FlexiDex: Flexible Indexing for Similarity Search with Logic-Based Query Models

Marcel Zierenberg and Maria Bertram

Brandenburg University of Technology Cottbus
Institute of Computer Science, Information and Media Technology
Chair of Database and Information Systems
P.O. Box 10 13 44, 03013 Cottbus, Germany
zieremar@tu-cottbus.de, maria.bertram@berner-mattner.com

Abstract. The flexibility of an indexing approach plays an important role for its applicability, especially for logic-based similarity search. A flexible approach allows the use of the same precomputed index structure even if query elements like weights, operators, monotonicity or used features of the aggregation function change in the search process (e.g., when using relevance feedback). While state-of-the-art approaches typically fulfill some of the needed flexibility requirements, none provides all of them. Consequently, this paper present *FlexiDex* , an efficient indexing approach for logic-based similarity search that is more flexible and also more efficient than known techniques.

1 Introduction

Similarity search is used in domains like image and text retrieval, DNA sequencing, biometric devices, and so forth. It generally seeks to find the most similar objects in a set of database objects, according to a given *query object*. The (dis-)similarity of two objects is determined by a *distance function* operating on *features* extracted from the database objects. Since the number of database objects is usually high and the computation of distances can be expensive in terms of CPU and I/O costs, efficient similarity search is an important research topic.

Early research efforts in similarity search were mainly concerned with search according to a single feature. In recent years, similarity search with multiple features has received more attention. Increasing the number of used features improves the expressiveness of queries and can lead to more effective similarity search. However, in regard of efficiency, adding features significantly increases CPU and I/O costs.

The combination of features is typically achieved through *aggregation functions*, which combine the *partial distances* of multiple features into an *aggregated distance*. While in general arbitrary aggregation functions can be used, the focus of this paper is similarity search with *logic-based query models*. Logic-based similarity search depends on the formulation of queries with the help of Boolean operators. Queries are then converted into arithmetic aggregation functions by means of specific transformation rules. Examples for logic-based query models in similarity search are the *Fuzzy Logic* [1] or the *Commuting Quantum Query Language* (*CQQL*) [2].

B. Catania, G. Guerrini, and J. Pokorný (Eds.): ADBIS 2013, LNCS 8133, pp. 274–287, 2013.

Indexing approaches for similarity search try to exclude as many objects as early as possible from the search, consequently decreasing CPU- and I/O costs. In addition to the efficiency of the indexing approach, its *flexibility* plays an important role in logic-based similarity search. A flexible indexing approach allows the use of the same precomputed index structure even if query elements like weights, operators or monotonicity of the aggregation function change in the search process.

The *flexibility requirements* will be denoted by (R1)–(R6) in the following. Flexibility in terms of features (R1) means that the indexing should be independent from the structure of the underlying features. As changes of weights (R2) and operators (R3) in the aggregation function are typical, for example when using *relevance feedback*, the index should be built independently from the used aggregation function(s). Even though the number of features can be high, in most cases only a subset (R4) of all indexed features is used in a query. A flexible index should process such a query without loss of efficiency. Most indexing approaches for multiple features focus on globally monotone aggregation functions (e.g., weighted sum) [3, 4]. However, in logic-based similarity search not all aggregation functions are globally monotone and other types of monotonicity (R5) need to be supported as well. Finally, the index should be dynamic (R6) in the sense that new objects and new features can be added efficiently without rebuilding the whole index.

While a number of specialized indexing approaches for similarity search with multiple features exist [3–6], none of them fulfills all of the flexibility requirements (see Section 2). Consequently, we present *FlexiDex* , an index that is more flexible than state-of-the-art approaches. As a *metric indexing* approach [7], FlexiDex relies solely on distances for indexing and is independent from the structure of features (R1). The index is built without knowledge of the aggregation function and therefore naturally supports changing weights (R2) and operators (R3). Index data is stored on an HDD in the form of one separate signature file per feature, which allows efficient querying for subsets (R4) of the indexed features. All types of logic-based queries are supported by FlexiDex, even queries with aggregation functions that are not globally monotone (R5). Furthermore, FlexiDex is dynamic (R6) as new objects and new features can be simply appended to an existing index.

The paper is structured as follows. A deeper insight into related work is given in Section 2. Section 3 defines the notations and terms used throughout the paper. As the monotonicity of an aggregation function plays an important role in the indexing process, Section 4 presents different types of monotonicity classes. Section 5 details the index creation and search process. The experimental evaluation shown in Section 6 demonstrates that FlexiDex significantly reduces I/O and CPU costs and even outperforms a less flexible state-of-the-art approach. Section 7 summarizes the work and gives an outlook on future research.

2 Related Work

This section gives a deeper insight into related work. Indexing approaches are divided into different categories, examples for each category are given and their flexibility is examined in detail.

Table 1. Indexing approaches for multiple features and their general flexibility

Indexing based on	combined to single feature	multiple features	aggregated distances		partial distances
Index type	R-Tree [8] VA-File [5] (naïve spatial)	GeVAS [9]	M-Tree [10] LAESA [11] (naïve metric)	Multi-Metric [4]	M^2-Tree [6] Combiner [3] *FlexiDex*
Flexibility	low	medium	low	medium	high

Table 1 shows an overview of indexing approaches for *multiple features* and their general flexibility. The flexibility mainly depends on where the indexing is used in the similarity search process. Indexing based directly on the feature data (single feature, multiple features) is independent of the aggregation function (R2, R3). However, it is typically restricted to vector spaces (*multi-dimensional* or *spatial indexing* [7]), making the index dependent on the structure of the features (R1). On the other side, indexing based on aggregated distances depends on the used aggregation function, disallowing the change of weights (R2) and operators (R3). Only approaches that utilize partial distances, like [3, 6] or FlexiDex, can theoretically support all flexibility requirements. With the exception of the M^2-Tree [6], all presented approaches consider only globally monotone increasing aggregation functions (R5).

