VLDBJ manuscript No.
(will be inserted by the editor)

Adaptive Disk Storage for Interaction Graphs

Robert Soulé · Buğra Gedik

Received: date / Accepted: date

Abstract TODO

Keywords First keyword · Second keyword · More

1 Introduction

An *interaction graph* is an append-only graph, where new edges and vertices are added as time progresses.

Typical interaction graphs have data associated with their vertices and edges. We call this data *attributes* or *properties*. Most algorithms access this data as they traverse the graph. However, not all of the data is accessed by all the queries. Typically, there are correlations among the attributes accessed by different queries, such as Q1 and Q5 accessing attributes a and b, and Q2, Q3, Q4 accessing attributes c and d, and so on.

In an interaction graph database organized as blocks of temporal neighborlists (as in Gedik et al. [2]), it is important to store the edge and vertex properties locally. For instance, if the edge properties are kept away in a relational table, there will be almost no benefit to the locality optimizations performed for block organization (discussed in Gedik et al. [2], as we would have to go back and forth between the disk blocks to access the edge attributes).

On the other hand, putting all the edge attributes into the disk blocks containing the graph structure is expected to cause significant overhead when only a few of these attributes are read. This is somewhat similar to the problem with row-oriented databases, where the entire row needs to be accessed from the disk despite the query needing only a small fraction of it. Unfortunately, there is no clear correspondence to a column-oriented database layout for the graph databases.

An important characteristic of interaction graphs are that they are temporal. As such, the co-access correlations for the attributes can be different for different temporal regions. It might also be unknown at the insertion time and may be discovered later depending on the workload. This points to the need for adaptively optimizing the layout (somewhat similar to H2O [1] and HYRISE [3]).

I imagine a solution to this problem, which I name the 'rail layout'. The idea is to start with large blocks that contain the entire data. As we learn about the access properties for different time regions, we might start splitting such blocks into smaller blocks that run parallel to each other, almost like having two or more graph databases for certain time regions, each containing a different subset of the attributes, but with a link between them in case a query needs to access both.

2 Problem Description

The problem is to determine a partitioning of the disk blocks into sub-blocks that would minimize query I/O. Of course, partitioning into sub-blocks results in an increase in storage cost. The storage cost must stay below some specified limit.

There are two possibilities: *overlapping* and *non-overlapping* partitioning. In the overlapping case, attributes can be replicated in the sub-blocks. The non-overlapping case is a true partition.

We start by defining the cost model, which formalizes the query I/O and storage costs.

R. Soulé is with University of Lugano
E-mail: robert.soule@usi.ch

B. Gedik is with Bilkent University,
E-mail: bgedik@cs.bilkent.edu.tr

2.1 Cost Model

Let Q be the query workload, where each query $q \in Q$ accesses a set of attributes $q.A$ and traverses parts of the graph for the time range $q.T = [q.t_s, q.t_e]$. We denote the set of all attributes as A. Given a block B, we denote its time range as $B.T$, which is the union of the time ranges of its temporal neighborlists. Let $s(a)$ denote the size of an attribute a. We use $c_n(B)$ to denote the number of temporal neighborlists within block B and $c_e(B)$ to denote the total number of edges in the temporal neighborlists within the block. We overload the notation for block size and use $s(B)$ to denote the size of a block B. We have:

$$s(B) = c_e(B) \cdot \Big(16 + \sum_{a \in B.A} s(a)\Big) + c_n(B) \cdot 12 \qquad (1)$$

Here, 16 corresponds to the cost of storing the edge id and the timestamp, and 12 corresponds to the cost of storing the head vertex (8 bytes) plus the number of entries (4 bytes) for a temporal neighborlist.

