $LB_{\texttt{ptol}}$ (Eq. 3.7) presents better results and provides more efficient pruning of non-interesting objects [15, 51].

Our main objective is to retain the standard $LB_\triangle$ and to increase its filtering power for such cases in which $LB_\triangle$ does not prune objects. The idea is simply to take an additional lowerbound LB^+ which is (in specific cases) supposed to provide better distance approximations. This will improve the triangle lowerbound pruning and we denote this method as *Triangle$^+$ Lowerbounding.*

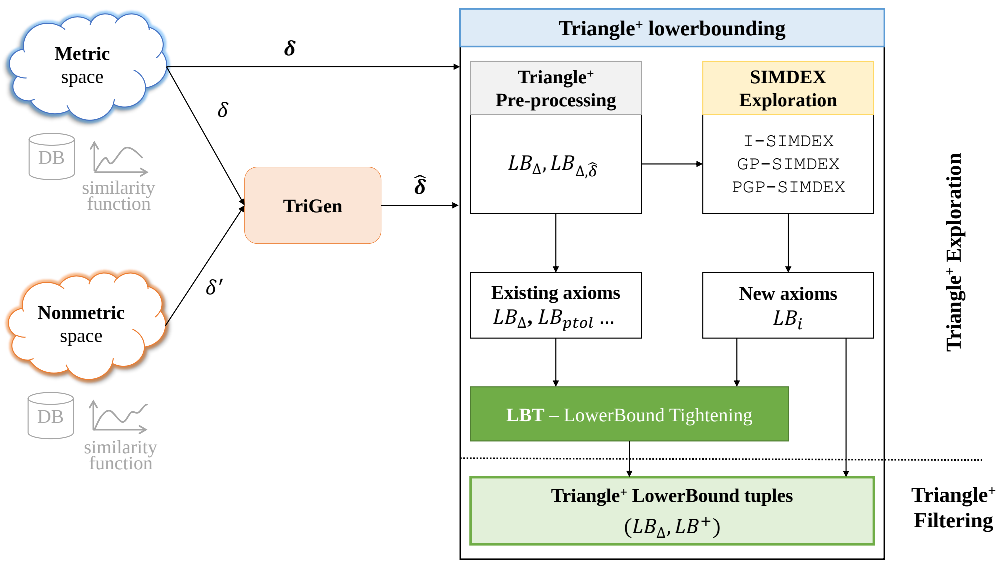


Figure 8.1: Triangle$^+$ lowerbounding overview

This opens a question whether we could enhance the triangle lowerbound with the result from any of the existing SIMDEX variants. Because all previous variants (`I-SIMDEX`, `GP-SIMDEX`, or `PGP-SIMDEX`) deal with general axiom spaces, applicable (not strictly limited) to metric spaces, this approach is feasible.

We depict the concept of *Triangle$^+$ Lowerbounding* in Fig. 8.1 which shows relationships between individual components. We distinguish the exploration phase (see Section 8.2.1) from the filtering stage (see Section 8.2.2). While the first one occurs once when the pivot table is constructed, we apply the latter one during each query evaluation.

Observe, that it is plausible to convert general (nonmetric) spaces to metric ones with TriGen algorithm (see Section 3.2.5) as the initial step. Then, we work with the modified space (given by the distance $\widehat{\delta}$ as opposed to δ) and we apply $LB_{\triangle,\widehat{\delta}}$[1] instead of $LB_\triangle$. For simplicity, we suppose only metric spaces and we take into account $LB_\triangle$ as the basis.

8.2.1 Triangle$^+$ Exploration

We would like to enhance the triangle lowerbound $LB_\triangle$ with an extra lowerbound LB^+ in order to increase the efficiency of query evaluation. Then, the combined lowerbound $LB^+_\triangle$ will provide better distance approximations and greater filtering power.

To obtain the appropriate lowerbound LB^+ from SIMDEX, we first modify the input parameter. Instead of using the whole distance matrix $M_{\delta,S}$, we consider only such objects that we cannot filter out by $LB_\triangle$. Note, that this is not a big issue and it means only a slight modification of the evaluation function. The other thing we need to ensure, is to use LAESA-like evaluation (see Section 4.4.2) to simulate real similarity queries. This is important in order to define the mining field correctly; otherwise we will not get relevant results.

[1] $LB_{\triangle,\widehat{\delta}}$ denotes $LB_\triangle$ that works with the modified distance $\widehat{\delta}$

Algorithm 8 Triangle$^+$ Preprocessing ($M_{\delta,S}$, *query*, $\mathcal{P}$, *radius*)

```
Require: Distance matrix $M_{\delta,S}$, a query object $query$, set of pivots $\mathcal{P}$, range
    query radius $radius$
1: for all $object \in M_{\delta,S}$ do
2:     $LB_\triangle \leftarrow$ TriangleLowerbound($query$, $object$, $\mathcal{P}$)
3:     $object$.Filtered = $LB_\triangle > radius$
4: end for
```

To fulfill these requirements, we use the Triangle$^+$ Preprocessing algorithm as depicted in Algorithm 8 and for mining purposes we consider only objects for which *object*.Filtered = `false`. If we use multiple queries, we can store the sets of objects to be filtered out for each query in a hashtable for fast access. Here, we do not need to compute any distance, as we deal with pure lowerbounds.

Even though we outline the preprocessing for range query $\mathrm{RQ}_\delta(q, radius)$ evaluation, it is easily adjustable to kNN queries by maintaining the priority queue with the distance to the kth nearest neighbor as the radius (see Section 3.1).

Having done this preprocessing phase, we are able to run SIMDEX instance to discover appropriate lowerbound(s) LB^+. Note that during the exploration we can reveal several suitable relations with same fitness values. Then, we either pick a single output based on additional requirements (e.g., a minimal number of pivots or a minimal number of total lowerbound computations), or we apply multiple lowerbounds LB_i^+ in order to obtain the additional lowerbound value and choose the maximal value. For initial evaluation, we pick only a single lowerbound LB^+.

We admit, that LB^+ might not be a true lowerbound for the whole distance matrix $M_{\delta,S}$ according to the formal definition (Def. 3.1) because we limit the mining space to a considerable extent. However, the way we will apply it (see Section 8.2.2) allows us to consider it as a valid lowerbound.

For now, we skip the description of *LowerBound Tightening* component which we introduce in Section 8.4. Also, it is an optional post-processing step.

8.2.2 Triangle$^+$ Filtering

With the resulting pairs $(LB_\triangle, LB^+)$ obtained from the previous (exploration) step, we are able to apply the advanced filtering during the query processing. For objects that we are able to filter out using standard $LB_\triangle$, nothing changes. Otherwise, (for non-filtered objects) we compute the additional lowerbound value using a specific lowerbound LB^+. Here, we expect that this value will be greater $value(LB^+) \geq value(LB_\triangle)$ and so it has a higher probability of filtering out some additional irrelevant objects.

This concept is similar to existing lowerbound techniques which combine multiple lowerbounds and take the maximum lowerbound value (Eq. 8.1) in order to filter out irrelevant objects. However, the additional LB^+ does not have to be a true lowerbound, as we expect it to provide better approximations only for some objects.

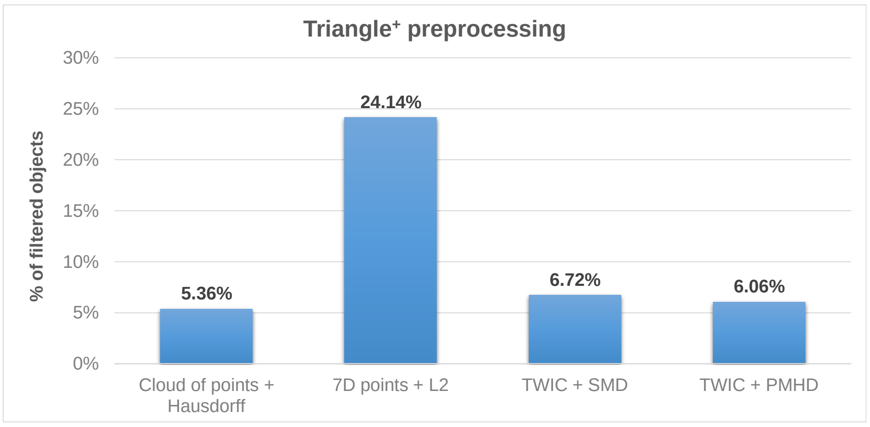


Figure 8.2: Triangle$^+$ preprocessing – Ratio of filtered objects

8.3 Evaluating Triangle$^+$ with GP-SIMDEX

In order to reveal the potential of *Triangle$^+$ lowerbounding* method which employs `GP-SIMDEX` variant, we use the following list of similarity models:

- *Clouds of points* with 360 dimensions using Hausdorff distance and L_2 as the ground distance (see Section 2.2.1)

- *7D points* – random objects in low-dimensional space constrained by the hypercube in which values in each dimension are limited to the interval $\langle 0, 100 \rangle$ with L_2 distance (see Section 2.2.1)

- *TWIC* (Thematic Web Images Collection) dataset [147] with *Signature Matching Distance* (SMD) for content-based image retrieval [148]

- *TWIC* (Thematic Web Images Collection) dataset [147] with *Perceptually Modified Hausdorff Distance* (PMHD) [149]

First two datasets consist of randomly generated data in high- and low-dimensional spaces and we apply basic and more complex metric models. The other two datasets employ the recently introduced TWIC dataset [147] that is designed for testing purposes in the content-based image retrieval area. All datasets are normalized, so that the values are limited to $\langle 0, 1 \rangle$.

8.3.1 GP-SIMDEX Exploration

For the exploration purposes, we use `GP-SIMDEX` and its standardized settings (see Section 6.5) with the population size of 80 individuals that evolve in 1,000 generations. We use LAESA-like evaluation function with 10 sample queries, 5 pivots, and kNN ($k = 5$) queries on the dataset that consists of 500 objects, tweaked by Triangle$^+$ preprocessing phase. Figure 8.2 summarizes the number of objects filtered by the triangle inequality and thus the number of objects involved in the evaluation during the exploration. The lower the ratio, the bigger the number of objects that remain for the exploration.

