The University of Melbourne

COMP90050 - Advanced Database Systems

Topic Survey - Group W8-9

Survey of Top-k Queries in Databases

Cameron Chandler - 993990, Dongcheng Ding - 952328

Jiawei Luo - 1114028, Jie Zhang Soon - 1112427

Abstract

The past decades have seen the development of top-k query algorithms in a range of contexts. Many approaches exist for designing such algorithms, and there are several key properties to consider to in each specific kind of query. This survey comprehensively compares, categorises, and contrasts some of these most fundamental query algorithms to provide a basis for understanding what problems they are trying to solve, and how they do so. These algorithms are compared by query type, data certainty and data access.

1 Introduction

When querying a database, users are commonly only interested in the top results as measured by a scoring function. For example, when considering kitchen appliances a customer may like to see the top 10 results by some function of cost and warranty. Explicitly, if x_1 and x_2 represent warranty and cost respectively, the scoring function $f(x_1, x_2) = x_1 - x_2$ could be used to rank appliances, (higher scores are better).In the area of top-k queries it is convention to assume attributes have been scaled to have values between 0 and 1. The naive approach to this problem is to assign every appliance a score, sort the appliances by score, and retain the top-k appliances. This approach becomes unfeasible in the face of enormous data sets and expensive join operations.

The aforementioned example was a top-k *selection* as the query was performed on a single data table where each score was associated with one object. However, there are other types of top-k queries that also require efficient algorithms. Top-k *joins* involve at least two relational data tables where the result will contain the top-k join results with the highest combined score. Lastly, top-k *aggregate* queries involve grouping objects together by some common attribute value and scoring the group as a whole, naturally resulting in the top-k groups being output. As an example consider taking a database of individual houses and retrieving the top ten most expensive suburbs as measured by median house price.

There are two types of databases that require different approaches to answer top-k queries. The most fundamental algorithms are designed to address databases with *certain data* where it is assumed that the data is complete and correct. Alternatively, there is *uncertain data* that usually arises from inaccurate or incomplete data sources, transformations between data granularity, or even when external factors like privacy protection disallow accurate data to be queried. Algorithms that function in such contexts have seen increasing relevance in the field as large scale sensor networks and distributed databases become more common.

Top-k query algorithms typically require at least one sorted attribute list L associated with the data table(s); Many algorithms required such lists for all attributes. These lists will be henceforth referred to as *streams* and are composed of two parts: The object identifier (oid) and the attribute value. Methods that access these sorted lists are said to use sorted access. Many query algorithms will sequentially access the next item from each stream. For two streams with sorted cost and warranty values respectively, an algorithm may look at the first cost, then the first warranty, then the second cost and the second warranty etc. Alternatively, if given an oid an algorithm can perform a *random access* where all associated attributes for a given object are located. This is usually much more expensive than sorted access depending on the data structures and indexing in use. Because of this, some algorithms forgo random access altogether and use just sorted access.

Most top-k algorithms are designed to only work with *monotone* scoring functions. Formally, a function f is monotone if $f(x_1, ..., x_m) \leq f(x'_1, ..., x'_m) \iff x_i \leq x'_i \forall i$. This means if object A's attribute set is element-wise entirely greater than or equal to object B's, then $f(A)$ must be greater than or equal to $f(B)$. This constraint is in place as many common stopping mechanisms rely on the fact that attributes are sorted and the function is monotone to guarantee algorithmic correctness with efficient early termination. Initially this design may appear limiting, but in reality most simple user-defined scoring functions satisfy this property (such as averages, sums and maximums). Some algorithms specifically cater for the case when a desired scoring function is not monotone, but instead could have any form. An algorithm designed for this context is said to work with *generic* scoring functions.

2 Related Works

In this section, a variety of top-k query methods are described. These methods are first simply classified for deterministic or uncertain data. More detailed comparisons will be made in section 3.

2.1 Certain Data

Algorithms designed for certain data can assume that data is complete and correct, greatly simplifying the problem of top-k queries.

