Computational aspects around preference queries

Karim Alami
supervised by Sofian Maabout
Univ. Bordeaux, CNRS, LaBRI, UMR 5800, F-33400
Talence, France
karim.alami@u-bordeaux.fr

ABSTRACT

Preference queries present two main challenges: difficulty for users to express their preferences and the computational complexity. For skyline queries, the preferences can be on attributes, e.g., some user may look for the best flights regarding price and number of stops, and others may look for the best flights regarding number of stops and duration. In addition, preference can be expressed as a (partial) order on attributes domains, e.g., some user may prefer flight company A over B while another one may have the opposite preference. For top-k queries, users define a score function to rank objects, e.g., users who give more importance to price could define the following score function: price $* 2 +$ duration of the flight. In general, several rounds are required before converging towards a satisfying answer where at each round, more precise preferences are given by the user. This is due to the difficulty to figure out the precise formulation of the user's preferences. Therefore, a more or less high number of queries need to be evaluated. Our research work aims to make these queries answering faster through dedicated index structures and precomputed views. The main challenges when adopting this strategy are (i) lightweight memory consumption and (ii) fast maintenance process. Our first step was NSC, an index structure that optimizes skyline queries. However, the structure was designed for a static context making it unsuitable when data can be inserted/deleted. We redesigned NSC to cope with dynamic data and in some cases, we proposed further approaches when the structure is not suitable. In this paper, we summarize our previous contributions and present some perspective research regarding the link between regret minimization queries and what we did so far.

1. INTRODUCTION

Preference queries aim to retrieve points among a set of points that can be considered interesting regarding the preferences of the user over a set of parameters. They are efficient tools which reduce the amount of data returned to the user ,and consequently, avoids him an endless comparison of data.

The information retrieval regarding user preferences have historically been a ranking problem, i.e., given a set of key words, retrieve the elements that "best" match these keywords, we call it often "top-k" query . Its adaptation to relational databases has been addressed in [7]. Users are requested to put weights on the attributes in order to select points that have the higher values on attributes with the higher weights. Top-k queries gives the advantage of controlling the size of the output, however the selection of the weights remain very hard. There has been several works to simplify this process, e.g., selection of a range of weights, elicitation and regret minimizing, among others. Authors in [6] proposed the skyline operator as an alternative to ranking queries. A skyline query returns a set of points which are not dominated by any other point of the dataset. A point x dominates a point y iff x is better or equal on all attributes and strictly better on at least one attribute. The skyline query provides the advantage to not rely on a score function however, the size of the output is not controllable and it requires a quadratic computational time regarding the size of the dataset. Many studies have been done to optimize the skyline query execution time either by optimizing the number of comparisons or by indexation.

In previous work [9], the structure NSC has been presented as an index to optimize skyline queries. Its main idea consists in comparing every record r of a dataset to all remaining records r' and store the subspaces where r' dominates r in a form of pair of subspaces $compare(r, r') = \langle X|Y\rangle$ where X represents the attributes where r' is strictly better than r and Y represents the attributes where r and r' are equal. Now given a subspace Z, a record r belongs to $Sky(Z)$, i.e. the skyline over the subspace Z, if and only if there does not exist a pair $\langle X|Y\rangle$ associated to r that *covers* Z, i.e. $Z \subseteq XY$ and $Z \neq Y$. We denote by $cover(\langle X|Y\rangle)$ the set of subspaces covered by $\langle X|Y\rangle$. For example, consider a dataset with attributes A, B and C. The pair $\langle AB|C\rangle$ covers the subspaces $\{A, B, AB, AC, BC, ABC\}$. The time complexity of NSC is quadratic wrt the size of the dataset as well as the space complexity. However, not every pair should be kept. Let $Pairs(r)$ be the set of pairs associated to r, this set can be minimized by computing a subset $Q \subseteq Pairs(r)$ such that $cover(Q) = cover(Pairs(r))$, i.e. the set of subspaces covered by $Pairs(r)$ are covered by Q as well. We say that Q is an equivalent subset of $Pairs(r)$, $Q \equiv Pairs(r)$. The minimization problem is NP-Hard and a polynomial greedy approximate algorithm has been proposed.

