


# Algorithms and Hardness for Multidimensional Range Updates and Queries

Joshua Lau 

Sydney, Australia

joshua.cs.lau@gmail.com

Angus Ritossa 

University of New South Wales, Sydney, Australia

a.ritossa@unsw.edu.au

---

## Abstract

Traditional orthogonal range problems allow queries over a static set of points, each with some value. Dynamic variants allow points to be added or removed, one at a time. To support more powerful updates, we introduce the GRID RANGE class of data structure problems over arbitrarily large integer arrays in one or more dimensions. These problems allow range updates (such as filling all points in a range with a constant) and queries (such as finding the sum or maximum of values in a range). In this work, we consider these operations along with updates that replace each point in a range with the minimum, maximum, or sum of its existing value, and a constant. In one dimension, it is known that segment trees can be leveraged to facilitate any  $n$  of these operations in  $\tilde{O}(n)$  time overall. Other than a few specific cases, until now, higher dimensional variants have been largely unexplored.

Despite their tightly-knit complexity in one dimension, we show that variants induced by *subsets* of these operations exhibit polynomial separation in two dimensions. In particular, no truly subquadratic time algorithm can support certain pairs of these updates simultaneously without falsifying several popular conjectures. On the positive side, we show that truly subquadratic algorithms can be obtained for variants induced by other subsets.

We provide two general approaches to designing such algorithms that can be generalised to online and higher dimensional settings. First, we give almost-tight  $\tilde{O}(n^{3/2})$  time algorithms for single-update variants where the update and query operations meet a set of natural conditions. Second, for other variants, we provide a general framework for reducing to instances with a special geometry. Using this, we show that  $O(m^{3/2-\epsilon})$  time algorithms for counting paths and walks of length 2 and 3 between vertex pairs in sparse graphs imply truly subquadratic data structures for certain variants; to this end, we give an  $\tilde{O}(m^{(4\omega-1)/(2\omega+1)}) = O(m^{1.478})$  time algorithm for counting simple 3-paths between vertex pairs.

**2012 ACM Subject Classification** Theory of computation → Data structures design and analysis; Mathematics of computing → Graph algorithms

**Keywords and phrases** Orthogonal range, Range updates, Online and Dynamic Data Structures, Fine-grained complexity, Cycle counting

**Digital Object Identifier** 10.4230/LIPIcs.ITCS.2021.35

**Related Version** A full version of the paper is available at <https://arxiv.org/abs/2101.02003>.

**Acknowledgements** We thank Tunan Shi, for suggesting the reduction from 2RANGEINVERSION-SQUERY to STATIC TRELLISED 2D GRID RANGE (+, set). We also thank Ray Li and anonymous reviewers for helpful suggestions and comments.

## 1 Introduction

Orthogonal range query problems are ubiquitous across various fields of Computer Science. In the simplest of these problems, a data set is modelled as a set of points in  $\mathbb{Z}^d$ , and the task is to design a data structure that can efficiently answer queries which ask: how many points lie within the (axis-aligned) orthogonal range  $[l_1, r_1] \times \dots \times [l_d, r_d]$ ? One can extend this



© Joshua Lau and Angus Ritossa;

licensed under Creative Commons License CC-BY

12th Innovations in Theoretical Computer Science Conference (ITCS 2021).

Editor: James R. Lee; Article No. 35; pp. 35:1–35:20

Leibniz International Proceedings in Informatics



Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

definition by assigning a value to each point in the input, and having queries ask instead for some aggregate (e.g. the maximum value) of the values of the points within the query range. This has been studied extensively [7, 4, 11, 22] (more in full version), along with models which ask to report all points in the range [2, 3]. *Dynamic* models, where a single point may be inserted or removed in a single operation, have also been studied [13, 12], corresponding to the addition or deletion of a single record. Queries may then be interspersed between these update operations, providing insight into the data set as it changes over time. Modelling data sets in this way has proven useful for Online Analytical Processing (OLAP) [15, 24] of databases.

In practice, however, one may wish to employ more powerful updates. In this work, we examine data structures which support updating the values of all points that fall within a range in addition to range queries. This models updates to the records in a database table whose numerical field values fall within designated ranges. For instance, this could be giving all employees who have been employed between 5 to 10 years, and have KPIs between 80 and 90 an “A” rating. We formalise this as follows.

► **Definition 1.1.** *Let  $d$  be a positive integer constant,  $(\mathbb{Z}, q)$  be a commutative semigroup<sup>1</sup> and  $U = \{u_j\}$  be a set of integer functions. The (dD) RANGE  $(q, U)$  problem gives as input a set  $P = \{x_1, \dots, x_p\}$  of  $p$  points in  $\mathbb{Z}^d$ . A corresponding integer value  $v_i$  is also given for each point  $x_i$ , each initially 0. It requests a series of operations, each taking one of the following forms:*

- *update $_j((l_1, r_1), \dots, (l_d, r_d))$ : for each  $x_i \in [l_1, r_1] \times \dots \times [l_d, r_d]$ , set  $v_i := u_j(v_i)$*
- *query $((l_1, r_1), \dots, (l_d, r_d))$ : compute and return  $q(P')$ , where  $P'$  is the multiset  $\{v_i : x_i \in [l_1, r_1] \times \dots \times [l_d, r_d]\}$ .*

*Importantly, the operations may include updates of different types, and operations may occur in any order. All operations are given **online** and no information is known about them before the preceding operations are performed. There are  $n = n_u + n_q$  operations in all, with  $n_u$  and  $n_q$  of the first and second forms, respectively.*

We are most interested in the case where update functions take the form  $*_c(x) = x * c$ , where  $*$  is a binary operation over the integers, and  $c$  is an operation-specific constant. Through a slight abuse of notation, we also use  $*$  to denote the set of all functions of the form  $*_c$ . We write  $*$  as a member of  $U$  as shorthand for  $*_c$  being a member of  $U$ , for all  $c$ . For example, RANGE  $(+, \{+, \max\})$  allows updates of the form  $+_c$  (increasing values by  $c$ ),  $\max_c$  (replacing values less than  $c$  with  $c$ ), and queries which ask for the sum of values in a range. For a single update function or binary operation  $u$ , we write RANGE  $(q, u)$  for short.

The GRID RANGE variants are those whose point set is  $P = [s]^d$ ;  $P$  is not given explicitly, but rather, is described in the input by the positive integer  $s$ . Hence, the size of input (and thus the running time) can be measured as a function of the number  $n$  of operations which occur, rather than the size of  $P$ . GRID RANGE problems will form our primary focus, but we first describe the context surrounding RANGE problems as a whole.

## 1.1 Motivation

In one dimension, RANGE problems can be viewed as operating over an array. In this case, balanced binary trees (such as binary search trees or *segment trees* [8]) over the array have been effective tools in solving RANGE problems. Such trees allow any range of the array to be canonically expressed as the disjoint union of  $O(\log p)$  subtrees of the tree.

<sup>1</sup> A *semigroup*  $(S, +)$  is a set  $S$  equipped with an associative binary operator  $+: S \times S \rightarrow S$ . A semigroup is *commutative* if  $x + y = y + x$  for all  $x, y \in S$ .

When the functions in  $U$  are closed under composition and each distributes<sup>2</sup> over  $q$ , the folklore technique of *lazy propagation* over a binary tree solves 1D RANGE ( $q, U$ ) in  $O(\log p)$  time in the worst case, per operation. Lazy propagation can be extended to support several types of updates: it can be applied to solve 1D RANGE ( $\max, \{+, \min, \text{set}, \max\}$ ) in worst-case  $O(\log p)$  time per operation, where set is the binary operation that returns its second operand. These techniques are described in more detail in Section 2. Ji [18] considered the 1D RANGE ( $+, \max$ ) problem, where the update operation does not distribute over the query operation, by introducing a technique known as a Ji-Driver Tree (informally “segment tree beats”), generalising lazy propagation. Using this technique, he showed that 1D RANGE ( $+, \{+, \min, \text{set}, \max\}$ ) can be solved in amortised  $O(\log^2 p)$  time per operation.

Let  $\mathfrak{B}$  be the set of RANGE ( $q, U$ ) problems with  $q \in \{+, \max\}$  and  $U \subseteq \{+, \min, \text{set}, \max\}$ . Motivated by the one dimensional results for problems in  $\mathfrak{B}$ , we seek general techniques addressing RANGE problems in two or more dimensions. This has been asked as an open question in the competitive programming community [19].

The RANGE problem on a  $p$  element array can be generalised to multiple dimensions in two natural ways: either a set of  $p$  points in  $\mathbb{Z}^d$  is provided explicitly, or the point set is considered to be  $[s]^d$ . In the former case, one dimensional techniques can be generalised to higher dimensions with the aid of an orthogonal space partition tree (hereafter simply *partition tree*), such as a *kd-tree* [7].

► **Lemma 1.2.** *If 1D RANGE ( $q, U$ ) on  $p$  points can be solved in time  $T(p)$  per operation and  $q$  is computable in  $O(1)$  time, then  $d$ D RANGE ( $q, U$ ) can be solved in time  $O(p^{1-1/d}T(p))$  per operation.*

We prove this result and provide almost-tight  $\Omega(p^{1/2-o(1)})$  time per-operation lower bounds conditioned on the Online-Matrix Vector (OMv) Conjecture of Henzinger et al. [16], for all RANGE problems in  $\mathfrak{B}$  when  $d = 2$ , in Section 7.