2.1.1 Threshold Algorithm (TA)

TA [5] is a fundamental algorithm in the realm of top-k queries due to both its early development and its simplicity; Many of the algorithms in this survey are based upon it. The basic idea behind TA is to find new candidates with sorted access, calculate candidate scores with random access, and stop when it is impossible for any new candidates to be better than the current k. TA works in two phases:

1. Sequentially access the next object o in each of the m streams L_i. Do random access to find all attributes $\{x_1, ..., x_m\}$ of object o. Compute score $f(o) = f(x_1, ..., x_m)$. If $f(o)$ is in the k highest thus far, store $(o, f(o))$ to avoid redundant random access, (only the top-k are stored).

2. For each sorted list L_i, $\underline{x_i}$ is the last attribute accessed. Let threshold $\tau = f(\underline{x_1}, ..., \underline{x_m})$. If k objects with score $\geq \tau$ are stored, then stop. Else return to phase 1.

For an unseen object to achieve the highest possible score, it would need to have attribute values $\{\underline{x_1}, ..., \underline{x_m}\}$, (based upon the assumption of a monotonic scoring function). This means that the threshold τ is the upper bound for unseen object scores. At termination, the algorithm has found k objects greater than or equal to this, so it is impossible that any unseen object could be higher and belong in the top-k.

TA uses only $O(k)$ memory since it only maintains the top-k results at any given time. However, this bounded buffer can cause more random accesses to occur than otherwise necessary and poor performance in some special cases. This is a prominent weakness of the algorithm, especially when random access is significantly more expensive than sorted access.

The threshold can be improved to reduce the running time of TA by a factor of $O(m)$ as is achieved by the Best Position Algorithm (BPA) [1]. For each stream L_i, BPA maintains a list of attribute positions that have been accessed P_i. The *best position* bp_i is defined as the greatest position in P_i such that all positions in L_i up to bp_i have also been accessed. Let $\underline{x_i}$ be the attribute value at position bp_i and threshold $\tau = f(\underline{x_1}, ..., \underline{x_m})$. Then the threshold is used the same way as in TA.

2.1.2 Stream-Combine

In some environments random access may be very expensive or even impossible. Stream-Combine [6] addresses these issues by avoiding random access altogether. The streams are combined in five phases:

1. Get the first object o and attribute value x_i from each of the m streams L_i.
2. Initialise a data structure to hold candidate objects. Each row will contain objects' observed values and upper bounds for unobserved values. Place the first objects and values into the table.
3. For each stream L_i, let $\underline{x_i}$ be the last attribute accessed. The upper bounds for unseen values are set to the associated $\underline{x_i}$. Score all objects using the current observed and unobserved values.
4. If all attributes of object o have been observed and it has the highest candidate score, remove it from the table and output it as the next best object. If k objects have been output, then stop.
5. Get the highest scoring candidate o, and access the next value in the sorted list L_i corresponding to any of its unobserved attributes x_i. Add this to the candidate table and return to phase 3.

To further optimise the algorithm and promote early termination, *expansion indicators* can be used to select the most promising stream to expand next.

2.1.3 Linear Programming approach for Threshold Algorithm (LPTA)

LPTA [4] uses pre-computation to speed-up top-k queries, thereby introducing a trade-off between space and time efficiency. Although an adaption of TA, LPTA is more than a trivial extension and utilises linear programming on views of previous top-k query results. LPTA takes scoring function f and M views V_j that were calculated with scoring functions $f_j \forall j \in \{1, ..., M\}$. The algorithm works in three phases:

1. Sequentially access the next object o from each of the sorted views V_j. Do random access to get the attributes x_i of each object.
2. Score each object as $f(o) = f(x_1, ..., x_m)$ and maintain only the top-k objects so far.
3. Use linear programming to calculate a threshold τ representing the upper bound for scores of unseen objects. Terminate if k objects have been seen with scores $\geq \tau$.

