Range Updates and Range Sum Queries on Multidimensional Points with Monoid Weights

Shangqi Lu ✉
Chinese University of Hong Kong, New Territories, Hong Kong

Yufei Tao ✉
Chinese University of Hong Kong, New Territories, Hong Kong

Abstract

Let P be a set of n points in $\mathbb{R}^d$ where each point $p \in P$ carries a *weight* drawn from a commutative monoid $(\mathcal{M}, +, 0)$. Given a d-rectangle r_{upd} (i.e., an orthogonal rectangle in $\mathbb{R}^d$) and a value $\Delta \in \mathcal{M}$, a *range update* adds Δ to the weight of every point $p \in P \cap r_{\text{upd}}$; given a d-rectangle r_{qry}, a *range sum query* returns the total weight of the points in $P \cap r_{\text{qry}}$. The goal is to store P in a structure to support updates and queries with attractive performance guarantees. We describe a structure of $\tilde{O}(n)$ space that handles an update in $\tilde{O}(T_{\text{upd}})$ time and a query in $\tilde{O}(T_{\text{qry}})$ time for arbitrary functions $T_{\text{upd}}(n)$ and $T_{\text{qry}}(n)$ satisfying $T_{\text{upd}} \cdot T_{\text{qry}} = n$. The result holds for any fixed dimensionality $d \geq 2$. Our query-update tradeoff is tight up to a polylog factor subject to the OMv-conjecture.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis

Keywords and phrases Range Updates, Range Sum Queries, Data Structures, Lower Bounds

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2022.14

Funding This research was supported in part by GRF Projects 14207820 and 14203421 from HKRGC.

1 Introduction

This paper studies range sum queries on multidimensional points where the point weights are drawn from a commutative monoid and can be modified by range updates. Specifically, let P be a set of n points in $\mathbb{R}^d$ for some constant $d \geq 1$. Denote by $(\mathcal{M}, +, 0)$ an arbitrary commutative monoid[1] where each element in $\mathcal{M}$ is called a *weight*. Each point $p \in P$ carries a weight $w(p) \in \mathcal{M}$; initially, the weights are 0 for all the points. We want to store P in a data structure to support two operations with attractive performance guarantees:

- *Range (sum) query:* given a d-rectangle[2] r_{qry}, the query returns the total weight of all the points $p \in P \cap r_{\text{qry}}$ (where sum is defined using the monoid's operator $+$);
- *Range update:* given a d-rectangle r_{upd} and a weight $\Delta \in \mathcal{M}$, the update adds Δ to the weight of every point $p \in P \cap r_{\text{upd}}$.

We will refer to the above as *the "range sum with range updates"* (RSRU) *problem.* Our complexity analysis assumes the standard unit-cost RAM model and holds on all commutative monoids $(\mathcal{M}, +, 0)$ satisfying: (i) each weight $w \in \mathcal{M}$ can be stored in one word, and (ii) $w_1 + w_2$ can be computed in constant time for any $w_1, w_2 \in \mathcal{M}$.

[1] A commutative monoid $(\mathcal{M}, +, 0)$ is defined by a set $\mathcal{M}$, an operator $+$: $\mathcal{M} \times \mathcal{M} \to \mathcal{M}$ obeying associativity and commutativity, and an identity element $0 \in \mathcal{M}$ satisfying $0 + w = w$ for every $w \in \mathcal{M}$.
[2] Defined as $[a_1, b_1] \times \ldots \times [a_d, b_d]$.

33rd International Symposium on Algorithms and Computation (ISAAC 2022).
Editors: Sang Won Bae and Heejin Park; Article No. 14; pp. 14:1–14:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1.1 Previous Results

Supporting range queries and range updates has important implications in geographical information systems (GIS), online analytical processing (OLAP), and database management systems (DBMS); the reader may refer to [16, 19, 22, 24] for the relevant applications.

For $d = 1$, the RSRU problem admits a folklore structure[3] of $O(n)$ space that supports each query and update in $O(\log n)$ time. The problems become rather challenging as soon as d reaches 2. For any $d \geq 2$, the standard *range tree* [2, 10] uses $\tilde{O}(n)$ space and answers a query in $\tilde{O}(1)$ time (throughout the paper, the notation $\tilde{O}(.)$ suppresses a polylog n factor). It also supports a "point update" — an update whose rectangle r_{upd} degenerates into a point — in $\tilde{O}(1)$ time. Given an update with an arbitrary r_{upd}, however, the range tree issues a point update for each $p \in P \cap r_{\text{upd}}$ and thus can incur a cost of $\tilde{O}(n)$.

For $d \geq 2$, Lau and Ritossa [19] developed an $O(n)$-space structure that supports each query and update in $\tilde{O}(n^{1-1/d})$ time. They also showed a connection to the *OMv-conjecture* [12], which has been widely utilized to characterize the hardness of problems involving dynamic data structures [1, 3–9, 11, 13–15, 17, 18, 20, 21, 23]:

> In *online matrix-vector multiplication* (OMv), an algorithm A is allowed to preprocess an $n \times n$ boolean matrix $\mathbf{M}$ in poly(n) time and then, in the online phase, needs to compute $\mathbf{M}\boldsymbol{v}_i$ for $n \times 1$ boolean vectors $\boldsymbol{v}_1, ..., \boldsymbol{v}_n$ (additions and multiplications are as in the boolean semi-ring). The vectors are supplied in succession, i.e., $\boldsymbol{v}_{i+1}$ arrives only after A has output $\mathbf{M}\boldsymbol{v}_i$. The *cost* of A is the total time it spends in the online phase. The *OMv-conjecture* states that no algorithm can guarantee a cost of $O(n^{3-\delta})$ no matter how small the constant $\delta > 0$ is.

For $d = 2$, Lau and Ritossa [19] proved that, subject to the OMv-conjecture, no structure with update time T_{upd} and query time T_{qry} can guarantee $\max\{T_{\text{upd}}, T_{\text{qry}}\} = O(n^{1/2-\delta})$, regardless of the constant $\delta > 0$. Hence, their aforementioned structure can no longer be improved significantly in 2D space.

The results of [19] leave two intriguing questions. First, the hardness result does not shed much light on the *tradeoff* between T_{upd} and T_{qry}. For example, if we insist on $T_{\text{qry}} = \tilde{O}(1)$, is it possible to improve the update cost $\tilde{O}(n)$ of the range tree by a polynomial factor? Conversely, if T_{upd} must be $\tilde{O}(1)$, what is the best query time achievable? As yet another example, can we hope to obtain $T_{\text{upd}} = \tilde{O}(n^{0.5})$ and $T_{\text{qry}} = \tilde{O}(n^{0.49})$, thereby improving *only* the query time of [19] polynomially? The second question concerns the scenario of $d \geq 3$, where there remains a large gap between the upper and (conditional) lower bounds of [19]. We will answer all these questions in this paper.

The RSRU problem has a degenerated *array* version that has received special attention. In that version, $P := [m]^d$ where $m \geq 1$ is an integer (given an integer $x \geq 1$, $[x]$ represents the set $\{1, 2, ..., x\}$). In other words, P has exactly $n = m^d$ points, and each point's coordinate is an integer in $[m]$ on every dimension; equivalently, P can be regarded as a d-dimensional array. This RSRU variant can be settled by a structure of $O(n)$ space that supports a query and an update both in $O(\log^{d+1} n)$ time [24]. Furthermore, if the monoid is multiplicative[4], the query and update time can be reduced to $O(\log^d n)$ [24]; see also [16, 22] for (array-RSRU) structures designed for the monoid $(\mathbb{R}, +, 0)$ (that is, each weight is a real value).

---

[3] `https://cp-algorithms.com/data_structures/segment_tree.html`.

[4] A monoid $(\mathcal{M}, +, 0)$ is *multiplicative* if, for any weight $w \in \mathcal{M}$ and any integer $c \geq 1$, $c \cdot w := \underbrace{w + w + ... + w}_{c}$ can be calculated in constant time.