The latter case corresponds to the GRID RANGE class of problems, which is the subject of the remainder of this work. The same technique does not apply in these cases, as the number of points is  $p = s^d$ .

## 1.2 Prior work

Prior work on GRID RANGE problems has been limited to a few specific cases.

Since balanced binary trees have been rather effective in solving problems in one dimension, it is natural to ask whether they can be easily generalised to higher dimensions. Lueker [21] and Chazelle [13] generalised binary search trees and segment trees to higher dimensions to answer various  $d$ -dimensional range query problems (without updates) on sets of  $n$  points in  $O(\log^d n)$  time per query. This technique can be used to solve GRID RANGE problems in the special cases where all update ranges affect a single point, or when all query ranges contain a single point and the update functions are commutative. The latter can be seen as a generalisation of RECTANGLE STABBING [12] (which accepts a series of  $d$ -dimensional boxes and supports queries for the number of boxes covering a given point), a dual of traditional range queries. The same structure was used with scaling by Ibtehaz et al. [17] to solve GRID RANGE ( $+, +$ ) in worst-case  $O(\log^d s)$  time per operation; without scaling, a straightforward solution for GRID RANGE ( $\max, \max$ ) can be obtained in the same time. These upper bounds contrast with the abovementioned  $\Omega(p^{1/2-o(1)})$  time conditional lower bounds for

<sup>2</sup> An integer function  $u$  distributes over  $q$  if  $u(q(a, b)) = q(u(a), u(b))$

the corresponding 2D RANGE problems. For other problems in  $\mathfrak{B}$ , the lazy propagation technique used in one dimension cannot be directly applied over this structure, as it requires a partition tree of the coordinate space.

In the KLEE'S MEASURE problem,  $n$  rectangles are given in  $d$  dimensions, and one is asked to find the volume of their union. In WEIGHTED DEPTH, the rectangles each have a weight, and one is asked to find the maximum sum of weights of rectangles that cover a single point. In DYNAMIC versions of these problems, a single rectangle can be added or removed in a single operation, and the new answer must be returned after each one. These can be seen as special cases of GRID RANGE problems.

When only additions are supported, DYNAMIC KLEE'S MEASURE is a special case of GRID RANGE (+, set) or GRID RANGE (+, max). Overmars and Yap [23] gave a  $O(n^{(d+1)/2} \log n)$  time solution over  $n$  updates when *the rectangles' coordinates are known during preprocessing*. Chan [9] gave a sublogarithmic time improvement over this method and showed that this is nearly tight in two dimensions, giving an  $\Omega(\sqrt{n})$  time per update worst-case lower bound by reducing from DYNAMIC MATRIX-VECTOR MULTIPLICATION.

DYNAMIC WEIGHTED DEPTH is the special case of GRID RANGE (max, +), where the maximum value in the entire grid is queried after each update. It is closely related to DYNAMIC KLEE'S MEASURE, in that every known algorithm for one has been adaptable to the other, with running times differing by a sublogarithmic factor. Hence, with a slight modification of the result of Overmars and Yap [23], an  $\tilde{O}(n^{(d+1)/2})$  time algorithm for GRID RANGE (max, +) can be obtained. Chan [9] showed that DYNAMIC WEIGHTED DEPTH is solvable in time  $O(n^{(d+1)/2} \log^{d/2} \log n)$ , with sublogarithmic improvements specific to entire-grid queries.

When there are no updates, KLEE'S MEASURE and WEIGHTED DEPTH can be reduced to  $O(n)$  updates on a  $d - 1$  dimensional DYNAMIC instance with a sweepline, however Chan [10] gave faster  $O(n^{d/2 - o(1)})$  time algorithms for these. Reductions by Chan [9] and Backurs et al. [6] showed that solving static variants of KLEE'S MEASURE or WEIGHTED DEPTH in time  $o(n^{d\omega/6})$  or  $O(n^{d/2 - \epsilon})$  for some  $\epsilon > 0$ , respectively, would improve longstanding upper bounds for CLIQUE or MAX CLIQUE, respectively. It follows that these problems are W[1]-complete for parameter  $d$ . The same reductions can be appropriated for  $d = 3$  to show that an  $O(n^{3/2 - \epsilon})$  time algorithm for 3D WEIGHTED DEPTH implies an  $O(n^{3 - 2\epsilon})$  time algorithm for NEGATIVE TRIANGLE. A truly subcubic algorithm for NEGATIVE TRIANGLE exists if and only if one exists for APSP [26], so an  $O(n^{3/2 - \epsilon})$  time algorithm for 2D GRID RANGE (max, +) for some  $\epsilon > 0$  would falsify the APSP Conjecture.

### 1.3 Our contribution

Our main contributions are conditional lower bounds, which serve to elucidate and categorise the expressiveness of these data structures, and upper bounds in the form of general algorithms and frameworks for solving GRID RANGE problems (such as those in  $\mathfrak{B}$ ) in two or more dimensions. We do so on a Word RAM with  $\Omega(\log n)$ -bit words. In the sequel, we use  $\mathfrak{B}'$  to denote  $\mathfrak{B}$  without the problems GRID RANGE (+, +) and GRID RANGE (max, max), for which per-operation polylogarithmic time upper bounds are already known.

**Lower bounds.** First, we consider algorithms whose per-operation time complexity is a function of  $s$ , the side length of the grid. In two dimensions, we show that there is no algorithm running in time  $O(s^{1 - \epsilon})$  per-operation for any  $\epsilon > 0$ , for any problem in  $\mathfrak{B}'$ , unless the OMv Conjecture is false. Further, for GRID RANGE (+, {+, max}) we obtain identical lower bounds, conditioned on the “extremely popular conjecture” of Abboud et al. [1] that at

■ **Table 1** Lower and upper bounds for GRID RANGE problems in  $\mathfrak{B}$ , exhibiting polynomial separation. All results are in two dimensions where  $d$  is unspecified, and in  $d$  dimensions otherwise, where  $d$  is a constant. All lower bounds hold offline, except those for OMv, and all upper bounds hold in fully online settings.

$q$	$U$	Lower bounds	Upper bound
max	max		$O(n \log^d s)$ (extension of segment trees)
+	+		$O(n \log^d s)$ [17]
max	+	$\Omega(n^{3/2-o(1)})$ APSP [6] or OMv $\Omega(n^{(d+1)/2-o(1)})$ MAXCLIQUE [6]	$\tilde{O}(n^{3/2})$ and $O(n \log^2 n)$ space $\tilde{O}(n^{(d+1)/2})$ [9, 10]
max	min	$\Omega(n^{3/2-o(1)})$ OMv	$\tilde{O}(n^{(d+1)/2})$
max	set		
max	{min, set, max}		
+	set		
+	{+, set}		
+			
max	{+, min}	$\Omega(n^{d-o(1)})$ $d$ -OV	$\tilde{O}(n^d)$
max	{+, min, set, max}		
+	{+, max}	$\Omega(n^{2-o(1)})$ 3SUM $\Omega(n^{d-o(1)})$ $d$ -OV	
+	{+, min, set, max}		

least one of the APSP, 3SUM and 2-OV (Orthogonal Vectors) Conjectures are true (the latter of which is implied by the Strong Exponential Time Hypothesis [27]). Hence, we cannot solve this variant in two dimensions in  $O(s^{1-\epsilon})$  time per operation for any  $\epsilon > 0$  without making a powerful breakthrough across several fields of Theoretical Computer Science. For GRID RANGE (max, {+, min}) and GRID RANGE (+, {+, max}), we generalise our results to  $d$  dimensions under the  $d$ -OV Conjecture to obtain an  $\Omega(s^{(d-1)-o(1)})$  time per-operation lower bound. All these lower bounds are almost-tight, as  $\tilde{O}(s^{d-1})$  time per-operation upper bounds can be obtained by maintaining  $s^{d-1}$  one dimensional instances.

These results, however, do not preclude the existence of efficient and practical algorithms for GRID RANGE problems whose overall complexity is a function of the number of operations,  $n$  – the true size of the input – rather than  $s$ . Our aforementioned lower bounds under the OMv and APSP Conjectures translate to conditional  $\Omega(n^{3/2-o(1)})$  lower bounds for all problems in  $\mathfrak{B}'$ , but our lower bound under the 3SUM Conjecture translates to conditional lower bounds of  $\Omega(n^{2-o(1)})$  for 2D GRID RANGE (+, {+, max}). In  $d$  dimensions, our lower bounds under the  $d$ -OV Conjecture translate to  $\Omega(n^{d-o(1)})$  time conditional lower bounds for GRID RANGE (max, {+, min}) and GRID RANGE (+, {+, max}). Our lower bounds are summarised in Table 1 and proven in Section 3.

**Upper bounds.** By reducing to algorithms in one dimension,  $\tilde{O}(n^d)$  time algorithms can be found for all these problems. Hence, we aim to determine which problems in  $\mathfrak{B}'$  can be solved more efficiently, by seeking truly subquadratic time algorithms in two dimensions. To this end, we provide two general frameworks for developing such algorithms, both based on the approach of Overmars and Yap [23] for DYNAMIC KLEE’S MEASURE. Their algorithm constructs a partition tree of the grid such that the rectangles intersecting each leaf region form a “trellis” pattern. This requires the coordinates of all rectangles to be known during precomputation.

First, we provide a fully-online generalisation of this approach, that does not require coordinates to be known ahead of time.