Each round of phase 1 and 2 encompasses one iteration. After d iterations, the last object read from each view V_j is o_d^j with score $f(o_d^j)$. The current k^{th} top score is $\text{Topk}_{\text{min}} = f(o_k)$. Any unseen objects o from view V_j satisfy $f_j(o) \leq f(o_d^j)$, and have all attributes are $0 \leq x_i \leq 1$. This system of inequalities defines a convex region of possible unseen objects. Linear programming is used to calculate τ as the maximal value of f in the region, and therefore the largest score that any unseen object could have.

In the case of high-dimensional databases, LPTA can suffer from significant computational overhead due to calling linear programming subroutines at every iteration to calculate upper bounds on the achievable values of candidates. LPTA+[17] implements a check on top of LPTA to determine if the threshold τ could change and therefore if it needs to invoke linear programming, thus reducing the computational overhead.

2.1.4 J*

J* [13] enables top-k queries over joins with arbitrary predicates, supporting both multiple join levels and nested join hierarchies. Based on the A* search algorithm, J* maintains a priority queue of partial and complete join conditions to enable incremental joins, only computing the most promising join results. This avoids redundant joins where the result could not possibly be in the top-k. J* maintains the exact score of completely joined objects and calculates an upper bound for partially joined tuples to determine which join are worth pursuing.

Figure 1 demonstrates how the J* algorithm begins to find the top-k results in a join of tables A and B and scoring function $f(A, B) = 0.7 \times A + 0.3 \times B$. Note that A and B must be sorted by score. The first join ($A0$, $B0$) is initially placed in the priority queue (sorted by score upper bound) and the maximum possible score for A and B is assumed to calculate an upper bound for the join result. Onto step 1, two new items are added to the priority queue: One from locking in the first oid and randomly accessing its score, and one from trying the next object from table A. The new upper bound for the score is recalculated with the randomly accessed value and the maximum possible value for ($A0$, $B0$) is now 0.9. This process continues until k fully joined objects have been taken off the priority queue.

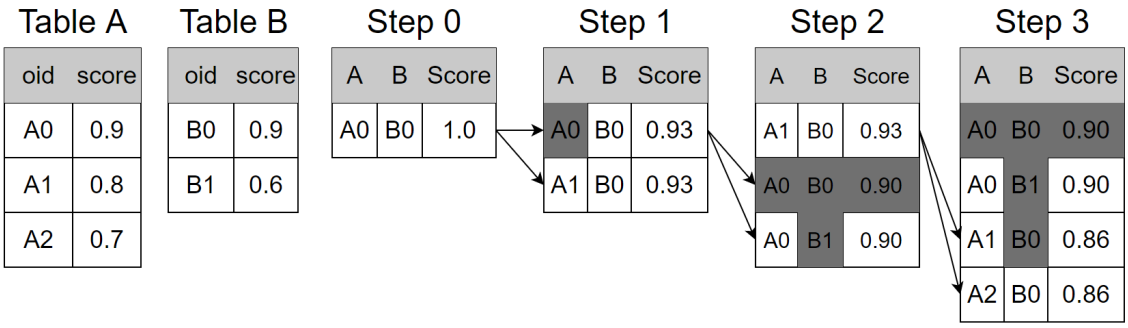

Figure 1: J* Algorithm in Progress

2.1.5 Minimal Probing (MPro)

MPro [3] is designed to work efficiently with a scoring function f that is based on expensive Boolean predicates. In such a context, the expensive operation of determining whether or not an object satisfies the predicate is referred to as a *probe*, and the minimal number of probes is desired. The algorithm works on a principle of *necessary probes* to reduce the cost of exploration. In short, a probe becomes *necessary* if the top-k results cannot be obtained without performing the probe, regardless of any other probes.

Much like J*, MPro initialises a priority queue that is at all times sorted on the upper bound of object scores. The algorithm will pop objects from the top of the queue and probe the next unevaluated predicate to update their upper bounds before reinserting them into the queue. Popped objects with all predicates evaluated must be in the top-k and are retained for the final results. Termination occurs when k objects are output as such.

The algorithm employs a schedule to determine the order of predicate evaluation. The schedule should order predicates from cheapest to most expensive as the first predicates will be evaluated the most. Determining this expense is NP-hard, so a greedy search algorithm is utilised for an approximation.