Naïve approaches for indexing multiple features rely on combining multiple features into a single feature (spatial) or using aggregated distances (metric). This allows the utilization of a plethora of well-known indexing approaches for similarity search with a single feature (e.g., *R-Tree* [8], *VA-File* [5], *M-Tree* [10] or *LAESA* [11]). Unfortunately, it also leads to inflexibility since queries using only subsets of all indexed features (R4) cannot be processed efficiently and adding new features is not possible without rebuilding the whole index (R6). Additionally, in case of naïve metric indexing, weights (R2) or operators (R3) of the aggregation function can not be changed, as the index was created according to a specific (metric) aggregation function.

The spatial indexing approach *GeVAS* [9] is an extension of the VA-File to multiple features. By using a separate signature file for each feature, it allows efficient queries with subsets (R4) and dynamically adding new objects and new features (R6). It is therefore more flexible than the naïve approach. However, the main problem of spatial indexing, the limitation to vector spaces, remains.

Multi-metric indexing [4] defines a framework to transform metric indexing approaches for single features into metric indexing approaches for multiple features using partial and aggregated distances at the same time. Even though this technique allows changing weights in the aggregation function (R2), it is restricted to aggregation functions that are metrics (e.g., weighted sum) and the index is still created based on a specific aggregation function, preventing the change of operators (R3). Since aggregated distances are used, querying with subsets (R4) is less efficient and new features can not be added dynamically (R6).

The *M^2-Tree* [6] is in general flexible enough to support changing weights (R2) and operators (R3). However, all features are used in the construction of the tree. This means, that with increasing numbers of features, the tree suffers from the curse of dimensionality [7]. Also, the clustering may prove inefficient if only a subset of all indexed features

Table 2. Fulfillment of flexibility requirements (R1)–(R6) for indexing approaches

Index type	Features (R1)	Weights (R2)	Operators (R3)	Subsets (R4)	Monotonicity (R5)	Dynamic (R6)
naïve (spatial)	-	✓	✓	-	-	-
GeVAS	-	✓	✓	✓	-	✓
naïve (metric)	✓	-	-	-	-	-
Multi-Metric	✓	✓	-	-	-	-
M^2-Tree	✓	✓	✓	-	(✓)	-
Combiner	✓	✓	✓	✓	-	✓
FlexiDex	✓	✓	✓	✓	✓	✓

is present in a query (R4). Even though the M^2-Tree considers globally and locally monotone functions, it still lacks support for some logic-based aggregation functions (R5) (e.g., $x_1 \oplus x_2$ (XOR), see Section 4). New objects can be added dynamically to the M^2-Tree, but adding new features requires rebuilding the whole tree (R6).

Combiner algorithms [3] are inherently independent of the underlying indexing approach and provide almost all flexibility requirements. Their drawback is that they only consider aggregation functions that are globally monotone increasing (R5), which, as stated before, is not always the case for logic-based queries.

Table 2 summarizes the flexibility of the presented indexing approaches and shows that FlexiDex constitutes the most flexible approach.

While *approximate similarity search* [7] can significantly decrease search time, it comes at the cost of effectiveness, since the results only have a certain probability to be the most similar objects. The focus of our research is exact similarity search and therefore approximate indexing approaches like [12] are not applicable.

3 Preliminaries

This section defines the notations and terms used throughout this paper.

Similarity search can be performed by means of a *k-Nearest Neighbor search*. A $k\text{NN}(q)$ in the universe of objects $\mathbb{U}$ returns k objects out of a database of objects $D = \{o^1, o^2, \ldots, o^n\} \subseteq \mathbb{U}$ that are closest (most similar) to the query object $q \in \mathbb{U}$. The distance between objects is computed by a distance function $\delta : \mathbb{U} \times \mathbb{U} \mapsto \mathbb{R}_{\geq 0}$ that operates on the features q' and o' extracted from the objects. The result is a (non-deterministic) set K with $|K| = k$ and $\forall o^i \in K, o^j \in D \setminus K : \delta(q, o^i) \leq \delta(q, o^j)$.

A *multi-feature kNN query* consists of m features for each object $q' = (q_1, q_2, \ldots, q_m)$ and $o' = (o_1, o_2, \ldots, o_m)$. A distance function δ_j is assigned to each single feature to compute *partial distances* $\delta_j(q_j, o_j)$. An *aggregation function* $\text{agg} : \mathbb{R}^m_{\geq 0} \mapsto \mathbb{R}_{\geq 0}$ combines all partial distances to an *aggregated distance* d_{agg} and the k nearest neighbors are then determined according to the aggregated distance.

Logic-based queries combine multiple features by Boolean operators. Query models like the *Fuzzy Logic* [1] or *CQQL* [2] transform the resulting Boolean expressions into arithmetic formulas. Table 3 shows examples of transformation rules. CQQL uses algebraic transformation rules and, contrary to Fuzzy Logic, normalizes the expressions

Table 3. Transformation rules for logic-based queries

Boolean	Zadeh	Algebraic
$\neg a$	$1-a$	$1-a$
$a \wedge b$	$\min(a,b)$	$a*b$
$a \vee b$	$\max(a,b)$	$a+b-a*b$
$(c \wedge a) \vee (\neg c \wedge b)$	(analog)	$a+b$

Table 4. Embedding of operand weights with CQQL

Boolean	Embedding
$a \wedge_{\theta_1;\theta_2} b$	$(a \vee \neg\theta_1) \wedge (b \vee \neg\theta_2)$
$a \vee_{\theta_1;\theta_2} b$	$(a \wedge \theta_1) \vee (b \wedge \theta_2)$

before their transformation. This preserves important properties like the distributivity and idempotency. For example, query $(a \wedge b) \vee (a \wedge c)$ results in formula $a*b+a*c-a*b*a*c$ using algebraic Fuzzy Logic. In contrast, the normalized formula used by CQQL is $a \wedge (b \vee c)$, which results in $a*(b+c-b*c)$.