► **Theorem 1.3.** *Suppose  $q$  and  $u$  are associative, commutative binary operations, computable in  $O(1)$  time, such that  $u$  distributes over  $q$ , and  $0$  is an identity of  $u$ . Then GRID RANGE  $(q, u)$  can be solved in  $\tilde{O}(n^{(d+1)/2})$  time.*

This algorithm does not apply to problems such as the RANGE  $(+, \text{set})$  problem described in the abstract, as set does not distribute over  $+$ . Motivated by this, in Section 5 we show that in two dimensions, efficient solutions to static GRID RANGE  $(q, U)$  instances where the update ranges form the same “trellis” pattern can be used to give fully-online, truly subquadratic time solutions to many GRID RANGE  $(q, U)$  problems. We also extend these results to multiple dimensions, giving a detailed proof in the full version.

As an application, we use this approach to give a truly subquadratic time algorithm for 2D GRID RANGE  $(\max, \{\min, \text{set}, \max\})$ . We do the same for 2D GRID RANGE  $(+, \text{set})$  and 2D GRID RANGE  $(+, \{+, \text{set}\})$  in Section 6, by drawing an equivalence and a reduction between the respective “STATIC TRELLISED” instances and counting the number of 2- and 3-edge paths between vertex pairs, respectively. To this end, we prove the following result.

► **Theorem 1.4.** *Let  $G$  be a graph with  $m$  edges and  $O(m)$  vertices. The number of 3-edge walks between each of  $q$  vertex pairs in  $G$  can be found in  $O(m^{2\omega/(2\omega+1)}(m+q)^{(2\omega-1)/(2\omega+1)})$  time.*

In this way, queries on static graphs yield efficient, fully-online dynamic GRID RANGE data structures. We find it somewhat surprising that, though all of the problems in  $\mathfrak{B}$  can be solved in  $\tilde{O}(n)$  time in one dimension, our upper and lower bounds imply likely polynomial separation in two or more dimensions (see Table 1).

Lastly, we provide a fully-online algorithm for 2D GRID RANGE  $(\max, +)$  that uses  $O(n \log^2 n)$  space, and runs in a time comparable to that of existing algorithms.

► **Theorem 1.5.** *2D GRID RANGE  $(\max, +)$  can be solved in  $\tilde{O}(n^{3/2})$  time, and  $O(n \log^2 n)$  space.*

The proof of this result is omitted for space, and proven in the full version.

While our complexity is slower than existing results by Chan [9] by a polylogarithmic factor, those require  $O(n^{3/2+o(1)})$  space and are not fully-online. Overmars and Yap [23] also gave an  $O(n)$  space algorithm for static KLEE’S MEASURE in  $d$  dimensions using a swepline, but this does not apply in the dynamic case.

In the remaining sections, due to space constraints, we omit some proofs and sketch others. We refer the reader to the full version for full details and proofs.

## 2 Preliminaries

**Model of computation.** All results are described are for a Word RAM over  $l$ -bit words, with  $l = \Omega(\log n)$ . We further assume that any coordinates or values given in the inputs can be represented in a constant number of words, and that basic arithmetic and the (binary) operations used in range problems can be performed on a constant number of words in constant time. In particular,  $s = O(n^c)$ , for some constant  $c$ .

**Notation.** We use the notation  $\tilde{O}(f(n)) = O(f(n)\text{poly log } n)$  to hide polylogarithmic factors. Note that  $\log^c s = \log^c n$  for any constant  $c$ . Where our algorithms and proofs use positive or negative  $\infty$  as a value, this can be replaced with a suitably large value, for a given input instance.

Where  $x \leq y$  are real numbers, we denote by  $[x, y]$  the set of all *integers* between  $x$  and  $y$ , inclusive. When  $y \geq 1$ , we write  $[y]$  as shorthand for  $[1, y]$ .

The binary operation *set* is the operation whose value is its second operand. That is,  $\text{set}(a, b) = b$ .

$\omega < 2.37286$  [5] is the exponent of multiplying two  $n \times n$  integer matrices. We also write  $\omega(a, b, c)$  for the time taken to multiply an  $a \times b$  matrix by a  $b \times c$  matrix.

**Ancillary problems and variants.** In RANGE problems, we say that the  $i$ th operation (update or query) occurs at *time*  $i$ . In OFFLINE variants, all operations are provided together with the initial input, and in STATIC variants, it is guaranteed that all updates precede all queries.

We formalise and appropriate the “trellis” pattern observed by Overmars and Yap [23] for our use, as follows. Call a GRID RANGE instance TRELLISED if for each update, there is a dimension  $d^*$  such that  $[l_{d'}, r_{d'}] = (-\infty, \infty)$  for all  $d' \in [d] \setminus \{d^*\}$ . When  $d = 2$ , updates must either cover all points in a range of rows, or all points in a range of columns, which we call *row updates* and *column updates*, respectively.

**Segment trees and lazy propagation.** Let  $s$  be a power of two. A *segment tree* over an array  $A$  containing  $s$  elements is a complete rooted binary tree of ranges over  $[s]$ . The root is  $[1, s]$ , and each node  $[a, b]$  has two children:  $[a, h]$ ,  $[h + 1, b]$ , where  $h = (a + b - 1)/2$ . Hence, there are  $O(s)$  nodes in the tree, with a depth of  $\log s$ . Given an interval  $I \subseteq [s]$ , we can write  $I$  as a canonical disjoint union of a set  $\text{base}(I)$  of  $O(\log s)$  nodes. These are defined as the nodes closest to the root that are fully contained in  $I$ , and can be found recursively.

Segment trees can be used to prove the following folklore proposition.

► **Proposition 2.1** (Lazy propagation). *Suppose  $U$  is a set of update functions, and  $q$  is a query function, computable in  $O(1)$  time. If there is a set  $\bar{U}$  such that:*

1.  $U \subseteq \bar{U}$  are sets of functions that can be represented and composed in  $\tilde{O}(1)$  space and time, such that the composition of any series of at most  $n$  (possibly non-distinct) functions of  $U$  results in a function in  $\bar{U}$ ; and
  2. For each  $u \in \bar{U}$ ,  $u$  distributes over  $q$
- then 1D GRID RANGE  $(q, U)$  is solvable in  $\tilde{O}(n)$  time.

**Hardness conjectures.** We base hardness on the following popular conjectures. The first is a conjecture of Henzinger et al. [16].

► **Conjecture 2.2** (OMv Conjecture). *No (randomized) algorithm can process a given  $m \times m$  boolean matrix  $M$ , and then in an online way compute the  $(\vee, \wedge)$ -product  $Mv_i$  for any  $m$  boolean vectors  $v_1, \dots, v_m$  in total time  $O(m^{3-\epsilon})$ , for any  $\epsilon > 0$ .*

In the OuMv problem, the same matrix  $M$  is given during preprocessing, and  $m$  pairs of boolean query vectors  $(u_1, v_1), \dots, (u_m, v_m)$  are given online. For each, the value of the product  $u_i^T Mv_i$  is requested. Note that the answer to each of these queries is a single bit. Henzinger et al. showed that if OuMv can be solved in  $O(m^{3-\epsilon})$  time, then OMv can be solved in  $O(m^{3-\epsilon/2})$  time, so these problems are subcubic equivalent.

We refer the reader to the survey by Vassilevska Williams [28] for more details on the remaining conjectures.

► **Conjecture 2.3** (APSP Conjecture). *No (randomized) algorithm can solve ALL-PAIRS SHORTEST PATHS (APSP) in  $O(v^{3-\epsilon})$  time for  $\epsilon > 0$ , on  $v$  vertex graphs with edge weights in  $\{-v^c, \dots, v^c\}$  and no negative cycles, for large enough  $c$ .*

► **Definition 2.4** ( $k$ -OV problem). *Let  $k \geq 2$  be a constant, and  $z = \omega(\log n)$ . Given  $k$  sets  $A_1, \dots, A_k \subseteq \{0, 1\}^z$  with each  $|A_i| = m$ , determine if there exist  $a_1 \in A_1, \dots, a_k \in A_k$  such that  $a_1 \cdot \dots \cdot a_k = 0$ , where  $a_1 \cdot \dots \cdot a_k := \sum_{i=1}^z \prod_{j=1}^k a_{ji}$ .*

► **Conjecture 2.5** ( $k$ -OV Conjecture). *No (randomized) algorithm can solve  $k$ -OV in  $m^{k-\epsilon} \text{poly}(z)$  time, for any  $\epsilon > 0$ .*

► **Conjecture 2.6** (3SUM Conjecture). *Any algorithm requires  $m^{2-o(1)}$  time in expectation to determine whether a set  $S \subset \{-m^3, \dots, m^3\}$  of  $m$  integers contains three distinct elements  $a, b, c \in S$  with  $a + b = c$ .*

### 3 Conditional lower bounds

In this section, we establish conditional hardness for problems in  $\mathfrak{B}'$  under popular conjectures. We do so by considering per-operation time complexity in terms of  $s$  (the side length of the grid), and overall complexity in terms of  $n$  (the number of operations).

Backurs et al. [6] gave a reduction from MAX  $k$ -CLIQUE to  $k$ D WEIGHTED DEPTH. When  $k = 3$ , MAX  $k$ -CLIQUE is equivalent to NEGATIVE TRIANGLE, which is subcubic equivalent to APSP [26]. Adapting this reduction with a sweepline implies conditional lower bounds for 2D GRID RANGE (max, +).