2.1.6 Rankaggr

Rankaggr [10] is a principled framework for efficient processing of ad-hoc top-k aggregate queries in OnLine Analytical Processing (OLAP) environments. To deal with such complexity, the problem is broken down and addressed with two basic principles: Group-Ranking and Tuple-Ranking. These provide theoretical support for optimisation of aggregate queries.

The algorithm incrementally explores objects within groups to avoid redundant computation. The Group-Ranking principle relates to ordering groups when selecting the next object o to process. The principle states that a group in the current top-k ranked must be further explored if not yet fully evaluated. This principle saves query process time that would be otherwise spent on evaluating aggregations that could not possibly be in the top-k. When each of the current top-k groups are fully evaluated, the algorithm terminates. The Tuple-Ranking principle addresses the order of object exploration within in a group (that was selected with the Group-Ranking principle). After following the Group-Ranking principle to avoid unnecessary object evaluation, the details of finding an optimal object exploration order are not necessary to understand how Rankaggr works.

Based on those two principles, the Rankaggr operator maintains both a queue of groups and the individual object scores to establish upper limits. Each iteration, the upper limit is updated until termination. With the introduction of new theory and query operators, Rankaggr has brought significant performance improvements. Although restricted to monotone aggregate functions, Rankaggr was the first algorithm framework to support ad-hoc top-k aggregation, and remains useful for data warehousing and decision support applications.

2.1.7 OPT*

OPT*[19] is the only algorithm covered in this survey with the ability to work with generic scoring functions. It reframes top-k queries as optimisation problems, and then reframes the optimisation problem as a search problem. OPT* introduces Boolean function $\mathcal{B}(o) \in \{0, 1\}$ which takes an object o and returns 1 if the object satisfies the query predicates (else 0). Then a goal function G is defined as $G(o) = f(o)\mathcal{B}(o)$. By multiplying the scoring function by the Boolean predicate function, the goal function is maximised only when an object satisfies the predicates, otherwise it has value 0. [2]

The algorithm then transforms this problem of maximising $G(o)$ into a search problem where the index space is instead viewed as paths and discrete search states at different levels of granularity. Answering the top-k query is equivalent to maximising $G(o)$ which is equivalent to minimising the path length, hence creating a context where shortest-path algorithms can be of use. The core of OPT* is based upon A* with correct

and admissible heuristics to satisfy the conditions of the optimisation. This is how OPT* can guarantee correctness whilst maintaining efficiency in such a challenging domain.

OPT* does not function well when the shape of $G(o)$ is relatively flat with few or no regions of distinction. For example, if few objects satisfy the predicates, $\mathcal{B}(o)$ will often be 0 and therefore so will $G(o)$. In such contexts the A* core of OPT* struggles to determine the next best direction to search.

2.1.8 KLEE

KLEE [12] is a high performance and flexible framework for top-k queries in distributed databases. Network latency, bandwidth consumption, and local computation all significantly impact query cost in such environments. KLEE assumes that the index list of data attributes are distributed among peer nodes.

Initially, each of the M peers P_j creates a candidate list with their local top-k objects. Each peer then locally calculates the expected bandwidth savings by filtering their candidates using communication with peers. The next phase utilises peer communication to reduce the size of local top-k candidate lists. Finally, the final top-k result is calculated by collating and processing all peers' candidate lists.

KLEE can bring robust performance improvements for top-k queries over distributed systems, reduce network bandwidth usage, query response time, increase accuracy and achieve lighter peer-to-peer load. Query algorithms in distributed systems are hypothesised to only become more relevant in the coming decade as organisations turn to distributed storage.

2.2 Uncertain Data

The algorithms described above are designed to handle certain data, however uncertain data is prevalence in practice. Three fundamental approaches to answer top-k queries in such environments will be described here. *Possible world* semantics are used as an aide to describe the results of these queries and define a space of possibilities with associated probabilities.