Additionally, CQQL also supports the direct embedding of operand weights $\theta_i \in \mathbb{R}_{\geq 0}$ into the logic (see Table 4). Therefore, only aggregation functions created by CQQL are examined hereinafter. However, note that the stated results are by no means restricted to CQQL and can be adapted to other logic-based query models as well.

Logic-based query models assume similarity values in the interval $[0,1]$, where 1 means most similar (identity) and 0 means least similar. Hence, the aforementioned partial distances have to be transformed into *partial similarities* by *transformation functions* $\mathrm{t_j} : \mathbb{R}_{\geq 0} \mapsto [0,1]$ before the aggregation function can be applied. The aggregated similarity s_{agg} is computed by combining all m partial similarities with the (logic-based) aggregation function $\mathrm{agg} : [0,1]^m \mapsto [0,1]$.

A *metric* [7] is a distance function with the properties *positivity* ($\forall x \neq y \in \mathbb{U} : \delta(x,y) > 0$), *symmetry* ($\forall x,y \in \mathbb{U} : \delta(x,y) = \delta(y,x)$), *reflexivity* ($\forall x \in \mathbb{U} : \delta(x,x) = 0$) and *triangle inequality* ($\forall x,y,z \in \mathbb{U} : \delta(x,z) \leq \delta(x,y) + \delta(y,z)$).

Metric indexing approaches exclude objects from search by computing *bounds* on the distance from the query object to database objects. The lower bound d^{lb} and upper bound d^{ub} on the distance $\delta(q,o)$ between query object q and database object o can be computed with the help of the triangle inequality and precomputed distances to a *reference object* (*pivot*) p as follows:

$$d^{lb} = |\delta(q,p) - \delta(p,o)| \leq \delta(q,o) \leq \delta(q,p) + \delta(p,o) = d^{ub}. \tag{1}$$

For logic-based queries, distance bounds have to be transformed into similarity bounds s^{lb} and s^{ub}. Notice that the meaning of lower and upper bound exchanges with this transformation. The upper bound for distances is the maximum dissimilarity, while the upper bound for similarities is the maximum similarity.

Bounds s^{lb}_{agg} and s^{ub}_{agg} on the aggregated similarity s_{agg} can be computed by inserting partial similarity bounds into the aggregation function.

4 Monotonicity and Computation of Aggregated Bounds

Most indexing approaches for similarity search with multiple features consider only aggregation functions that are monotone increasing in each of their arguments (globally

monotone increasing functions). However, to allow all different kinds of logic-based queries, other classes of monotonicity have to be supported as well (R5). Consequently, this section describes the computation of bounds on aggregated similarities based on the bounds on partial similarities for three different classes of monotonicity: *globally*, *locally* and *flexible monotone aggregation functions*.

Although another naming convention was used, globally and locally monotone functions have already been considered by [13] and [6]. However, queries like $x_1 \oplus x_2$ (XOR) result in aggregation functions that are neither globally nor locally monotone. Therefore, the new class of flexible monotone functions is introduced and we show that the three presented classes of monotonicity are sufficient to support all kinds of (logic-based) aggregation functions created by CQQL.

4.1 Globally and Locally Monotone Functions

Definition 1. *An aggregation function* agg *is monotone increasing in the i-th argument with* $i \in \{1,2,\ldots,m\}$, $x = (x_1,x_2,\ldots,x_i,\ldots,x_m)$ *and* $x' = (x_1,x_2,\ldots,x'_i,\ldots,x_m)$ *iff:*

$$\forall x,x' \in [0,1]^m : x_i < x'_i \implies \text{agg}(x) \leq \text{agg}(x')\,. \tag{2}$$

This means, if all arguments except x_i are constant and x_i is increased, the result of the aggregation function will also increase (or be constant).

Definition 2. *An aggregation function* agg *is monotone decreasing in the i-th argument iff* $\text{agg}' = -\text{agg}$ *is monotone increasing in the i-th argument.*

Definition 3. *An aggregation function* agg *is globally monotone increasing (decreasing) iff it contains only monotone increasing (decreasing) arguments.*

An example for a globally monotone increasing function is $\text{agg}(x_1,x_2) = x_1 * x_2$, which is the result of the query $x_1 \wedge x_2$.

It can easily be shown that the lower (upper) bound s^{lb}_{agg} (s^{ub}_{agg}) on the aggregated similarity of a globally monotone increasing function is computed by inserting all lower (upper) bounds on the partial similarities into the aggregation function:

$$s^{lb}_{agg} = \text{agg}\left(s^{lb}_1, s^{lb}_2, \ldots, s^{lb}_m\right)\,. \tag{3}$$

Definition 4. *An aggregation function* agg *is locally monotone iff it contains only monotone increasing and monotone decreasing arguments.*

The aggregation function $\text{agg}(x_1,x_2) = x_1 * (1 - x_2)$ based on the query $x_1 \wedge \neg x_2$ is an example for a locally monotone function.