Exploration Results

Based on the previous settings, we pick the best individuals from the exploration process. We discover the following LB^+ relations suitable for lowerbound filtering:

- *Clouds of points* will be enriched by
$$\delta(q,o) \geq |(\delta(q,p) - \delta(o,p)|^{0.717061}$$
- *7D points* will use
$$\delta(q,o) \geq |\delta(q,p) - \delta(o,p)|^{0.918389}$$
- *TWIC* with *Signature Matching Distance* (SMD) is empowered by
$$\delta(q,o) \geq (\ln[\delta(o,p_1) + \delta(q,p_1)] + \ln \delta(q,p_1))^2 \cdot (\max\{\delta(o,p_1), \delta(q,p_2)\})^2$$
- *TWIC* with *Perceptually Modified Hausdorff Distance* (PMHD)
$$\begin{aligned}\delta(q,o) &\geq \ln \max\{\delta(o,p), (0.4905^{\delta(q,p)} + |\delta(o,p)|)\} \\ &= \ln[0.4905^{\delta(q,p)} + \delta(o,p)]\end{aligned}$$

For the last dataset, we provide two equivalent expressions to highlight that resulting formulas do not have to hold optimal forms. During the exploration, the engine might recombine two expressions which results in a correct mathematical expression that is however useless. Here, the value $0.4905^{\delta(q,p)}$ is always positive, so we can discard the `max` operator with no negative effect.

8.3.2 Indexing Evaluation

With the obtained results and expressions, we perform the real indexing tasks. In all cases, we average the values over 10 different kNN similarity queries ($k = 5$). With the requirement of 100% precision, we execute these queries and look for the number of distance computations required to obtain final results.

We compare the triangle lowerbounding (*Triangle*) with the proposed *Triangle*$^+$ *lowerbounding* (*Triangle+ (GP)*) which computes the additional lowerbound value for non-filtered objects based on the results from `GP-SIMDEX`.

For completeness, we compare the results with the triangle lowerbounding of TriGen FP-modifiers (*TriGenFP*) with following weights: $w_{\text{Clouds}} = -0.019235$, $w_{\text{7D}} = -0.198429$, $w_{\text{SMD}} = 0.116880$, and $w_{\text{PMHD}} = 0.033072$.

As we can see in Fig. 8.3, for most cases, *Triangle*$^+$ *(GP)* as the combination of $LB_\triangle$ with the additional (highly suitable) lowerbound LB^+ decreases the number of DCs to a considerable extent and also outperforms *TriGenFP* approach.

The overall improvements of *Triangle*$^+$ *(GP)* and *TriGenFP* methods compared to triangle lowerbounding are highlighted in Fig. 8.4. For TWIC datasets, *TriGenFP* performs poorly probably due to non-optimal weights w. If we discard the lowest value for the problematic dataset (TWIC with SMD) for which neither *Triangle* nor *Triangle+ (GP)* performs efficiently, for the remaining datasets, we get the query efficiency improvements of *Triangle*$^+$ *(GP)* that range in 14–34%.

These results highlight strong improvements and also the fact that we cannot improve the query performance for any dataset. It remains as an open question for further research to solve for which databases and similarity models this happens.

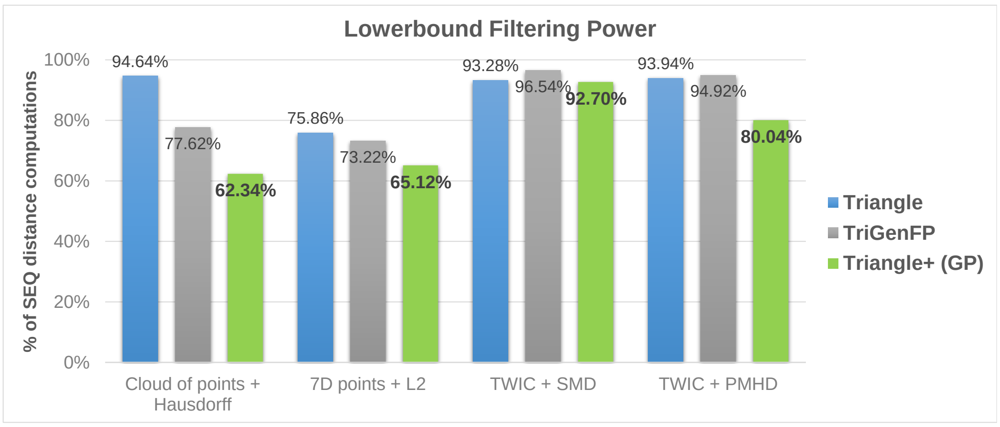


Figure 8.3: Comparing DCs of SEQ scan (always 100%), the standard $LB_{\triangle}$ (*Triangle*), the triangle lowerbound with TriGen FP-modifier $LB_{\triangle,\widehat{\delta}}$ (*TriGenFP*), and the combined $LB^{+}_{\triangle}$ resulted from `GP-SIMDEX` (*Triangle+ (GP)*)

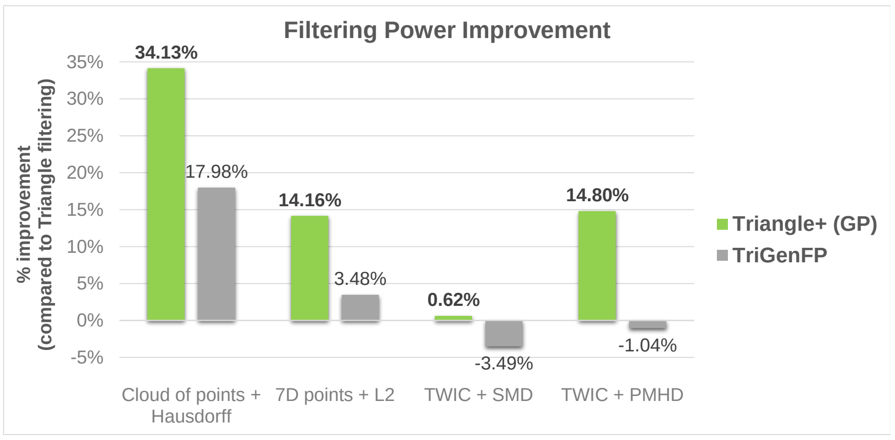


Figure 8.4: The filtering power of `GP-SIMDEX` Triangle^{+} lowerbound *Triangle+ (GP)* and the triangle lowerbound with TriGen FP-modifier (*TriGenFP*) compared to the standard $LB_{\triangle}$ (always 0%)

8.4 LowerBound Tightening

If we analyze the previous results of Triangle^{+} lowerbounding, we observe that the secondary lowerbound LB^{+} which helps the standard triangle lowerbound $LB_{\triangle}$ to filter out additional objects is very often only a modification of the original $LB_{\triangle}$. Suppose we apply the *tree edit distance* δ_{TED} (see Section 2.2.1) to lowerbounds and we get the number of operators and/or constants we need to modify in order to convert the first lowerbound into the second one. Then, we could search axiom space only for candidate lowerbounds LB for which the distance between the lowerbound LB and original $LB_{\triangle}$ is relatively small: $\delta_{TED}(LB, LB_{\triangle}) < \theta$.

Based on this observation, we propose a new technique called *LowerBound Tightening* (LBT). Its main purpose is to modify any valid lowerbound expression LB in a consistent and effective way by strengthening its filtering power with better distance estimations using *Expression Tightening Algorithm* in order to increase lowerbound's efficiency.

8.4.1 Expression Tightening Overview

The idea of *tightening* a lowerbound LB stems from mathematical properties of normalized spaces in which all items (distances in our case) are strictly limited to the interval $\langle 0, 1\rangle$ and is also motivated by the success of TriGen algorithm [8, 52] (for details see Section 3.2.5). More precisely, having a number $0 \leq x \leq 1$, we can observe two facts

$$x < x^y \qquad \forall y \in (0, 1) \tag{8.2}$$

$$x > x^z \qquad \forall z \in (1, \infty) \tag{8.3}$$

We want to leverage these properties for lowerbounding and filtering. This means that if we power the lowerbound LB to any number y from the interval $y \in (0, 1)$, we increase the lowerbound value and thus improve its filtering power. Yet, we need to ensure the correctness of the resulting lowerbound: $\delta \geq (LB)^y$.

On the other hand, if we power the lowerbound LB to the number from the interval $z \in (1, \infty)$, we decrease its value and thus decrease its filtering power. This applies to cases, in which the lowerbound LB provides incorrect values: $\delta < LB$. Here, we can select appropriate z for which $\delta \geq (LB)^z$.

If we are able to find the appropriate power value $w \in (0, \infty)$ for the lowerbound LB, we modify it and get a new lowerbound $(LB)^w$ with better filtering capabilities. If $w = 1$, we get the same lowerbound. This concept applies to any valid lowerbound expression LB, while the only requirement remains the *normalized distance space.*

8.4.2 Expression Tightening Algorithm

Based on the discussion in the previous sections, we outline the algorithm which aims at discovering the best *power value* $w \in (0, \infty)$ for the input lowerbound expression LB. It uses the principle of tuning the power value w in a similar way as TriGen algorithm (see Section 3.2.5). We start with $w = 1$ and in each step we either (a) increase its value whenever the lowerbound $(LB)^w$ is incorrect, or (b) decrease its value to provide tighter lowerbound value. To do this, we use the complementary function *ComputeError* which computes the error ratio and sets the future direction as shown in Algorithm 9.

This approach is general enough to be involved as the post-processing step in any algorithm. However, we try to find an additional lowerbound LB^+ to enrich $LB_\triangle$, so we set one extra requirement – instead of sampling random objects from the dataset, we consider only objects not filtered by $LB_\triangle$ because our goal is to enrich its filtering power (see Triangle$^+$ Preprocessing in Algorithm 8).

Difference from TriGen Algorithm

It might not be clear at the first sight but the *expression tightening algorithm* is conceptually different from the TriGen algorithm. We do not modify the input data as TriGen does and the behavior of tightening comes from the exploration phase with strictly limited axiom space inspired by SIMDEX. The tuning phase of a single parameter w is however similar for both methods and we acknowledge that our motivation for this process comes from TriGen.