A possible world is defined with the following [18]: A possible relation R^p is defined by the tuple $< R, p, \mathcal{C} >$, Where R is a certain relation, p is a probability function that maps each object o to a probability between 0 and 1, and C_i is a partition such that $\sum_{o \in C_i} p(o) \leq 1, \forall C_i \in \mathcal{C}$. Note that $\mathcal{C}$ is simply the set of possible ways to validly partition R such that each object o only appears in one partition C_i. A possible world W is a certain relation and $W \subseteq R$.

U-Topk [16] and **U-kRanks** [16] are two query algorithms that represent different interpretations of what it means to get the top-k results in an uncertain database. U-Topk provides the top-k result that is most probable in the aggregate. However, this does not guarantee that each object in the top-k result will be the most probable. For instance, the first object may not be the most likely object to be the first object, despite being part of the most likely overall result. Contrary to this, U-kRanks computes the most probable object for each position $1...k$. However this means that the result is not guaranteed to be the most likely result. The trade-off relies on the user's desired interpretation of a top-k result with uncertain data.

The space of possible worlds grows exponentially with the data size for these methods. Combined with a lack of heuristic pruning, they are limited in their ability to process possible worlds. Their stopping mechanisms rely on upper bound scores of other candidates and requires specifying the states of all possible worlds. This causes U-Topk and U-kRanks to be inefficient.

PT-k [7, 8] computes a threshold p and only considers k objects from all possible worlds where the probability of candidates exceeds p. PT-k does not find the space of all possible worlds and avoids duplicate retrieval with pruning strategies, thus reducing the required computational complexity.

2.2.1 Multisimulation (MS_TopK)

MS_TopK [14] employs Monte Carlo simulation on the top-k buckets to rearrange the ranking of the candidate results. Like the previous methods, it works on uncertain data where each object has an associated probability. Although MS_TopK guarantees a correct ranking of objects, it only provides an approximate probability for each object. These probabilities are calculated only to the precision required to guarantee ranking correctness.

The MS_TopK algorithm utilises precision boundaries for each object o_i initialised as $[a_i^0, b_i^0] = [0, 1] \forall i \in \{1, ...n\}$. At each step the new boundary can shrink such that $[a^{N+1}, b^{N+1}] \in [a^N, b^N]$, and convergence is guaranteed so $\lim_{N\to\infty} a^N = \lim_{N\to\infty} b^N$.

The algorithm works by detecting candidate objects that have overlap in their probability precision boundaries and *simulating* them. That is, but sampling an object instance out of all possible worlds. This process gradually tightens the associated bounds of $[a_i, b_i]$ for each object o_i until the correct ranking can be guaranteed (as there are no more overlaps to consider).

3 Comparison of Key Approaches

Table 1 shows the different properties of the algorithms described in this paper. The categorisation in this survey is based on the definitions of the methods, many of which improve on the most basic method, TA. These are classified here as **TA-based algorithms**.

	Top-k Query Type			Data Certainty		Data Access	
	Selection	Join	Aggregate	Certain	Uncertain	Sorted	Random
TA & BPA	✓			✓		✓	✓
Stream-Combine	✓			✓		✓	
LPTA & LPTA+	✓	✓		✓		✓	
J*		✓		✓		✓	
MPro	✓			✓		✓	✓
Rankaggr			✓	✓		✓	
OPT*	✓	✓		✓			✓
KLEE	✓			✓		✓	✓
MS_TopK	✓	✓	✓		✓		✓
U-topk	✓	✓			✓	✓	
U-kRanks	✓	✓			✓	✓	
PT-k	✓	✓			✓		✓

Table 1: Problem attributes that top-k algorithms solve

3.1 TA-based algorithms

All TA-based algorithms are designed for certain databases, and they include **TA**, **BPA**, **Stream-Combine**, **MPro**, **LPTA** and **LPTA+**. Of these, BPA improves the stopping mechanism of TA to guarantee global optimality, reduce the computational complexity and increase the query speed.

Both MPro and Stream-Combine improve upon TA's sequential sorted data access. MPro employs probing to increase efficiency, while Stream-Combine defines an expansion indicator to promote better decisions in stream exploration. This allows Stream-Combine to forgo random access altogether as opposed to other TA-based algorithms. MPro calculates a schedule and uses a greedy algorithm for controlled random access, reducing the number of random accesses.