Our goal is to create a potentially overlapping partitioning of a block, denoted by $\mathcal{P}(B)$. We call the partitions *sub-blocks*. We have $\bigcup_{B' \in \mathcal{P}(B)} B'.A = A$. We aim to find the function $\mathcal{P}$ that minimizes the query I/O over B, while keeping the relative storage overhead beyond a limit, say $1+\alpha$ times the original. If we represent the query I/O as $L(\mathcal{P}, B)$ and the relative storage overhead as $H(\mathcal{P}, B)$, our goal is to find:

$$\mathcal{P} \leftarrow \text{argmin}_{\{\mathcal{P} : H(\mathcal{P}(B)) < \alpha\}} L(\mathcal{P}, B) \qquad (2)$$

The storage overhead can be formalized as follows for the non-overlapping case:

$$H(\mathcal{P}, B) = (|\mathcal{P}(B)| - 1) \cdot \Big(1 - \frac{c_e(B) \cdot \sum_{a \in A} s(a)}{s(B)}\Big) \qquad (3)$$

For the general case (including overlapping), we can formulate the storage overhead as follows:

$$H(\mathcal{P}, B) = \frac{\sum_{B' \in \mathcal{P}(B)} s(B')}{s(B)} - 1 \qquad (4)$$

2.1.1 Query I/O

Let m be a function that maps a query q to the set of sub-blocks that are accessed to satisfy it for a relevant block B under a given partitioning $\mathcal{P}$. For the case of non-overlapping attributes, we have:

$$m(\mathcal{P}, B, q) = \{B' : B' \in \mathcal{P}(B) \wedge q.A \cap B'.A \neq \varnothing\} \qquad (5)$$

For the case of overlapping attributes, we use a simple heuristic to define the set of sub-blocks to be used for answering the query. Algorithm 1 captures it. The idea is to select sub-blocks that bring the highest relative marginal gain, which is defined as the size of attribute data that contributes to the query result, relative to the sub-block size. While computing the marginal gain, attributes that are covered by sub-blocks that are already selected are not considered. The query I/O cost

Algorithm 1: m-overlapping($\mathcal{P}, B, q$)

```
Data: $\mathcal{P}$: partitioning function, $B$: block, $q$: query
$S \leftarrow \varnothing; R \leftarrow \varnothing$        ▷ Selected attributes; Resulting sub-blocks
while $S \subset q.A$ do        ▷ While unselected attributes remain
    $B' \leftarrow \text{argmax}_{B' \in \mathcal{P}(B) \setminus R} \sum_{a \in B'.A \cap q.A \setminus S} \frac{c_e(B') \cdot s(a)}{s(B')}$
    $S \leftarrow S \cup B'.A$        ▷ Extend the selected attributes
    $R \leftarrow R \cup B'$        ▷ Extend the selected sub-blocks
end
return R        ▷ Final set of sub-blocks covering the query attributes
```

for a block is then given by:

$$L(\mathcal{P}, B) = \sum_{q \in Q} \mathbf{1}(q.T \cap B.T \neq \varnothing) \cdot \sum_{B' \in m(\mathcal{P}, B, q)} s(B') \qquad (6)$$

3 Partitioning

This section is still a work-in-progress. We are trying to find the algorithms.

3.1 Non-Overlapping Attributes

Problem. Find a true partitioning of attributes that minimizes the query I/O and bounds the storage cost by some upper limit.

3.1.1 Integer Linear Program Formulation

We present an ILP formuation of the problem. For this purpose, we define a number of binary (0 or 1) variables:

– $x_{a,p}$: 1 if attribute a is in partition p, 0 otherwise.
– $y_{p,q}$: 1 if partition p is used by query q, 0 otherwise.
– $z_{a,p,q}$: 1 if partition p is used by query q and attribute a is in partition p, 0 otherwise.
– u_p: 1 if partition p is assigned at least 1 attribute, 0 otherwise.

Each of these variables serve a purpose:

– xs define the attribute-to-partition assignment.
– ys help formulate the query I/O contribution of partitions, excluding their assigned attributes.
– zs help formulate the query I/O contribution of partitions, only considering their assigned attributes.
– us help formulate the storage overhead requirement.

We also define a helper notation for representing whether a variable is accessed by a query or not: $q(a) \equiv \mathbf{1}(a \in q.A)$, where $\mathbf{1}$ is the indicator function. We assume that there are a maximum of k partitions. k can be taken as $|A|$, as there cannot be more partitions than there are attributes.