Space	Update, Query	Ref	Remark
$\tilde{O}(n)$	$\tilde{O}(n), \tilde{O}(1)$	[2]	$d \geq 2$
$O(n)$	$\tilde{O}(\sqrt{n}), \tilde{O}(\sqrt{n})$	[19]	$d = 2$
$O(n)$	$\tilde{O}(n^{1-1/d}), \tilde{O}(n^{1-1/d})$	[19]	$d \geq 3$
$\tilde{O}(n)$	any $\tilde{O}(T_{\text{upd}})$, $\tilde{O}(T_{\text{qry}})$ satisfying $T_{\text{upd}} \cdot T_{\text{qry}} = n$	this paper	$d \geq 2$
–	$\max\{T_{\text{upd}}, T_{\text{qry}}\} = O(n^{1/2-\delta})$ impossible	[19]	monoid $(\mathbb{R}, +, 0)$, $d = 2$ cond. on OMv-conjecture
–	$O(n^a), O(n^b)$ with $a + b < 1$ impossible (a, b are constants)	this paper	monoid $(\mathbb{R}, +, 0)$, $d = 2$ cond. on OMv-conjecture

Table 1 A comparison of our and previous results on the RSRU problem

1.2 New Results

For the RSRU problem, we establish a smooth trade-off between the update and query time under fixed dimensions $d \geq 2$:

▶ **Theorem 1.** *For the RSRU problem, there is a structure of $\tilde{O}(n)$ space that supports an update in $\tilde{O}(T_{\text{upd}})$ time and a query in $\tilde{O}(T_{\text{qry}})$ time for arbitrary functions $T_{\text{upd}}(n) \geq 1$ and $T_{\text{qry}}(n) \geq 1$ satisfying $T_{\text{upd}} \cdot T_{\text{qry}} = n$. The result holds for any constant dimension $d \geq 2$.*

By setting $T_{\text{upd}} = T_{\text{qry}} = \sqrt{n}$, we obtain a structure of $\tilde{O}(n)$ space that handles an update/query in $\tilde{O}(\sqrt{n})$ time for any d. Compared to [19], for $d = 2$ we obtain the same update and query time (up to a polylog factor), whereas for $d \geq 3$ our update and query time is better by a polynomial factor. The theorem, interestingly, also captures the range tree as a special case with $T_{\text{upd}} = n$ and $T_{\text{qry}} = 1$. By adjusting T_{upd} and T_{qry}, one can obtain a series of structures with different update-query tradeoffs that were not known previously. Our structures are drastically different from the ones in [19] and do not deteriorate with d (ignoring polylog factors).

We further prove that Theorem 1 is nearly tight subject to the OMv-conjecture.

▶ **Theorem 2.** *Consider the RSRU problem defined on $d = 2$ and the monoid $(\mathbb{R}, +, 0)$. Fix any constant c satisfying $0 \leq c < 1$ and an arbitrarily small constant $\delta > 0$. Subject to the OMv-conjecture, the following holds for any structure constructible in* poly(n) *time:*
- *if the update time $T_{\text{upd}} = O(n^c)$, then the query time T_{qry} cannot be $O(n^{1-c-\delta})$;*
- *if $T_{\text{qry}} = O(n^c)$, then T_{upd} cannot be $O(n^{1-c-\delta})$.*

The above clearly implies the impossibility of $\max\{T_{\text{upd}}, T_{\text{qry}}\} = O(n^{1/2-\delta})$, as was already proved in [19]. On the other hand, our conditional lower bounds are much more informative; for example, they reveal, somewhat unexpectedly, the range tree — with $T_{\text{qry}} = \tilde{O}(1)$ and $T_{\text{upd}} = \tilde{O}(n)$ — can no longer be improved significantly without breaking the OMv-conjecture. Putting together Theorems 1 and 2, we now have a complete picture on the query-update tradeoff achievable for the RSRU problem under any fixed dimension up to a sub-polynomial factor. Table 1 summarizes the comparison of our and previous results.

1.3 New Techniques

Our structures stem from a new observation on the inherent characteristics of the RSRU problem. The observation, described below, is interesting in its own right and illustrates what separates the RSRU problem from its array variant (defined in Section 1.1).

For any point $p \in \mathbb{R}^d$, we use $p[i]$ $(i \in [d])$ to represent its coordinate on dimension i. Similarly, given a d-rectangle $r := [a_1, b_1] \times ... \times [a_d, b_d]$, we use $r[i]$ to represent its i-th projection $[a_i, b_i]$. Given a subset $S \subseteq [d]$, we define an *S-rectangle* r as a d-rectangle where $r[i] := (-\infty, \infty)$ for every $i \in [d] \setminus S$, namely, r can have a bounded range $r[i]$ only on the dimensions $i \in S$.

Given an update with rectangle r_{upd} and some weight, we call it a *U-update* for some $U \subseteq [d]$ if r_{upd} is a U-rectangle. Likewise, given a query with rectangle r_{qry}, we call it a *Q-query* for some $Q \subseteq [d]$ if r_{qry} is a Q-rectangle.

▶ **Definition 3.** *Fix two (possibly overlapping) subsets U and Q of $[d]$. A (U, Q)-*structure *is a structure that supports only U-updates and Q-queries.*

Our objective in the RSRU problem is to design a $([d], [d])$-structure. We are now ready to state our characteristic observation:

▶ **Theorem 4.** *For the RSRU problem, suppose that, given any* <u>*disjoint*</u> *$U \subseteq [d]$ and $Q \subseteq [d]$, there is a (U, Q)-structure of $\tilde{O}(n)$ space that guarantees update time T_{upd} and query time T_{qry}. Then, there is a $([d], [d])$-structure of $\tilde{O}(n)$ space that handles an update in $O(T_{\text{upd}} \cdot \log^d n)$ time and a query in $O(T_{\text{qry}} \cdot \log^d n)$ time.*

The theorem indicates that the core of RSRU lies in dealing with updates and queries that concern *disjoint* sets of dimensions. For example, in 2D space, the core boils down to supporting $U = \{1\}$ and $Q = \{2\}$, namely, every update rectangle r_{upd} is a vertical slab while every query rectangle r_{qry} is a horizontal slab. Interestingly, this is precisely what separates general RSRU from its array variant. As we will see, when P is a 2D array, there is a trivial (U, Q)-structure of $O(1)$ space ensuring $O(\log n)$ update and query time (the time can even be reduced to $O(1)$ if the monoid is multiplicative); in contrast, when P is a generic set of Euclidean points, the hardness in Theorem 2 applies!

Theorem 4 has yet another notable implication: it "trivializes" the array version of RSRU and allows us to recover all the existing results from [16, 22, 24] (reviewed in Section 1.1) with a simple structure. The details can be found in Appendix A.

2 A Dimension Elimination Technique

This section is devoted to proving Theorem 4. Our strategy is to incrementally remove a common dimension of U and Q until the two dimension sets become disjoint, at which point we can apply the U-Q disjoint structure stated in the theorem's assumption statement. The core is to establish the following lemma.

▶ **Lemma 5.** *Consider any overlapping subsets U and Q of $[d]$. Let $i \in [d]$ be an arbitrary dimension in $U \cap Q$. Suppose that we have a $(U \setminus \{i\}, Q)$-structure and a $(U, Q \setminus \{i\})$-structure both of which use $O(n \log^c n)$ space (where $c \geq 0$ is a constant) and support an update in $O(T_{\text{upd}})$ time and a query in $O(T_{\text{qry}})$ time. Then, there is a (U, Q)-structure of $O(n \log^{c+1} n)$ space that handles an update in $O(T_{\text{upd}} \log n)$ time and a query in $O(T_{\text{qry}} \log n)$ time.*

Before proving the lemma, let us first see how it leads to Theorem 4.

Proof of Theorem 4. We will establish a more general claim: fix any integer $k \in [0, d]$; for any subsets U and Q of $[d]$ such that $|U \cap Q| = k$, there is a (U, Q)-structure of $\tilde{O}(n)$ space that guarantees update and query time $O(T_{\text{upd}} \log^k n)$ and $O(T_{\text{qry}} \log^k n)$, respectively. When $k = 0$, U and Q are disjoint and the claim directly follows from the theorem's assumption.

Next, we will prove the claim for $k = k_0 + 1$, assuming the claim's correctness on $k = k_0 \geq 0$. Identify an arbitrary $i \in U \cap Q$; i must exist because $|U \cap Q| = k_0 + 1 \geq 1$. By the inductive assumption, there exist a $(U \setminus \{i\}, Q)$-structure and a $(U, Q \setminus \{i\})$-structure, both of which use $\tilde{O}(n)$ space and ensure update time $O(T_{\text{upd}} \log^{k_0} n)$ and query time $O(T_{\text{qry}} \log^{k_0} n)$. We now apply Lemma 5 to obtain a (U, Q)-structure of $\tilde{O}(n)$ space with update and query time $O(T_{\text{upd}} \log^{k_0+1} n)$ and $O(T_{\text{qry}} \log^{k_0+1} n)$ time, respectively. This completes the proof. ◀

The rest of the section serves as a proof of Lemma 5. Section 2.1 will describe our structure as well as the update and query algorithms. Section 2.2 will present our analysis.