Many of these algorithms utilise sorted lists of objects and attribute values, (streams). This includes stream-combine, TA and BPA. However LPTA instead uses materialised views of previous top-k queries,

thereby manipulating past work to hasten future computation. These views can speed up data access and insertion with fast access and local algorithms, and can use historical cache information to reduce redundancy in sorting calculations.

3.2 Uncertain algorithms

TA-based algorithms operate on certain databases, but all fail to perform on uncertain databases. Described uncertain database top-k query algorithms include **U-Topk**, **U-kRanks**, **PT-k** and **MS_TopK**. U-Topk and U-kRanks were some of the first and more naive query methods proposed. They apply possible world semantics to compute probabilities but do not adjust the possible world data space, thus resulting in avoidable inefficiencies. Instead, PT-k implements pruning techniques algorithm for the possible world space, and many subsequent algorithms have followed suit to achieve more efficient computation.

Although possible world is well defined for algorithms on uncertain databases, the computation of probabilities between individual worlds computed according to the rule is still complex. The naive algorithm requires the computation of probabilities for all worlds, resulting in a high computational complexity. MS_TopK proposes an efficient method that expects to compute only the top-k probabilities to reduce redundancy. The algorithm accepts an approximation for each probability while maintaining a guarantee of correctness. Monte Carlo simulation is used to achieve this by simulating each node of the search until a certain depth or a particular state, and back-propagates the values to calculate the probabilities. Compared to the naive calculation used by the other three methods, is empirically much faster [14].

3.3 Search-based algorithms

OPT* and J* are both based on A* which is a classical search algorithm that bounds the cost of reaching a state with heuristics. OPT* explicitly frames the top-k query problem as an optimisation problem before this, something that no other algorithm in this survey does. The primary advantage of the OPT* over other algorithms is that the scoring function need not be monotone as it converts the scoring function into the non-monotone goal function anyway. It optimises arbitrary goal functions by converting the index traversal and optimisation problem into a generic search problem. The algorithm is adaptable to any form of access path, and the same purpose can be achieved by traversing the leaf nodes for the sorted and random accesses required by TA.

The key difference between the A* search used in OPT* and J* is that OPT* uses A* to optimise goal function, whereas J* uses A* to join tables. The objective of J* is to find assignments for all variables subject to the join constraint while maximising the score. A* always guarantees an optimal solution for both J* and OPT*. Although viewed by A* as the shortest path, OPT* sees the solution as the lowest cost index path, and J* minimises the number of states to process.

3.4 Parallel and distributed top-k

KLEE and Rankaggr are general frameworks for top-k queries. Although their targets and optimization goals are different, they focus on connecting multiple nodes to answer the query. For OLAP in an ad-hoc environment, Rankaggr focuses on aggregation methods for data analysis and storage. Compared with traditional data aggregation, Rankaggr provides effective query sorting to improve performance. KLEE focuses on performance and query flexibility. It applies to more widely distributed data structures where the quality of node connections can vary significantly. Rankaggr does not address flexibility like KLEE, which is likely a result of OLAP being commonly used for long-term statistical reports and regular analysis. The performance requirements are not high and not sensitive to data delay, so the flexibility in KLEE that allows a trade-off between bandwidth and communication phases has no great demand in OLAP.

As a top-k query algorithm designed for distributed systems, KLEE is born to provide support for parallel computing and controlling through the connection between nodes, and Chang [3] in the MPro article also provides two kinds of extension ideas to support parallel computing. For KLEE, because the data is stored in blocks in a distributed database environment, queries can be run independently on different nodes simultaneously to improve speed. An extension method called Data-Parallel MPro has a similar idea. It partitions the database and uses multi-threading technology for retrieval at the same time. The other method is quite different. It is to parallelize the probes and use the probes in parallel in each layer of exploration to speed up the process of finding necessary probes. Although there is a certain degree of similarity in the idea of parallelization, it is evident that KLEE considers the communication between wide-area distributed database nodes more specifically.