We are now ready to state the ILP formulation. We start with the objective function, that is the total query I/O:

$$\sum_{q\in Q}\sum_{p=1}^{k} \quad (16 \cdot c_e(B) + 12 \cdot c_n(B)) \cdot y_{p,q} + \sum_{a\in A} s(a) \cdot c_e(B) \cdot z_{a,p,q} \tag{7}$$

In Eq. 7, we simply sum for each query and each partition, and add the I/O cost of reading in the structural information found in a sub-block, if the partition is used by the query. We then sum over each attribute as well, and add the I/O cost of reading in the attributes. Note that $z_{a,p,q}$ could have been replaced with $x_{a,p} \cdot y_{p,q}$, but that would make the objective function non-linear.

We are now ready to state our constraints. Our first constraint is that, each attribute must be assigned to a single partition. Formally:

$$\forall_{a\in A}, \sum_{p=1}^{k} x_{a,p} = 1 \tag{8}$$

Our second constraint is that, if a query q contains an attribute a assigned to a partition p, then partition p is used by the query, i.e., $y_{p,q} = 1$. In essence, we want to state: $\forall_{\{p,q\}\in[1..k]\times Q}, y_{p,q} = \mathbf{1}(\sum_{a\in A} q(a) \cdot x_{a,p} > 0)$.

In order to formulate this constraint, we use the following ILP construction: Assume we have two variables, β_1 and β_2, where $\beta_2 \in [0, 1]$ and $\beta_1 \geq 0$. We want to implement the following constraint: $\beta_2 = \mathbf{1}(\beta_1 > 0)$. This could be expressed as a linear constraint as follows, where K is a large constant guaranteed to be larger than β_1 for practical purposes:

$$\begin{aligned} &\beta_1 - \beta_2 \geq 0 \\ &K \cdot \beta_2 - \beta_1 \geq 0 \end{aligned} \tag{9}$$

We now apply this construction to our second constraint, where $\beta_1 = \sum_{a\in A} q(a) \cdot x_{a,p}$ and $\beta_2 = y_{p,q}$. This results in the following linear constraints:

$$\begin{aligned} &\forall_{\{p,q\}\in[1..k]\times Q}, \ \sum_{a\in A} q(a) \cdot x_{a,p} - y_{p,q} \geq 0 \\ &\forall_{\{p,q\}\in[1..k]\times Q}, \ K \cdot y_{p,q} - \sum_{a\in A} q(a) \cdot x_{a,p} \geq 0 \end{aligned} \tag{10}$$

Our third constraint is that, if an attribute a is assigned to a partition p, and partition p is used by a query q, then the corresponding z variable must be set to 1. That is, we want: $\forall_{\{a,p,q\}\in A\times[1..k]\times Q}, z_{a,p,q} = \mathbf{1}(x_{a,p} = y_{p,q} = 1)$. We express this as a linear constraint, as follows:

$$\forall_{\{a,p,q\}\in A\times[1..k]\times Q}, \ z_{a,p,q} - (x_{a,p} + y_{p,q}) \geq -1 \tag{11}$$

In Eq. 11, when the x and y variables are both 1, the z variable is simply forced to be 1. Otherwise, the z variable can be either 0 or 1, but since the z variables appear in the objective fucntion as positive terms, the solver will set them to 0, which is what we want.

Our fourth constraint is that, if a partition is non-empty, then its corresponding u variable must be set to 0. In other words, we want $\forall_{p\in[1..k]}, u_p = \mathbf{1}(\sum_{a\in A} x_{a,p} > 0)$. This is expressed as linear constraints, as follows:

$$\begin{aligned} &\forall_{p\in[1..k]}, \ \sum_{a\in A} x_{a,p} - u_p \geq 0 \\ &\forall_{p\in[1..k]}, \ K \cdot u_p - \sum_{a\in A} x_{a,p} \geq 0 \end{aligned} \tag{12}$$