Basic Notations and Concepts. Let U and Q be the dimension sets in Lemma 5. Assume, w.l.o.g., that the value i in the lemma is 1, i.e., $1 \in U \cap Q$. For convenience, we will refer to dimension 1 as the "x-dimension". Accordingly, given a point $p \in \mathbb{R}^d$, its "x-coordinate" is $p[1]$. We will represent an update as (r_{upd}, Δ), where r_{upd} is a d-rectangle and Δ is a weight in $\mathcal{M}$; recall that the update adds Δ to the weight of every point $p \in P \cap r_{\text{upd}}$. We will use $r_{\text{upd}}[2:d]$ to denote the projection of r_{upd} onto dimensions $2, 3, ..., d$, namely, $r_{\text{upd}}[2:d]$ is a $(d-1)$-dimensional rectangle.

Given a set S of n real values, a binary search tree (BST) on S is a binary tree $\mathcal{T}$ such that (i) $\mathcal{T}$ has height $O(\log n)$, (ii) $\mathcal{T}$ has n leaves each storing a different value in S as its *key*, (iii) every internal node has two children, (iv) for each internal node, the elements of S in its left subtree are strictly less than those in its right subtree, and (v) each internal node stores a *key*, which is the smallest element of S in its right subtree. For each leaf/internal node u, denote its key as $key(u)$. The parent of a non-root node u is represented as $parent(u)$ and the root of $\mathcal{T}$ as $root(\mathcal{T})$.

We associate each node u of $\mathcal{T}$ with a *slab* $\sigma(u)$ defined recursively as follows. If $u = root(\mathcal{T})$, then $\sigma(u) := (-\infty, \infty)$. Otherwise, let $v := parent(u)$. If u is the left child of v, $\sigma(u) := \sigma(v) \cap (-\infty, key(v))$; otherwise, $\sigma(u) := \sigma(v) \cap [key(v), \infty)$. Slabs have several easy-to-verify properties:

- If node v is an ancestor of node u, then $\sigma(u) \subseteq \sigma(v)$.
- If u and v have no ancestor-descendant relationships, then $\sigma(u)$ and $\sigma(v)$ are disjoint.
- For each node u, $\sigma(u) \cap S$ is the set of elements stored in the subtree of u.

2.1 Structure and Algorithms

Denote by S the set of *distinct* x-coordinates of the points in P. Build a BST $\mathcal{T}$ on S. For each node u of $\mathcal{T}$, define

$$P_u \quad := \quad \{p \in P \mid p[1] \in \sigma(u)\}$$

namely, the set of points $p \in P$ whose x-coordinates are in the slab $\sigma(u)$ of u. We associate each u with a $(U \setminus \{1\}, Q)$-structure and a $(U, Q \setminus \{1\})$-structure both constructed on P_u. Recall that the two structures are already available by the assumption of Lemma 5. We will call each of them a *secondary structure* on P_u. This completes the description of our (U, Q)-structure.

Each $p \in P$ is in $O(\log n)$ secondary structures. For each secondary structure Υ, define

$$\text{weight of } p \text{ in } \Upsilon \quad := \sum_{(r_{\text{upd}}, \Delta) \in \mathcal{U}_\Upsilon : p \in r_{\text{upd}}} \Delta$$

where $\mathcal{U}_\Upsilon$ is the set of updates[5] ever performed on Υ.

[5] More specifically, each update $(r_{\text{upd}}, \Delta) \in \mathcal{U}$ should be treated as a pair with an id because two updates

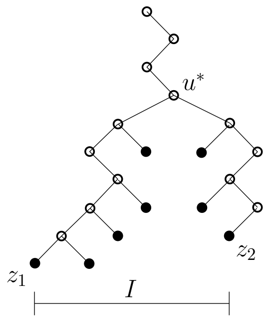


Figure 1 White dots are the internal path nodes of I and black dots are the canonical nodes of I.

Canonical and Internal Path Nodes of an Interval. To pave the way for our discussion, next we define what are the canonical and internal path nodes of an interval $I := [x_1, x_2]$, where both x_1 and x_2 belong to S. Let z_1 and z_2 be the leaves whose keys equal x_1 and x_2, respectively. Denote by π_1 (resp., π_2) the path from $root(\mathcal{T})$ to z_1 (resp., z_2).

- We call u an *internal path node* of I if u is an internal node on π_1 or π_2.
- We call u a *canonical node* of I if
  - $u = z_1$ or z_2, or
  - $parent(u)$ is in $\pi_1 \cup \pi_2$, u itself is not in $\pi_1 \cup \pi_2$, and $\sigma(u)$ is covered by I.

Let $\mathcal{C}_I$ be the set of canonical nodes of I. We must have $|\mathcal{C}_I| = O(\log n)$.

As another way to understand $\mathcal{C}_I$, one can first identify the lowest node $u^* \in \pi_1 \cap \pi_2$ (this is the node where π_1 and π_2 diverge). If u^* is a leaf, it means $\pi_1 = \pi_2$ and u^* is the only node in $\mathcal{C}_I$. Now consider the case where u^* is an internal node. Let us descend the path π'_1 from u^* to z_1. Every time we descend into the left child of a node $v \neq u^*$ on π'_1, we add to $\mathcal{C}_I$ the right child of v (nothing is added if we descend into the right child of v). Perform also a symmetric process for the path from u^* to z_2. The $\mathcal{C}_I$ at this moment contains all the canonical nodes. See Figure 1 for an illustration.

Update Algorithm. Consider a U-update (r_{upd}, Δ) on our (U, Q)-structure (remember the structure only needs to support U-updates). W.o.l.g., assume that the x-range of r_{upd} has the form $[x_1, x_2]$ where both x_1 and x_2 belong to S.[6] We carry out the update using the following algorithm.

`update` (r_{upd}, Δ)
1. $I_{\text{upd}} \leftarrow r_{\text{upd}}[1]$ /* the x-range of r_{upd} */
2. $r'_{\text{upd}} \leftarrow (-\infty, \infty) \times r_{\text{upd}}[2:d]$ /* r'_{upd} replaces the x-range with $(-\infty, \infty)$ */
3. **for** each internal path node u of I_{upd} **do**
4. perform an update (r_{upd}, Δ) on the $(U, Q \setminus \{1\})$-structure of P_u
5. **for** each canonical node u of I_{upd} **do**
6. perform an update $(r'_{\text{upd}}, \Delta)$ on the $(U \setminus \{1\}, Q)$-structure of P_u

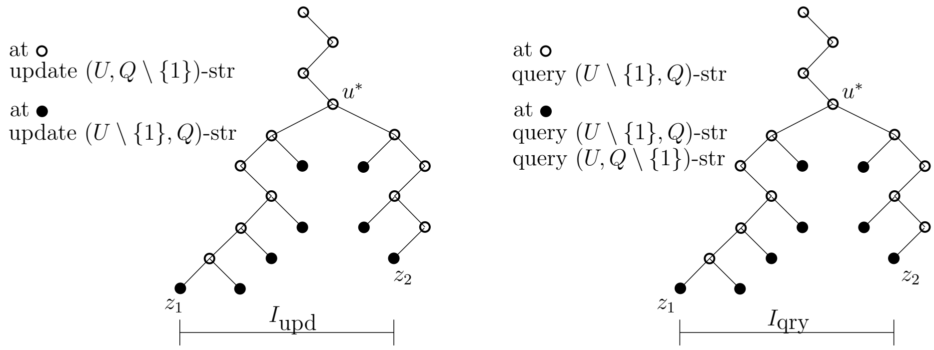
It is worth pointing out that r'_{upd} is a $U \setminus \{1\}$-rectangle. Hence, the update $(r'_{\text{upd}}, \Delta)$ at Line 6 is permitted on the $(U \setminus \{1\}, Q)$-structure of P_u. See Figure 2(a) for an illustration.

▶ **Proposition 6.** *Let Υ be a structure updated at Line 4 or 6 of* `update`*. Suppose that it is a secondary structure of P_u. For each $p \in P_u$, its weight in Υ increases by Δ if and only if $p \in r_{\text{upd}}$.*

---

can have the same (r_{upd}, Δ).

[6] This assumption can be easily fulfilled by performing predecessor/successor search in $O(\log n)$ time.