4 Discussion

The top-k query algorithms presented in this survey can be broadly categorised into algorithms for certain data and those for uncertain data. Of these, many algorithms are based on TA, the majority are applied to certain databases, and all contribute fundamental improvements to TA.

Although possible world semantics are widely used in the current literature of uncertain data queries, they are unavoidably complex. Possible worlds are inefficient when dealing with huge quantities of uncertain data that can be produced by large scale sensor networks. On the other hand, many of the algorithms of possible worlds make independent assumptions that are not applicable in uncertain databases with complex inter-object aggregation rules.

Although TA-based algorithms are both the most commonly used and researched tools for top-k queries, many of them share a common limitation: Most of these algorithms assume that the existing data is sorted and partly stored in a cache. With the rapid growth of Big Data, the amount of data generated is climbing exponentially annually. The resulting problem is that as the volume and dimensionality of data increases, it becomes difficult for algorithms to insert and sort the data. At the same time, the rise of modern distributed systems and applications such as cloud computing and large scale sensor networks contribute to the significant increase in the uncertainty of data. These topics have received the most attention recently in the domain of top-k queries, challenging existing algorithms and hardware performance [9].

Recent work has demonstrated that with the accelerated development of GPUs, parallelisation of the Single Instruction Multi-Threading (SIMT) model using GPUs can significantly accelerate the data sorting in Bitonic Sorting Network and Radix sorting [15]. It also has shown that parallel computation accelerated by multiple cores and threads performs significantly better on GPUs than on CPUs, despite CPUs' lower bandwidth usage than GPUs. Such acceleration is becoming increasingly common in data analysis tasks.

Furthermore, with the development of cloud computing there is significant concern about data security in outsourced cloud databases. Many companies outsource projects to other companies and cloud service providers for a cost-effective, high-performance service. As outsourcing companies should not always be trusted, many companies do not release all database access. Top-k queries play an essential role in securely providing the outsourced project with some of the data required and securing the database. As the literature stands, top-k queries on encrypted data have limited efficiency [11]. However, a protocol has been introduced for encrypted top-k queries based on uncertain data and its security and effectiveness have been empirically demonstrated [11].

5 Conclusion

This survey covered how top-k queries can be used and what important attributes should be considered when designing an algorithm to solve them. These features include the query type, (selection/join/aggregate), data certainty, (certain/uncertain), data access, (sorted/random), and scoring function, (monotone/generic). A comprehensive range of algorithms and frameworks were covered including classic algorithms for traditional local databases, P2P type data indexing, and data storage under a distributed architecture. The logic behind each algorithm as well as their different applications and classifications were also explained in detail. This is in addition to the difficulties encountered by these algorithms and some possible solutions.

The classical TA algorithm has a good performance on sorted lists and many innovative modifications continue to extend the usefulness of the model. But TA has many limitations such as the assumption of access to sorted lists and certain data. The possible worlds model was outline d to address uncertain data, as well as the advantages and disadvantages of this model and several algorithms that utilise it. Lastly, use of indexing, pre-computation and optimisation was discussed in order to develop frameworks to run top-k queries in environments as complex and volatile as distributed systems.

5.1 Future Directions

After examining possible future research directions in light of recent technological developments, it appears that the field of top-k queries has many more avenues to explore. The development of big data and recommender systems by large internet shopping and video streaming organisations inevitably involves consumer demand for faster and more accurate top-k queries. Further work must be completed to reduce the running time of modern top-k query algorithms while maintaining a satisfactory level of accuracy.

Similarly, for uncertain databases, the increasing prevalence of uncertainty in the era of big data advocates the need for more efficient algorithms in this space.The famous possible worlds model struggles to meet user requirements in terms of efficiency, rendering many algorithms for uncertain databases infeasible for big data. Future work is expected to result in new models and algorithms to work with uncertain data. Likewise, as hardware improves, parallelisation requires more attention to fully exploit the performance of hardware such as GPUs.