► **Proposition 3.1** (Modified from [6]). *If OFFLINE 2D GRID RANGE (max, +) can be solved in amortised  $O(s^{1-\epsilon})$  time per update and  $O(s^{2-\epsilon})$  time per query, or in  $O(n^{3/2-\epsilon})$  time overall, for any  $\epsilon > 0$ , then the APSP Conjecture is false.*

We now establish more general linear per-operation lower bounds for 2D GRID RANGE problems in terms of  $s$ , based on the OMv Conjecture.

► **Lemma 3.2.** *Suppose  $(\mathbb{Z}, +, 0)$  is a monoid<sup>3</sup> (resp. group<sup>4</sup>),  $(\mathbb{Z}, \cdot)$  is a commutative semigroup such that  $0r = r0 = 0$  for all  $r \in \mathbb{Z}$  and that there exists  $x \in \mathbb{Z}$  such that  $0 \in \{xz, (x+x)z, (x+x+x)z\}$  if and only if  $z = 0$ . Then, 2D GRID RANGE  $(\cdot, +)$  cannot be solved in worst-case (resp. amortised)  $O(s^{1-\epsilon})$  time per update and  $O(s^{2-\epsilon})$  time per query, for any  $\epsilon > 0$ , unless the OMv Conjecture is false. If  $(\mathbb{Z}, +, 0)$  is a group, 2D GRID RANGE  $(\cdot, +)$  also cannot be solved in  $O(n^{3/2-\epsilon})$  time overall, for any  $\epsilon > 0$ , unless the OMv Conjecture is false.*

**Proof.** We reduce OuMv to an instance of 2D GRID RANGE  $(\cdot, +)$  with  $s = m$ . Let  $A_{(i,j)}$  denote the value of the point  $(i, j)$ . Initially, each  $A_{(i,j)} = 0$ . In preprocessing, for each  $M_{ij} = 0$ , add  $x$  to  $A_{(i,j)}$ . We now say the data structure is in its *ready state*.

Let  $(u, v)$  denote a pair of input vectors. For each  $u_i = 0$ , add  $x$  to the value of all points in row  $i$ , and for each  $v_j = 0$ , add  $x$  to the value of all points in column  $j$ . Every point now has a value in  $\{0, x, x+x, x+x+x\}$ . Now some point has value 0 if and only if the answer

<sup>3</sup>  $(\mathbb{Z}, +, 0)$  is a *monoid* if  $(\mathbb{Z}, +)$  is a semigroup, and the identity of  $+$  is 0.

<sup>4</sup>  $(\mathbb{Z}, +, 0)$  is a *group* if it is a monoid and  $+$  is invertible.



to the OuMv query is 1, so we establish this with a single range query. We then restore the data structure to its ready state, either by keeping a journal of updates (semigroup) or by updating with additive inverses (group). The reduction uses  $O(m^2)$  updates and  $O(m)$  queries, implying the stated conditional lower bounds. ◀

This gives a  $\Omega(n^{3/2-o(1)})$  time conditional lower bound on 2D GRID RANGE (max, +), matching that of Proposition 3.1. Through different reductions, we are able to obtain matching lower bounds for the other problems in  $\mathfrak{B}'$ , under the same conjecture.

► **Lemma 3.3.** *If 2D GRID RANGE (+, max), 2D GRID RANGE (+, min) or 2D GRID RANGE (max, min) can be solved in  $O(n^{3/2-\epsilon})$  time overall, for some  $\epsilon > 0$ , then the OMv Conjecture is false.*

The proof is similar to Lemma 3.2, with a different ready state, and is in the full version.

► **Lemma 3.4.** *If 2D GRID RANGE (+, set) or 2D GRID RANGE (max, set) can be solved in  $O(n^{3/2-\epsilon})$  time overall, for some  $\epsilon > 0$ , then the OMv Conjecture is false.*

**Proof sketch.** We reduce from OuMv. In our data structure, we have  $s = m^2$  and utilise an  $m \times m^2$  area of this grid, with updates and queries restricted to this area. In preprocessing, we perform updates so that all points in the  $j$ -th column have a value of  $(j - 1 \bmod m) + 1$ . Each  $M_{ij}$  is represented in  $A$  by a  $1 \times m$  section of points with values  $[1, 2, 3, \dots, m]$ . For each  $M_{ij} = 0$ , we perform an additional update to set its section of points to 0.

For the  $k$ -th query, we only consider columns that were assigned a value of  $m - k + 1$  during preprocessing. Among these, we set to 0 columns  $j$  where  $v_j = 0$ , and query rows  $i$  where  $u_i = 1$  to check for the presence of  $m - k + 1$ . ◀

Together, these give us conditional lower bounds for each of the single-update variants in  $\mathfrak{B}'$ .

► **Corollary 3.5.** *If  $q \in \{+, \max\}$  and  $u \in \{+, \text{set}, \min, \max\}$  and  $q \neq u$ , then 2D GRID RANGE ( $q, u$ ) cannot be solved in worst-case  $O(s^{1-\epsilon})$  time per update and  $O(s^{2-\epsilon})$  time per query, or in  $O(n^{3/2-\epsilon})$  time overall, for some  $\epsilon > 0$ , unless the OMv Conjecture is false. If  $u = +$ , then the lower bounds are amortised rather than worst-case, as addition is invertible.*

When measuring complexity in terms of  $s$ , it appears difficult to improve upon the naive solution which maintains a 1D instance for each column of the grid, for these problems. However, when we measure complexity in terms of  $n$ , there is a polynomial gap between the  $\Omega(n^{3/2-o(1)})$  time lower bound, and the  $\tilde{O}(n^2)$  time naive algorithm. Indeed, Chan [9] gave a  $\tilde{O}(n^{3/2})$  time solution for GRID RANGE (max, +). This might lead one to ask if there exists a general mechanism to adapt  $\tilde{O}(n)$  time algorithms in one dimension to  $\tilde{O}(n^{3/2})$  time algorithms in two dimensions, as there is for RANGE problems on a set of  $n$  explicitly provided points. Alas, when we consider variants with two simultaneous types of updates, we can obtain stronger reductions from the  $d$ -OV and 3SUM Conjectures, suggesting that it is unlikely that such a mechanism exists.

► **Lemma 3.6.** *Let  $d \geq 1$  be a constant. If OFFLINE STATIC TRELLISED  $d$ D GRID RANGE (+,  $\{+, \max\}$ ) or OFFLINE STATIC TRELLISED  $d$ D GRID RANGE (max,  $\{+, \min\}$ ) can be solved in amortised  $O(s^{(d-1)-\epsilon})$  time per update and amortised  $O(s^{d-\epsilon})$  time per query, or in  $O(n^{d-\epsilon})$  time overall, for any  $\epsilon > 0$ , then the  $d$ -OV Conjecture is false.*

**Proof sketch.** We reduce from  $d$ -OV with  $z = \log^2 n$  to an instance of  $d$ D GRID RANGE (+,  $\{+, \max\}$ ) or  $d$ D GRID RANGE (max,  $\{+, \min\}$ ) with  $s = m$  and  $n = \tilde{O}(m)$ , over the points  $[m]^d$ . Consider the first entry in each vector. Let  $v_{ji}$  be the  $i$ th vector in  $A_j$ . For each  $j \in [d]$

and each vector  $v_{ji} \in A_j$ , using the data structure, add  $(v_{ji})_1$  to all points with  $x_j = i$ . Now a point  $x = (x_1, \dots, x_d) \in [m]^d$  has value  $d$  if and only if  $(v_{jx_j})_1 = 1$  for every  $j \in [d]$ . We then undo these updates by repeating the operations with the negations of the added values, to restore every point to 0. We repeat this procedure for each of the  $z$  entries in the vectors.

Now observe that  $v_{1x_1} \cdot \dots \cdot v_{dx_d} = 0$  if and only if  $C_x$  never attained a value of  $d$  throughout this process. We check if such an  $x$  exists, by adapting a trick of Ji [18] to our data structure. This trick utilises max or min updates, in addition to the  $+$  updates described above.

We note that this reduction can be done offline, uses  $O(mdz) = \tilde{O}(m)$  updates (each of the form  $x_j = i$ ) and a single query spanning the whole grid, giving the required lower bounds. ◀

► **Lemma 3.7.** *If OFFLINE TRELLISED 2D GRID RANGE  $(+, \{+, \max\})$  can be solved with amortised  $O(s^{1-\epsilon})$  time per update and query, or in  $O(n^{2-\epsilon})$  time overall, and any  $\epsilon > 0$ , then the 3SUM Conjecture is false.*

The proof of this result is omitted for space, and proven in the full version.

A modification of Proposition 3.1, together with the results above imply strong conditional hardness for OFFLINE 2D GRID RANGE  $(+, \{+, \max\})$ , when complexity is measured per-operation.

► **Corollary 3.8.** *If OFFLINE 2D GRID RANGE  $(+, \{+, \max\})$  can be solved in amortised  $O(s^{1-\epsilon})$  time per update and query, or in  $O(n^{3/2-\epsilon})$  overall, for any  $\epsilon > 0$ , then the APSP, 2-OV and 3SUM Conjectures are all false.*

Our lower bounds show that a general approach adapting almost-linear one dimensional algorithms for GRID RANGE problems to truly subquadratic solutions for two dimensional instances is unlikely to exist. However, these results do not preclude the existence of efficient and practical algorithms for specific problems, so we would like to classify which 2D GRID RANGE problems can (and can't) be solved in truly subquadratic time. To this end, we now describe truly subquadratic algorithms to some of the as-of-yet unclassified problems in  $\mathfrak{B}'$ , and generalise these to specific subclasses of 2D GRID RANGE problems.