Eq. 12 uses the same construction as the second constraint, where $\beta_1 = \sum_{a\in A} x_{a,p}$ and $\beta_2 = u_p$.

$$
\begin{aligned}
\text{minimize} \sum_{q\in Q}\sum_{p=1}^{k} \quad & (16\cdot c_e(B) + 12\cdot c_n(B))\cdot y_{p,q} \\
& + \sum_{a\in A} s(a)\cdot c_e(B)\cdot z_{a,p,q} \\
\text{subject to} & \\
\forall_{a\in A}, \quad & \sum_{p=1}^{k} x_{a,p} = 1 \\
\forall_{\{p,q\}\in[1..k]\times Q}, \quad & \sum_{a\in A} q(a)\cdot x_{a,p} - y_{p,q} \geq 0 \\
\forall_{\{p,q\}\in[1..k]\times Q}, \quad & K\cdot y_{p,q} - \sum_{a\in A} q(a)\cdot x_{a,p} \geq 0 \\
\forall_{\{a,p,q\}\in A\times[1..k]\times Q}, \quad & z_{a,p,q} - (x_{a,p} + y_{p,q}) \geq -1 \\
\forall_{p\in[1..k]}, \quad & \sum_{a\in A} x_{a,p} - u_p \geq 0 \\
\forall_{p\in[1..k]}, \quad & K\cdot u_p - \sum_{a\in A} x_{a,p} \geq 0 \\
& \sum_{p=1}^{k} u_p \leq 1 + \frac{\alpha}{1 - \frac{c_e(B)\cdot\sum_{a\in A} s(a)}{s(B)}}
\end{aligned}
$$

Fig. 1 ILP formulation for the non-overlapping partitioning

Our fifth, and the last, constraint deals with the storage overhead. We want to make sure that the storage overhead does not go over α. The storage overhead depends on the number of partitions used. That means that the only ILP variables it depends on is the us. In particular, the number of partitions used is given by $\sum_{p=1}^{k} u_p$. This results in the following linear constraint:

$$\sum_{p=1}^{k} u_p \leq 1 + \frac{\alpha}{1 - \frac{c_e(B)\cdot\sum_{a\in A} s(a)}{s(B)}} \tag{13}$$

The final ILP formulation for the non-overlapping partitioning is given in Figure 1.

3.1.2 Heuristic Solution

Approach. TODO

Algorithm 2: Algorithm for partitioning blocks into sub-blocks with non-overlapping attributes.

```
Data: $B$: block, $Q$: set of queries
$c^* \leftarrow \infty$                          ▷ Lowest cost over all # of partitions
for $k = 1$ to $|A|$ do                          ▷ For each possible # of partitions
    $R[i] \leftarrow \varnothing, \forall i \in [1..k]$          ▷ Initialize partitions
    for $a \in A$, in decr. order of $f(a)$ do   ▷ For each attribute
        $c \leftarrow \infty$                    ▷ Lowest cost over all assignments
        $j \leftarrow -1$                        ▷ Best partition assignment
        for $i \in [1..k]$ do                    ▷ For each partition assignment
            $R[i] \leftarrow R[i] \cup \{a\}$    ▷ Assign attribute
            if $L(R,B,Q) < c$ then               ▷ If query cost is lower
                $c \leftarrow L(R,B,Q)$          ▷ Update the lowest cost
                $j \leftarrow i$                 ▷ Update the best partition
            end
            $R[i] \leftarrow R[i] \setminus \{a\}$   ▷ Un-assign attribute
        end
        $R[j] \leftarrow R[j] \cup \{a\}$        ▷ Assign to best partition
    end
    ▷ If solution is feasable and has lower cost
    if $H(R,B,Q) \geq \alpha \wedge L(R,B,Q) < c^*$ then
        $c^* \leftarrow L(R,B,Q)$                ▷ Update the lowest cost
        $\mathcal{P}(B) \leftarrow R$            ▷ Update the best partitioning
    end
end
return $\mathcal{P}(B)$                          ▷ Final set of sub-blocks
```

3.2 Overlapping Attributes

Problem. Find an overlapping partitioning of attributes that minimizes the query I/O and bounds the storage cost by some upper limit.