Despite organisations' increasing reliance on cloud computing, there are still many issues regarding security in top-k queries such as trust in public clouds and querying encrypted data. Research in this field is hypothesised to expand as cloud computing continues to become an everyday part of business, government and private life.

References

[1] Reza Akbarinia, Esther Pacitti, and Patrick Valduriez. 2011. Best position algorithms for efficient top-k query processing. *Information Systems*, 36, 6, 973–989.

[2] Burçak Baş. 2016. *The construction of beauty by mobile applications*. PhD thesis. Bilkent University.

[3] Kevin Chen-Chuan Chang and Seung-won Hwang. 2002. Minimal probing: supporting expensive predicates for top-k queries. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, 346–357.

[4] Gautam Das, Dimitrios Gunopulos, Nick Koudas, and Dimitris Tsirogiannis. 2006. Answering top-k queries using views. In *Proceedings of the 32nd international conference on Very large data bases*, 451–462.

[5] Ronald Fagin, Amnon Lotem, and Moni Naor. 2003. Optimal aggregation algorithms for middleware. *Journal of computer and system sciences*, 66, 4, 614–656.

[6] J Guntzer, W-T Balke, and Werner Kießling. 2001. Towards efficient multi-feature queries in heterogeneous environments. In *Proceedings International Conference on Information Technology: Coding and Computing*. IEEE, 622–628.

[7] Ming Hua, Jian Pei, Wenjie Zhang, and Xuemin Lin. 2008. Efficiently answering probabilistic threshold top-k queries on uncertain data. In *2008 IEEE 24th International Conference on Data Engineering*. IEEE, 1403–1405.

[8] Ming Hua, Jian Pei, Wenjie Zhang, and Xuemin Lin. 2008. Ranking queries on uncertain data: a probabilistic threshold approach. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, 673–686.

[9] Wang Hui, Guan Zhitao, Yang Tingting, and Xu Yue. 2014. Top-k query framework in wireless sensor networks for smart grid. *China Communications*, 11, 6, 89–98.

[10] Chengkai Li, Kevin Chen-Chuan Chang, and Ihab F Ilyas. 2006. Supporting ad-hoc ranking aggregates. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, 61–72.

[11] Xianrui Meng, Haohan Zhu, and George Kollios. 2018. Top-k query processing on encrypted databases with strong security guarantees. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, 353–364.

[12] Sebastian Michel, Peter Triantafillou, and Gerhard Weikum. 2005. Klee: a framework for distributed top-k query algorithms. In *Proceedings of the 31st international conference on Very large data bases*, 637–648.

[13] Apostol Natsev, Yuan-Chi Chang, John R Smith, Chung-Sheng Li, and Jeffrey Scott Vitter. 2001. Supporting incremental join queries on ranked inputs. In *VLDB*. Volume 1, 281–290.

[14] Christopher Re, Nilesh Dalvi, and Dan Suciu. 2007. Efficient top-k query evaluation on probabilistic data. In *2007 IEEE 23rd International Conference on Data Engineering*. IEEE, 886–895.

[15] Anil Shanbhag, Holger Pirk, and Samuel Madden. 2018. Efficient top-k query processing on massively parallel hardware. In *Proceedings of the 2018 International Conference on Management of Data*, 1557–1570.

[16] Mohamed A Soliman, Ihab F Ilyas, and Kevin Chen-Chuan Chang. 2007. Top-k query processing in uncertain databases. In *2007 IEEE 23rd International Conference on Data Engineering*. IEEE, 896–905.

[17] Min Xie, Laks VS Lakshmanan, and Peter T Wood. 2013. Efficient top-k query answering using cached views. In *Proceedings of the 16th International Conference on Extending Database Technology*, 489–500.

[18] Xi Zhang and Jan Chomicki. 2009. Semantics and evaluation of top-k queries in probabilistic databases. *Distributed and parallel databases*, 26, 1, 67–126.

[19] Zhen Zhang, Seung-won Hwang, Kevin Chen-Chuan Chang, Min Wang, Christian A Lang, and Yuan-chi Chang. 2006. Boolean+ ranking: querying a database by k-constrained optimization. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, 359–370.