## 4 Solving GRID RANGE problems with a Dynamic Partition

In the full version of the paper, we describe an extension of the partition of Overmars and Yap [23]. Notably, our data structure is fully-online in that it does not require the coordinates in the input to be known during preprocessing. We give a brief overview of the construction and provide some applications, leaving further details to the full version.

### 4.1 Dynamic Structure

First, we introduce some terminology to reason about orthogonal objects in  $d$  dimensions.

A *box* is a  $d$ -dimensional (orthogonal) range. For two boxes  $R_1 = \prod_{i=1}^d [l_i^1, r_i^1]$  and  $R_2 = \prod_{i=1}^d [l_i^2, r_i^2]$ , we say that  $R_1$  is an  *$i$ -pile* with respect to  $R_2$  if  $[l_j^2, r_j^2] \subseteq [l_j^1, r_j^1]$  for all dimensions  $j \in [d] \setminus \{i\}$ . Separately, we say that  $R_1$  *partially covers*  $R_2$  if  $\emptyset \subsetneq R_1 \cap R_2 \subsetneq R_2$ . Similarly,  $R_1$  *completely covers*  $R_2$  if  $R_2 \subseteq R_1$ .

An  *$i$ -slab* ( $i \in [d]$ ) is a box of the form  $[l_1, r_1] \times \dots \times [l_i, r_i] \times \mathbb{Z}^{d-i}$ . We define  $\mathbb{Z}^d$  to be a 0-slab. A *partition tree* is a rooted tree where

1. All nodes are orthogonal ranges of  $\mathbb{Z}^d$
2. The root is  $\mathbb{Z}^d$
3. Every non-leaf node is the disjoint union of its immediate children.

A partition tree is a *level partition tree* if it has depth  $d$ , and the nodes at depth  $i$  ( $i \in [0, d]$ ) are  $i$ -slabs. Hence, a node at depth  $i$  in a level partition tree is *cut* at a series of coordinates in dimension  $i + 1$  to form its children.

► **Theorem 4.1.** *There is a data structure that maintains a set  $V$  of boxes, and supports  $n$  fully-online box insertions to  $V$ . Throughout, it can maintain a level partition tree of  $\mathbb{Z}^d$  whose leaves partition  $\mathbb{Z}^d$  into a set of  $O(n^{d/2})$  axis-aligned regions (colloquially “ $t$ -regions”, short for “trellised-regions”) such that:*

1. *Every box in  $V$  does not intersect, completely covers or is a pile with respect to each  $t$ -region;*
2. *Each box partially covers  $O(n^{(d-1)/2})$   $t$ -regions;*
3. *Each  $t$ -region is partially covered by at most  $O(\sqrt{n})$  boxes;*
4. *Any line parallel to a coordinate axis intersects at most  $O(n^{(d-1)/2})$   $t$ -regions; and*
5. *A list of the boxes that partially cover each  $t$ -region is maintained.*

*This all can be done in amortised  $\tilde{O}(n^{(d-1)/2})$  time per insertion.*

**Proof sketch.** Overmars and Yap [23] construct such a partition tree during precomputation from the boxes’ coordinates, however this is not possible in a fully-online setting. Instead, we maintain the required properties as boxes are added by periodically rebuilding subtrees which exceed a certain size and strategically inserting additional boxes to preserve balance. The cost of rebuilding amortises over the  $n$  operations, giving the stated time complexities. ◀

Using this structure, we can reduce GRID RANGE problems to TRELLISED GRID RANGE problems when the update operation distributes over the query operation.

► **Theorem 4.2.** *Suppose  $q$  and  $u$  are associative operations, both computable in  $O(1)$  time, such that  $q$  is commutative,  $u$  distributes over  $q$ , and  $0$  is an identity of  $u$ . If TRELLISED  $d$ D GRID RANGE  $(q, u)$  can be solved in  $\tilde{O}(n)$  time, then GRID RANGE  $(q, u)$  can be solved in  $\tilde{O}(n^{(d+1)/2})$  time.*

**Proof sketch.** We use the structure given by a dynamic level partition tree  $J$  from Theorem 4.1. Recall that there are  $O(n^{(d-1)/2})$   $(d - 1)$ -slabs in  $J$ , and each is a parent of  $O(\sqrt{n})$   $t$ -regions, which are leaves of  $J$ . For each  $(d - 1)$ -slab, we maintain a data structure supporting range updates and queries within the slab. Conceptually, we maintain a 1D array over its children, in order of their  $d$ -coordinate, with each array entry containing a TRELLISED instance for the corresponding  $t$ -region. We support operations whose ranges completely cover multiple children by using a modified version of the 1D GRID RANGE  $(q, u)$  structure (see Proposition 2.1). Operations affecting only part of a child are handled by the corresponding TRELLISED instance.

Operations are then performed by iterating over all  $(d - 1)$ -slabs, and updating and querying the respective data structures, as necessary. Our data structure is maintained in such a way that over its lifetime, there will be  $O(n^{d/2})$  TRELLISED instances, each facilitating  $O(\sqrt{n})$  operations, giving the required time complexity. ◀

## 4.2 Applications

We apply Theorem 4.2 to give  $\tilde{O}(n^{(d+1)/2})$  time algorithms for particular problems.

First, we show that TRELLISED variants can be solved as separate one dimensional instances, when the conditions of Theorem 4.2 are met and  $u$  is also *commutative*.

## 35:12 Algorithms and Hardness for Multidimensional Range Updates and Queries

► **Lemma 4.3.** *Suppose  $q$  and  $u$  are associative, commutative binary operations, computable in  $O(1)$  time, such that  $u$  distributes over  $q$ , and  $0$  is an identity of  $u$ . Then TRELLED GRID RANGE  $(q, u)$  is solvable in  $\tilde{O}(n)$  time.*

**Proof.** For notational convenience we provide a proof for the case where  $q$  is  $\max$  and  $u$  is  $+$ ; other operations are proven identically. By the definition of TRELLED, we can associate every update with a dimension  $i$  such that the update range is an  $i$ -pile with respect to  $\mathbb{Z}^d$ ; we call this an  $i$ -update for short. At any given point in time, let  $U_i(x)$  be the sum (with respect to  $u$ ) of all  $i$ -updates with coordinate  $x$  in dimension  $i$ .

Now consider a query over the range  $R = [l_1, r_1] \times \dots \times [l_d, r_d]$ . The answer to the query can be written as

$$\max_{(x_1, \dots, x_d) \in R} \sum_{i \in [d]} U_i(x_i) = \sum_{i \in [d]} \max_{(x_1, \dots, x_d) \in R} U_i(x_i) = \sum_{i \in [d]} \max_{x_i \in [l_i, r_i]} U_i(x_i)$$

by distributivity. Hence, we can reduce to  $d$  instances of 1D RANGE  $(q, u)$ , which each can be solved in  $\tilde{O}(n)$  time, by Proposition 2.1. ◀

This proves Theorem 1.3, giving efficient fully-online algorithms in these cases.

► **Corollary 4.4.** *GRID RANGE  $(\max, +)$  and GRID RANGE  $(\max, \min)$  can be solved in  $\tilde{O}(n^{(d+1)/2})$  time.*

There also exists some instances where  $u$  is not commutative for which an  $\tilde{O}(n)$  solution to TRELLED GRID RANGE  $(q, u)$  exists, giving us algorithms with the same time complexity.

► **Lemma 4.5.** *TRELLED GRID RANGE  $(\max, \text{set})$  can be solved in  $\tilde{O}(n)$  in  $d$  dimensions. Hence, GRID RANGE  $(\max, \text{set})$  can be solved in  $\tilde{O}(n^{(d+1)/2})$ .*

### 5 Reducing to STATIC TRELLED instances

The technique from the previous section does not work on all variants. For instance, consider the 2D GRID RANGE  $(\max, \{\min, \max\})$  problem. While the update operations ( $\min$  and  $\max$ ) are individually associative, commutative and distribute over the query operation ( $\max$ ), they do not commute with each other. Given that the order of operations matters greatly, it is difficult to decompose this into separate one dimensional problems.

In this section, we describe a general framework for reducing multidimensional range problems to STATIC TRELLED instances, using 2D GRID RANGE  $(\max, \{\min, \max\})$  as an example. For simplicity, we first give a reduction for OFFLINE instances. We describe how to generalise this approach to online settings, leaving the proof to the full version.

**General approach.** Our algorithm operates by partitioning operations into chronologically contiguous batches of at most  $k$  operations, for some function  $k$  of  $n$ . Each batch may contain both updates and queries. Let  $B$  be a particular batch, and let  $G_B$  be the state of the grid at the start of  $B$ . We will show how to answer the queries within  $B$ .