(a) Update (b) Query

Figure 2 Illustration of the update and query algorithms

Proof. This is obvious if Υ is a $(U, Q \setminus \{1\})$-structure of P_u (Line 4). Consider, instead, Υ as a $(U \setminus \{1\}, Q)$-structure of P_u (Line 6). It follows that u is a canonical node of I_{upd} and hence $p[1] \in I_{\text{upd}}$. By the assumption of Lemma 5, Υ increases the weight of p if and only if $p \in r'_{\text{upd}}$. Our claim holds because $p \in r'_{\text{upd}}$ if and only if $p \in r_{\text{upd}}$. ◀

Query Algorithm. Consider a Q-query with search rectangle r_{qry} on our (U, Q)-structure. W.o.l.g., we assume that the x-range of r_{qry} has the form $[x_1, x_2]$ where both x_1 and x_2 belong to S. Our query algorithm is shown below.

`query` (r_{qry})
1. $I_{\text{qry}} \leftarrow r_{\text{qry}}[1]$; $r'_{\text{qry}} \leftarrow (-\infty, \infty) \times r_{\text{qry}}[2:d]$
2. OUT $\leftarrow 0$
3. **for** each internal path node u of I_{qry} **do**
4. OUT $\leftarrow$ OUT + output of the query r_{qry} on the $(U \setminus \{1\}, Q)$-structure of P_u
5. **for** each canonical node u of I_{qry} **do**
6. OUT $\leftarrow$ OUT + output of the query r'_{qry} on the $(U \setminus \{1\}, Q)$-structure of P_u
7. OUT $\leftarrow$ OUT + output of the query r'_{qry} on the $(U, Q \setminus \{1\})$-structure of P_u
8. **return** OUT

The reader should note that r'_{qry} is a $Q \setminus \{1\}$-rectangle and hence also a Q-rectangle. Therefore, the queries at Lines 6 and 7 are permitted. See Figure 2(b) for an illustration.

▶ **Proposition 7.** *Let Υ be a structure searched at Line 4, 6, or 7 of* `query`. *Suppose that it is a secondary structure of P_u. For each $p \in P_u$, its weight in Υ is added into* OUT *if and only if $p \in r_{\text{qry}}$.*

Proof. This is obvious if Υ is a $(U \setminus \{1\}, Q)$-structure at Line 4. If Υ is a $(U \setminus \{1\}, Q)$-structure at Line 6 or a $(U, Q \setminus \{1\})$-structure at Line 7, u must be a canonical node of I_{qry} and hence $p[1] \in I_{\text{qry}}$. By the assumption of Lemma 5, when Υ is searched with r'_{qry}, its output incorporates the weight of p if and only if $p \in r'_{\text{qry}}$. Our claim holds because $p \in r'_{\text{qry}}$ if and only if $p \in r_{\text{qry}}$. ◀

2.2 Analysis

Space and Time Complexities. The update time and query time are clearly $O(T_{\text{upd}} \log n)$ and $O(T_{\text{qry}} \log n)$, respectively. The secondary structures of a node u in $\mathcal{T}$ occupy space $O(|P_u| \log^c n)$. As each point $p \in P$ appears in the P_u of $O(\log n)$ nodes u, the total space of our (U, Q)-structure is $O(n \log^{c+1} n)$.

Correctness. It remains to prove that all queries are answered correctly. Let us start with a concept crucial for our argument: update atom. Formally, each update (r_{upd}, Δ) generates an *atom* $(r_{\text{upd}}, \Delta, p)$ for every $p \in P \cap r_{\text{upd}}$. The atom describes the fact that the update should increase $w(p)$ by Δ. Conceptually, the effect of (r_{upd}, Δ) is achieved by "executing" all of its atoms.

Given a query with search rectangle r_{qry}, we will show that the output OUT of algorithm `query` is exactly $\sum_{p \in P \cap r_{\text{qry}}} w(p)$. Define

- $\mathcal{U}$ as the set of updates that have ever been performed on our (U, Q)-structure;
- $\mathcal{A}$ as the collection of atoms generated by the updates in $\mathcal{U}$.

Each atom $(r_{\text{upd}}, \Delta, p) \in \mathcal{A}$ is said to be *relevant* if $p \in r_{\text{qry}}$. For each $p \in P$, it holds that

$$w(p) = \sum_{(r_{\text{upd}}, \Delta, p) \in \mathcal{A}} \Delta$$

which yields

$$\sum_{p \in P \cap r_{\text{qry}}} w(p) = \sum_{p \in P \cap r_{\text{qry}}} \Big(\sum_{(r_{\text{upd}}, \Delta, p) \in \mathcal{A}} \Delta \Big) = \sum_{\text{relevant } (r_{\text{upd}}, \Delta, p) \in \mathcal{A}} \Delta. \tag{1}$$

Let Υ be a secondary structure searched at Line 4, 6, or 7 of `query`(r_{qry}). Denote by u the node that Υ is associated with. Define:

- $\mathcal{U}_\Upsilon$ as the set of updates $(r_{\text{upd}}, \Delta) \in \mathcal{U}$ such that algorithm `update`(r_{upd}, Δ) modifies Υ at either Line 4 or 6;
- $\mathcal{A}_\Upsilon$ as the collection of atoms $(r_{\text{upd}}, \Delta, p)$ generated by the updates in $\mathcal{U}_\Upsilon$ satisfying $p \in P_u$.

We will refer to $\mathcal{A}_\Upsilon$ as the *atom set* of Υ. By Proposition 6, it holds for each point $p \in P_u$:

$$\text{weight of } p \text{ in } \Upsilon \quad := \sum_{(r_{\text{upd}}, \Delta, p) \in \mathcal{A}_\Upsilon} \Delta.$$

By Proposition 7, when searched in algorithm `query`(r_{qry}), Υ returns:

$$\sum_{p \in P_u \cap r_{\text{qry}}} \text{weight of } p \text{ in } \Upsilon = \sum_{p \in P \cap r_{\text{qry}}} \Big(\sum_{(r_{\text{upd}}, \Delta, p) \in \mathcal{A}_\Upsilon} \Delta \Big) = \sum_{\text{relevant } (r_{\text{upd}}, \Delta, p) \in \mathcal{A}_\Upsilon} \Delta.$$

It follows from the above discussion that

$$\text{OUT} \quad = \sum_{\text{searched } \Upsilon} \Big(\sum_{\text{relevant } (r_{\text{upd}}, \Delta, p) \in \mathcal{A}_\Upsilon} \Delta \Big). \tag{2}$$

Our mission is to draw equivalence between (1) and (2). We achieve the purpose with the following lemma.

▶ **Lemma 8.** *Every relevant atom $(r_{\text{upd}}, \Delta, p) \in \mathcal{A}$ appears in the atom set $\mathcal{A}_\Upsilon$ of exactly one secondary structure Υ searched by* `query`(r_{qry}).

Proof. Consider any relevant atom $(r_{\text{upd}}, \Delta, p) \in \mathcal{A}$. Let $I_{\text{qry}} := r_{\text{qry}}[1]$. By definition of relevance, $p \in r_{\text{qry}}$. Among the canonical nodes of I_{qry}, there is exactly one node — denoted as u_{qry} — satisfying the condition that $p[1]$ falls in the slab $\sigma(u_{\text{qry}})$ of u_{qry}. Similarly, let $I_{\text{upd}} := r_{\text{upd}}[1]$. By definition of atom, $p \in r_{\text{upd}}$. Among the canonical nodes of I_{upd}, there is exactly one node — denoted as u_{upd} — satisfying $p[1] \in \sigma(u_{\text{upd}})$. Nodes u_{qry} and u_{upd} must have an ancestor-descendant relationship.

Fix a secondary structure Υ searched by `query`(r_{qry}) (at Line 4, 6, or 7). The next two facts follow from how `update`(r_{upd}, Δ) and `query`(r_{qry}) execute (as illustrated in Figure 2).

Fact 1. Suppose that Υ is the $(U \setminus \{1\}, Q)$-structure of node v. Then, $(r_{\text{upd}}, \Delta, p)$ appears in $\mathcal{A}_\Upsilon$ if and only if

- $v = u_{\text{upd}}$, and
- v is an ancestor of u_{qry} (this includes the case $v = u_{\text{qry}}$).

Fact 2. Suppose that Υ is the $(U, Q \setminus \{1\})$-structure of v. Then, $(r_{\text{upd}}, \Delta, p)$ appears in $\mathcal{A}_\Upsilon$ if and only if

- $v = u_{\text{qry}}$, and
- v is an internal path node of I_{upd}.