Algorithm 9 Expression Tightening Algorithm (LB, NFO)

```
Require: Lowerbound expression $LB$, set of non-filtered objects $NFO$
 1: $w = w_{\texttt{best}} = 1;\ w^* = \infty;$
 2: if ComputeError($NFO, LB$) $> 0$ then
 3:    $w_{\texttt{LB}} = 1; w_{\texttt{UB}} = \infty; w = 2$  {improve – decrease lowerbound value}
 4: else
 5:    $w_{\texttt{LB}} = 0; w_{\texttt{UB}} = 1; w = 0.5$  {worsen – increase lowerbound value}
 6: end if
 7: for ($i = 0$; $i <$ MaxIterations; $i$++) do
 8:    if ComputeError($NFO, (LB)^w$) $> 0$ then
 9:       $w_{\texttt{LB}} = w;$
10:       if $w_{\texttt{UB}} = \infty$ then
11:          $w = 2 \cdot w$
12:       else
13:          $w = (w_{\texttt{LB}} + w_{\texttt{UB}})/2$
14:       end if
15:    else {best result so far}
16:       $w^* = w;\ w = (w_{\texttt{LB}} + w_{\texttt{UB}})/2$
17:    end if
18: end for
19: if $w^* \neq \infty$ then
20:    $w_{\texttt{best}} = w^*$
21: end if
22: return $(LB)^{w_{\texttt{best}}}$  {the best power modifier for the lowerbound expression}
```

8.4.3 Tightening Triangle Lowerbound

To verify proposed LBT approach empowered by *expression tightening algorithm* for real-world data indexing using Triangle$^+$ filtering, we examine its behavior on several datasets. We fix the database size to 900 objects, use 5 pivots and average the results over 10 kNN queries ($k = 5$).

For initial evaluation, we completely limit the axiom space only to relations $\delta(q,o) \geq |\delta(q,p) - \delta(p,o)|^w$. This means that the exploration phase equals to applying the expression tightening algorithm to the triangle lowerbound $LB_\triangle$.

We use same databases as for `GP-SIMDEX` experiments (see Section 8.3) and compare both approaches. Here, we obtain the following results:

- *Clouds of points* – the best power value is $w = 0.776897$ as opposed to $w_{GP} = 0.717061$ found by `GP-SIMDEX` ($\lambda = w - w_{GP} = 0.059836$).
- *7D points* – the best power value $w = 0.972691$ as opposed to $w_{GP} = 0.939630$ discovered by `GP-SIMDEX` ($\lambda = w - w_{GP} = 0.033061$).
- *TWIC* (Thematic Web Images Collection) dataset [147] with *Signature Matching Distance* (SMD) for content-based image retrieval [148] results with $w = 0.957558$.
- *TWIC* (Thematic Web Images Collection) dataset [147] with *Perceptually Modified Hausdorff Distance* (PMHD) [149] outputs the best power value $w = 0.896825$.

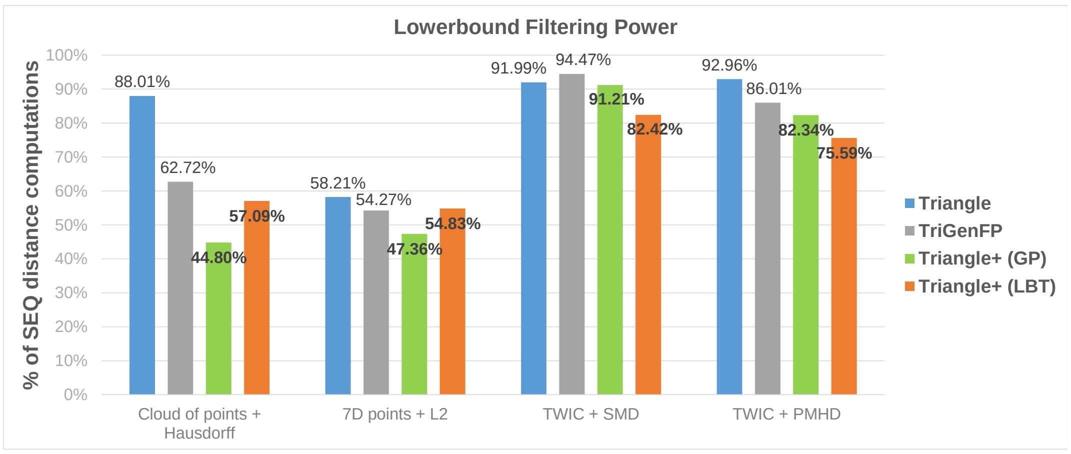


Figure 8.5: Comparing DCs of the standard $LB_\triangle$ (*Triangle*), the triangle lowerbound with TriGen FP-modifier $LB_{\triangle,\widehat{\delta}}$ (*TriGenFP*), the combined $LB_\triangle^+$ resulted from `GP-SIMDEX` (*Triangle+ (GP)*) , and the combined $LB_\triangle^+$ resulted from LBT approach (*Triangle+ (LBT)*)

In cases when $\lambda > 0$ (the difference between power values), we expect worse query performance due to the properties of normalized spaces (Section 8.4.1).

The obtained results from LBT method by applying the *expression tightening algorithm* to $LB_\triangle$ provide the basis for indexing tasks. For LBT results, the combined lowerbound $LB_\triangle^+$ always corresponds to the pair $(LB_\triangle, LB_\triangle^w)$.

In Fig. 8.5, we compare these *Triangle+ (LBT)* results together with previous results from `GP-SIMDEX` exploration phase (*Triangle+ (GP)*; see Section 8.3.2), and Trigen FP-modifiers (*TriGenFP*). For first two datasets, the performance of the best expression discovered by `GP-SIMDEX` provides higher query efficiency. The reason for this is that the increased database size brings additional objects that invalidate the previous lowerbound performance. Yet, these objects do not get to the final result set, so the incorrect distance estimations for some objects introduced by `GP-SIMDEX` lowerbound do not matter.

For the remaining two datasets, we reveal lowerbounds with higher filtering power which additionally improve the query performance. In this case, our method outperforms *TriGenFP* efficiency. This makes *Triangle+ (LBT)* a better and stronger alternative for efficient indexing and querying in (non)metric spaces.

8.4.4 Triangle Tightening Behavior Analysis

We study deeply the behavior of *Triangle*$^+$ *filtering* with the tightened triangle lowerbound $LB_\triangle^w$. For this purpose, we generate 9 synthetic datasets as random objects in N-dimensional space constrained by the hypercube in which values in each dimension are limited to the interval $\langle 0, 100\rangle$. We apply the standard L_2 distance to get the metric similarity model. We depict the general information regarding the datasets in Table 8.1 with details in Appendix A.2.

With the increasing dimensionality, the number of DCs rises for both lowerbounds $LB_\triangle$ and $LB_\triangle^w$ (see Fig. 8.6) also due to increasing IDim values (Fig A.3 or Section 3.2.3) and the *curse of dimensionality* [56, 6, 58] but it does not increase proportionally.

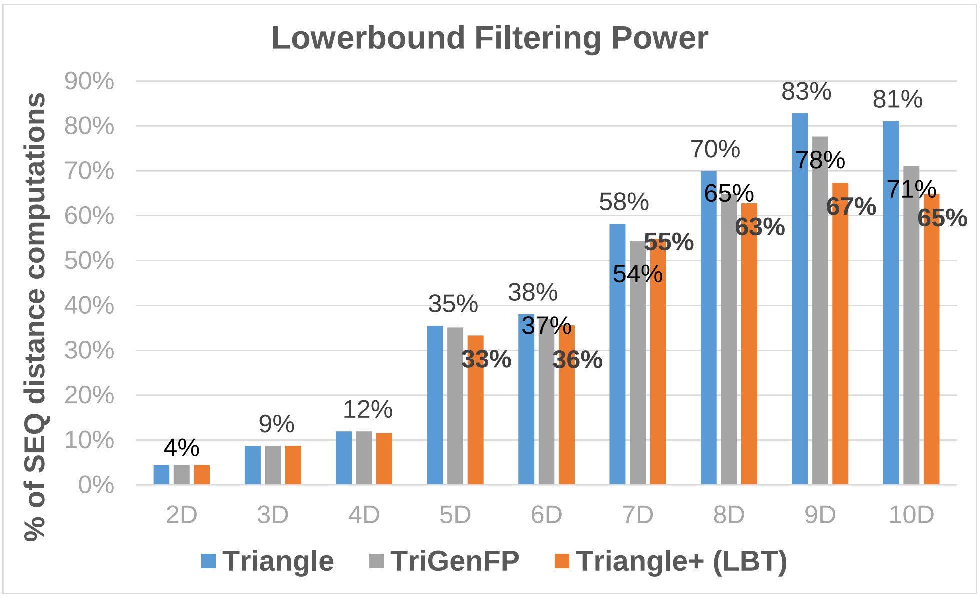


Figure 8.6: Comparing DCs of the standard $LB_{\triangle}$ (*Triangle*), the triangle lowerbound with TriGen FP-modifier $LB_{\triangle,\widehat{\delta}}$ (*TriGenFP*), and the combined $LB_{\triangle}^{+}$ resulted from LBT approach (*Triangle+ (LBT)*)

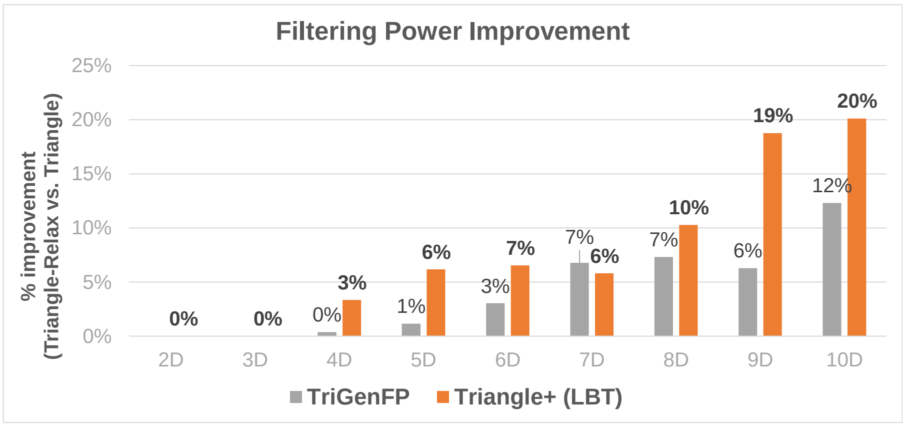


Figure 8.7: The filtering power of the combined $LB_{\triangle}^{+}$ obtained from the LBT approach (*Triangle+ (LBT)*) and the triangle lowerbound with TriGen FP-modifier (*TriGenFP*) compared to the standard $LB_{\triangle}$ (always 0%)

Starting with 3-dimensional space, *Triangle+ (LBT)* gains the first (minor) improvements of 3% which further rise with high-dimensional spaces (see Fig. 8.7) and result in 20% improvement of the query evaluation for 10-dimensional space compared to pure triangle lowerbound $LB_{\triangle}$. In this particular case, we drop the percentage of DCs from 81% to 65% (as opposed to 71% given by *TriGenFP*) simply by employing additional lowerbound $LB_{\triangle}^{w}$ for non-filtered objects.

