Efficient Vague Joins Processing in the VQS

Tran Khanh DANG, Josef KÜNG, Roland WAGNER
FAW-Institute
Johannes Kepler University of Linz
Altenberger Strasse 69, A-4040, Linz
Austria/Europe
E-mail: {khanh, jkueng, rrwagner}@faw.uni-linz.ac.at

ABSTRACT

Joins are among the most expensive operations in database management systems (DBMSs). In the context of the VQS (Vague Query System), which is a flexible query answering system built on top of DBMSs to provide them with similarity search/retrieval capabilities, vague joins are prohibitively expensive in terms of both IO-cost and CPU-cost because they must undergo intermediate processing steps with the sheer volume of multidimensional data in multiple feature spaces. This article presents problems arisen when processing complex vague joins in the VQS and introduces a new approach to efficiently solve these problems. This new approach not only reduces the costs significantly, but also returns to users the best matches or approximate nearest neighbors with a certain tolerant error $\varepsilon$ with respect to given vague join predicates. The proposed approach is based on a novel, flexible, and efficient solution introduced recently for solving (approximate) complex vague queries in the VQS.

Keywords: Similarity search, multidimensional index structures, complex vague queries, similarity joins, information retrieval, flexible query answering systems.

1. INTRODUCTION

Along with the development of the database technology, join processing has been supported as a primitive operation in all commercial strength relational DBMSs due to its important applicability. In the context of spatial databases, spatial joins are also among the most prohibitively expensive operations even when powerful indexes are employed (see [5]). There, a join operation is typically involved in similarity or spatial predicates\(^1\), which do not like the relational joins in traditional DBMSs. In fact, a two-way similarity join is also a join in the relational sense in which two datasets $R$ and $S$ are combined into one such that the new set contains pairs of data objects that fulfill a join condition. The main difference appears in the join condition and data objects participating in the join. The data objects here can be feature vectors or metric objects rather than ordinary tuples/records of some type and, moreover, the join condition is composed by some similarity predicates instead of only the traditional Boolean expressions. Even though, Eq. (1) still holds in any case:

\[ R \bowtie S \subseteq R \times S \]  

Such two-way similarity joins have been extensively investigated in the research community as \([1, 29, 7, 18, 30, 22, 2, 3, \text{etc.}]\). Furthermore, due to the evolution of Geographical Information Systems (GISs), large collections of spatial data having various thematic contents are currently available. As a result, the interest of users is not limited to simple (two-way) spatial joins (and selections), but complex query types that implicate numerous spatial inputs become more common \([23]\). Although lots of research activities have been carried out to deal with the problem of two-way similarity joins, there is only limited work that has been concentrated on processing and optimization of multi-way similarity and spatial joins such as \([20, 27, 24, 23, 31, 26]\). In a recent publication \([4]\), the author proposed a classification scheme for two-way similarity join approaches based on the view point as well as the processing methods of the problem. Here we will base on this classification scheme (with some modifications) to classify and briefly discuss approaches to two-way and multi-way similarity joins. Figure 1 presents the modified classification scheme.

![Figure 1: A similarity join classification scheme](image)

As we see in this modified classification scheme, approaches to similarity joins are of two main categorizations that are based either on a distance range $\varepsilon$, or on nearest neighbor processing techniques. The former (also called $\varepsilon$-join) is intuitive and formally defined as follows' given two datasets $R$, $S$ and a non-negative real $\varepsilon$:

\[ R \bowtie_{H_{\varepsilon}} S \]

\(^1\) Note that, spatial joins also include similarity joins for spatial/multidimensional objects. In this paper, we shall mainly focus only on similarity joins. With spatial joins, we have more possible join predicates such as intersect(), contain(), adjacent(), northwest(), etc. (see, e.g., \([16, 17]\)).
R \bowtie \subseteq S = \{(r, s) \in R \times S : \|r-s\| \leq \varepsilon\}

(2)

The join operation result of formal definition (2) can be represented by the SQL like notation as follows:

```
SELECT * FROM R, S
WHERE \|R.object – S.object\| \leq \varepsilon
```

(3)

The distance range join has become most popular and best evaluated join. In the literature, it is also usually called the similarity join. Its application range is wide and one of the most important application is for data mining (clustering problem) but with distance range self join, in which two datasets participating in the join are identical, i.e. R=S. Besides, a main disadvantage for users of the distance range join is that the result cardinality of the join is hard to manage: If \(\varepsilon\) is too small, the result set may be empty, but if \(\varepsilon\) is too large, all pairs from the Cartesian product RxS will be produced [4]. Obviously, the worst case complexity when processing the distance range join is at least \(O(|R|\times |S|)\).