3.2.1 Integer Linear Program Formulation

We present an ILP formuation of the problem, as we did for the case of non-overlapping partitions in Section 3.1.1. We use the same set of variables and the same objective function. However, the formulation of the constraints differ.

Our first constraint is that, each attribute must be assigned to at leat one partition. Formally:

$$\forall_{a \in A}, \sum_{p=1}^{k} x_{a,p} \geq 1 \tag{14}$$

As our second constraint, we require that for each attribute used by a query, there needs to be a partition that is used by that query and that contains the attribute in question. Formally:

$$\forall_{\{a,q\} \in A \times Q}, \sum_{p=1}^{k} z_{a,p,q} \geq q(a) \tag{15}$$

As our third constraint, we require that if a query is using an attribute from a partition, then that partition must contain the attribute. I.e., we need to link the z variables with the x variables as $\forall_{\{a,p,q\} \in A \times [1..k] \times Q}, (z_{a,p,q} = 1) \implies (x_{a,p} = 1)$. This can be state as linear constraints:

$$\forall_{\{a,p,q\} \in A \times [1..k] \times Q}, x_{a,p} - z_{a,p,q} \geq 0 \tag{16}$$

As our fourth constraint, we require that if a query is using at least one attribute from a partition, then that partition must be used by the query. I.e., we need to link the z variables with the y variables as $\forall_{\{p,q\} \in [1..k] \times Q}, y_{p,q} = \mathbf{1}(\sum_{a \in A} z_{a,p,q} > 0)$. As before, we use the ILP construction from Eq. 9 for this, where $\beta_2 = y_{p,q}$ and $\beta_1 = \sum_{a \in A} z_{a,p,q}$. We get:

$$\begin{aligned} &\forall_{\{p,q\} \in [1..k] \times Q}, \ \sum_{a \in A} z_{a,p,q} - y_{p,q} \geq 0 \\ &\forall_{\{p,q\} \in [1..k] \times Q}, \ K \cdot y_{p,q} - \sum_{a \in A} z_{a,p,q} \geq 0 \end{aligned} \tag{17}$$

Our fifth constraint is that, if an attribute a is assigned to a partition p, and partition p is used by a query q, then the corresponding z variable must be set to 1. This is same as the formulation for the non-overlapping case from Eq. 11.

Our sixth constraint is that, if a partition is non-empty, then its corresponding u variable must be set to 0. Again, this is same as the formulation for the non-overlapping case from Eq. 12.

$$\begin{aligned}
\text{minimize} \sum_{q \in Q} \sum_{p=1}^{k} \ & (16 \cdot c_e(B) + 12 \cdot c_n(B)) \cdot y_{p,q} \\
& + \sum_{a \in A} s(a) \cdot c_e(B) \cdot z_{a,p,q} \\
\text{subject to} & \\
\forall_{a \in A}, \ & \sum_{p=1}^{k} x_{a,p} \geq 1 \\
\forall_{\{a,q\} \in A \times Q}, \ & \sum_{p=1}^{k} z_{a,p,q} \geq q(a) \\
\forall_{\{a,p,q\} \in A \times [1..k] \times Q}, \ & x_{a,p} - z_{a,p,q} \geq 0 \\
\forall_{\{p,q\} \in [1..k] \times Q}, \ & \sum_{a \in A} z_{a,p,q} - y_{p,q} \geq 0 \\
\forall_{\{p,q\} \in [1..k] \times Q}, \ & K \cdot y_{p,q} - \sum_{a \in A} z_{a,p,q} \geq 0 \\
\forall_{\{a,p,q\} \in A \times [1..k] \times Q}, \ & z_{a,p,q} - (x_{a,p} + y_{p,q}) \geq -1 \\
\forall_{p \in [1..k]}, \ & \sum_{a \in A} x_{a,p} - u_p \geq 0 \\
\forall_{p \in [1..k]}, \ & K \cdot u_p - \sum_{a \in A} x_{a,p} \geq 0 \\
\sum_{p=1}^{k} (16 \cdot c_e(B) \ & + \ 12 \cdot c_n(B)) \cdot u_p \\
& + \sum_{a \in A} s(a) \cdot c_n(B) \cdot x_{a,p} \leq s(B) \cdot (1 + \alpha)
\end{aligned}$$