TriGenFP approach also improves the query performance but its behavior is in average worse than *Triangle+ (LBT)*. This validates the value of *Triangle*$^{+}$ *lowerbounding* for practical indexing tasks with a potential of better performance.

Dataset	IDim	Triangle$^+$ LBT weight w	Average DCs		
			Triangle $LB_\triangle$	*TriGenFP* $LB_{\triangle,\widehat{\delta}}$	*Triangle+ (LBT)* $LB^w_\triangle$
2D	2.1887	1.000000	39.5	39.5	39.5
3D	3.4805	0.999859	78.3	78.3	78.3
4D	4.7912	0.997222	107.5	107.1	**103.9**
5D	6.1878	0.993441	319.4	315.7	**299.7**
6D	7.5951	0.993160	342.5	332.1	**320.1**
7D	8.9323	0.990897	523.9	**488.4**	493.5
8D	10.2821	0.982229	629.6	583.5	**564.9**
9D	11.5424	0.960257	745.5	698.6	**605.6**
10D	13.1199	0.953754	729.7	639.9	**582.9**

Table 8.1: Characteristics of synthetic N-dimensional datasets

8.4.5 LowerBound Tightening Summary

The results confirm the validity of *LowerBound Tightening* as a simple approach for empowering lowerbounds for better filtering and completes the overall Triangle$^+$ concept (Fig. 8.1). In the future, we want to study (1) whether there exist better candidates than $LB_\triangle$ for a general applicability of the expression tightening algorithm (see Section 8.4.2), (2) how to identify objects for which the lowerbound values are irrelevant, as they will not get to the result set, and (3) how to involve LBT technique as the post-processing step to SIMDEX to additionally increase the performance of the discovered axioms.

8.5 SIMDEX and Triangle$^+$ Synergy

The Smart Pivot Table concept with *Triangle$^+$ lowerbounding* ($LB^+_\triangle$) reveals the benefits and the applicability of SIMDEX Framework for practically oriented general-purpose indexing tasks. If we incorporate preprocessing and exploration phases directly within the Pivot Table index and build the Smart Pivot Table, the users would not recognize the difference from the classic Pivot Table. The initial cost required for the discovery of additional lowerbound will be returned by faster query evaluation in the future.

Here, we need to admit that the advantages of using multiple lowerbounds is suitable only for similarity models in which computing a single distance computation is far more expensive than evaluating multiple lowerbounds in order to get a tight lowerbound value.

In the future, we would like to research potential combinations with TriGen modifiers (see Section 3.2.5) for nonmetric spaces. Also, we would like to find out the ideal ”zero” point in which the initial costs are returned in terms of faster query evaluation.

Chapter 9

Conclusion

Our motivation in this thesis is to enable efficient indexing techniques for different communities of domain experts who employ arbitrary (generally nonmetric) similarity models. To address the challenges of existing indexing methods (Section 3), we propose the algorithmic framework SIMDEX that is capable of discovering strong alternatives to existing indexing techniques (Section 4).

Besides the general overview and framework fundamentals, we describe three existing and different variants of SIMDEX Framework – `I-SIMDEX`, `GP-SIMDEX`, and `PGP-SIMDEX` together with a novel technique of *LowerBound Tightening* that is applicable as the post-processing step to any of them.

The final outcome of our work includes the concept of *Smart Pivot Table* (Section 8) with *Triangle*$^+$ *lowerbounding* (Section 8.2). Here we apply SIMDEX Framework to practice and confirm its potential.

In the following text, we review individual variants in a consolidated manner and highlight individual strengths and weaknesses of these methods.

I-SIMDEX

`I-SIMDEX` (see Section 5) is the first and very naive implementation of SIMDEX Framework which leverages incremental and iterative exploration of the *axiom space*. Although this is quite straightforward and easy to implement, it is slow in terms of time, as it re-evaluates various forms of the same mathematical expressions. This occurs very often, results in multiple repetitive tasks with the same output, and thus degrades the overall performance.

Also, there is no feedback about the "potential" of the *candidate relations* being evaluated. Although we can estimate that the *relation* will not provide good results or does not have a strong potential, it will be still evaluated, as we cannot simply apply good heuristics. This has a negative impact on the performance. However, the iterative exploration is attractive, as it is easy to implement, deterministic, and always returns the same results for the fixed input.

GP-SIMDEX

`GP-SIMDEX` (see Section 6) follows the idea of SIMDEX Framework but mix it with the inspiring theoretical concept of genetic programming. The custom GP-based *inequality symbolic regression* algorithm reveals a great potential of intelligent axiom exploration with subsequent populations of candidate relations.

The advantage is that we explore the *axiom space* in a more consolidated way, as we take into account the feedback from previous evaluations. Also, such an intelligent discovery is intended to find more complex relations faster which makes the exploration more efficient.

However, from the nature of GP-based algorithms, also `GP-SIMDEX` suffers from the non-deterministic behavior which needs to be overcome by multiple executions (runs) of the algorithm to get acceptable and/or stable results. Additionally, the implementation of such an approach is dramatically more complex than for `I-SIMDEX`.

What remains as a minor issue here is the re-evaluation of the same mathematical expressions as there is much smaller probability that the outcome of genetic operations will contain a different form of the same (previously tested) mathematical expression and because we test the reproduced relations only once.

PGP-SIMDEX

`PGP-SIMDEX` (see Section 7) pushes the idea of GP-based axiom exploration to the next level, modifies `GP-SIMDEX` approach, and introduces the parallel processing in the distributed environment. It addresses the challenges of single-threaded `GP-SIMDEX` executions while leveraging the fact that the genetic operations in the real world apply to thousands or millions of individuals simultaneously.

For the exploration, we use the same *inequality symbolic regression* algorithm together with *island-population* model and *map-reduce* techniques in order to improve the qualitative results by a nontrivial parallel algorithm.

This method eliminates the need for multiple executions of `GP-SIMDEX` to get acceptable results, because it executes multiple instances at the same time. Moreover, the additional intelligence ensures that the best-so-far individuals are propagated between instances to boost the efficiency and overall performance. Even though the computation is distributed amongst several nodes, we have once again the non-deterministic behavior. With the increased complexity of such a solution, we also have to guarantee the synchronization of all running instances.

The drawback is the necessity of multi-threaded or multi-CPU environment and corresponding high performance in order to obtain qualitative results.

Table 9.1 compares the major pros and cons of all variants in a consolidated manner. Note that we consider individual advantages and disadvantages from the database researcher‘s point of view. Not all of these items apply to final end-user (domain researches).

Smart Pivot Table

We apply these principles to practice and propose *Smart Pivot Table* indexing scheme (Section 8) for efficient filtering. It employs data-driven discovery of indexing techniques using only the stored data. Together with SIMDEX, this results in *Triangle*$^+$ *lowerbounding* and filtering for metric spaces. This approach outperforms the state-of-the-art methods such as the triangle lowerbound $LB_\triangle$.

Moreover, we introduce *LowerBound Tightening* as the candidate for post-processing steps to existing SIMDEX variants because it further improves the filtering power of given expressions. We also analyze its superior behavior while individually tightening the triangle lowerbound $LB_\triangle$ (Section 8.4.4).

Variant	**Advantages (+)**	**Disadvantages (-)**
`I-SIMDEX`	Straightforward	Slow
	Easy to use	Multiple evaluations of same expressions[1]
	Fixed (deterministic) evaluations	No feedback
`GP-SIMDEX`	Intelligent discovery	Random and unpredictable results
	Quickly finds complex relations	Multiple executions (runs) required
	Feedback from evaluations	More complex to implement
`PGP-SIMDEX`	Intelligent discovery at high scale	Random and unpredictable results
	Huge parallelism and distributed environment	Synchronization between islands required
	Propagating best-so-far individuals	Very complex to implement

Table 9.1: Comparing SIMDEX variants

9.1 Future Work

We review several variants of SIMDEX Framework that confirm its viability and as we outline and describe, all of these approaches have positive and negative sights. What remains as a challenge for the future research is how to develop such a variant that efficiently searches the given *axiom space*, provides (sub)optimal results quickly, and is truly deterministic. Also, we need to ensure the crucial requirement that the exploration is easily adopted before / within the indexing.

One such concept is the proposed *Smart Pivot Table*, however, it is initially validated mainly for metric spaces. The tasks of extensively studying its behavior on nonmetric similarity models and of combining triangle lowerbound modified by appropriate FP-modifier ($LB_{\triangle,\widehat{\delta}}$) with an additional lowerbound LB^{+} remain as the future work.

[1]Observe, that we put the *Multiple evaluations of same expressions* only to `I-SIMDEX` variant, although it generally applies to all variants. The reason is that while building the candidate relations (and thus inequality expressions) iteratively, this occurs far more frequently than in randomized SIMDEX versions (`GP-SIMDEX` or `PGP-SIMDEX`).

Bibliography

[1] C. Faloutsos, R. Barber, M. Flickner, J. Hafner, W. Niblack, D. Petkovic, and W. Equitz. Efficient and Effective Querying by Image Content. *J. Intell. Inf. Syst.*, 3:231–262, July 1994.

[2] Wayne Niblack, Ron Barber, William Equitz, Myron Flickner, Eduardo H. Glasman, Dragutin Petkovic, Peter Yanker, Christos Faloutsos, and Gabriel Taubin. The QBIC Project: Querying Images by Content, Using Color, Texture, and Shape. In *Storage and Retrieval for Image and Video Databases (SPIE)'93*, pages 173–187, 1993.