By definition, every globally monotone function also is a locally monotone function with only monotone increasing/decreasing arguments.

The following equations compute lower bounds for locally monotone aggregation functions. The computation of upper bounds follows the same equations. Only the values *lb* and *ub* in function f^{agg}_i have to be exchanged.

$$s^{lb}_{agg} = \text{agg}\left(\text{f}^{\text{agg}}_1\left(s^{lb}_1, s^{ub}_1\right), \text{f}^{\text{agg}}_2\left(s^{lb}_2, s^{ub}_2\right), \ldots, \text{f}^{\text{agg}}_m\left(s^{lb}_m, s^{ub}_m\right)\right) \tag{4}$$

$$\text{f}^{\text{agg}}_i(lb,ub) = \begin{cases} lb, & \text{if agg is monotone increasing in the } i\text{-th argument} \\ ub, & \text{if agg is monotone decreasing in the } i\text{-th argument} \end{cases} \tag{5}$$

4.2 Flexible Monotone Functions

Definition 5. *An aggregation function* agg *is flexible monotone in the i-th argument with* $i \in \{1,2,\ldots,m\}$, $x = (x_1,x_2,\ldots,x_i,\ldots,x_m)$ *and* $x' = (x_1,x_2,\ldots,x'_i,\ldots,x_m)$ *iff:*

$$\begin{aligned} \forall x \in [0,1]^m : \\ \left(\forall x' \in [0,1]^m : x_i < x'_i \implies \mathrm{agg}(x) \leq \mathrm{agg}(x')\right) \\ \vee \left(\forall x' \in [0,1]^m : x_i < x'_i \implies \mathrm{agg}(x) \geq \mathrm{agg}(x')\right) . \end{aligned} \tag{6}$$

In other words, if all arguments except x_i are constant and x_i is increased, the result of the aggregation function will either always increase (monotone increasing) or always decrease (monotone decreasing) or be constant for the current combination of fixed $m-1$ argument values. The difference between a monotone increasing/decreasing argument and a flexible monotone argument is that the monotonicity of a flexible monotone argument depends on the values of the other arguments and can therefore change. For monotone increasing/decreasing arguments the monotonicity is always the same, independent of the values of the other arguments.

Definition 6. *An aggregation function* agg *is flexible monotone iff it contains only flexible monotone arguments.*

Figure 1 shows a flexible monotone aggregation function based on the query $x_1 \oplus x_2$. As can be seen, the monotonicity of argument x_2 changes between monotone increasing and monotone decreasing, depending on the value of x_1.

By definition, every monotone increasing/decreasing argument also is a flexible monotone argument. A locally monotone function also is a flexible monotone function where the monotonicity in the flexible monotone arguments never changes.

Theorem 1. *The upper and lower bound on the aggregated similarity of a flexible monotone function are always located in the corners of the hyper-rectangle given by the m lower and upper bounds on the partial similarities* $\left\{(s_1^{lb},s_1^{ub}),(s_2^{lb},s_2^{lb}),\ldots,(s_m^{lb},s_m^{ub})\right\}$.

Proof. Let aggregation function agg be a flexible monotone function and the partial similarity bounds for $1 \leq i \leq m$ be $(s_i^{lb},s_i^{ub}) = (0,1)$. Without loss of generality now consider a tuple $x \in [0,1]^m$ with $\mathrm{agg}(x)$ to be a global maximum. For an arbitrarily chosen i assume $0 < x_i < 1$, which means x is not located in a corner of the hyper-rectangle $[0,1]^m$. Now consider two tuples $x',x'' \in [0,1]^m$ with $x' = (x_1,x_2,\ldots,x'_i,\ldots,x_m)$ and $x'' = (x_1,x_2,\ldots,x''_i,\ldots,x_m)$ for which $x'_i < x_i < x''_i$ is true. Because $\mathrm{agg}(x)$ is a maximum, $\mathrm{agg}(x') \leq \mathrm{agg}(x)$ and $\mathrm{agg}(x'') \leq \mathrm{agg}(x)$ is true. However, this contradicts

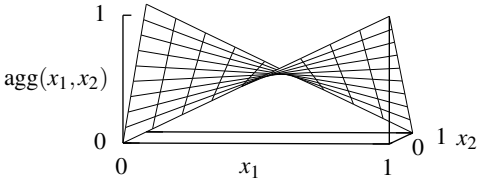

Fig. 1. Flexible monotone aggregation function $x_1 * (1-x_2) + (1-x_1) * x_2$.

the assumption of a flexible monotone function for which $\text{agg}(x') \leq \text{agg}(x) \leq \text{agg}(x'')$ or $\text{agg}(x'') \leq \text{agg}(x) \leq \text{agg}(x')$ should be true. □

From Theorem 1 immediately follows that lower (upper) bounds s^{lb}_{agg} (s^{ub}_{agg}) on the aggregated similarities of a flexible monotone function can be computed by inserting all different combinations of lower and upper bounds for flexible monotone arguments into the aggregation function and subsequently selecting the minimum (maximum) result.