The  $\bar{k} = O(k)$  coordinates of  $B$ 's operations partition the grid into a  $\bar{k} \times \bar{k}$  overlay grid of overlay regions. Any update or query will concern all points that fall within a 2D range of whole overlay regions. Since the update operations  $\min_c$  and  $\max_c$  are monotonically increasing functions for any constant  $c$ , it suffices to know the maximum value within each overlay region according to  $G_B$ , to answer the queries of  $B$ . We use these maximums as initial values for a 2D GRID RANGE  $(\max, \{\min, \max\})$  instance over the overlay grid, which we solve by keeping  $\bar{k}$  1D RANGE instances, one for each column, and facilitating operations in  $\tilde{O}(k)$  time, by iterating over each one.

It remains to find the maximum value within each overlay region, according to  $G_B$ . To do so, consider an alternate partition of the grid into t-regions according to Theorem 4.1. In each t-region  $Y$ , we will form an instance of STATIC TRELLISED 2D GRID RANGE ( $\max, \{\min, \max\}$ ) with  $O(\sqrt{n})$  updates; we describe how to do so below. Then, to find the maximum value within an overlay region  $O$ , we issue queries to the instances corresponding to the t-regions intersecting  $O$ .

**Forming STATIC TRELLISED instances.** By construction, each t-region  $Y$  is affected by up to  $n$  whole updates which completely cover  $Y$ , and up to  $\sqrt{n}$  partial updates which cover all points in a range of rows or range of columns of  $Y$ . We can afford to include each partial update in our instance, but need to find a succinct way to represent whole updates.  $Y$  has at most  $\sqrt{n} + 1$  ranges of time between each of its partial updates, and with the aid of a lazy propagation structure over the t-region tree, we can compress the whole updates occurring during each of these ranges into a single update. Hence, we can form an instance of STATIC TRELLISED 2D GRID RANGE ( $\max, \{\min, \max\}$ ) with  $O(\sqrt{n})$  updates, as required.

We now analyse the time complexity of our approach.

► **Lemma 5.1.** *If there an algorithm for STATIC TRELLISED 2D GRID RANGE ( $\max, \{\min, \max\}$ ) running in  $\tilde{O}(n_u^{c-\gamma}(n_u + n_q)^\gamma)$  time for some  $c \geq 1$  and  $\gamma \in [0, 1]$ , then 2D GRID RANGE ( $\max, \{\min, \max\}$ ) can be solved in  $\tilde{O}(n^{5/4+c/2})$  time.*

**Proof.** We process each of the batches separately, and consider the time taken to answer the queries within a particular batch  $B$ .

Using the methods above, in  $\tilde{O}(n^{3/2})$  time we form an instance of STATIC TRELLISED 2D GRID RANGE ( $\max, \{\min, \max\}$ ) with  $n_u = O(\sqrt{n})$  updates in each of  $O(n)$  t-regions. Next, we bound the number of queries made to such instances within  $B$ . Consider the partition  $\mathcal{P}$  of the grid produced by refining each t-region by the overlay grid. A query to a STATIC TRELLISED instance is made for each region in  $\mathcal{P}$ , and the number of these regions and thus, queries, is  $O((k + \sqrt{n})^2) = O(k^2 + n)$ : this is the number of intersections found when the overlay grid is laid atop the t-regions.

For a given t-region  $Y$ , let the number of queries made to the instance in  $Y$  be  $n_{q_Y}$ . First, consider the regions  $y$  where  $n_{q_y} < \sqrt{n}$ . There are  $O(n)$  regions in total, so we spend  $\tilde{O}(n^{1+c/2})$  time answering queries for these regions.

Now consider the regions  $Y$  where  $n_{q_Y} \geq \sqrt{n}$ . Since  $n_{q_Y} = \Omega(n_u)$ , the running time of  $Y$ 's TRELLISED instance is subadditive with respect to  $n_{q_Y}$ . Thus, the total running time for these regions is maximised when the queries are distributed evenly among as many regions as possible. Subject to the constraint on these regions, this maximum is achieved, within a constant multiplicative factor, when there are at most  $O((k^2 + n)/\sqrt{n})$  regions, each with  $\sqrt{n}$  queries. Hence, in this case, the total running time is bounded by  $\tilde{O}((k^2 + n)n^{(c-1)/2})$ . We thus spend time  $\tilde{O}(n^{3/2} + n^{1+c/2} + k^2n^{(c-1)/2})$  for each of  $n/k$  batches. For balance, we choose  $k = n^{3/4}$ , giving a running time of  $\tilde{O}(n^{5/4+c/2})$  overall. ◀

It remains to show that we can solve STATIC TRELLISED GRID RANGE ( $\max, \{\min, \max\}$ ) efficiently; we do so in the full version. By observing that  $\text{set}_z = \min_z \circ \max_z$ , we obtain the following result.

► **Corollary 5.2.** *2D GRID RANGE ( $\max, \{\min, \text{set}, \max\}$ ) can be solved in  $\tilde{O}(n^{7/4})$  time.*

Most of the steps in our approach are not specific to the update and query operations in our example. To generalise this approach to fully-online and multidimensional settings, we introduce the following “partial information” variant of 1D RANGE.

► **Definition 5.3** (1D PARTITIONED RANGE  $(q, U)$ ). Let  $A_0$  be an integer array of length  $s$  and let  $0 = a_0 < \dots < a_\rho = s$  be a sequence of indices. Let  $Q$  be a corresponding sequence of  $\rho$  integers, denoting the values  $q(A_0[a_0 + 1, a_1]), \dots, q(A_0[a_{\rho-1} + 1, a_\rho])$ . Initially,  $\rho = 1$ .

Let  $J$  be a list of range updates applicable to  $A_0$  in chronological order, initially empty. We write  $A_J$  for the result of applying the updates of  $J$  to  $A_0$ , in order.

Given an integer  $s$ , and the value of  $q(A_0[1, s])$ , support the following operations:

- *split* $(i, a, q_l, q_r)$ : given  $i \in [0, \rho - 1]$  and  $a_i < a < a_{i+1}$ , add  $a$  to the sequence of indices, and update  $Q$  with the knowledge that  $q(A_0[a_i + 1, a]) = q_l$  and  $q(A_0[a + 1, a_{i+1}]) = q_r$
- *update* $_j(a_l, a_r)$ : append to  $J$ , the update: “for each  $i \in [a_l + 1, a_r]$ , set  $A[i] := u_j(A[i])$ ”
- *query* $(a_l, a_r)$ : return  $q(A_J[a_l + 1, a_r])$

It is guaranteed that every  $a_l$  or  $a_r$  provided as input will already be in the sequence.

Note that this problem is not solvable for all choices of  $q$  and  $U$ : one may need to know the individual values of  $A_0$ , and not just the result of  $q$  over some ranges, to facilitate queries after certain types of updates.

We use this to obtain the following general result, which we prove in the full version.

► **Theorem 5.4.** Suppose  $U$  is a set of update functions, and  $q$  is a query function, computable in  $O(1)$  time. If there is a set  $\bar{U}$  such that:

1.  $U \subseteq \bar{U}$  are sets of functions that can be represented and composed in  $\tilde{O}(1)$  space and time, such that the composition of any series of at most  $n$  (possibly non-distinct) functions of  $U$  results in a function in  $\bar{U}$ ;
2. There is an algorithm for 1D PARTITIONED RANGE  $(q, U)$  that performs both updates and queries in  $\tilde{O}(1)$  time;
3. There is an algorithm for STATIC TRELLISED  $d$ D GRID RANGE  $(q, \bar{U})$  that runs in  $\tilde{O}(n_u^{c-\gamma}(n_u + n_q)^\gamma)$  time for some  $\gamma \in [0, 1]$

then GRID RANGE  $(q, U)$  can be solved in  $\tilde{O}(n^{\frac{c+d+1}{2} - \frac{1}{2d}})$  time. When  $d = 2$ , this is  $\tilde{O}(n^{5/4+c/2})$  time.

This gives a  $\tilde{O}(n^{(d^2+2d-1)/2d})$  time algorithm for GRID RANGE  $(\max, \{\min, \text{set}, \max\})$ .

In the next section, we give two additional applications of this theorem.

## 6 Truly subquadratic set updates and + queries by counting paths

In this section, we apply Theorem 5.4 to give truly subquadratic algorithms for 2D GRID RANGE  $(+, \text{set})$  and 2D GRID RANGE  $(+, \{+, \text{set}\})$ .

### 6.1 2D GRID RANGE $(+, \text{set})$ by counting inversions

The first condition of Theorem 5.4 is met for  $U = \bar{U} = \{\text{set}\}$ , since  $\text{set}_a \circ \text{set}_b = \text{set}_a$  for any  $a$  and  $b$ . The second condition can be met with a data structure for 1D GRID RANGE  $(+, \text{set})$ , maintaining the invariant that a sum query over a range  $R$  in this structure yields the same result as a sum query over  $R$  in the PARTITIONED RANGE structure. Operations each occur in  $O(\log n) = \tilde{O}(1)$  time.

Finally, we address the third condition by drawing an equivalence between STATIC TRELLISED 2D GRID RANGE  $(+, \text{set})$  and a class of range query problems over arrays. The RANGE EQPAIRS QUERY accepts an array of size  $n$  as input, and asks for the number of pairs of equal elements within each of  $q$  given ranges. Duraj et al. [14] defined this weighted analogue for counting inversions between pairs of ranges.