[3] Pavel Zezula, Giuseppe Amato, Vlastislav Dohnal, and Michal Batko. *Similarity Search: The Metric Space Approach.* Advances in Database Systems. Springer-Verlag, USA, 2005.

[4] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. In *Proc. of the 6th conf. on Symp. on Oper. Systems Design & Impl.*, pages 10–10, USA, 2004. USENIX Association.

[5] K. Shvachko, Hairong Kuang, S. Radia, and R. Chansler. The Hadoop Distributed File System. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, pages 1–10, May 2010.

[6] Edgar Chávez, Gonzalo Navarro, Ricardo Baeza-Yates, and José Luis Marroquín. Searching in metric spaces. *ACM Comp. Surveys*, 33(3):273–321, 2001.

[7] Hanan Samet. *Foundations of Multidimensional and Metric Data Structures.* Morgan Kaufmann Publishers Inc., USA, 2005.

[8] Tomáš Skopal. Unified Framework for Fast Exact and Approximate Search in Dissimilarity Spaces. *ACM Transactions on Database Systems*, 32(4):1–46, 2007.

[9] Benjamin Bustos and Tomáš Skopal. Non-metric similarity search problems in very large collections. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on*, pages 1362–1365, 2011.

[10] Tomáš Skopal and Benjamin Bustos. On nonmetric similarity search problems in complex domains. *ACM Comp. Surv.*, 43:1–50, 2011.

[11] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197, March 1981.

[12] Jakub Galgonek, David Hoksza, and Tomáš Skopal. SProt: sphere-based protein structure similarity algorithm. *Proteome Science*, 9:1–12, 2011.

[13] Phillipe Salembier and Thomas Sikora. *Introduction to MPEG-7: Multimedia Content Description Interface.* John Wiley & Sons, Inc., New York, NY, USA, 2002.

[14] Christian Beecks, Merih Seran Uysal, and Thomas Seidl. Signature Quadratic Form Distance. In *Proceedings of the ACM International Conference on Image and Video Retrieval*, CIVR '10, pages 438–445, New York, NY, USA, 2010. ACM.

[15] Jakub Lokoč, Magnus Lie Hetland, Tomáš Skopal, and Christian Beecks. Ptolemaic Indexing of the Signature Quadratic Form Distance. In *Proceedings of the Fourth International Conference on SImilarity Search and APplications*, SISAP '11, pages 9–16, New York, NY, USA, 2011. ACM.

[16] Martin Kruliš, Jakub Lokoč, and Tomáš Skopal. Efficient Extraction of Feature Signatures Using Multi-GPU Architecture. In Shipeng Li, Abdulmotaleb Saddik, Meng Wang, Tao Mei, Nicu Sebe, Shuicheng Yan, Richang Hong, and Cathal Gurrin, editors, *Advances in Multimedia Modeling*, volume 7733 of *Lecture Notes in Computer Science*, pages 446–456. Springer Berlin Heidelberg, 2013.

[17] Elena Deza and Michel Deza. *Dictionary of distances.* North-Holland, 2006.

[18] Alan Eckhardt, Tomáš Skopal, and Peter Vojtáš. On Fuzzy vs. Metric Similarity Search in Complex Databases. In *FQAS '09: Proceedings of the 8th International Conference on Flexible Query Answering Systems*, pages 64–75, Berlin, Heidelberg, 2009. Springer-Verlag.

[19] Tomáš Bartoš, Alan Eckhardt, and Tomáš Skopal. Fuzzy Approach to Non-metric Similarity Indexing. In *Proceedings of the Fourth International Conference on SImilarity Search and APplications*, SISAP '11, pages 115–116, New York, NY, USA, 2011. ACM.

[20] Tomáš Skopal, Tomáš Bartoš, and Jakub Lokoč. On (Not) Indexing Quadratic Form Distance by Metric Access Methods. In *Proceedings of the 14th International Conference on Extending Database Technology*, EDBT/ICDT '11, pages 249–258, New York, NY, USA, 2011. ACM.

[21] M Ioka. A Method of Defining the Similarity of Images on the Basis of Color Information. Technical Report Tech. Report RT-0030, IBM Tokyo Research Lab, 1989.

[22] James Hafner, Harpreet S. Sawhney, Will Equitz, Myron Flickner, and Wayne Niblack. Efficient Color Histogram Indexing for Quadratic Form Distance Functions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17:729–736, 1995.

[23] Mihael Ankerst, Bernhard Braunmüller, Hans-Peter Kriegel, and Thomas Seidl. Improving Adaptable Similarity Query Processing by Using Approximations. In *Proc. 24th int. conf. on Very large data bases (VLDB)*, pages 206–217. Morgan Kaufmann, 1998.

[24] Mihael Ankerst, Gabi Kastenmüller, Hans-Peter Kriegel, and Thomas Seidl. 3D Shape Histograms for Similarity Search and Classification in Spatial Databases. In *Advances in Spatial Databases*, volume 1651 of *Lecture Notes in Computer Science*, pages 207–226. Springer Berlin / Heidelberg, 1999.

[25] Mihael Ankerst, Gabi Kastenmüller, Hans-Peter Kriegel, and Thomas Seidl. Nearest Neighbor Classification in 3D Protein Databases. In *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology*, pages 34–43. AAAI Press, 1999.

[26] Thomas Fober and Eyke Hüllermeier. Similarity Measures for Protein Structures based on Fuzzy Histogram Comparison. In *IEEE World Congress on Computational Intelligence*. IEEE, 2010.

[27] Thomas Fober, Marco Mernberger, Gerhard Klebe, and Eyke Hüllermeier. Efficient Similarity Retrieval for Protein Binding Sites based on Histogram Comparison. In *German Conference on Bioinformatics*, 2010.

[28] T. Bernas, E.K. Asem, J.P. Robinson, and B. Rajwa. Quadratic form: a robust metric for quantitative comparison of flow cytometric histograms. *Cytometry, Part A.*, 73(8):715–726, 2008.

[29] Yoshiharu Ishikawa, Ravishankar Subramanya, and Christos Faloutsos. MindReader: Querying Databases Through Multiple Examples. In *Proceedings of the 24th International Conference on Very Large Data Bases*, VLDB '98, pages 218–227, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[30] Thomas Seidl and Hans-Peter Kriegel. Efficient User-Adaptable Similarity Search in Large Multimedia Databases. In *Proceedings of the 23rd International Conference on Very Large Data Bases*, VLDB '97, pages 506–515, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[31] Christian Beecks, Merih Seran Uysal, and Thomas Seidl. Signature quadratic form distances for content-based similarity. In *Proceedings of the seventeen ACM international conference on Multimedia*, MM '09, pages 697–700, New York, NY, USA, 2009. ACM.

[32] Robert A. Wagner and Michael J. Fischer. The String-to-String Correction Problem. *J. ACM*, 21(1):168–173, January 1974.

[33] David Sankoff and Joseph B. Kruskal. *Time warps, string edits, and macromolecules*. Cambridge University Press, Cambridge, England, 2000.

[34] Sudipto Guha, H. V. Jagadish, Nick Koudas, Divesh Srivastava, and Ting Yu. Approximate XML Joins. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, SIGMOD '02, pages 287–298, New York, NY, USA, 2002. ACM.

[35] G. Cobena, S. Abiteboul, and A. Marian. Detecting changes in XML documents. In *Data Engineering, 2002. Proceedings. 18th International Conference on*, pages 41–52, 2002.

[36] Sudipto Guha, H. V. Jagadish, Nick Koudas, Divesh Srivastava, and Ting Yu. Integrating XML Data Sources Using Approximate Joins. *ACM Trans. Database Syst.*, 31(1):161–207, March 2006.

[37] Jiří Novák, Tomáš Skopal, David Hoksza, and Jakub Lokoç. Non-metric similarity search of tandem mass spectra including posttranslational modifications. *Journal of Discrete Algorithms*, 13(0):19 – 31, 2012. ¡ce:title¿Best Papers from the 3rd International Conference on Similarity Search and Applications (SISAP 2010)¡/ce:title¿.

[38] Peter Howarth and Stefan Rüger. Fractional distance measures for content-based image retrieval. In *In 27th European Conference on Information Retrieval*, pages 447–456. Springer, 2005.

[39] Ricardo A. Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[40] Yossi Rubner, Jan Puzicha, Carlo Tomasi, and Joachim M. Buhmann. Empirical Evaluation of Dissimilarity Measures for Color and Texture. *Comput. Vis. Image Underst.*, 84(1):25–43, October 2001.

[41] Donald J. Berndt and James Clifford. Using Dynamic Time Warping to Find Patterns in Time Series. In Usama M. Fayyad and Ramasamy Uthurusamy, editors, *KDD Workshop*, pages 359–370. AAAI Press, 1994.

[42] Eamonn Keogh and Chotirat Ann Ratanamahatana. Exact Indexing of Dynamic Time Warping. *Knowl. Inf. Syst.*, 7(3):358–386, March 2005.

[43] Tomáš Bartoš and Tomáš Skopal. Revisiting Techniques for Lowerbounding the Dynamic Time Warping Distance. In Gonzalo Navarro and Vladimir Pestov, editors, *Similarity Search and Applications*, volume 7404 of *Lecture Notes in Computer Science*, pages 192–208. Springer Berlin Heidelberg, 2012.

[44] Hiroaki Sakoe. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics,Speech,and Signal Processing*, 26:43–49, 1978.

[45] Lawrence Rabiner and Biing-Hwang Juang. *Fundamentals of speech recognition.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

[46] Eamonn Keogh. Exact indexing of dynamic time warping. In *Proceedings of the 28th international conference on Very Large Data Bases*, VLDB '02, pages 406–417. VLDB Endowment, 2002.

[47] Byoung-Kee Yi, H. V. Jagadish, and Christos Faloutsos. Efficient Retrieval of Similar Time Sequences Under Time Warping. In *Proceedings of the*

Fourteenth International Conference on Data Engineering, ICDE '98, pages 201–208, Washington, DC, USA, 1998. IEEE Computer Society.

[48] Gonzalo Navarro. Analyzing Metric Space Indexes: What For? In *Proceedings of the 2009 Second International Workshop on Similarity Search and Applications*, SISAP '09, pages 3–10, Washington, DC, USA, 2009. IEEE Computer Society.