Proceedings of the VLDB 2019 PhD Workshop, August 26th, 2019. Los Angeles, California

In the literature, works that optimize skyline queries can be divided in three groups, works that (i) design fast algorithms without precomputation, (ii) design index structures (iii) materialize the results. We note BSkyTree [11] the state of the art algorithm to process skyline queries without precomputing. We note HashCube [4], a bitmap like index structure which associates a 2^d Boolean vector v to every record r where d is the number of attributes. $v[i]$ is set iff r belongs to $Sky(Z)$ such that Z is encoded by i. HashCube is highly efficient for skyline query answering, however, authors only proposed an algorithm to construct the structure from scratch in [5]. No obvious maintenance procedure is noticed. Materialized skyline views are very time and memory consuming as the number of skylines wrt to a dataset is exponential to the number of dimension. However, they provide the best query answering performance.

The experiments performed in [9] assess the construction time and memory performance of NSC against its competitors as well as query execution time. However, the structure was designed for a static context. An insertion/deletion of a record or a change in an attribute domain order may require rebuilding the structure from scratch.

For this thesis, we studied the ability of NSC to deal with updates. Precisely, we addressed its maintenance in case of (i) dynamic data, i.e., points are removed and inserted at any time and (ii) streaming data, i.e., append-only database and sliding window model. We redesigned the structure to cope with each context. Presently, we are working on skyline queries optimization over a dataset with nominal attributes and dynamic order. We found that NSC is not suitable for this context, hence we established a novel approach based on views. Finally, we aim to investigate relationship between multidimensional skylines and regret minimization queries.

2. SKYLINE QUERIES OPTIMIZATION IN PRESENCE OF UPDATES

In the following, we present the adaptation of NSC for, first, dynamic data, i.e., insertion/deletion of one or multiple records at any time, then, streaming data, i.e., insertion of records at regular intervals. We studied separately both contexts because they present different challenges.

2.1 Dynamic Data

In the case of dynamic data, i.e., insertion/deletion of one or multiple records at any time, NSC should be updated in accordance to the new dataset. The baseline approach is to build NSC structure from scratch, however this approach is costly.

Related work pointed out that dealing with deletion is harder than insertion. We note the work [17] which addressed the maintenance of a materialized skyline view in case of deletion. For NSC, deletion is a hard task as well. Let r^- be the deleted record then for every record r, we remove from $Pairs(r)$ the pair p^- computed wrt r^-. However, p^- may have *exclusively* covered some pairs that have been deleted from $Pairs(r)$ during the minimization process. Recovering these pairs requires recomputing $Pairs(r)$ from scratch. We propose a solution that detects when $Pairs(r)$ should be recomputed. For every distinct pair p in $Pairs(r)$, we set a counter of how many records are the source of p. Hence, we recompute $Pairs(r)$ only if p^- is in $Pairs(r)$ and its counter set to 1. In theory, this approach could be seen as building the structure from scratch at each deletion, however, in practice, only a small fraction of the dataset requires their set of pairs to be recomputed. In terms of time complexity, let the size of the dataset be n. Let I be the set of records for which their respective set of pairs requires to be recomputed. Identifying I takes $O(n)$ time and recomputing $Pairs(r)\ \forall r \in I$ takes $O(|I| \cdot n)$. Regarding memory consumption, NSC size is augmented by a constant due to the additional counter.

Now for insertions, let r^+ be a record to be inserted then we compute $Pairs(r^+)$. Regarding the existing records in the dataset, let r be one, we compute $p^+ = compare(r, r^+)$. However, $Pairs(r)$ is already a minimal set of pairs, hence we address the problem: should $Pairs(r) \cup compare(r, r^+)$ be minimized now? or should the minimization process be triggered after a number of pairs appended? In absence of an incremental greedy algorithm, we propose a linear algorithm wrt the size of $Pairs(r)$, called *min_inclusion*, which takes every pair $p \in Pairs(r)$ and compare it to p^+. The pair covered by the other one is discarded. This algorithm does not provide any approximation guarantee. However, in practice, *min_inclusion* allows a good compression ratio wrt the greedy algorithm.