► **Definition 6.1** (WEIGHTED 2RANGEINVERSIONSQUERY). *Given an integer array  $A$ , an integer array of weights  $W$ , both of length  $n$ , and a sequence of  $q$  pairs of non-overlapping ranges  $([l'_1, r'_1], [l''_1, r''_1]), \dots, ([l'_q, r'_q], [l''_q, r''_q])$ , with  $r'_i < l''_i$ , compute for each pair  $([l', r'], [l'', r''])$  the quantity*

$$\sum_{i \in [l', r']} \sum_{j \in [l'', r'']} 1_{A[i] > A[j]} \cdot W[i] \cdot W[j].$$

2RANGEINVERSIONSQUERY is the problem with the added restriction that every weight is 1.

They showed that RANGEQPAIRSQUERY is equivalent, up to polylogarithmic factors, to 2RANGEINVERSIONSQUERY, even when the time complexity is expressed as a function of both  $n$  and  $q$ . We extend this equivalence to STATIC TRELLISED 2D GRID RANGE (+, set).

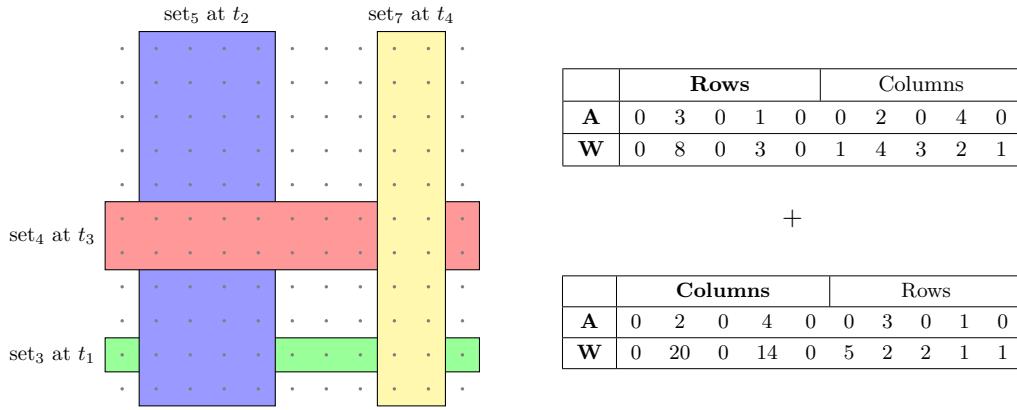
► **Lemma 6.2.** *STATIC TRELLISED 2D GRID RANGE (+, set), WEIGHTED 2RANGEINVERSIONSQUERY and 2RANGEINVERSIONSQUERY all have the same complexity, up to polylogarithmic factors. This holds even when the queries are presented online, and with the complexity measured as a function of two variables,  $n$  and  $q$ .*

**Proof.** (STATIC TRELLISED 2D GRID RANGE (+, set)  $\rightarrow$  WEIGHTED 2RANGEINVERSIONSQUERY). By adding a dummy update with value 0 covering the whole grid at time  $-1$ , we may assume – without loss of generality – that every row (column) is covered by at least one row (column) update. Now the value of a given point (after all updates) is equal to the value of the later of the latest row update and the latest column update affecting this point's row and column, respectively. Thus, for each row (column), it suffices to keep the latest row (column) update covering it. With a sort and sweep, we can process the row updates into  $O(n)$  disjoint row updates which preserve this property. Hence, without loss of generality, we may assume that the row updates are pairwise disjoint, as are the column updates. This preprocessing takes  $O(n \log n)$  time.

Consider a query over the points  $[l_1, r_1] \times [l_2, r_2]$ . We will show how to compute the contribution (sum of values) of those points whose value is determined by a row update: we can treat the contribution from column updates identically, and the sum of these contributions is the answer to the query. To form our instance of WEIGHTED 2RANGEINVERSIONSQUERY, create an array  $A$ , whose values are equal to the update time of each row update in order, from top to bottom, concatenated with the update time of each column update in order, from left to right. For the weights of our instance, let the weight corresponding to the row update with value  $v$  over rows  $[l_1, r_1]$  be  $v \times (r_1 - l_1 + 1)$ , and the weight corresponding to a column update over columns  $[l_2, r_2]$  be  $(r_2 - l_2 + 1)$ , irrespective of its value. This describes our instance (see Figure 1): we will now describe the queries made to this instance.

Since row (column) updates are disjoint, there is a contiguous range of row (column) updates in this array which lie entirely inside  $[l_1, r_1]$  ( $[l_2, r_2]$ ), whose indices can be found with binary search. The contribution of row updates to the points from these ranges is the result of a query on our instance over the corresponding ranges.  $O(1)$  parts of the query range may fall within part of a row or column update: the contribution from these can be found by scaling the result of a similarly constructed query.

For (WEIGHTED 2RANGEINVERSIONSQUERY  $\rightarrow$  2RANGEINVERSIONSQUERY) and (2RANGEINVERSIONSQUERY  $\rightarrow$  STATIC TRELLISED 2D GRID RANGE (+, set)) reductions, refer to full version. ◀



■ **Figure 1** Reducing STATIC TRELLISED 2D GRID RANGE (+, set) to WEIGHTED 2RANGEINVERSIONSQUERY. Updates occur at times  $t_1$  through  $t_4$ . Separate instances for row and column contributions.

Duraj et al. [14] gave an  $\tilde{O}(n^{(2\omega-2)/(\omega+1)}(n+q)^{2/(\omega+1)})$  time<sup>5</sup> algorithm for RANGEEQPAIRSQUERY, so we obtain the following truly subquadratic time algorithm for 2D GRID RANGE (+, set).

► **Theorem 6.3.** 2D GRID RANGE (+, set) can be solved in  $\tilde{O}(n^{5/4+\omega/(\omega+1)}) = \tilde{O}(n^{1.954})$  time.

### 6.2 2D GRID RANGE (+, {+,set}) by counting 3-paths

We will once again employ Theorem 5.4 to give a truly subquadratic time algorithm for 2D GRID RANGE (+, {+,set}). The first two conditions are met with slight modifications to that in the previous section; fulfilling the third condition is the subject of the remainder of this section. We begin by defining the following graph problems.

► **Definition 6.4.** The  $k$ -WALKQUERY (resp. SIMPLE $k$ -PATHQUERY) problem gives, as input, a simple graph with  $m$  edges and  $O(m)$  vertices, and poses  $q$  online queries, each asking for the number of  $k$ -edge walks (resp. simple  $k$ -edge paths) between a given pair of vertices.

Duraj et al. [14] proved an equivalence between RANGEEQPAIRSQUERY when  $n = q$ , and counting the number of triangles each edge is contained in, in a  $n$ -edge,  $O(n)$ -vertex graph. When the restriction  $n = q$  is relaxed, an equivalence can be drawn with 2WALKQUERY instead. In the same vein, we reduce STATIC TRELLISED 2D GRID RANGE (+, {+,set}) to 3WALKQUERY via a generalisation of RANGEEQPAIRSQUERY.

► **Lemma 6.5.** If 3WALKQUERY can be solved in  $T(m, q)$  time then STATIC TRELLISED 2D GRID RANGE (+, {+,set}) can be solved in  $\tilde{O}(T(n_u, n_q))$  time.

To solve 3WALKQUERY, we generalise the 4-cycle detection and counting algorithms of Yuster and Zwick [29] and Vassilevska Williams et al. [25] to solve 3WALKQUERY. Specifically, we partition vertices into three groups based on their degree, and consider each

<sup>5</sup> The multivariate running time given in [14] is slightly better than this when  $q \leq n$ , but this simplified form suffices our purposes.



of the possible configurations of vertices in the 3-Walk with respect to these groups. For each configuration, we use a combination of rectangular matrix multiplication and enumerating edges, both during precomputation and on-the-fly for each query. Finally, we perform a multivariate analysis of the running time, obtaining the following result.

► **Theorem 6.6.** *3WALKQUERY is equivalent to SIMPLE3PATHQUERY, and both can be solved in  $O(m^{2\omega/(2\omega+1)}(m+q)^{(2\omega-1)/(2\omega+1)})$  time.*

Hence, this yields the following algorithm, by Theorem 5.4.

► **Theorem 6.7.** *2D GRID RANGE (+, {+, set}) can be solved in time  $\tilde{O}(n^{5/4+(4\omega-1)/(4\omega+2)}) = O(n^{1.989})$ .*

## 7 RANGE problems over an explicit point set

► **Lemma 1.2.** *If 1D RANGE ( $q, U$ ) on  $p$  points can be solved in time  $T(p)$  per operation and  $q$  is computable in  $O(1)$  time, then  $d$ D RANGE ( $q, U$ ) can be solved in time  $O(p^{1-1/d}T(p))$  per operation.*

**Proof.** We construct a  $kd$ -tree [7]  $T$  over the points in  $P$ . There is a single point of  $P$  in each of the leaves of  $T$ : we assign these labels from 1 to  $p$ , according to the order of their appearance in a preorder traversal of  $T$ .