Combined with Equation (4), the computation of bounds for all three classes of monotonicity is therefore defined as follows. Again, the computation of upper bounds s^{ub}_{agg} follows the same principle. Only the values *lb* and *ub* in the first two conditions of function g^{agg}_i have to be exchanged.

$$C = \text{g}^{\text{agg}}_1\left(s^{lb}_1, s^{ub}_1\right) \times \text{g}^{\text{agg}}_2\left(s^{lb}_2, s^{ub}_2\right) \times \ldots \times \text{g}^{\text{agg}}_m\left(s^{lb}_m, s^{ub}_m\right) \quad \textit{(combinations)} \tag{7}$$

$$\text{g}^{\text{agg}}_i(lb, ub) = \begin{cases} \{lb\}, & \text{if agg is monotone increasing in the } i\text{-th argument} \\ \{ub\}, & \text{if agg is monotone decreasing in the } i\text{-th argument} \\ \{lb, ub\}, & \text{if agg is flexible monotone in the } i\text{-th argument} \end{cases} \tag{8}$$

$$s^{lb}_{agg} = \min_{c \in C} \text{agg}(c) \quad \textit{(selection)} \tag{9}$$

4.3 Logic-Based Queries and Monotonicity Classes

As stated in [13], every logic-based aggregation function which contains each argument only once is either globally or locally monotone. For functions that contain an argument twice, [13] proposes to handle those occurrences as independent arguments. Unfortunately, no proof for correctness was given and it can easily be shown that the approach tends to create bounds less tight than Equation (9).

The disadvantage of Equation (9) is that the number of combinations that have to be aggregated increases exponentially with the number of flexible monotone arguments (maximum 2^m). It is therefore only applicable for aggregation functions with a low number of flexible monotone arguments.

Every aggregation function created by CQQL (*CQQL formula*) is flexible monotone. For space reasons only a sketch of the proof is given[1]: It can be proven that every multivariate linear polynomial is a flexible monotone function by bringing it into the form $\text{f}(x) = x_i * c_1 + c_2$, where c_1 and c_2 are multivariate linear polynomials that do not contain x_i. It can also be shown that every CQQL formula is a multivariate linear polynomial. Thus, every CQQL formula is flexible monotone.

5 FlexiDex

This section explains the index creation and *k*NN search process of FlexiDex. Figure 2 gives an overview of the process, which is divided into the stages *creation & preparation*, *filtering* and *refinement*. The figure also shows, where the flexibility requirements (R1)–(R6) are involved.

[1] Detailed version: `http://tiny.cc/8t4wuw` (`http://dbis.informatik.tu-cottbus.de/down/pdf/CQQL-Monotonicity.pdf`)

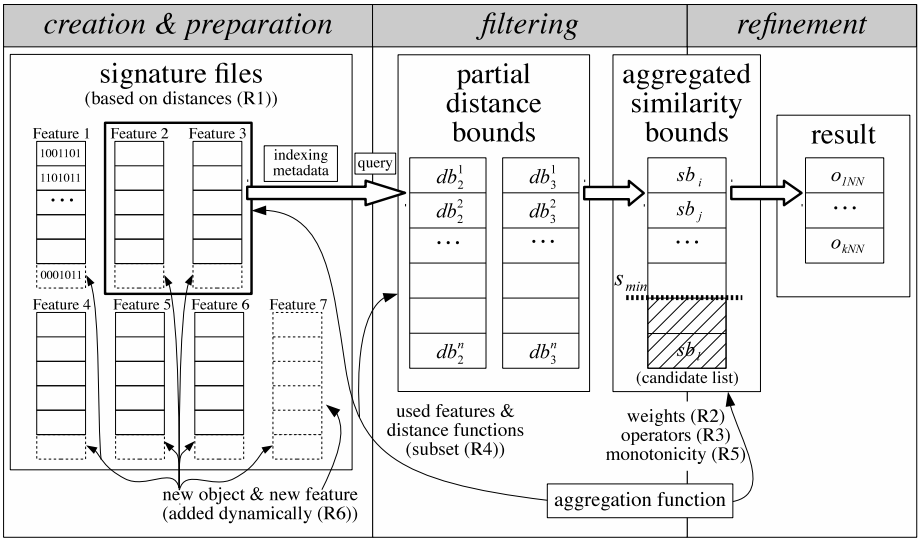

Fig. 2. Overview of the creation and search process of FlexiDex

5.1 Index Creation

FlexiDex follows the GeVAS [9] approach, creating one *signature file* for each feature. However, to ensure independence from the structure of the features (R1), metric indexing is used instead of spatial indexing. This means, signatures are not based on feature vectors but instead are comprised of precomputed distances to a set of pivots. Since one separate signature file is used for each feature, each signature file can be read from disk independently. Therefore, FlexiDex can efficiently process queries using subsets of all indexed features (R4) and also is dynamic (R6), since new objects can simply be appended to an existing signature file and new signature files can be created at any time.

In the following the index creation process for a single feature is described. The creation process for multiple features works similar. The only constraint is that the order of objects has to be the same in all signature files. The process of signature creation resembles [14], where metric indexing and signature files were used for similarity search with a single feature. Also note that the parameters explained in the following can be set individually for each feature.

Index creation starts with the selection of P pivot objects. Pivots can be selected randomly or by a specific selection strategy like *incremental selection* [15]. Additionally, 2^B *quantization intervals* $i^0, i^1, \ldots, i^{2^B-1}$ are defined which will be used for the compression of the distances. The number of used bits B determines the precision of the intervals. A higher number of bits results in better distance approximations but also increases the time needed for reading the signature file. Selected pivots and indexing parameters are stored on disk as metadata.

When adding a new database object o to the index, the distances $\delta(p^k, o)$ between object o and all pivot objects p^k are computed. Each distance is then replaced with the

interval number of the corresponding quantization interval. The *object signature S* is the result of the concatenation of each binary coded interval number and simply appended to the existing signature file.