[49] Jeffrey K. Uhlmann. Satisfying general proximity / similarity queries with metric trees. *Information Processing Letters*, 40(4):175 – 179, 1991.

[50] Magnus Lie Hetland. Ptolemaic Indexing. `arXiv:0911.4384 [cs.DS]`, 2009.

[51] Magnus Lie Hetland, Tomáš Skopal, Jakub Lokoč, and Christian Beecks. Ptolemaic access methods: Challenging the reign of the metric space model. *Information Systems*, 38(7):989 – 1006, 2013.

[52] Tomáš Skopal. On Fast Non-metric Similarity Search by Metric Access Methods. In *Proc. 10th International Conference on Extending Database Technology (EDBT'06)*, LNCS 3896, pages 718–736. Springer, 2006.

[53] Christian Böhm, Stefan Berchtold, and D Keim. Searching in High-Dimensional Spaces – Index Structures for Improving the Performance of Multimedia Databases. *ACM Computing Surveys*, 33(3):322–373, 2001.

[54] Antonin Guttman. R-trees: A Dynamic Index Structure for Spatial Searching. *SIGMOD Rec.*, 14(2):47–57, June 1984.

[55] Stefan Berchtold, Daniel A. Keim, and Hans-Peter Kriegel. The X-tree: An Index Structure for High-Dimensional Data. In *Proceedings of the 22th International Conference on Very Large Data Bases*, VLDB '96, pages 28–39, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc.

[56] Roger Weber, Hans-Jörg Schek, and Stephen Blott. A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. In *Proceedings of the 24rd International Conference on Very Large Data Bases*, VLDB '98, pages 194–205, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[57] R. Bayer and E. McCreight. Organization and Maintenance of Large Ordered Indices. In *Proceedings of the 1970 ACM SIGFIDET (Now SIGMOD) Workshop on Data Description, Access and Control*, SIGFIDET '70, pages 107–141, New York, NY, USA, 1970. ACM.

[58] Richard Ernest Bellman. *Dynamic Programming.* Dover Publications, Incorporated, 2003.

[59] Kevin S. Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When Is "Nearest Neighbor" Meaningful? In *ICDT '99: Proceedings of the 7th International Conference on Database Theory*, pages 217–235, London, UK, 1999. Springer-Verlag.

[60] CharuC. Aggarwal, Alexander Hinneburg, and DanielA. Keim. On the Surprising Behavior of Distance Metrics in High Dimensional Space. In Jan Bussche and Victor Vianu, editors, *Database Theory — ICDT 2001*, volume 1973 of *Lecture Notes in Computer Science*, pages 420–434. Springer Berlin Heidelberg, 2001.

[61] Paolo Ciaccia, Marco Patella, and Pavel Zezula. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *Proceedings of the 23rd International Conference on Very Large Data Bases*, VLDB '97, pages 426–435, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[62] David Novak, Michal Batko, and Pavel Zezula. Metric Index: An Efficient and Scalable Solution for Precise and Approximate Similarity Search. *Information Systems*, 36(4):721–733, June 2011.

[63] Peter N. Yianilos. Data Structures and Algorithms for Nearest Neighbor Search in General Metric Spaces. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '93, pages 311–321, Philadelphia, PA, USA, 1993. Society for Industrial and Applied Mathematics.

[64] E V Ruiz. An Algorithm for Finding Nearest Neighbours in (Approximately) Constant Average Time. *Pattern Recogn. Lett.*, 4(3):145–157, July 1986.

[65] María Luisa Micó, José Oncina, and Enrique Vidal. A New Version of the Nearest-neighbour Approximating and Eliminating Search Algorithm (AESA) with Linear Preprocessing Time and Memory Requirements. *Pattern Recogn. Lett.*, 15(1):9–17, January 1994.

[66] Sergey Brin. Near Neighbor Search in Large Metric Spaces. In *Proceedings of the 21th International Conference on Very Large Data Bases*, VLDB '95, pages 574–584, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[67] Gonzalo Navarro. Searching in Metric Spaces by Spatial Approximation. *The VLDB Journal*, 11(1):28–46, August 2002.

[68] Benjamin Bustos, Gonzalo Navarro, and Edgar Chávez. Pivot selection techniques for proximity searching in metric spaces. *Pattern Recognition Letters*, 24(14):2357 – 2366, 2003.

[69] B. Bustos, G. Navarro, and E. Chavez. Pivot selection techniques for proximity searching in metric spaces. In *Computer Science Society, 2001. SCCC '01. Proceedings. XXI Internatinal Conference of the Chilean*, pages 33–40, 2001.

[70] R. Bayer and E.M. McCreight. Organization and maintenance of large ordered indexes. *Acta Informatica*, 1(3):173–189, 1972.

[71] Paolo Ciaccia and Marco Patella. The M2-tree: Processing Complex Multi-Feature Queries with Just One Index. In *DELOS Workshop: Information Seeking, Searching and Querying in Digital Libraries*, 2000.

[72] Xiangmin Zhou, Guoren Wang, Jeffrey Xu Yu, and Ge Yu. M+-tree: A New Dynamical Multidimensional Index for Metric Spaces. In *Proceedings of the 14th Australasian Database Conference - Volume 17*, ADC '03, pages 161–168, Darlinghurst, Australia, Australia, 2003. Australian Computer Society, Inc.

[73] Benjamin Bustos and Tomáš Skopal. Dynamic Similarity Search in Multi-metric Spaces. In *Proceedings of the 8th ACM International Workshop on Multimedia Information Retrieval*, MIR '06, pages 137–146, New York, NY, USA, 2006. ACM.

[74] Tomáš Skopal, Pokorný Jaroslav, and Václav Snášel. Nearest Neighbours Search Using the PM-Tree. In Lizhu Zhou, BengChin Ooi, and Xiaofeng Meng, editors, *Database Systems for Advanced Applications*, volume 3453 of *Lecture Notes in Computer Science*, pages 803–815. Springer Berlin Heidelberg, 2005.

[75] Tomáš Skopal and Jakub Lokoč. NM-Tree: Flexible Approximate Similarity Search in Metric and Non-metric Spaces. In *Proceedings of the 19th International Conference on Database and Expert Systems Applications*, DEXA '08, pages 312–325, Berlin, Heidelberg, 2008. Springer-Verlag.

[76] Christos Faloutsos and King-Ip Lin. FastMap: A Fast Algorithm for Indexing, Data-mining and Visualization of Traditional and Multimedia Datasets. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, SIGMOD '95, pages 163–174, New York, NY, USA, 1995. ACM.

[77] Vassilis Athitsos, Jonathan Alon, Stan Sclaroff, and George Kollios. BoostMap: An Embedding Method for Efficient Nearest Neighbor Retrieval. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30(1):89–104, January 2008.

[78] J. T.L. Wang, Xiong Wang, D. Shasha, and Kaizhong Zhang. MetricMap: An Embedding Technique for Processing Distance-based Queries in Metric Spaces. *Trans. Sys. Man Cyber. Part B*, 35(5):973–987, October 2005.

[79] J. Bourgain. On lipschitz embedding of finite metric spaces in Hilbert space. *Israel Journal of Mathematics*, 52(1):46–52, March 1985.

[80] Gísli R. Hjaltason and Hanan Samet. Properties of Embedding Methods for Similarity Searching in Metric Spaces. *IEEE Trans. Pattern Anal. Mach. Intell.*, 25(5):530–549, May 2003.

[81] Gene H. Golub and Charles F. Van Loan. *Matrix computations (3rd ed.)*. Johns Hopkins University Press, Baltimore, MD, USA, 1996.

[82] David Novak and Michal Batko. Metric Index: An Efficient and Scalable Solution for Similarity Search. In *Proceedings of the 2009 Second International Workshop on Similarity Search and Applications*, SISAP '09, pages 65–73, Washington, DC, USA, 2009. IEEE Computer Society.

[83] H. V. Jagadish, Beng Chin Ooi, Kian-Lee Tan, Cui Yu, and Rui Zhang. iDistance: An Adaptive B+-tree Based Indexing Method for Nearest Neighbor Search. *ACM Trans. Database Syst.*, 30(2):364–397, June 2005.

[84] Matthew Skala. Counting Distance Permutations. *J. of Discrete Algorithms*, 7(1):49–61, March 2009.

[85] Andrea Esuli. MiPai: Using the PP-Index to Build an Efficient and Scalable Similarity Search System. In *Proceedings of the 2009 Second International Workshop on Similarity Search and Applications*, SISAP '09, pages 146–148, Washington, DC, USA, 2009. IEEE Computer Society.

[86] Andrea Esuli. Use of permutation prefixes for efficient and scalable approximate similarity search. *Information Processing & Management*, 48(5):889 – 902, 2012. ¡ce:title¿Large-Scale and Distributed Systems for Information Retrieval¡/ce:title¿.

[87] Kurt Mehlhorn. *Data Structures and Algorithms 2: Graph Algorithms and NP-Completeness*, volume 2 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1984.

[88] Edgar Chavez Gonzalez, Karina Figueroa, and Gonzalo Navarro. Effective Proximity Retrieval by Ordering Permutations. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30(9):1647–1658, September 2008.

[89] Tomás Skopal. Pivoting M-tree: A Metric Access Method for Efficient Similarity Search. In *DATESO*, pages 27–37. CEUR-WS.org, 2004.

[90] Tomás Skopal, Jaroslav Pokorný, and Václav Snásel. PM-tree: Pivoting Metric Tree for Similarity Search in Multimedia Databases. In *ADBIS (Local Proceedings)*, 2004.

[91] Peter Vojtáš and Alan Eckhardt. Using tuneable fuzzy similarity in non-metric search. In Tomáš Skopal and Pavel Zezula, editors, *SISAP 2009: Second international workshop on similarity search and applications*, pages 163–164, Prague, Czech Republic, 2009. IEEE.

[92] Hans W. Guesgen. Reasoning About Distance Based on Fuzzy Sets. *Applied Intelligence*, 17:265–270, September 2002.

[93] P. Bosc and M. Galibourg. Indexing principles for a fuzzy data base. *Information Systems*, 14(6):493 – 499, 1989.