We proceed by discussing two cases separately:

Case 1: u_{upd} is a proper descendant of u_{qry}. Atom $(r_{\text{upd}}, \Delta, p)$ cannot belong to the atom set of any $(U \setminus \{1\}, Q)$-structure Υ searched by $\texttt{query}(r_{\text{qry}})$. Otherwise, Υ must be associated with u_{upd} (first bullet of Fact 1), but then the second bullet of Fact 1 contradicts u_{upd} being a proper descendant of u_{qry}. On the other hand, as a proper ancestor of u_{upd}, u_{qry} must be an internal path node of I_{upd}. Fact 2 thus shows that $(r_{\text{upd}}, \Delta, p)$ exists in the atom set of only one $(U, Q \setminus \{1\})$-structure searched by $\texttt{query}(r_{\text{qry}})$: the one at node u_{qry}.

Case 2: u_{upd} is an ancestor of u_{qry}. Atom $(r_{\text{upd}}, \Delta, p)$ cannot belong to the atom set of any $(U, Q \setminus \{1\})$-structure Υ searched by $\texttt{query}(r_{\text{qry}})$. To see why, suppose that such a Υ exists. By Fact 2, Υ must be associated with node u_{qry}, and u_{qry} must be an internal path node of I_{upd}. This is impossible because u_{upd} (being a canonical node of I_{upd}) cannot have any descendant that is an internal path node of I_{upd}. Finally, Fact 1 shows that $(r_{\text{upd}}, \Delta, p)$ appears in the atom set of only one $(U \setminus \{1\}, Q)$-structure searched by $\texttt{query}(r_{\text{qry}})$: the one at node u_{upd}. ◀

This completes the proof of Lemma 5.

3 U-Q Disjoint Structures

Equipped with Theorem 4, we can now concentrate on designing (U, Q)-structures with disjoint U and Q. We will prove:

▶ **Lemma 9.** *Fix an integer $k \geq 1$ and consider the RSRU problem under dimensionality $d = k$. Suppose that, for any disjoint $U, Q \subseteq [d]$, there is a (U, Q)-structure of $\tilde{O}(n)$ space supporting an update in $\tilde{O}(T_{\text{upd}})$ time and a query in $\tilde{O}(T_{\text{qry}})$ time for any functions $T_{\text{upd}}(n) \geq 1$ and $T_{\text{qry}}(n) \geq 1$ satisfying $T_{\text{upd}} \cdot T_{\text{qry}} = n$. Then, the following holds for dimensionality $d = k + 1$: for any disjoint $U, Q \subseteq [d]$, we can build a (U, Q)-structure of $\tilde{O}(n)$ space supporting an update in $\tilde{O}(T_{\text{upd}})$ and a query in $\tilde{O}(T_{\text{qry}})$ time for any functions $T_{\text{upd}}(n) \geq 1$ and $T_{\text{qry}}(n) \geq 1$ satisfying $T_{\text{upd}} \cdot T_{\text{qry}} = n$.*

Before delving into the proof, let us see how the lemma leads to Theorem 1.

Proof of Theorem 1. At $d = 1$, it is easy to obtain a $([1], [1])$-structure of $O(n)$ space and $O(\log n) = \tilde{O}(1)$ update and query time (see Section 1.1). The structure can serve as the basis solution for $k = 1$ and any $T_{\text{upd}}(n) \geq 1, T_{\text{qry}}(n) \geq 1$ with $T_{\text{upd}} \cdot T_{\text{qry}} = n$. Lemma 9 then asserts that, for any constant d and any disjoint $U, Q \subseteq [d]$, we can build a (U, Q)-structure that uses $\tilde{O}(n)$ space and handles an update in $\tilde{O}(T_{\text{upd}})$ and a query in $\tilde{O}(T_{\text{qry}})$ time for any $T_{\text{upd}}(n) \geq 1, T_{\text{qry}}(n) \geq 1$ satisfying $T_{\text{upd}} \cdot T_{\text{qry}} = n$. Combining this with Theorem 4 establishes Theorem 1. ◀

The rest of the subsection serves as a proof of Lemma 9. Let us first eliminate the case of $U = \emptyset$. In this scenario, the rectangle r_{upd} of an update is fixed to $\mathbb{R}^d$ and hence all points in P have the same weight. It suffices to maintain the $w(p^*)$ of an arbitrary $p^* \in P$. In addition, build a standard *range count* structure on P such that uses $\tilde{O}(n)$ space and,

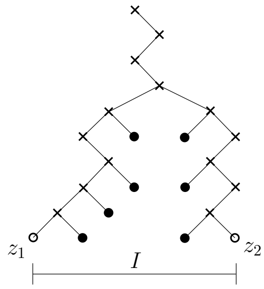


Figure 3 White dots are the path leaves of I and black dots are the non-path canonical nodes.

given a rectangle r_{qry}, outputs $|P \cap r_{\text{qry}}|$ in $\tilde{O}(1)$ time; the range tree [10] fulfills our purpose here. To answer a query with rectangle r_{qry}, we first obtain $c := |P \cap r_{\text{qry}}|$ and then return $c \cdot w(p^*)$. The query time is $\tilde{O}(1)$, noticing that $c \cdot w(p^*)$ can be calculated in $O(\log c)$ time[7].

Next, we assume $U \neq \emptyset$ and, w.l.o.g., consider that (i) U contains the x-dimension (i.e., dimension 1), (ii) $n := |P|$ is a power of two, and (iii) the points in P have distinct coordinates on each dimension. Fix any $T_{\text{upd}}(n) \geq 1$ and $T_{\text{qry}}(n) \geq 1$ satisfying $T_{\text{upd}} \cdot T_{\text{qry}} = n$.

Structure. We will describe a binary tree $\mathcal{T}$ of $O(\log T_{\text{qry}})$ levels and $O(T_{\text{qry}})$ nodes. Each node u in $\mathcal{T}$ is associated with a subset $P_u \subseteq P$ and an interval $\sigma(u)$ as its slab. If $u = \mathit{root}(\mathcal{T})$, $P_u := P$ and $\sigma(u) := (-\infty, \infty)$. In general, if $|P_u| \leq T_{\text{upd}}$, u is a leaf of $\mathcal{T}$. Otherwise, we split P_u evenly into P_1 and P_2 at some value x such that P_1 (resp., P_2) includes all the points of P_u whose x-coordinates are less (resp., greater) than x. The left and right children of u are associated with P_1 and P_2, respectively, and have slab $\sigma(u) \cap (-\infty, x)$ and $\sigma(u) \cap [x, \infty)$, respectively. The total number of nodes in $\mathcal{T}$ is $O(n/T_{\text{upd}}) = O(T_{\text{qry}})$.

Each internal node u in $\mathcal{T}$ is associated with a $(U \setminus \{1\}, Q)$-structure $\mathcal{T}_u$ on P_u. Since $(U \setminus \{1\}) \cap Q = \emptyset$ and $|(U \setminus \{1\}) \cup Q| \leq k$, we already know how to construct such a structure (see the assumption of Lemma 9). We parameterize $\mathcal{T}_u$ such that it supports an update on P_u in $\tilde{O}(T_{\text{upd}})$ time and answers a query on P_u in $\tilde{O}(|P_u|/T_{\text{upd}})$ time; its space is $\tilde{O}(|P_u|)$.

For each leaf z in $\mathcal{T}$, create a range tree $\mathcal{T}_z$ on P_z. As discussed in Section 1.1, $\mathcal{T}_z$ uses $\tilde{O}(|P_z|)$ space, answers a query on P_z in $\tilde{O}(1)$ time, and supports an update on P_z in $\tilde{O}(|P_z|) = \tilde{O}(T_{\text{upd}})$ time.

Each $p \in P$ appears in $O(\log T_{\text{qry}})$ secondary structures Υ. For every such Υ, define

$$\text{weight of } p \text{ in } \Upsilon \quad := \sum_{(r_{\text{upd}}, \Delta) \in \mathcal{U}_\Upsilon : p \in r_{\text{upd}}} \Delta$$

where $\mathcal{U}_\Upsilon$ is the set of updates ever performed on Υ.