Example 1. Consider a new database object o with two features o_1 and o_2 ($m = 2$). Indexing parameters are set individually for each feature and distinguished by their subscript. The number of pivots is $P_1 = 4$, $P_2 = 2$ and the interval boundaries are $i_1^0 = [0,4), i_1^1 = [4,8), i_1^2 = [8,12), i_1^3 = [12,16]$ ($B_1 = 2$) and $i_2^0 = [0,2), i_2^1 = [2,4]$ ($B_2 = 1$). Now, let exemplary distances to the pivots for the first feature be given by $\delta_1(p_1^1,o_1) = 5, \delta_1(p_1^2,o_1) = 16, \delta_1(p_1^3,o_1) = 7$ and $\delta_1(p_1^4,o_1) = 1$ and for the second feature by $\delta_2(p_2^1,o_2) = 0$ and $\delta_2(p_2^2,o_2) = 3$. By replacing the distances with their binary coded interval numbers the following object signatures are obtained for object o: $S_1 = (01\ 11\ 01\ 00)$ and $S_2 = (0\ 1)$.

5.2 Nearest Neighbor Search

Since the creation process is completely independent from the aggregation functions used in *k*NN search, changing weights (R2) and operators (R3) of the aggregation function at query time is supported naturally. With the help of the monotonicity classes defined in Section 4, all types of logic-based queries can be processed (R5).

For *k*NN search the well-known concept of *filter refinement* [5, 14] is utilized. The *filtering phase* reads each signature file sequentially from disk and afterward computes partial distance bounds for each object with the help of the triangle inequality. Equation (1) has to be adapted slightly to incorporate the fact that exact distance $\delta(p,o)$ is replaced by the corresponding interval boundaries i^{lb} and i^{ub}:

$$d^{lb} = \max\left\{0, i^{lb} - \delta(q,p), \delta(q,p) - i^{ub}\right\} \leq \delta(q,o) \leq \delta(q,p) + i^{ub} = d^{ub}. \qquad (10)$$

Note that only signature files that are actually present in the current aggregation function are loaded. Distance bounds are then transformed into similarity bounds and combined to aggregated similarity bounds, using Equation (9). Objects having a smaller upper bound s_{agg}^{ub} than the k-th greatest lower bound s_{agg}^{lb} (s_{min}) can be excluded from the search.

In the *refinement phase* exact aggregated similarities are computed for the objects that could not be excluded in the filtering phase. A priority queue PQ, sorted in descending order according to s_{agg}^{lb}, acts as a *candidate list* for refinement. Since objects are reinserted into PQ after the computation of the exact aggregated similarity s_{agg}, the search can be terminated as soon as k exactly computed objects have reappeared at the top of PQ.

6 Evaluation

This section compares FlexiDex against linear scan and a combiner algorithm. FlexiDex was implemented as part of the multimedia retrieval system *PythiaSearch*[2] [16].

All experiments were conducted on a 2 x 2.26 GHz Quad-Core Intel Xeon with 8 GB RAM and an HDD with 7,200 rpm. The image collection *Caltech-256 Object Category*

[2] `http://tiny.cc/mc5wuw` (`https://saffron.informatik.tu-cottbus.de/livingfeatures`)

Table 5. Used features and their optimal index parameters

Feature	δ	ρ	*P*	*B*	*D*	*T* in s
ScalableColor (*scal*) [20]	L_1	2.91	8	8	873 (2.9 %)	1.57 (8.1 %)
Tamura (*tam*) [21]	L_1	2.89	8	8	1,205 (3.9 %)	1.93 (10.6 %)
FCTH (*fcth*) [22]	L_1	5.77	32	8	1,658 (5.4 %)	2.57 (13.0 %)
DominantColor (*dom*) [20]	EMD + L_2	2.16	24	10	1,755 (5.7 %)	2.70 (11.7 %)
ColorStructure (*cs*) [20]	L_2	5.87	56	8	3,143 (10.3 %)	3.66 (21.7 %)
ColorHistCenter (*chc*) [20]	L_1	3.89	56	8	2,999 (9.8 %)	3.70 (20.3 %)
ColorLayout (*cl*) [20]	weighted L_2	5.05	48	8	6,814 (22.3 %)	6.42 (30.5 %)
ColorHistBorder (*chb*) [20]	L_1	5.82	64	8	7,507 (24.5 %)	6.77 (36.2 %)
AutoColorCorrel. (*auto*) [20]	L_2	8.07	64	8	7,789 (25.4 %)	7.17 (36.3 %)
ColorHistogram (*ch*) [20]	L_2	11.48	64	8	10,697 (34.9 %)	9.14 (46.6 %)
BIC (*bic*) [23]	L_1	10.23	56	8	12,908 (42.2 %)	9.18 (49.8 %)
EdgeHistogram (*edge*) [20]	weighted L_1	8.55	48	10	12,056 (39.4 %)	10.48 (50.7 %)
CEDD (*cedd*) [24]	L_2	12.05	64	8	10,863 (35.5 %)	11.03 (45.2 %)

Dataset [17] was used, which consists of 30,607 pictures. Efficiency was assessed by measuring the average number of distance computations and the average *k*NN search time (wall-clock time) of 100 randomly chosen query objects.

To provide a comparison to the state-of-the-art, a combiner algorithm (*threshold algorithm* [3]) was implemented. Sorted lists for the combiner algorithm are provided based on the same signature files as FlexiDex and the implementation of *getNext* is similar to [18].

Table 5 shows the 13 different color, texture and form features extracted for the collection. While most features utilize some type of *Minkowski (L_p) distance* function, the feature *dom* employs the *Earth Mover's distance function (EMD)* [19].