Approaches to similarity joins based on NN processing techniques are commonly in the form of k-closest pair queries or k-NN joins. In the database context, such approaches were introduced by Hjaltason and Samet [18]. Formally, k-closest pair query result set is the smallest subset of RxS that contains k pairs of objects and satisfies the condition as shown in Eq. (4). Its SQL like notation is described as shown in statement (5).

\[
\forall (r_i, s_j) \in R \times S \forall (r_k, s_L) \in R \times S \setminus R \bigcup_{k \neq P} S:
\|r_i-s_j\| \leq \|r_k-s_L\|
\]

(4)

```
SELECT * FROM R, S
ORDER BY \|R.object – S.object\|
STOP AFTER k
```

(5)

Interestingly, although STOP AFTER clause was introduced in [6], it has been still not yet supported as a primitive operation in commercial DBMSs. To accomplish this “stop after” functionality, users can employ cursor-based approaches at the application level, which have been supported in all DBMSs. Also, the k-closest pair queries can also be solved without knowing the k value in advance (cf. [28, 18]). In this case, the results are incrementally ranked instead.

Besides, k-NN joins are also of interest. Basically, this kind of join combines each object of the first dataset R with its k-nearest neighbors in the second dataset S. A formal definition for two-way k-NN join is as follows [4]: The result set of a k-NN join between two datasets R and S is the smallest subset of RxS so that for each object of R it contains k objects of S and the condition as shown in Eq. (6) holds. Moreover, its SQL like notation is also given as in statement (7) with an optional STOP AFTER clause to manually limit the result cardinality.

\[
\forall (r_i, s_j) \in R \bigcup_{k \leq P} S \forall (r_k, s_L) \in R \times S \setminus R \bigcup_{k \neq P} S:
\|r_i-s_j\| \leq \|r_k-s_L\|
```

(6)

```
SELECT * FROM R, S
GROUP BY R.object
ORDER BY \|R.object – S.object\|
[STOP AFTER n]
```

(7)

1. In [18], the authors called k-closest pair queries distance join, and called k-NN join distance semi-join.

Main applications of k-NN joins are problems related to the clustering of data objects (cf. [4]). Note that, the k-NN join operation is not symmetric, meaning that:

```
R \bowtie_{k-NN} S \neq S \bowtie_{k-NN} R
```

In addition, concerning approaches to similarity joins based on NN processing techniques, in [4], the author did not consider cases that are similar to k-NN joins or k-closest pair queries as discussed above but in which the result set of similarity joins may consists only of good matches (but not the best ones) [20] or some of join predicates are special predicates that make the system have to do further processing steps (post-filtering steps) before outputting the result. This is the case appeared in [24] and all variants of an approach introduced in [13] as [15, 25]. In those approaches, the problem must be death with is that one or more additional predicates on attributes (may be the key as well) of involved tuples must held, except for join predicates/computations have to be dealt with as usual. Therefore, we name such similarity joins complex similarity joins and classify them into a new sub-class in our classification scheme as shown in Figure 1, i.e. approximate/good match/extended k-NN joins and k-closest pair queries. This problem is also one of the most interesting problems arising in the VQS [19]. It introduces more new challenges as well as opportunities to further improve and enhance the VQS, one of flexible query answering systems (FQASs). In this paper, we are going to present an approach for solving such complex similarity joins in the VQS and then propose a generalized approach to this problem, which can also be applied to other systems and application domains. The problem of approximate complex similarity join processing will also be addressed.

The rest of this paper is organized as follows: Section 2 presents the vague/complex similarity join problem in the VQS. In section 3 and 4, we introduce efficient approaches adapted from our approaches to complex vague queries [10, 11] and approximate complex vague queries [12] for addressing the problem of complex and approximate complex similarity join processing in the VQS. Discussions and ways to parallelize the join processing will be given in section 5. In section 6, we generalize our adapted approaches in order to facilitate them in being applied to other application domains and systems. Last, section 7 presents concluding remarks for the paper.

2. VAGUE JOINS IN THE VQS

In [9], we discussed the VQS in detail and pointed out all shortcomings existing in that system. Basically, the VQS [19] has been introduced to deal with the problem of empty result sets in the conventional DBMSs. It extends the query facility of the conventional DBMSs with similarity search capabilities. Concretely, when available data in a conventional database do not match a user’s query precisely, the corresponding DBMS will only return an empty result set to the user. This limits the applicability of the conventional DBMSs to domains where only crisp answers are meaningful. In many other application domains, however, users also expect not only the crisp results returned but also some other results that are relevant or close to the query in a certain sense. The VQS is such a FQAS. It has been designed to work “on top” of the conventional DBMSs in
order to return to users tuples of the query relation/view that do not match the query criteria exactly.

The main features of the VQS are to employ the concept of NCR-Tables (Numeric Coordination Representation Tables) and to introduce a new query language called the Vague Query Language (VQL) that is an extended version of the SQL. NCR-Tables store (multidimensional) semantic metadata of attributes of the query relation/view. The VQL has introduced a new operator “IS” (“similar to”) to formulate similar (vague) queries in the system. Unfortunately, the operator “IS” cannot be used directly in a join condition. An extension of the VQL has been carried out to realize vague joins in the VQS [20]. Nevertheless, for the sake of reducing costs, users can only get the good matches, not the best ones and the new system gives no guarantee about the goodness of the results. To make the problem clearer, we depict in Figure 2 a graphical view over the complex similarity (or vague) join problem in the VQS. Besides, the approach to complex similarity joins7 [20] has had the following main weaknesses:

• It returns only good match results for a given complex similarity join. That means there may have other better match combined tuples but they may be dismissed during the join processing. This is one kind of false dismissals.
• The user must provide a value of the parameter CONFIDENCE for each join predicate to express the expectation of the closeness degree of the join result with respect to each join predicate. However, eventually, the user will not receive any feedback about the goodness of the results returned.
• Choosing a right/good value for CONFIDENCE of each join predicate is not intuitive and convenient for naive users. This is also difficult as the returned results do not satisfy the user and he/she has to tune such CONFIDENCE values for join predicates.
• The running join must be performed again, i.e. from scratch, whenever the value of some CONFIDENCE is readjusted and re-submitted. This could lead to a low response time for the join, especially with large datasets, which cannot be accepted in many cases, e.g. in interactive query processing environments.