Non-path Canonical Nodes and Path Leaves of an Interval. We now adapt the concepts "canonical" and "path nodes" from Section 2.1 to our context here. Consider an interval $I := [x_1, x_2]$. Let z_1 and z_2 be the leaves of $\mathcal{T}$ such that $x_1 \in \sigma(z_1)$ and $x_2 \in \sigma(z_2)$. Denote by π_1 (resp., π_2) the path from $\mathit{root}(\mathcal{T})$ to z_1 (resp., z_2).

- We call each of z_1 and z_2 a *path leaf* of I.
- We call u a *non-path canonical node* of I if $\mathit{parent}(u)$ is in $\pi_1 \cup \pi_2$, u itself is not in $\pi_1 \cup \pi_2$, and $\sigma(u)$ is covered by I.

See Figure 3 for an illustration.

[7] E.g., $15w = w + 2w + 4w + 8w$, where $4w$ (resp. $8w$) can be derived from $2w$ (resp. $4w$) in constant time.

Update. Consider an update (r_{upd}, Δ). Define $I_{\text{upd}} := r_{\text{upd}}[1]$ and $r'_{\text{upd}} := (-\infty, \infty) \times r_{\text{upd}}[2:d]$. At each non-path canonical node u of I_{upd}, perform an update $(r'_{\text{upd}}, \Delta)$ on $\mathcal{T}_u$. At each path leaf z of I_{upd}, perform an update (r_{upd}, Δ) on $\mathcal{T}_z$.

Query. Given a query with rectangle r_{qry}, we simply access every node u in $\mathcal{T}$ and issue a query with the same rectangle r_{qry} on the secondary structure $\mathcal{T}_u$. Then, we return the sum of the weights returned by those structures.

Analysis. It should have become straightforward that our structure uses $\tilde{O}(n)$ space overall and supports an update in $\tilde{O}(T_{\text{upd}})$ time. Next, we analyze the query time. As $\mathcal{T}$ has $O(T_{\text{qry}})$ leaves and a query spends $\tilde{O}(1)$ time on each leaf, the time spent on all the leaves is $\tilde{O}(T_{\text{qry}})$. Let us now attend to the internal nodes. Consider the i-th level of $\mathcal{T}$.[8]. There are $O(2^i)$ internal nodes and $|P_u| = O(n/2^i)$ for every such node u. The time spent on all the level-i nodes is $\tilde{O}(2^i \cdot (n/2^i)/T_{\text{upd}}) = \tilde{O}(n/T_{\text{upd}}) = \tilde{O}(T_{\text{qry}})$. As $\mathcal{T}$ has $\tilde{O}(1)$ levels, the overall query cost is $\tilde{O}(T_{\text{qry}})$.

It remains to show the correctness of our $(k+1)$-dimensional structure. For this purpose, let us first observe:

▶ **Proposition 10.** *For any $p \in P$, $w(p) = \sum_{\textit{node } u \textit{ in } \mathcal{T}: p \in P_u}(\text{weight of } p \text{ in } \mathcal{T}_u)$.*

Proof. The proposition obviously holds after the structure has just been constructed. Consider an update (r_{upd}, Δ). Define $I_{\text{upd}} := r_{\text{upd}}[1]$. Denote by z_1, z_2 the two path leaves of I_{upd} and by $\mathcal{C}$ the set of non-path canonical nodes of I_{upd}. It is easy to verify:

- for any distinct nodes u, v in $\{z_1, z_2\} \cup \mathcal{C}$, P_u and P_v are disjoint;
- $\bigcup_{u \in \{z_1, z_2\} \cup \mathcal{C}} (P_u \cap r_{\text{upd}}) = P \cap r_{\text{upd}}$.

For each point $p \in P \cap r_{\text{upd}}$, there is a unique node $u \in \{z_1, z_2\} \cup \mathcal{C}$ satisfying $p \in P_u$. Our update procedure increases the weight of p in $\mathcal{T}_u$ by Δ and does not change its weight in any other secondary structure. On the other hand, if $p \notin r_{\text{upd}}$, the procedure will not change its weight in any secondary structure. Therefore, if the proposition holds before the update, it still does afterwards. ◀

Fix any query with rectangle r_{qry}. For each node u in $\mathcal{T}$, denote by OUT_u the answer returned by the structure $\mathcal{T}_u$. The value OUT_u equals $\sum_{p \in P_u \cap r_{\text{qry}}}$ (weight of p in $\mathcal{T}_u$). The final answer returned is

$$\sum_{\text{node } u \text{ in } \mathcal{T}} \sum_{p \in P_u \cap r_{\text{qry}}} \text{weight of } p \text{ in } \mathcal{T}_u = \sum_{p \in P \cap r_{\text{qry}}} \Big(\sum_{\text{node } u \text{ in } \mathcal{T}: p \in P_u} \text{weight of } p \text{ in } \mathcal{T}_u \Big)$$
$$= \sum_{p \in P \cap r_{\text{qry}}} w(p)$$

where the last equality used Proposition 10. With this, we have established the correctness of our structure and thus conclude the proof of Lemma 9.

4 Hardness of RSRU

This section will establish Theorem 2. Let us first review the *γ-uMv problem* from [13]:

[8] The root is at level 0 and the level number increases by 1 each time we descend into a child.

> Fix a constant $\gamma > 0$, and choose two integers n_1 and n_2 satisfying $n_1 = \lfloor n_2^\gamma \rfloor$. In the *$\gamma$-uMv problem*, an algorithm A is allowed to preprocess an $n_1 \times n_2$ boolean matrix $\mathbf{M}$ in $\mathrm{poly}(n_1, n_2)$ time, after which A receives a $1 \times n_1$ boolean vector $\boldsymbol{u}$ and an $n_2 \times 1$ boolean vector $\boldsymbol{v}$, and needs to compute $\boldsymbol{u}\mathbf{M}\boldsymbol{v}$ (additions and multiplications are as in the boolean semi-ring). The *cost* of A is the time it spends on computing $\boldsymbol{u}\mathbf{M}\boldsymbol{v}$.

The following result is due to Henzinger et al. [13]:

▶ **Lemma 11** ([13]). *Fix an arbitrary constant $\gamma > 0$. Subject to the OMv-Conjecture, no algorithm can solve the γ-uMv problem with cost $O(n_1^{1-\delta} \cdot n_2 + n_1 \cdot n_2^{1-\delta})$, no matter how small the constant $\delta > 0$ is.*

Given an RSRU structure defying Theorem 2, we will show how to utilize it to develop an algorithm to beat Lemma 11. We use $\mathbf{M}[i, j]$ to denote the entry of $\mathbf{M}$ at the i-th row and j-th column, $\boldsymbol{u}[i]$ to denote the i-th component of $\boldsymbol{u}$, and $\boldsymbol{v}[j]$ to denote the j-th component of $\boldsymbol{v}$, where $i \in [n_1]$ and $j \in [n_2]$.

Proof of the First Bullet of Theorem 2. Consider the RSRU problem under $d = 2$ and monoid $(\mathbb{R}, +, 0)$ and let constants $c \in [0, 1)$ and $\delta > 0$ be chosen as in Theorem 2. Define $U := \{1\}$ and $Q := \{2\}$. We will prove that, subject to the OMv-conjecture, no (U, Q)-structure constructible in $\mathrm{poly}(n)$ time can guarantee update time $O(n^c)$ and query time $O(n^{1-c-\delta})$. This will imply the first bullet of the theorem.

Assume that such a structure Υ exists. Set $\gamma := \frac{1-c-\delta/2}{c+\delta/2}$. Next, we will describe an algorithm for the γ-uMv problem. In preprocessing, we create a set P of 2D points as follows: P has a point (i, j) if and only if $\mathbf{M}[i, j] = 1$ for each $i \in [n_1]$ and $j \in [n_2]$. Initialize $w(p) := 0$ for all $p \in P$ and then create a (U, Q)-structure Υ on P. The preprocessing time is $\mathrm{poly}(n_1, n_2)$ because $|P| \le n_1 \cdot n_2$. Given vectors $\boldsymbol{u}$ and $\boldsymbol{v}$, we compute $\boldsymbol{u}\mathbf{M}\boldsymbol{v}$ by issuing at most n_1 U-updates and at most n_2 Q-queries. For each $i \in [n_1]$, if $\boldsymbol{u}[i] = 1$, we perform an update with rectangle $(r_{\mathrm{upd}}, 1)$ with $r_{\mathrm{upd}} := [i, i] \times (-\infty, \infty)$ on P, which effectively adds 1 to the weight of every point $p \in P$ satisfying $p[1] = i$. Then, for each $j \in [n_2]$, if $\boldsymbol{v}[j] = 1$, we perform a query with $r_{\mathrm{qry}} := (-\infty, \infty) \times [j, j]$ on P, which effectively checks whether any point $p \in P$ with $p[2] = j$ has a positive $w(p)$. The reader can verify that $\boldsymbol{u}\mathbf{M}\boldsymbol{v} = 1$ if and only if at least one of the queries returns a non-zero value.