Optimization of indexing parameters was performed separately for each feature. Pivots were chosen randomly. The tested precision values B ranged from 4 to 12 and the number of pivots P from 8 to 64. For both parameters higher values decreased the number of needed distance computations but simultaneously increased the time needed for reading the signature files. Optimal parameters were selected based on the average search time for queries with $k = 10$.

The results of the optimization process are presented in Table 5. Search times (T) and number of distance computations (D) were significantly reduced in comparison to the *linear scan* (see percentage values). Since FlexiDex and the combiner algorithm work similar when using only a single feature, the determined optimal parameters apply to both approaches.

The optimal precision value B was 8 in almost all cases. For lower precision values the number of distance computations increased rapidly, resulting in significantly longer search times. Higher values than 8 gave only slight decreases in distance computations that could not make up for the increased time needed for reading the signature files. The number of used pivots P showed some correlation to the intrinsic dimensionality[3]. For

[3] Intrinsic dimensionality ρ is defined as $\rho = \frac{\mu^2}{2*\sigma^2}$ where μ is the mean and σ^2 is the variance of a distance distribution. It is frequently used as an estimator for indexability [25].

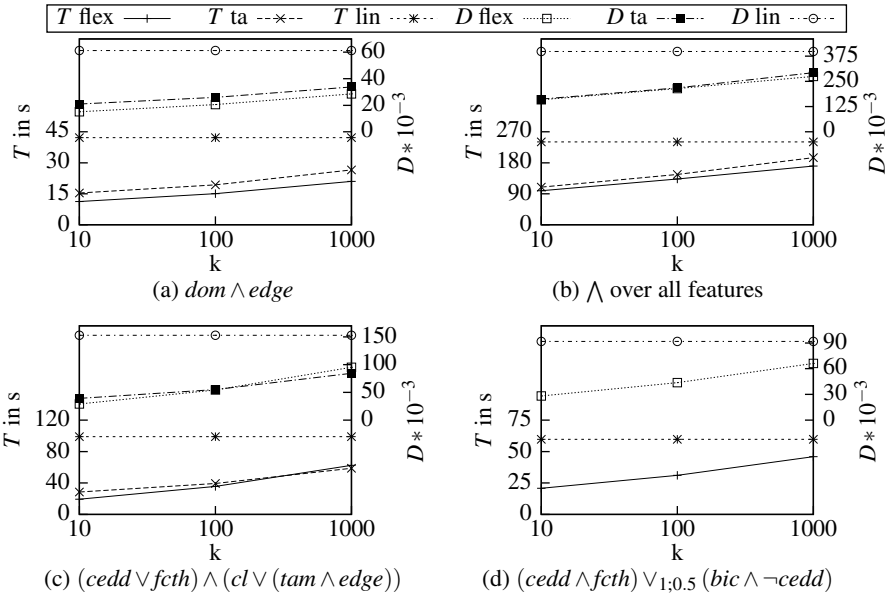

Fig. 3. Measured efficiency for selected logic-based *k*NN queries.

low intrinsic dimensionality (features *scal* or *dom*) a small number of pivots excluded more than 90 % of the objects. A greater number of pivots was needed when the intrinsic dimensionality increased.

Figure 3 depicts the average number of distance computations and search time of selected logic-based *k*NN queries for FlexiDex (*flex*), linear scan (*lin*) and the implemented combiner algorithm (*ta*). The selected queries are similar to those used in [26] and represent different types of logic-based queries: only conjunction (3a, 3b), disjunction and conjunction (3c) and weighted disjunction, conjunction and negation (3d).

FlexiDex performed clearly superior to linear scan and combiner algorithm in almost all cases. In the best case FlexiDex was about 33 % faster (query 3c with $k = 10$) than the combiner algorithm and needed about 25 % less distance computations. Only in case of query 3c with $k = 1000$ the combiner algorithm outperformed FlexiDex slightly ($\sim 6\,\%$ faster). Query 3d resulted in a flexible monotone aggregation function, which was not supported by the combiner algorithm.

7 Conclusion and Outlook

The flexibility of an indexing approach plays an important role in logic-based similarity search. It allows the use of the same precomputed index even if query elements like the used features or the monotonicity of the aggregation function change in the search process. Since none of the known approaches presented in Section 2 supports all flexibility requirements (R1)–(R6), we introduce the new index FlexiDex.

The adaption of the GeVAS [9] approach to metric indexing (Section 5) results in a flexible index that is independent from the structure of features (R1), supports changing

weights (R2) and operators (R3), allows efficient querying for subsets (R4) of the indexed features and can dynamically add new objects and features (R6). As indexing for logic-based queries depends on the monotonicity of the resulting aggregation functions, three classes of monotonicity are examined in Section 4 and it is shown that these are sufficient to support all types of logic-based queries (R5). The efficiency of FlexiDex is evaluated in Section 6, which proves that FlexiDex significantly outperforms the linear scan and a combiner algorithm in terms of distance computations and search time.

Future work will focus on an extended evaluation with large synthetic and real-world datasets. The definition of cost formulas will allow an analytical approach for parametrization of the index and support for other types of queries (e.g., range search) will be added.

Acknowledgements. Special thanks go to Robert Kuban for proving the monotonicity of CQQL formulas.