2.2 Streaming Data

We consider the following stream model. Records are streamed at regular interval time θ. They have a validity period of size ω which can also be seen as a sliding window over the dataset. Hence, records are considered outdated ω timestamps after their arrival time. The semantic of continuous queries [15] states that the query result should be available and accurate at each time. However, this condition is hardly attainable, especially for skyline queries. We point out two main difficulties for continuous skyline queries. Let Q_{sky} be a skyline query, then (i) a record r belonging to $Ans(Q_{sky})$ at a timestamp t may leave $Ans(Q_{sky})$ at timestamp $t' > t$ if a streamed record r' dominates r. And (ii) a record r can join $Ans(Q_{sky})$ at timestamp t' later than its arrival time if skyline records that dominate r get outdated. We note the contribution of [14]. Authors propose to maintain a skyline set $DBSky$ and a set of potentially skyline records $DBRest$, i.e., records which may join $DBSky$ in the future. They propose an algorithm to maintain these sets together with an event list recording timestamps where a record in $DBRest$ may join $DBSky$. Still, this approach makes a big number of records comparison which makes it non scalable wrt dimensionality and data cardinality. Moreover it maintains a *single* skyline query, hence maintaining several skylines may require an exponential space wrt to the number of attribute. We note as well the approach presented in [12] which is based on R-trees. Regarding NSC structure, the approaches presented for dynamic data are not suitable for two main reasons: (i) the timestamp where a record is no more valid is known and (ii) the frequency of streaming is often high. Generally, real time data processing is hard in streaming context. Batch processing mode provides a larger time to process data, however, the query result is not accurate with the current dataset. We proposed a framework called $MSSD$ which handles three data structure, (i) a buffer B, (ii) a main dataset R and (iii) NSCt, a redesigned version for NSC. Incoming records are first buffered during an interval of time of size k, then inserted into the main

dataset R. NSCt only indexes records in R, hence, queries evaluated through NSCt, consider only records in R. We sacrifice the accuracy of the queries, nonetheless, we ensure a fast maintenance that allows faster query answering than state of the art works. Next we explain the main novelties for NSCt. We organize the set of pairs of a record r as a sequence of buckets of pairs $Pairs(r) = [Buck_1, \ldots, Buck_m]$, where each bucket corresponds to the pairs computed wrt a batch of data. We adopt this approach for the following purpose. Let b_1 and b_2 be two batches such that their respective timestamp are $TS(b_1)$ and $TS(b_2)$, and $TS(b_1) < TS(b_2)$. As we consider a sliding window data model with an interval of size ω, records in b_1 get outdated before records in b_2. Therefore, let r be a record, the bucket computed wrt b_1 is located before the bucket computed wrt b_2 in $Pairs(r)$. During maintenance, $Buck_1$, i.e. the oldest bucket, is simply discarded as it contains pairs computed wrt an outdated batch. Moreover, during the minimization process, a bucket $Buck_i$ is minimized only wrt successor buckets. We formalized the problem of buckets minimization for NSCt and proved its NP-Hardness by a reduction to **MSC** (Minimum Set Cover) problem. We provided a greedy approximate algorithm as well. We present in [2] early experiments we performed. It shows that despite the maintenance time, NSCt answers much more queries during the batch interval than BSkyTree [11].

3. SKYLINE QUERIES OPTIMIZATION IN PRESENCE OF DYNAMIC ORDER

In many real world applications, users are allowed to express their preferences over the values of nominal attributes. In such case, we say that the attribute domain has a *dynamic order*. For example, on a movie platform website, users want the best rated movies but have preferences over the genre as well. Also, on a flight booking website, users are interested in cheap and fast flights but may have preferences over the airline companies. To ease the comprehension, we consider datasets with only one nominal attribute A and some number of numerical attributes. A user preference over A is a partial order that can be written as a set of (a_1, a_2) such that $a_1, a_2 \in A$ and a_1 is preferred over a_2. In such scenario, NSC is not suitable to answer skyline queries as it is constructed depending on a given partial order over A. Hence it should be updated every time a query with a different partial order is issued, or it may require to construct NSC for each possible partial order over A. In the literature, there is two major approaches to handle the problem of answering skyline queries over datasets with attributes having dynamic order, (i) algorithms which, given a query Q, maps a nominal attribute into a set of virtual numerical attributes in accordance to the user preference $Q.R$. Then, a skyline query is processed over the transformed dataset [18]. And (ii) answers the given query through a set of cached views, each computed wrt a partial order [16, 10]. These works adopt *refinement* and *chain product decomposition* techniques in order to evaluate an issued query through materialized views. Let Q be an issued query and let Q' be a view then $Ans(Q) \subseteq Ans(Q')$ if $Q'.R \subseteq Q.R$. We say that Q is a *refinement* of Q'. Now let $\{Q_1, \ldots, Q_n\}$ a set of views such that $Q_i.R \; \forall i \in [1 \ldots n]$ is a chain, i.e., the values of A are totally ordered, then $Ans(Q) = \bigcup_{i \in [1 \ldots n]} Q_i$ if $Q.R = \bigotimes_{i \in [1 \ldots m]} Q_i.R$. We say that $\{Q_1, \ldots, Q_n\}$ is a *chain product decomposition* of Q. We adopted a novel approach which theoretically and experimentally provided better results than the above approaches. We define the single partial order *spo* over an attribute domain $Dom(A)$ as a partial order where only two values in $Dom(A)$ are comparable and all other values are incomparable. Given a query Q with user preference $Q.R$, $Ans(Q)$ is the intersection of the skylines wrt *spo*s composing $Q.R$. Every *spo* skyline query is considered a view whether it is materialized or not. Our experiments showed that answering a query through non materialized *spo* views is even faster than online algorithms, mainly because skyline algorithms are highly weakened by dimensionality added by mapping a nominal attribute to several numerical attributes. Moreover, this approach, namely the *single partial order decomposition*, presents the advantage of easily selecting the right views in order to evaluate a given query compared to the chain product decomposition which is an NP Hard Problem. Regarding the memory consumption, let $|dom(A)| = m$, there exists $m!$ total orders wrt $dom(A)$, hence $m!$ views to store for chain product decomposition approach. For refinement approach, the higher the number of views stored, the faster will be query answering. Note that there exists an exponential number of partial orders wrt m. Our approach requires $2 \cdot C_m^2$ views as for every two values, two views are stored. This can be further optimized by selecting only a subset of *spo* views to materialize. We addressed the following problem: given a workload Q and an integer k, select a set of views of size at most k to materialize such that the **cost** of answering the queries in Q through the views is minimum. We are working on the proof of the Hardness of this problem. Finally we extended the work to the case where datasets have several nominal attributes.