Fig. 2 ILP formulation for the overlapping partitioning

Our seventh, and the last, constraint deals with the storage overhead. However, the storage overhead formulation for the overlapping case is different from the one for the non-overlapping. This is because the overhead does not merely depend on the number of partitions, as attributes might have to be loaded multiple times from different partitions. As a result, we express the overhead using base variables as in the objective function. Formally:

$$\begin{aligned} &\sum_{p=1}^{k} (16 \cdot c_e(B) \ + \ 12 \cdot c_n(B)) \cdot u_p \\ &\quad + \sum_{a \in A} s(a) \cdot c_n(B) \cdot x_{a,p} \leq s(B) \cdot (1 + \alpha) \end{aligned} \tag{18}$$

The final ILP formulation for the overlapping partitioning is given in Figure 2.

3.2.2 Heuristic Solution

Approach. We start the algorithm with a partitioning based on what queries we have seen. Every query gets its own sub-block. This is the“ideal” partitioning, because the I/O cost would be minimized for every query that we would have seen. As input to the algorithm, we provide a function dist, which is used to compute the distance between two blocks.

$$\text{dist} : block \to block \to int$$

Note that the dist function can be chosen arbitrarily. Examples of useful dist functions are the frequencies with which the queries are accessed, or the Jaccard index, which measures the set-similarity of the attributes.

Using this function, the algorithm iteratively combines the two partitions that are closest together. Algorithm 4 shows the pseudocode for a brute-force approach to finding the closest paris that runs in $O(n^2)$ time. However, there is also a divide-and-conquer algorithm [?] for solving closest pair of points problem in $O(n \log n)$. After each combination of partitions, the algorithm calculate the storage cost for the partitioning. The algorithm stops when the storage cost is below some specified threshold. The result is the block partitioning.

Algorithm 3: Algorithm for partitioning blocks into sub-blocks with overlapping attributes.

```
Data: $B$: block, $Q$: set of queries
$\mathcal{P}(B) \leftarrow \varnothing$;                          ▷ Sub-blocks;
for $q \in Q$ do                       ▷ Every query gets its own sub-block
    $\mathcal{P}(B) = \mathcal{P}(B) \cup q.A$
end
while $H(\mathcal{P}, B) \geq max$ do    ▷ While storage exceeds threshold
    x, y ← closestPair()
    $\mathcal{P}(B) \leftarrow \mathcal{P}(B) - \{x, y\}$
    $\mathcal{P}(B) \leftarrow \mathcal{P}(B) \cup \{x \cup y\}$
end
return $\mathcal{P}(B)$                          ▷ Final set of sub-blocks
```

Algorithm 4: Brute-force algorithm for closest points in $O(n^2)$.

```
$minDist \leftarrow \infty$
for $i \leftarrow 1$ to length($P$) $- 1$ do
    for $j \leftarrow i + 1$ to length($P$) do
        $p \leftarrow P[i]$
        $q \leftarrow P[j]$
        if dist($p, q$) $< minDist$ then
            $minDist \leftarrow$ dist($p, q$)
            $closestPair \leftarrow (p, q)$
        end
    end
end
return $closestPair$
```

References

1. Alagiannis, I., Idreos, S., Ailamaki, A.: "h2o: A hands-free adaptive store". In: Proc. ACM SIGMOD International Conference on Management of Data (2014). To appear
2. Gedik, B., Bordawekar, R.: Disk-based Management of Interaction Graphs. IEEE Transactions on Knowledge and Data Engineering **10**(1109) (2014)
3. Grund, M., Zeier, A., KrÃijger, J., Plattner, H., Madden, S.: HYRISE A Main Memory Hybrid Storage Engine. The VLDB Journal **4**(2), 105–116 (2010)