To analyze the cost, set $\lambda := n_2^{1/(c+\delta/2)}$. As $n_1 = \lfloor n_2^\gamma \rfloor$, we have $n_1 = \Theta(\lambda^{1-c-\delta/2})$ and $n_2 = \Theta(\lambda^{c+\delta/2})$. The number of points in P is $O(n_1 \cdot n_2) = O(\lambda)$; hence, Υ ensures update time $O(\lambda^c)$ and query time $O(\lambda^{1-c-\delta})$. As the algorithm performs at most n_1 updates and at most n_2 queries, the total cost is

$$O(n_1 \cdot \lambda^c + n_2 \cdot \lambda^{1-c-\delta}) = O(\lambda^{1-\delta/2}) = O((n_1 \cdot n_2)^{1-\delta/2})$$

where the last step used $\lambda = \Theta(n_1 \cdot n_2)$. This contradicts Lemma 11.

Proof of the Second Bullet of Theorem 2. As before, define $U := \{1\}$ and $Q := \{2\}$. We will prove that, subject to the OMv-conjecture, no (U, Q)-structure constructible in $\mathrm{poly}(n)$ time can guarantee update time $O(n^{1-c-\delta})$ and query time $O(n^c)$. This will imply the second bullet of the theorem.

Assume that such a structure exists. We deploy it to tackle γ-uMv in the same way as before where $\gamma := \frac{c+\delta/2}{1-c-\delta/2}$. To analyze the cost, set $\lambda := n_2^{1/(1-c-\delta/2)}$. As $n_1 = \lfloor n_2^\gamma \rfloor$, we have $n_1 = \Theta(\lambda^{c+\delta/2})$, $n_2 = \Theta(\lambda^{1-c-\delta/2})$, and $|P| = O(n_1 \cdot n_2) = O(\lambda)$. The structure handles an update and query in $O(\lambda^{1-c-\delta})$ and $O(\lambda^c)$ time, respectively. Because at most n_1 updates and at most n_2 queries are performed, our algorithm's cost is $O(n_1 \cdot \lambda^{1-c-\delta} + n_2 \cdot \lambda^c) = O(\lambda^{1-\delta/2}) = O((n_1 \cdot n_2)^{1-\delta/2})$, contradicting Lemma 11.

Remark. We can extend the above lower bound to any monoid $(\mathcal{M}, +, 0)$ as long as there is a value $e^* \in \mathcal{M}$ satisfying $\sum_{i=1}^{c} e^* \neq 0$ for any $c \in [1, n]$. The only modification is in the online phase: for each $i \in [n_1]$ with $u[i] = 1$, add e^* (rather than 1) to $w(p)$ for all the points $p \in P$ satisfying $p[1] = i$. Then, we have $\boldsymbol{uMv} = 1$ if and only if at least one of the at most n_2 queries defined as before returns a non-zero value.

Appendix

A A Simpler Structure for the Array Variant of RSRU

Henceforth, we will focus on the array version of RSRU, defined in Section 1.1, where P is a d-dimensional array $[m]^d$ for some integer $m \geq 1$ (as a result, $n = m^d$). Our goal is to show:

▶ **Theorem 12.** *For the array variant of RSRU, there is a structure of $O(n)$ space that supports each query and update in $O(\log^{d+1} n)$ time. The query and update complexities can be improved to $O(\log^d n)$ if the underlying monoid is multiplicative.*

Recall that a monoid $(\mathcal{M}, +, 0)$ is *multiplicative* if $c \cdot w := \underbrace{w + w + ... + w}_{c}$ can be calculated in constant time for any weight $w \in \mathcal{M}$ and any integer $c \geq 1$. The monoid $(\mathbb{R}, +, 0)$ studied in [16, 22] is multiplicative; hence, the theorem subsumes the results in [16, 22] (reviewed in Section 1.1). For arbitrary commutative monoids, the extra $O(\log n)$ factor arises from the need to compute a multiplication $c \cdot w$ in $O(\log c)$ time; the integer c never exceeds n in our algorithms. In [24], Yang and Wan claimed a structure with query and update time $O(\log^d n)$, but a careful look at their definition reveals that their monoid is multiplicative; for non-multiplicative monoids, their query and update time both slow down by an $O(\log n)$ factor. Hence, Theorem 12 recovers the result of [24] as well. Our structures are drastically different from those in [16, 22, 24].

A.1 The Counterpart of Theorem 4

The characteristics of RSRU revealed by Theorem 4 extend to the array version as well:

▶ **Theorem 13.** *For the array variant of RSRU, suppose that, given any disjoint $U \subseteq [d]$ and $Q \subseteq [d]$, there is a (U, Q)-structure of $O(1)$ space that guarantees update time T_{upd} and query time T_{qry}. Then, there is a $([d], [d])$-structure of $O(n)$ space that handles an update in $O(T_{\text{upd}} \cdot \log^d n)$ time and a query in $O(T_{\text{qry}} \cdot \log^d n)$ time.*

To prove the theorem, we need the lemma below that echoes Lemma 5.

▶ **Lemma 14.** *Consider any two overlapping subsets U and Q of $[d]$. Let $i \in [d]$ be an arbitrary dimension in $U \cap Q$. Suppose that we have a $(U \setminus \{i\}, Q)$-structure and a $(U, Q \setminus \{i\})$-structure both of which use $O(m^{|U \cap Q|-1})$ space and support an update in $O(T_{\text{upd}})$ and a query in $O(T_{\text{qry}})$ time. Then, there is a (U, Q)-structure of $O(m^{|U \cap Q|})$ space that handles an update in $O(T_{\text{upd}} \log n)$ time and a query in $O(T_{\text{qry}} \log n)$ time.*

Proof. Due to symmetry, we assume $i = 1$. Let S be the set of *distinct* x-coordinates of the points in P. $|S| = m$ because P is an array. We use the same reduction in the proof Lemma 5 to obtain a (U, Q)-structure. Recall that $\mathcal{T}$ is a BST on S and $P_u := \{p \in P \mid p[1] \in \sigma(u)\}$ for every node u in $\mathcal{T}$. Associate each u with a $(U \setminus \{1\}, Q)$-structure and a $(U, Q \setminus \{1\})$-structure both constructed on P_u. The update and query algorithms require no changes and finish in $O(T_{\text{upd}} \log n)$ and $O(T_{\text{qry}} \log n)$ time, respectively. Since $\mathcal{T}$ has $O(m)$ nodes and the space at each node is $O(m^{|U \cap Q|-1})$, the total space is $O(m^{|U \cap Q|})$. ◀

Equipped with the above lemma, we will now prove a general claim: fix any integer $k \in [0, d]$; for any subsets U and Q of $[d]$ such that $|U \cap Q| = k$, there is a (U, Q)-structure of $O(m^k)$ space that guarantees update and query time $O(T_{\text{upd}} \log^k n)$ and $O(T_{\text{qry}} \log^k n)$, respectively. Theorem 13 then follows because $m^d = n$.

When $k = 0$, U and Q are disjoint and the claim holds from the theorem's assumption. Next, we will prove the claim for $k = k_0 + 1$, assuming the claim's correctness on $k = k_0 \geq 0$. Fix an arbitrary $i \in U \cap Q$. By the inductive assumption, there exist a $(U \setminus \{i\}, Q)$-structure and a $(U, Q \setminus \{i\})$-structure, both of which use $O(m^{k_0})$ space and ensure update and query time $O(T_{\text{upd}} \log^{k_0} n)$ and $O(T_{\text{qry}} \log^{k_0} n)$ time, respectively. We now apply Lemma 14 to obtain a (U, Q)-structure of $O(m^{k_0+1})$ space with update and query time $O(T_{\text{upd}} \log^{k_0+1} n)$ and $O(T_{\text{qry}} \log^{k_0+1} n)$ time, respectively. This completes the proof.

A.2 U-Q Disjoint Structures

Since P is a d-dimensional array $[m]^d$, henceforth, we consider only d-rectangles of the form $[a_1, b_1] \times ... \times [a_d, b_d]$, where $a_i \in [m]$ and $b_i \in [m]$ for all $i \in [d]$. Accordingly, a U-rectangle is redefined as a d-rectangle r satisfying $r[i] = [1, m]$ for every $i \in [d] \setminus U$, and similarly, a Q-rectangle r is a d-rectangle satisfying $r[i] = [1, m]$ for every $i \in [d] \setminus Q$.