4. REGRET MINIMIZING QUERIES AND MULTIDIMENSIONAL SKYLINES

In this section, we mainly present state of the art of regret optimization queries and we place some questions that we plan to investigate during this thesis.

Skyline and Top-k queries share the same objective which is selecting the best elements. However, on one hand, skyline queries return a non constrained size of result by relying on just the dominance relation between elements, and on the other hand, Top-k queries require a score function from the user to restrain the result size to k.

Recently [13] presented "regret minimizing set" to avoid the limitations of skyline and Top-k queries by not requiring a score function while bounding the result size. The main idea is to select a *representative* subset S of a dataset T. Let f be a score function, k be an integer, then let $f_k(T)$ be the score of the k^{th} ranked point using f. The regret of a subset S wrt f is $f_1(T) - f_1(S)$ and the regret ratio is $\frac{f_1(T) - f_1(S)}{f_1(T)}$. Given a family of functions $\mathcal{F}$, the problem here is to find S of size r which minimizes the maximal regret ratio among all $f \in \mathcal{F}$. A greedy approximate algorithm to solve this problem has been proposed in [13]. Consequently, [8] proposed a relaxation, namely the k-regret minimizing set: the regret of S considered here is $\max(f_k(T) - f_1(S), 0)$. Moreover, they proved the NP-hardness of the regret minimization problem, with or without the relaxation. [3] addressed the query evaluation optimization. They proposed a linear time dynamic programming algorithm based on the skyline

set for a two-dimensional dataset. In addition, for datasets with more than 3 dimensions, they proposed an approach where they discretise the family of linear function $\mathcal{F}$, into a set of function F wrt a parameter γ given by the user. This discretisation allows a controlled approximation of the optimal regret ratio. A notable contribution to the regret minimization line of research is given by [1]. They provide a reformulation of a regret representative set. Let $S \subset T$. Then S is a (k, ϵ)-set iff $\forall f \in \mathcal{F}$, $\frac{f_k(T)-f_1(S)}{f_k(T)} \leq \epsilon$. They formulate two regret minimizing set problems, (i) by size minimization, i.e., find the smallest (k, ϵ)-set, and (ii) by regret minimization, compute the (k, ϵ)-set of size r and ϵ is minimum.

Our previous work addressed two aspects of skyline queries: index structure to optimize multidimensional queries and its efficient maintenance upon updates. We aim to extend this material to optimize regret minimization queries. We plan to investigate more deeply the relationship between skyline sets and regret minimizing sets (RMS). Let T be a dataset over a set of dimensions $D = \{D_1, \ldots, D_d\}$. One question that could be interesting to investigate is to find a relationship between the regret minimization sets when considering subspaces of D. More precisely, let $X \subseteq Y \subseteq D$ and $S(X)$, resp. $S(Y)$, be the RMS wrt X, resp. Y. How these two sets do compare? Which situations make them comparable? Which functions families make them comparable? How can the RMS's wrt all subspaces can be computed efficiently? Consider the RMS frequency of a tuple to be the number of RMS's it belongs to, then retrieving the k most frequent tuples could be seen as a way to define the 'best" k tuples. Let $t \in T$. Is the knowledge we have about the skylines to which t belongs, can give us hints about its belonging to an RMS? Let the skyline frequency of a tuple be the number of skylines it belongs to. Let L be the set of tuples that are at least l skyline frequent, what approximation, if any, gives the restriction of regret minimizing set computation wrt L compared to the whole dataset? In case of a positive answer, it would be interesting to see how to adapt NSC to optimize the calculation of the set L and consequently, an approximate regret minimization set.