[94] P. Bosc, M. Galibourg, and G. Hamon. Fuzzy querying with SQL: Extensions and implementation aspects. *Fuzzy Sets and Systems*, 28(3):333 – 349, 1988.

[95] Erich Peter Klement, Radko Mesiar, and Endre Pap. Kluwer Academic Publisher, Dordrecht, 2000.

[96] Paolo Bolettieri, Andrea Esuli, Fabrizio Falchi, Claudio Lucchese, Raffaele Perego, Tommaso Piccioli, and Fausto Rabitti. CoPhIR: a Test Collection for Content-Based Image Retrieval. *CoRR*, abs/0905.4627v2, 2009.

[97] Li Junkui and Wang Yuanzhen. Early abandon to accelerate exact dynamic time warping. *Int. Arab J. Inf. Technol.*, 6(2):144–152, 2009.

[98] Sang-Wook Kim, Sanghyun Park, and Wesley W. Chu. An Index-Based Approach for Similarity Search Supporting Time Warping in Large Sequence Databases. In *Proceedings of the 17th International Conference on Data Engineering*, pages 607–614, Washington, DC, USA, 2001. IEEE Computer Society.

[99] Jin Shieh and Eamonn J. Keogh. *i*SAX: disk-aware mining and indexing of massive time series datasets. *Data Min. Knowl. Discov.*, 19(1):24–57, 2009.

[100] Alessandro Camerra, Themis Palpanas, Jin Shieh, and Eamonn Keogh. iSAX 2.0:Indexing and Mining One Billion Time Series. *IEEE International Conference on Data Mining*, 0:58–67, 2010.

[101] Eamonn Keogh, Xiaopeng Xi, Li Wei, and Chotirat (. Ratanamahatana. The UCR Time Series Classification/Clustering Homepage, 2006.

[102] Tomáš Skopal and Benjamin Bustos. On Index-Free Similarity Search in Metric Spaces. In Sourav S. Bhowmick, Josef Küng, and Roland Wagner, editors, *Database and Expert Systems Applications (DEXA)*, volume 5690 of *Lecture Notes in Computer Science*, pages 516–531. Springer Berlin Heidelberg, 2009.

[103] Tomás Skopal, Jakub Lokoc, and Benjamin Bustos. D-Cache: Universal Distance Cache for Metric Access Methods. *IEEE Trans. Knowl. Data Eng.*, 24(5):868–881, 2012.

[104] Pavel Zezula, Pasquale Savino, Giuseppe Amato, and Fausto Rabitti. Approximate Similarity Retrieval with M-trees. *The VLDB Journal*, 7(4):275–293, December 1998.

[105] Chen Li, Edward Chang, Hector Garcia-Molina, and Gio Wiederhold. Clustering for Approximate Similarity Search in High-Dimensional Spaces. *IEEE Trans. on Knowl. and Data Eng.*, 14(4):792–808, July 2002.

[106] Ertem Tuncel, Hakan Ferhatosmanoglu, and Kenneth Rose. VQ-index: An Index Structure for Similarity Searching in Multimedia Databases. In *Proceedings of the Tenth ACM International Conference on Multimedia*, MULTIMEDIA '02, pages 543–552, New York, NY, USA, 2002. ACM.

[107] Edgar Chávez and Gonzalo Navarro. A Probabilistic Spell for the Curse of Dimensionality. In *Revised Papers from the Third International Workshop on Algorithm Engineering and Experimentation*, ALENEX '01, pages 147–160, London, UK, UK, 2001. Springer-Verlag.

[108] Tomáš Skopal and Tomáš Bartoš. Algorithmic Exploration of Axiom Spaces for Efficient Similarity Search at Large Scale. In *SISAP 2012*, Lecture Notes in Computer Science, 7404, pages 40–53. Springer, 2012.

[109] Tomáš Bartoš, Tomáš Skopal, and Juraj Moško. Efficient Indexing of Similarity Models with Inequality Symbolic Regression. In *Proceeding of the Fifteenth Annual Conference on Genetic and Evolutionary Computation Conference*, GECCO '13, pages 901–908, New York, NY, USA, 2013. ACM.

[110] Tomáš Bartoš and Tomáš Skopal. Designing Similarity Indexes with Parallel Genetic Programming. In Nieves R. Brisaboa, Oscar Pedreira, and Pavel Zezula, editors, *SISAP*, volume 8199 of *Similarity Search and Applications - 6th International Conference, SISAP 2013, Lecture Notes in Computer Science*, pages 294–299. Springer, 2013.

[111] Noam and Chomsky. On certain formal properties of grammars. *Information and Control*, 2(2):137 – 167, 1959.

[112] Ralf Lämmel and Wolfram Schulte. Controllable combinatorial coverage in grammar-based testing. In *Proceedings of the 18th IFIP TC6/WG6.1 international conference on Testing of Communicating Systems*, TestCom'06, pages 19–38, Berlin, Heidelberg, 2006. Springer-Verlag.

[113] Roman Barták. Constraint Models for Reasoning on Unification in Inductive Logic Programming. In *Artificial Intelligence: Methodology, Systems, and Applications*, volume 6304 of *Lecture Notes in Computer Science*, pages 101–110. Springer, 2010.

[114] Seth Hettich and Stephen D. Bay. The UCI KDD Archive. http://kdd.ics.uci.edu, 1999.

[115] SISAP Metric Library. Listeria.

[116] Tomáš Bartoš, Tomáš Skopal, and Juraj Moško. Towards Efficient Indexing of Arbitrary Similarity: Vision Paper. *SIGMOD Record*, 42(2):5–10, July 2013.

[117] John R. Koza. *Genetic programming.* MIT Press, Cambridge, MA, USA, 1992.

[118] John R. Koza and Riccardo Poli. Genetic Programming. In Edmund K. Burke and Graham Kendall, editors, *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, chapter 5, pages 127–164. Springer, 2005.

[119] J.W.Davidson, D.A.Savic, and G.A.Walters. Symbolic and numerical regression: experiments and applications. *Information Sciences*, 150(1-2):95 – 117, 2003.

[120] Michael Schmidt and Hod Lipson. Distilling Free-Form Natural Laws from Experimental Data. *Science*, 324(5923):81–85, April 2009.

[121] J. H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, MI, USA, 1975.

[122] Nichael Lynn Cramer. A Representation for the Adaptive Generation of Simple Sequential Programs. In *Proc. of the 1st Int. Conf. on Genetic Algorithms*, pages 183–187, Hillsdale, NJ, USA, 1985. L. Erlbaum Associates Inc., USA.

[123] Darrell Whitley. A genetic algorithm tutorial. *Statistics and computing*, 4(2):65–85, 1994.

[124] David E. Goldberg and Kalyanmoy Deb. A Comparative Analysis of Selection Schemes Used in Genetic Algorithms. In Gregory J. E. Rawlins, editor, *FOGA*, pages 69–93. Morgan Kaufmann, 1990.

[125] David E. Goldberg. *Genetic Algorithms in Search Optimization and Machine Learning.* Addison-Wesley, 1989.

[126] Riccardo Poli, William B. Langdon, Nicholas F. Mcphee, and John R. Koza. Genetic programming an introductory tutorial and a survey of techniques and applications. Technical report, 2007.

[127] John R. Koza. Evolving the Architecture of a Multi-Part Program in Genetic Programming Using Architecture-Altering Operations. In *Evolutionary Programming IV: Proceedings of the Fourth Annual Conference on Evolutionary Programming*, pages 695–717. The MIT Press, 1995.

[128] John R. Koza, John R. Koza, David Andre, and David Andre. Classifying protein segments as transmembrane domains using architecture-altering operations in genetic programming. In *Advances in Genetic Programming.* MIT Press, 1996.

[129] MichaelF. Korns. Abstract Expression Grammar Symbolic Regression. In Rick Riolo, Trent McConaghy, and Ekaterina Vladislavleva, editors, *Genetic Programming Theory and Practice VIII*, volume 8 of *Genetic and Evolutionary Computation*, pages 109–128. Springer New York, 2011.

[130] Minkyu Kim, YingL. Becker, Peng Fei, and Una-May O'Reilly. ConstrainedGenetic Programming toMinimizeOverfitting in StockSelection. In *Genetic Programming Theory and Practice VI*, Genetic and Evolutionary Computation, pages 1–16. Springer US, 2009.

[131] Michael D. Schmidt and Hod Lipson. Co-evolving Fitness Predictors for Accelerating and Reducing Evaluations. In Rick L. Riolo, Terence Soule, and Bill Worzel, editors, *Genetic Programming Theory and Practice IV*, volume 5 of *Genetic and Evolutionary Computation*, pages 113–130. Springer, Ann Arbor, 11-13 May 2006.

[132] Flor Castillo, Arthur Kordon, and Carlos Villa. Genetic Programming Transforms in Linear Regression Situations. In Rick Riolo, Trent McConaghy, and Ekaterina Vladislavleva, editors, *Genetic Programming Theory and Practice VIII*, volume 8 of *Genetic and Evolutionary Computation*, pages 175–194. Springer New York, 2011.

[133] Guido F. Smits, Ekaterina Vladislavleva, and Mark E. Kotanchek. Scalable Symbolic Regression by Continuous Evolution with Very Small Populations. In Rick Riolo, Trent McConaghy, and Ekaterina Vladislavleva, editors, *Genetic Programming Theory and Practice VIII*, volume 8 of *Genetic and Evolutionary Computation*, pages 147–160. Springer New York, 2011.

[134] T. McConaghy and G. G E Gielen. Template-Free Symbolic Performance Modeling of Analog Circuits via Canonical-Form Functions and Genetic Programming. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 28(8):1162–1175, 2009.

[135] Trent McConaghy. FFX: Fast, Scalable, Deterministic Symbolic Regression Technology. In Rick Riolo, Ekaterina Vladislavleva, and Jason H. Moore, editors, *Genetic Programming Theory and Practice IX*, Genetic and Evolutionary Computation, pages 235–260. Springer New York, 2011.

[136] Tony Worm and Kenneth Chiu. Prioritized Grammar Enumeration: Symbolic Regression by Dynamic Programming. In *Proceeding of the Fifteenth Annual Conference on Genetic and Evolutionary Computation Conference*, GECCO '13, pages 1021–1028, New York, NY, USA, 2013. ACM.