References

[1] Zadeh, L.A.: Fuzzy Logic. Computer 21, 83–93 (1988)
[2] Schmitt, I.: QQL: A DB&IR Query Language. The VLDB Journal 17, 39–56 (2008)
[3] Fagin, R., Lotem, A., Naor, M.: Optimal Aggregation Algorithms for Middleware. In: Proceedings of the 20th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2001, pp. 102–113. ACM, Santa Barbara (2001)
[4] Bustos, B., Kreft, S., Skopal, T.: Adapting Metric Indexes for Searching in Multi-Metric Spaces. Multimedia Tools Appl. 58(3), 467–496 (2012)
[5] Weber, R., Schek, H.-J., Blott, S.: A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. In: Proceedings of the 24th International Conference on Very Large Data Bases, VLDB 1998, pp. 194–205. Morgan Kaufmann Publishers Inc., San Francisco (1998)
[6] Ciaccia, P., Patella, M.: The M^2-Tree: Processing Complex Multi-Feature Queries with Just One Index. In: DELOS Workshop: Information Seeking, Searching and Querying in Digital Libraries (2000)
[7] Samet, H.: Foundations of Multidimensional and Metric Data Structures. The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling. Morgan Kaufmann Publishers Inc., San Francisco (2005)
[8] Guttman, A.: R-Trees: A Dynamic Index Structure for Spatial Searching. In: Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data, SIGMOD 1984, pp. 47–57. ACM, Boston (1984)
[9] Böhm, K., Mlivoncic, M., Schek, H.-J., Weber, R.: Fast Evaluation Techniques for Complex Similarity Queries. In: Proceedings of the 27th International Conference on Very Large Data Bases, VLDB 2001, pp. 211–220. Morgan Kaufmann Publishers Inc., San Francisco (2001)
[10] Ciaccia, P., Patella, M., Zezula, P.: M-Tree: An Efficient Access Method for Similarity Search in Metric Spaces. In: Proceedings of 23rd International Conference on Very Large Data Bases, VLDB 1997, pp. 426–435. Morgan Kaufmann, Athens (1997)
[11] Micó, M.L., Oncina, J., Vidal, E.: A New Version of the Nearest-Neighbour Approximating and Eliminating Search Algorithm (AESA) with Linear Preprocessing Time and Memory Requirements. Pattern Recogn. Lett. 15, 9–17 (1994)
[12] Lange, D., Naumann, F.: Efficient Similarity Search: Arbitrary Similarity Measures, Arbitrary Composition. In: Proceedings of the 20th ACM International Conference on Information and Knowledge Management, CIKM 2011, pp. 1679–1688. ACM, Glasgow (2011)

[13] Ciaccia, P., Patella, M., Zezula, P.: Processing Complex Similarity Queries with Distance-Based Access Methods. In: Schek, H.-J., Saltor, F., Ramos, I., Alonso, G. (eds.) EDBT 1998. LNCS, vol. 1377, pp. 9–23. Springer, Heidelberg (1998)
[14] Balko, S., Schmitt, I.: Signature Indexing and Self-Refinement in Metric Spaces. Tech. rep. 06/12. Brandenburg University of Technology Cottbus, Institute of Computer Science (September 2012)
[15] Bustos, B., Navarro, G., Chávez, E.: Pivot Selection Techniques for Proximity Searching in Metric Spaces. Pattern Recogn. Lett. 24, 2357–2366 (2003)
[16] Zellhöfer, D., et al.: PythiaSearch: A Multiple Search Strategy-Supportive Multimedia Retrieval System. In: Proceedings of the 2nd ACM International Conference on Multimedia Retrieval, ICMR 2012, pp. 59:1–59:2. ACM, Hong Kong (2012)
[17] Griffin, G., Holub, A., Perona, P.: Caltech-256 Object Category Dataset. Tech. rep. 7694. California Institute of Technology (2007)
[18] Schmitt, I., Balko, S.: Filter Ranking in High-Dimensional Space. Data Knowl. Eng. 56, 245–286 (2006)
[19] Rubner, Y., Tomasi, C., Guibas, L.J.: The Earth Mover's Distance as a Metric for Image Retrieval. Int. J. Comput. Vision 40, 99–121 (2000)
[20] Sikora, T.: The MPEG-7 Visual Standard for Content Description-An Overview. IEEE Transactions on Circuits and Systems for Video Technology 11(6), 696–702 (2001)
[21] Tamura, H., Mori, S., Yamawaki, T.: Texture Features Corresponding to Visual Perception. IEEE Transactions on Systems, Man and Cybernetics 8(6) (1978)
[22] Chatzichristofis, S.A., Boutalis, Y.S.: FCTH: Fuzzy Color and Texture Histogram - A Low Level Feature for Accurate Image Retrieval. In: Proceedings of the 2008 9th International Workshop on Image Analysis for Multimedia Interactive Services, WIAMIS 2008, pp. 191–196. IEEE Computer Society, Washington, DC (2008)
[23] Stehling, R.O., Nascimento, M.A., Falcão, A.X.: A Compact and Efficient Image Retrieval Approach Based on Border/Interior Pixel Classification. In: Proceedings of the 11th International Conference on Information and Knowledge Management, CIKM 2002, pp. 102–109. ACM, McLean (2002)
[24] Chatzichristofis, S.A., Boutalis, Y.S.: CEDD: Color and Edge Directivity Descriptor: A Compact Descriptor for Image Indexing and Retrieval. In: Gasteratos, A., Vincze, M., Tsotsos, J.K. (eds.) ICVS 2008. LNCS, vol. 5008, pp. 312–322. Springer, Heidelberg (2008)
[25] Chávez, E., Navarro, G., Baeza-Yates, R., Marroquín, J.L.: Searching in Metric Spaces. ACM Comput. Surv. 33, 273–321 (2001)
[26] Zellhöfer, D., Schmitt, I.: A User Interaction Model Based on the Principle of Polyrepresentation. In: Proceedings of the 4th Workshop for Ph.D. Students in Information and Knowledge Management, PIKM 2011, pp. 3–10. ACM, Glasgow (2011)