Recently, in [24], the authors presented solutions to both problems of complex and approximate complex similarity joins. Nonetheless, although their solutions are very similar to our approaches will be shown later, they are essentially different. Our approaches are based on (adapted from) our other algorithms/approaches for complex multi-feature NN (M-FNN) queries and approximate complex similarity queries, which have been introduced in [10, 11] and [12], individually. In [12] we also pointed out the differences between our approaches and ones proposed in [24]. The next two subsections will present our approaches adapted for the concerned problems.

3. SOLVING VAGUE JOINS

In [10], we proposed an approach to finding k-NNs of a complex M-FNN query Q of n query conditions q. The problem of complex similarity join processing (CSJP) as posed above can be viewed as the problem of processing multiple complex M-FNN queries sequentially7. In detail, with the CSJP problem, if we consider n join predicates of the join condition as n query conditions in the problem of complex M-FNN query processing.

---

5 In fact, [10] is an improvement approach of [11], and so we will refer to only [10] hereafter.
6 It is interesting to note here that, although there are only two tables participating in the join, the problem of vague joins in the VQS still belongs to the multi-way join class because it must deal with join predicates over multiple NCR-Tables together with the two basic tables Tab1, Tab2.
7 In [20], the authors called such complex similarity joins vague joins.
8 Without loss of generality, we assume here that all tuples in the first joining relation/view will be processed to complete the join. That means the STOP AFTER clause is not considered here or its parameter value m=k.h, where h is the tuple number in the first joining relation/view.
and consider each tuple in the first joining table, say R, as the complex M-FNN query, our CSJP problem here then becomes as simple as the problem that we have already solved in [10].

Note that, except for n join predicates, one more condition ("extended" join condition) is always implied during the join processing: n NCR-Values satisfying these n join predicates must all belong to a certain tuple in the second joined relation/view, i.e., as called in [10], there is at least an appropriate tuple concerning a set of n NCR-Values in the query relation/view. In Figure 3 below we present high-level abstract steps to realize this idea using the available result, i.e., the approach to complex M-FNN queries introduced.

Input:
- Two joined relations/views R, S (assume that R is the first one in the statement submitted), in which R has h tuples and S has g≥k tuples. Here k is the number of tuples in S that each tuple in R will join to.
- A join condition Q of n join predicates qi (i=1, 2… n).
- Assume each feature space (or NCR-Table) related to Q is managed by a multidimensional index structure Fi as illustrated as in Figure 2.

Output:
- The resulting relation/view named myResultTable of the extended k-NN join problem dealing with.

Processing:
For each tuple ri (i=1…h) in R we carry out the following steps:

Step 1: Apply the k-NN algorithm as introduced in [10] for complex M-FNN queries to find k-NNs of ri in S. Here, n join predicates are equivalent to n query conditions of a complex M-FNN query; ri is considered as the query and S is considered as the query relation/view.

Step 2: Combine ri with its k-NNs in S to form k combined tuples and add these k new tuples to the resulting relation/view myResultTable.

Figure 3 Solving the CSJP problem in the VQS

We can observe that the adapted approach as described in Figure 3 overcomes all shortcomings existing in the previous approach proposed in [20] as discussed above. Depending on the base of our approach to complex M-FNN queries, the adapted approach to the CSJP problem results in the best match tuples for the join, i.e., no false dismissals occur. Here, the user does not need to provide or tune values for the parameters CONFIDENCE and thus the system also does not need to perform the join again after each change to such parameters.

4. SOLVING APPROXIMATE VAGUE JOINS

Almost work that has been done on the topic of two- or multi-way spatial joins is in general focused on the retrieval of all exact solutions (e.g. the best matches for the CSJP as just discussed above). As pointed out in [26], depending on the query and data properties, exhaustive processing of multi-way spatial joins can be prohibitively expensive due to the exponential nature of the problem. Furthermore, as similar to the main purpose that the VQS and other FQASs (see [9]) should address, if there is no any exact solution, the result will be empty even though there may exist solutions that match the query very closely. In this section, we will present an adaptation of our research results introduced in [12] in order to solve the problem of approximate complex similarity join processing (ACSJP) in the VQS.

Figure 4 Solving the ACSJP problem in the VQS