[137] Michael Schmidt and Hod Lipson. Symbolic Regression of Implicit Equations. In Rick Riolo, Una-May O'Reilly, and Trent McConaghy, editors, *Genetic Programming Theory and Practice VII*, Genetic and Evolutionary Computation, pages 73–85. Springer US, 2010.

[138] Francisco Fernandez, Giandomenico Spezzano, Marco Tomassini, and Leonardo Vanneschi. Parallel Genetic Programming. In Enrique Alba, editor, *Parallel Metaheuristics*, Parallel and Distributed Computing, chapter 6, pages 127–153. Wiley-Interscience, Hoboken, New Jersey, USA, 2005.

[139] Christian Gagné, Marc Parizeau, and Marc Dubreuil. A Robust Master-Slave Distribution Architecture for Evolutionary Computations. In *Late Breaking Papers, Genetic and Evolutionary Computation COnference (GECCO) 2003*, pages 80–87, July 12-16 2003.

[140] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.

[141] Reiko Tanese. Distributed Genetic Algorithms. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 434–439, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[142] Theodore C. Belding. The Distributed Genetic Algorithm Revisited, 1995.

[143] Enrique Alba and José M. Troya. A Survey of Parallel Distributed Genetic Algorithms. *Complex.*, 4(4):31–52, March 1999.

[144] Heinz Mühlenbein. Evolution in time and space - the parallel genetic algorithm. In *FOUNDATIONS OF GENETIC ALGORITHMS*, pages 316–337. Morgan Kaufmann, 1991.

[145] Darrell Whitley, Soraya Rana, and Robert B. Heckendorn. The Island Model Genetic Algorithm: On Separability, Population Size and Convergence. *Journal of Computing and Information Technology*, 7:33–47, 1998.

[146] George Marsaglia. Seeds for Random Number Generators. *Commun. ACM*, 46(5):90–93, May 2003.

[147] Jakub Lokoč, David Novák, Michal Batko, and Tomáš Skopal. Visual Image Search: Feature Signatures or/and Global Descriptors. In Gonzalo Navarro and Vladimir Pestov, editors, *Similarity Search and Applications*, volume 7404 of *Lecture Notes in Computer Science*, pages 177–191. Springer Berlin Heidelberg, 2012.

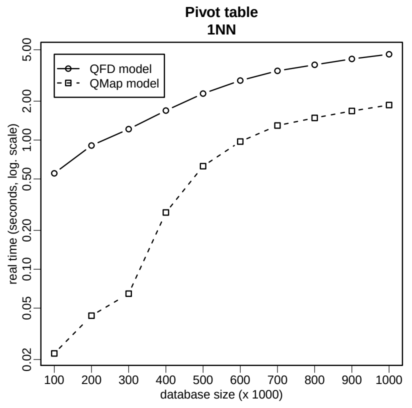
[148] Christian Beecks, Steffen Kirchhoff, and Thomas Seidl. Signature Matching Distance for Content-based Image Retrieval. In *Proceedings of the 3rd ACM Conference on International Conference on Multimedia Retrieval*, ICMR '13, pages 41–48, New York, NY, USA, 2013. ACM.

[149] BoGun Park, KyoungMu Lee, and SangUk Lee. A New Similarity Measure for Random Signatures: Perceptually Modified Hausdorff Distance. In Jacques Blanc-Talon, Wilfried Philips, Dan Popescu, and Paul Scheunders, editors, *Advanced Concepts for Intelligent Vision Systems*, volume 4179 of *Lecture Notes in Computer Science*, pages 990–1001. Springer Berlin Heidelberg, 2006.

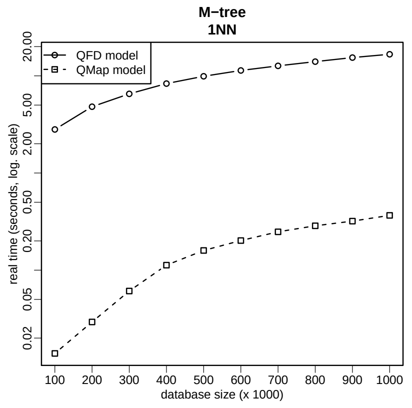
Appendix

A.1 QMap Evaluation

Figures A.1 and A.2 show the comparison of QFD and QMap models while evaluating 1NN similarity queries using Pivot table and M-tree index on real-world data[2]. For more details, we refer readers to the original paper [20].

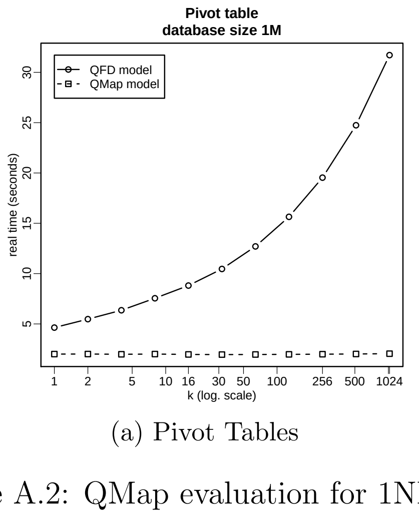


(a) Pivot Tables

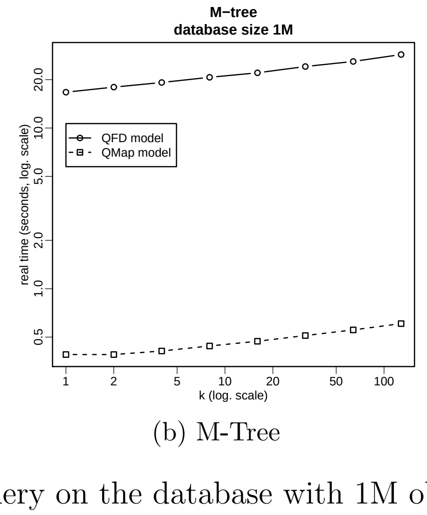


(b) M-Tree

Figure A.1: QMap evaluation for 1NN query on growing databases



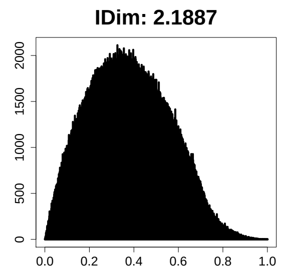
(a) Pivot Tables



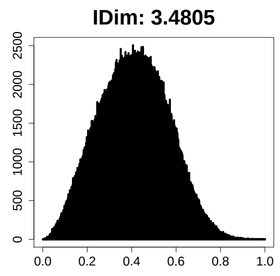
(b) M-Tree

Figure A.2: QMap evaluation for 1NN query on the database with 1M objects

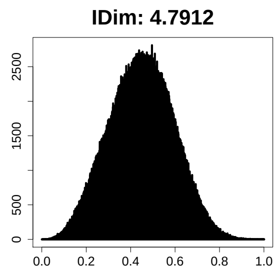
[2]Results acquired for 1,000,000 images represented by real-valued RGB histograms with the dimensionality of 512, where the R,G,B components are divided in 8 bins each, thus 8*8*8 = 512 bins. Each histogram is normalized to have the sum equal to 1. Then, for each bin we compute the color in the center of the bin $[(\frac{R_{\min}+R_{\max}}{2}, \frac{G_{\min}+G_{\max}}{2}, \frac{B_{\min}+B_{\max}}{2}]$ as the *color prototype*, and transform this color to CIE Lab color space [40].



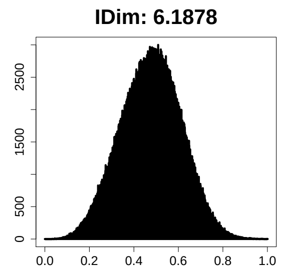
(a) 2D



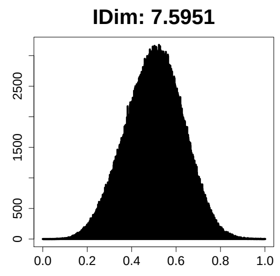
(b) 3D



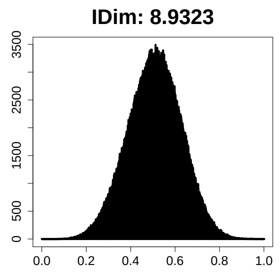
(c) 4D



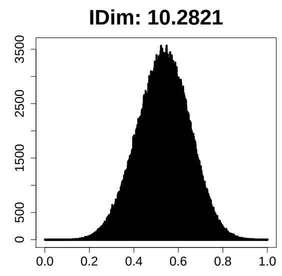
(d) 5D



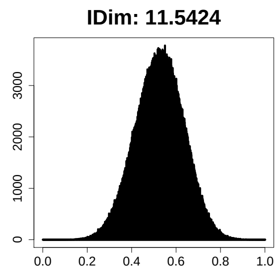
(e) 6D



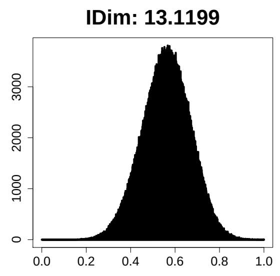
(f) 7D



(g) 8D



(h) 9D



(i) 10D

Figure A.3: Distance distribution histograms for N-dimensional datasets

A.2 Custom N-dimensional Datasets

Figure A.3 outlines the IDim values (Section 3.2.3) for n-dimensional datasets we use for evaluating *LowerBound Tightening* behavior in Section 8.4.4.

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

AESA	Approximating and Eliminating Search Algorithm
DC	Distance Computations
FP, FP-modifier	Fractional-Power Modifier
GA	Genetic Algorithm
GP	Genetic Programming
ISR	Inequality Symbolic Regression
k**NN**	k Nearest Neighbors
LAESA	Linear Approximating and Eliminating Search Algorithm
LB	Lowerbound
$\mathbf{LB}_{\triangle}$	Triangle Lowerbound
$\mathbf{LB}^{+}$	Triangle^{+} Lowerbound
LBT	LowerBound Tightening
MAE	Mean Absolute Error
MAM	Metric Access Method
MBR	Minimum Bounding Rectangle
SAM	Spatial Access Method
SEQ	Sequential Scanning
SR	Symbolic Regression
PGP	Parallel Genetic Programming
PT	Pivot Table (AESA or LAESA)
PtoAM	Ptolemaic Access Method