Acknowledgements

Experiments presented in this paper were carried out using the PlaFRIM experimental testbed, supported by Inria, CNRS (LABRI and IMB), Université de Bordeaux, Bordeaux INP and Conseil Régional d'Aquitaine
(see https://www.plafrim.fr/)

5. REFERENCES

[1] P. K. Agarwal, N. Kumar, S. Sintos, and S. Suri. Efficient algorithms for k-regret minimizing sets. In *SEA 2017, June 21-23, 2017, London, UK*, pages 7:1–7:23, 2017.

[2] K. Alami and S. Maabout. Multidimensional skylines over streaming data. In *International Conference on Database Systems for Advanced Applications*, pages 338–342. Springer, 2019.

[3] A. Asudeh, A. Nazi, N. Zhang, and G. Das. Efficient computation of regret-ratio minimizing set: A compact maxima representative. In *Proceedings of the 2017 ACM SIGMOD Conference*, pages 821–834. ACM, 2017.

[4] K. S. Bøgh, S. Chester, D. Sidlauskas, and I. Assent. Hashcube: A data structure for space and query efficient skycube compression. In *Proceedings of CIKM conference*, pages 1767–1770, 2014.

[5] K. S. Bøgh, S. Chester, D. Sidlauskas, and I. Assent. Template skycube algorithms for heterogeneous parallelism on multicore and GPU architectures. In *Proc. of SIGMOD Conference*, pages 447–462, 2017.

[6] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *Proc. of ICDE conf.*, pages 421–430, 2001.

[7] S. Chaudhuri and L. Gravano. Evaluating top-k selection queries. In *VLDB*, volume 99, pages 397–410, 1999.

[8] S. Chester, A. Thomo, S. Venkatesh, and S. Whitesides. Computing k-regret minimizing sets. *Proceedings of the VLDB Endowment*, 7(5):389–400, 2014.

[9] N. Hanusse, P. Kamnang-Wanko, and S. Maabout. Computing and summarizing the negative skycube. In *Proc. of CIKM Conference*, pages 1733–1742, 2016.

[10] Y.-L. Hsueh, C.-C. Lin, and C.-C. Chang. An efficient indexing method for skyline computations with partially ordered domains. *IEEE Transactions on Knowledge & Data Engineering*, pages 963–976, 2017.

[11] J. Lee and S.-W. Hwang. Scalable skyline computation using a balanced pivot selection technique. *Information Systems*, 39:1–21, 2014.

[12] M. Morse, J. M. Patel, and W. I. Grosky. Efficient continuous skyline computation. *Information Sciences*, 177(17):3411–3437, 2007.

[13] D. Nanongkai, A. D. Sarma, A. Lall, R. J. Lipton, and J. Xu. Regret-minimizing representative databases. *Proceedings of the VLDB Endowment*, 3(1-2):1114–1124, 2010.

[14] Y. Tao and D. Papadias. Maintaining sliding window skylines on data streams. *IEEE Trans. Knowl. Data Eng.*, 18(2):377–391, 2006.

[15] D. B. Terry, D. Goldberg, D. A. Nichols, and B. M. Oki. Continuous queries over append-only databases. In *Proceedings of the 1992 ACM SIGMOD*, pages 321–330, 1992.

[16] R. C.-W. Wong, J. Pei, A. W.-C. Fu, and K. Wang. Online skyline analysis with dynamic preferences on nominal attributes. *IEEE Transactions on Knowledge and Data Engineering*, 21(1):35–49, 2009.

[17] P. Wu, D. Agrawal, Ö. Egecioglu, and A. El Abbadi. Deltasky: Optimal maintenance of skyline deletions without exclusive dominance region generation. In *Proceedings of ICDE Conference*, pages 486–495, 2007.

[18] S. Zhang, N. Mamoulis, D. W. Cheung, and B. Kao. Efficient skyline evaluation over partially ordered domains. *Proceedings of the VLDB Endowment*, 3(1-2):1255–1266, 2010.