The ACSJP problem in the VQS is concisely described as follows: Given two relations/views R and S, and a positive real ε, join each tuple in R to its approximate k-NNs in S. The approximation between tuples is calculated based on a monotone scoring aggregation function of parameters di (i=1…n) which are distances from NCR-Values of joining tuples to NCR-Values of joined tuples with respect to n join predicates in a join condition. From this description of approximate complex similarity joins (ACSJs) appearing in the VQS, we see that the ACSJP problem can be altered to become the problem of processing multiple approximate complex similarity queries that we have solved in [12]. In fact, in [12], an approximate complex similarity query is posed by the user for the querying over a query relation/view, whereas with the ACSJP problem here we already have h approximate complex similarity queries
with respect to \( h \) tuples of the first joining relation/view \( R \), and \( S \) will take role of the query relation/view. Therefore, similarly to section 3, we present in Figure 4 above an approach adapted from one that aims at finding approximate \( k \)-NNs of a complex similarity query for the ACSJP problem in the VQS. The adapted approach is easy to understand so we shall not discuss more here. Furthermore, the correctness as well as efficiency of this adapted approach is the obvious consequence followed by those of the generalized \( \varepsilon \)-ISA introduced in [12] (also see [9]).

5. DISCUSSIONS

As shown in two previous sections, approaches for addressing our concerned problems in this paper are derived from our other approaches to complex and approximate complex M-FNN queries, which have been presented and their efficiency and effectiveness have also been evidenced by the intensive experiments with both synthesis and real multidimensional datasets. In this paper, we will therefore not give more experimental results for the CSJP and ACSJP problems presented above. A lot of valuable discussions and information about the experiments can also be found in [9].

Besides, with respect to the CSJP and ACSJP problems discussed, we also recognize that they can be processed in parallel ways very suitably. This will lead to the more cost-effectiveness and, especially, reduce CPU-cost significantly. From the graphical overview of the concerned problem as shown in Figure 2, we can intuitively see that data-partitioning or task-partitioning based parallelisms are both very suitable for the CSJP and ACSJP problems. Specially, a mixed strategy of both of them can also be applied: The search over each NCR-Table can be assigned to a computing element in the parallel system (i.e. the data-partitioning based parallelism), and the next tuple in the first joining relation/view \( R \) can be handled by the task-partitioning based parallelism policy, i.e. the first available computing element in the parallel system will fetch this tuple for the processing. Developing an efficient approach to process in parallel (approximate) complex similarity join problems is one of our future researches of great interest.

6. A GENERALIZATION OF VAGUE AND APPROXIMATE VAGUE JOINS

If we investigate more deeply the problem of extended \( k \)-NN joins/k-closest pair queries, we would perceive that all solved problems which are relevant to multi-way similarity (but not spatial) joins have a common nature: Apart from join predicates as usual, there are some additional (implicit or explicit) predicates included in the join condition that each tuple in the final result set must satisfy. Specially, such join-relevant problems have been particularly focused on modern database application domains, e.g., [13, 20, 21, 25, 15, 24, 14], etc.

Here, we want to introduce general concepts in order to formalize the problem of extended \( k \)-NN joins with multiple joined tables and multiple user-defined join predicates over multiple attributes. From analyses in previous sections and above we can simply describe the general problem of extended \( k \)-NN joins as follows: Given \( m \) multidimensional datasets \( T_1, T_2, \ldots, T_m \) and a set of multidimensional pivot objects \( Q = \{Q_1, Q_2, \ldots, Q_m\} \) with respect to these datasets. Data objects in each dataset \( T_i \) \((i = 1 \ldots m)\) are ordered according to their distance to a corresponding pivot \( Q_i \), i.e. ordered according to \(|Q_i-T_i.Object|\). Given a scoring aggregation function \( S \) defined over the set \( C = \{|Q_1-T_1.Object|, |Q_2-T_2.Object|, \ldots, |Q_m-T_m.Object|\} \) so that \( S \) is monotone with respect to each of its arguments (see [13] for more information about monotone functions). Let \( \theta \) be a certain Boolean predicate on the set \( A = \{T_i.Object, T_1.Object, \ldots, T_m.Object\} \). Our goal is to retrieve \( k \) m-tuples \( (T_1.Object, T_2.Object, \ldots, T_m.Object) \) so that \( \theta(T_1.Object, T_2.Object, \ldots, T_m.Object) = \text{TRUE} \) and with all other m-tuples \( (T_1.Object', T_2.Object', \ldots, T_m.Object') \) that also satisfy the condition \( \theta(T_1.Object', T_2.Object', \ldots, T_m.Object') = \text{TRUE} \), we will have \( S(|Q_1-T_1.Object|, |Q_2-T_2.Object|, \ldots, |Q_m-T_m.Object|) \leq S(|Q_1-T_1.Object'|, |Q_2-T_2.Object'|, \ldots, |Q_m-T_m.Object'|) \). The \( \theta \) predicate can be either implicitly or explicitly pointed out. In the other words, our overall goal is to retrieve \( k \) combined tuples from \( m \) given datasets for each set of pivot objects \( Q \) so that these \( k \) combined tuples have the smallest aggregation distances to all objects in \( Q \). This is a \( k \)-NN problem with multiple queries (i.e., pivots), multiple features (i.e., multidimensional datasets), and multiple constraints (i.e., the relationship predicate \( \theta \), which can also be composed by multiple sub-predicates \( \theta' \)).