We will show:

▶ **Lemma 15.** *Consider the array version of RSRU. For any disjoint $U \subseteq [d]$ and $Q \subseteq [d]$, there is a (U, Q)-structure of $O(1)$ space that supports an update and a query in $O(\log n)$ time. The update and query time can be improved to $O(1)$ if the underlying monoid $(\mathcal{M}, +, 0)$ is multiplicative.*

Combining Theorem 13 with the above lemma establishes Theorem 12. The rest of the subsection serves as a proof of Lemma 15.

Case 1: $Q = \emptyset$. In other words, the query rectangle r_{qry} always covers the whole $[m]^d$. It suffices to maintain the total weight of all the points: $s := \sum_{p \in P} w(p)$. A query obviously can be settled in $O(1)$ time. Given an update (r_{upd}, Δ), we first calculate the number c of points in P covered by r_{upd}. As P is a multidimensional array, this can be done in $O(1)$ time because $c = \prod_{i \in [d]} |r_{\text{upd}}[i] \cap [m]|$.[9] Then, we increase s by $c \cdot \Delta$, which takes $O(\log n)$ time, or $O(1)$ time if the monoid is multiplicative.

Case 2: $Q \neq \emptyset$. W.o.l.g., we will assume $Q = [\ell]$ for some integer $\ell \in [1, d]$; hence, $U \subseteq [\ell + 1, d]$. Given an *$\ell$-tuple* $t := (x_1, x_2, ..., x_\ell) \in [m]^\ell$, let $P(t) := \{t\} \times [m]^{d-\ell}$, i.e., the set of points $p \in P$ satisfying $p[i] = x_i$ for all $i \in [\ell]$. Define

$$w(t) := \sum_{p \in P(t)} w(p).$$

▶ **Proposition 16.** *For any ℓ-tuples t and t', it always holds that $w(t) = w(t')$.*

[9] If $r_{\text{upd}}[i] = [a_i, b_i]$, then $|r_{\text{upd}}[i] \cap [m]| = b_i - a_i + 1$.

Proof. Consider any update (r_{upd}, Δ). As r_{upd} is a U-rectangle, $r_{\text{upd}}[i] = [1, m]$ for each $i \in [\ell]$. The number c of points in $P(t) \cap r_{\text{upd}}$ is $\prod_{i \in [\ell+1,d]} |r_{\text{upd}}[i] \cap [m]|$. Likewise, $|P(t') \cap r_{\text{upd}}| = \prod_{i \in [\ell+1,d]} |r_{\text{upd}}[i] \cap [m]| = c$. Hence, both $w(t)$ and $w(t')$ will increase by $c \cdot \Delta$ after the update. The claim follows because $w(t) = w(t') = 0$ in the beginning (i.e., before the first update). ◀

Our structure simply maintains the $w(t^*)$ for an arbitrary ℓ-tuple t^*. Given a Q-query with rectangle r_{qry}, we first obtain in constant time the number c_1 of ℓ-tuples $t := (x_1, ..., x_\ell)$ satisfying $x_i \in r_{\text{qry}}[i]$ for every $i \in [\ell]$.[10] By Proposition 16 and the fact $r_{\text{qry}}[i] = [1, m]$ for every $i \in [\ell + 1, d]$ (r_{qry} is a Q-rectangle), the query answer is exactly $c_1 \cdot w(t^*)$, which can be computed in $O(\log n)$ time. Given an update (r_{upd}, Δ), we obtain in constant time the number c_2 of points in $P(t^*)$ covered by the U-rectangle r_{upd},[11] and then increase $w(t^*)$ by $c_2 \cdot \Delta$ in $O(\log n)$ time. Both the update and query time can be reduced to $O(1)$ if the monoid is multiplicative.

This completes the proof of Lemma 15.

References

1. Amir Abboud and Søren Dahlgaard. Popular conjectures as a barrier for dynamic planar graph algorithms. In *Proceedings of Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 477–486, 2016.
2. Jon Louis Bentley. Decomposable searching problems. *Information Processing Letters (IPL)*, 8(5):244–251, 1979.
3. Thiago Bergamaschi, Monika Henzinger, Maximilian Probst Gutenberg, Virginia Vassilevska Williams, and Nicole Wein. New techniques and fine-grained hardness for dynamic near-additive spanners. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1836–1855, 2021.
4. Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering conjunctive queries under updates. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 303–318, 2017.
5. Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering ucqs under updates and in the presence of integrity constraints. In *Proceedings of International Conference on Database Theory (ICDT)*, pages 8:1–8:19, 2018.
6. Christoph Berkholz and Maximilian Merz. Probabilistic databases under updates: Boolean query evaluation and ranked enumeration. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 402–415, 2021.
7. Katrin Casel and Markus L. Schmid. Fine-grained complexity of regular path queries. In *Proceedings of International Conference on Database Theory (ICDT)*, pages 19:1–19:20, 2021.
8. Raphaël Clifford, Allan Grønlund, Kasper Green Larsen, and Tatiana Starikovskaya. Upper and lower bounds for dynamic data structures on strings. In *Proceedings of Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 22:1–22:14, 2018.
9. Soren Dahlgaard. On the hardness of partially dynamic graph problems and connections to diameter. In *Proceedings of International Colloquium on Automata, Languages and Programming (ICALP)*, pages 48:1–48:14, 2016.
10. Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 3rd edition, 2008.

[10] $c_1 = \prod_{i \in [\ell]} |r_{\text{qry}}[i] \cap [m]|$.
[11] $c_2 = \prod_{i \in [\ell+1,d]} |r_{\text{upd}}[i] \cap [m]|$.

11 Maximilian Probst Gutenberg, Virginia Vassilevska Williams, and Nicole Wein. New algorithms and hardness for incremental single-source shortest paths in directed graphs. In *Proceedings of ACM Symposium on Theory of Computing (STOC)*, pages 153–166, 2020.

12 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proceedings of ACM Symposium on Theory of Computing (STOC)*, pages 21–30, 2015.

13 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. *CoRR*, abs/1511.06773, 2015.

14 Monika Henzinger, Andrea Lincoln, Stefan Neumann, and Virginia Vassilevska Williams. Conditional hardness for sensitivity problems. In *Innovations in Theoretical Computer Science (ITCS)*, pages 26:1–26:31, 2017.

15 Monika Henzinger, Andrea Lincoln, and Barna Saha. The complexity of average-case dynamic subgraph counting. *Electronic Colloquium on Computational Complexity*, page 157, 2021.

16 Nabil Ibtehaz, M. Kaykobad, and M. Sohel Rahman. Multidimensional segment trees can do range updates in poly-logarithmic time. *Theoretical Computer Science*, 854:30–43, 2021.

17 Ahmet Kara, Hung Q. Ngo, Milos Nikolic, Dan Olteanu, and Haozhe Zhang. Maintaining triangle queries under updates. *ACM Transactions on Database Systems (TODS)*, 45(3):11:1–11:46, 2020.

18 Ahmet Kara, Milos Nikolic, Dan Olteanu, and Haozhe Zhang. Trade-offs in static and dynamic evaluation of hierarchical queries. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 375–392, 2020.

19 Joshua Lau and Angus Ritossa. Algorithms and hardness for multidimensional range updates and queries. In *Innovations in Theoretical Computer Science (ITCS)*, pages 35:1–35:20, 2021.

20 Hung Le, Lazar Milenkovic, Shay Solomon, and Virginia Vassilevska Williams. Dynamic matching algorithms under vertex updates. In *Innovations in Theoretical Computer Science (ITCS)*, pages 96:1–96:24, 2022.

21 Shangqi Lu and Yufei Tao. Towards optimal dynamic indexes for approximate (and exact) triangle counting. In *Proceedings of International Conference on Database Theory (ICDT)*, pages 6:1–6:23, 2021.

22 Pushkar Mishra. On updating and querying sub-arrays of multidimensional arrays. *CoRR*, abs/1311.6093, 2013.

23 Yufei Tao and Ke Yi. Intersection joins under updates. *Journal of Computer and System Sciences (JCSS)*, 124:41–64, 2022.

24 Jason Yang and Jun Wan. On updating and querying submatrices. *CoRR*, abs/2010.13180, 2020.