It is an intuition from the above description that complex similarity joins (or vague joins) in the VQS are just a special case when the multidimensional datasets are substituted by NCR-Tables, the set of multidimensional pivot objects \( Q \) is sequentially changed according to each tuple \( T \) in the first joining relation/view (here, the NCR-Value of each involved attribute value of the joining tuple is a pivot object), and the additional relationship predicate \( \theta \) is understood (i.e., implicit) as mentioned before due to the special inputs and specifications of the problem. Note that, because of the operator \( IS \) of the VQL language, with respect to each set of pivots \( Q \), objects/NCR-Values in each NCR-Table are always ordered according their distance to the corresponding pivot in \( Q \).

In general, in the above description of the extended \( k \)-NN joins, the relationship predicate \( \theta \) can be decomposed into a subset of Boolean predicates \( \theta' \) that can also be arbitrary user-defined predicates depending on their concerned applications. If this set of join predicates \( \theta' \) contains only the equivalence relation on some attributes of the involved objects, the join is called equi-join. This is a similar case to the one of a definition presented in [24]. In addition, an approach to combining fuzzy information from multiple systems [13] (and all of its variants as [15, 25, etc.]) can be seen as the unique equi-join in which the equi-join is defined on the key attributes of concerned objects only. In the other words, our above description can be seen as a generalization of extended \( k \)-NN joins; hence complex/approximate complex similarity joins with respect to the classification scheme as shown in Figure 1.

The general problem of extended \( k \)-NN joins as discussed above is a question of great interest because it supports arbitrary user-defined join predicates (i.e., very flexible in being applied to various application domains) and has a broad application range, especially in many modern database applications where multidimensional data take the role as fundamental elements (e.g. in multimedia/image databases, GISs and TISs - tourist information systems, etc.). From our analyses in previous sections in this chapter, solving this general problem of extended \( k \)-NN joins turns out to be simple now. Generally, to address the basic problem we can employ the ISA (Incremental
hyper-Sphere Approach) as introduced in [10] in the similar way to one that has been employed to deal with complex similarity joins (vague joins) in the VQS as presented in section 3. Furthermore, dealing with the general problem of approximate extended k-NN joins is also not too complicated if we employ the ε-ISA introduced in [12] in the similar way to one that has been adapted in section 4.

7. REMARKS AND CONCLUSIONS

In this paper we have focused on approaches to complex and approximate complex similarity join processing. First of all, we discussed, identified and classified previous researches related into two main classifications: distance range and nearest neighbor based approaches. Our concerned problems are in a branch of nearest neighbor based approaches, that is named approximate/good match/extended k-NN joins and k-closest pair queries. Therein, apart from the similarity join predicates as usual, we must deal with additional (join) predicates that make the system have to further carry out expensive processing steps before outputting the final results.

The VQS system, a very promising FQAS, also introduces such a challenge with vague joins. Unfortunately, one previous research published in [20] into this problem for the VQS still has many deficiencies that lead it to become unpractical. Therefore, we have firstly developed efficient approaches for dealing with both the complex and approximate complex similarity join problems in the VQS. Our approaches are derived from our previous research results for complex and approximate complex multi-feature nearest neighbor queries. These approaches have been evidenced to be very efficient and they are now also the state-of-the-arts in these areas. Later, we generalized the problem and proposed generalized approaches to efficiently solving the complex and approximate complex similarity join problems. The capability of parallelizing our concerned problems is also discussed and it is also one of our future research activities of great interest.

8. REFERENCES

[29] H. Shin, B. Moon, S. Lee. Partition-Based Similarity Join in High Dimensional Data Spaces. DEXA’02, 741-750