We can represent the points in any orthogonal range as those in the disjoint union of  $O(p^{1-1/d})$  subtrees of  $T$  [20]. The labels of points within these subtrees correspond to disjoint ranges of  $[p]$ . Hence, we can keep an instance of 1D RANGE ( $q, U$ ), and perform updates and queries on the corresponding ranges of labels using this data structure. ◀

► **Theorem 7.1.** *If either 2D RANGE (+, +) or 2D RANGE (max, max) can be solved in amortised  $O(p^{1/2-\epsilon})$  time per update or query, for any  $\epsilon > 0$ , then the OMv Conjecture is false.*

**Proof sketch.** We use a slight modification of Lemma 3.2, with a point in  $P$  for each  $M_{ij} = 1$ . For each pair of query vectors  $(u, v)$ , we use the update operation to mark row  $i$  for each  $u_i = 1$ . For each  $v_j = 1$ , we perform a query over column  $j$  to check if any points in that column were marked. A trick for 2D RANGE (max, max) allows us to reuse the same instance for all query vector pairs. ◀

► **Theorem 7.2.** *If any of 2D RANGE (+, set), 2D RANGE (max, set), 2D RANGE (max, +), 2D RANGE (+, max) or 2D RANGE (max, min) can be solved in amortised  $O(p^{1/2-\epsilon})$  time per update and amortised  $O(p^{1-\epsilon})$  time per query, for any  $\epsilon > 0$ , then the OMv Conjecture is false.*

The proof is similar to that of Lemma 3.2, and is omitted.

## 8 Open Problems

We have shown that 2D GRID RANGE (max, {min, set, max}) and 2D GRID RANGE (max, +) can both be solved in truly subquadratic time, and found  $\Omega(n^{2-o(1)})$  time conditional lower bounds for 2D GRID RANGE (max, {+, min}). We also observe that 2D GRID RANGE (max, {+, set}) reduces to 2D GRID RANGE (max, {+, max}), since  $\text{set}_c = \max_c \circ +_{-\infty}$ . Hence, the remaining maximum query variants in  $\mathfrak{B}$  are each at least as hard as 2D GRID RANGE (max, {+, set}).

► **Open Problem 8.1.** *Can (OFFLINE) 2D GRID RANGE (max, {+, set}) be solved in truly subquadratic time?*

Among variants supporting sum queries, we gave  $\Omega(n^{2-o(1)})$  time conditional lower bounds for 2D GRID RANGE (+, {+, max}). Using the identity  $\text{set}_c = \text{max}_c \circ +_{-\infty}$  once again, one can see that this is at least as hard as 2D GRID RANGE (+, {+, set}), which we solved in  $O(n^{1.989})$  time, using Theorem 5.4. Another problem easier than 2D GRID RANGE (+, {+, max}) is simply 2D GRID RANGE (+, max), which does not support + updates. This is also the easiest among the remaining sum query variants in  $\mathfrak{B}$ .

We make several comments regarding the hardness of 2D GRID RANGE (+, max). First, we observe that it is also at least as hard as its set “counterpart”, since any instance of GRID RANGE (+, set) can be simulated with two instances of GRID RANGE (+, max).

► **Lemma 8.2.** *GRID RANGE (+, set) can be solved in the same time as GRID RANGE (+, max).*

We solved 2D GRID RANGE (+, set) in  $O(n^{1.954})$  time using Theorem 5.4. However, it is easy to see that there is no solution to 1D PARTITIONED RANGE (+, max), which is required as a precondition of Theorem 5.4: it is simply not enough to know the sum of a range of points. One might instead determine for each overlay region  $O$ , query range  $R$  and  $\text{max}_c$  update: the number of points in  $O \cap R$  with value at most  $c$  and the sum of points in  $O \cap R$  with value greater than  $c$ . This requires  $O(k^3)$  values returned per batch, limiting precomputation to  $O(n^{3/2-\epsilon})$  time, for some  $\epsilon > 0$ , if a truly subquadratic time algorithm overall is desired. This cannot be achieved with a direct application of Theorem 5.4, since there are  $O(n^{3/2})$  updates across the t-regions.

► **Open Problem 8.3.** *Can (OFFLINE) 2D GRID RANGE (+, max) be solved in truly subquadratic time?*

Finally, our  $\Omega(n^{3/2-o(1)})$  conditional lower bounds do not match the upper bounds we gave for 2D GRID RANGE (+, {set, +}), 2D GRID RANGE (+, set) and 2D GRID RANGE (max, {min, set, max}). We ask if the gap can be closed for these problems to see if there exists an in-between complexity class of 2D GRID RANGE problems. In particular, this would be resolved in the affirmative if Theorem 5.4 is a tight reduction for any of these problems.

► **Open Problem 8.4.** *Are there any 2D GRID RANGE problems solvable in  $O(n^{2-\epsilon})$  time, for some  $\epsilon > 0$ , but require  $\Omega(n^{3/2-o(1)})$  time?*

We have studied just a small subset of RANGE and GRID RANGE problems in this work. Additional update or query operations, such as addition modulo a prime, can also be considered. Many existing variants of range searching (see [4]) can also be adapted to these problem classes. In particular, we have not investigated problems which deal with data points that have a “colour” or “category”, and ask for the number of distinct colours in a range. These may be of particular interest, as set updates could be used to facilitate changing the colour of several data points at the same time.

---

## References

- 1 Amir Abboud, Virginia Vassilevska Williams, and Huacheng Yu. Matching triangles and basing hardness on an extremely popular conjecture. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 41–50. ACM, 2015. doi:10.1145/2746539.2746594.

- 2 Peyman Afshani, Lars Arge, and Kasper Dalgaard Larsen. Orthogonal range reporting in three and higher dimensions. In *50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, October 25-27, 2009, Atlanta, Georgia, USA*, pages 149–158. IEEE Computer Society, 2009. doi:10.1109/FOCS.2009.58.
- 3 Peyman Afshani, Lars Arge, and Kasper Dalgaard Larsen. Orthogonal range reporting: query lower bounds, optimal structures in 3-d, and higher-dimensional improvements. In David G. Kirkpatrick and Joseph S. B. Mitchell, editors, *Proceedings of the 26th ACM Symposium on Computational Geometry, Snowbird, Utah, USA, June 13-16, 2010*, pages 240–246. ACM, 2010. doi:10.1145/1810959.1811001.
- 4 Pankaj K. Agarwal. Range searching. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry, Second Edition*, pages 809–837. Chapman and Hall/CRC, 2004. doi:10.1201/9781420035315.ch36.
- 5 Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. *CoRR*, abs/2010.05846, 2020. arXiv:2010.05846.
- 6 Arturs Backurs, Nishanth Dikkala, and Christos Tzamos. Tight hardness results for maximum weight rectangles. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPICs*, pages 81:1–81:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICs.ICALP.2016.81.
- 7 Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975. doi:10.1145/361002.361007.
- 8 Jon Louis Bentley. Solutions to klee's rectangle problems. *Unpublished manuscript*, pages 282–300, 1977.
- 9 Timothy M. Chan. A (slightly) faster algorithm for klee's measure problem. In Monique Teillaud, editor, *Proceedings of the 24th ACM Symposium on Computational Geometry, College Park, MD, USA, June 9-11, 2008*, pages 94–100. ACM, 2008. doi:10.1145/1377676.1377693.
- 10 Timothy M. Chan. Klee's measure problem made easy. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 410–419. IEEE Computer Society, 2013. doi:10.1109/FOCS.2013.51.
- 11 Timothy M. Chan, Kasper Green Larsen, and Mihai Pătraşcu. Orthogonal range searching on the ram, revisited. In Ferran Hurtado and Marc J. van Kreveld, editors, *Proceedings of the 27th ACM Symposium on Computational Geometry, Paris, France, June 13-15, 2011*, pages 1–10. ACM, 2011. doi:10.1145/1998196.1998198.
- 12 Timothy M. Chan, Yakov Nekrich, and Michiel H. M. Smid. Orthogonal range reporting and rectangle stabbing for fat rectangles. In Zachary Friggstad, Jörg-Rüdiger Sack, and Mohammad R. Salavatipour, editors, *Algorithms and Data Structures - 16th International Symposium, WADS 2019, Edmonton, AB, Canada, August 5-7, 2019, Proceedings*, volume 11646 of *Lecture Notes in Computer Science*, pages 283–295. Springer, 2019. doi:10.1007/978-3-030-24766-9\_21.
- 13 Bernard Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM J. Comput.*, 17(3):427–462, 1988. doi:10.1137/0217026.
- 14 Lech Duraj, Krzysztof Kleiner, Adam Polak, and Virginia Vassilevska Williams. Equivalences between triangle and range query problems. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 30–47. SIAM, 2020. doi:10.1137/1.9781611975994.3.
- 15 Jim Gray, Adam Bosworth, Andrew Layman, and Hamid Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-total. In Stanley Y. W. Su, editor, *Proceedings of the Twelfth International Conference on Data Engineering, February 26 - March 1, 1996, New Orleans, Louisiana, USA*, pages 152–159. IEEE Computer Society, 1996. doi:10.1109/ICDE.1996.492099.

- 16 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 21–30. ACM, 2015. doi:10.1145/2746539.2746609.
- 17 Nabil Ibtihaz, M. Kaykobad, and M. Sohel Rahman. Multidimensional segment trees can do range queries and updates in logarithmic time. *CoRR*, abs/1811.01226, 2018. arXiv:1811.01226.
- 18 Ruyi Ji. Interval maximum value operation and historical maximum value problem. *Informatics Olympiad China National Team Candidates Essay Collection*, 2016. Article written in Chinese. The author (Ji) has written a blog post in English, outlining the main techniques: <https://codeforces.com/blog/entry/57319>.
- 19 Tuukka Korhonen. On multidimensional range queries. Technical report, University of Helsinki, 2019.
- 20 D. T. Lee and C. K. Wong. Worst-case analysis for region and partial region searches in multidimensional binary search trees and balanced quad trees. *Acta Informatica*, 9:23–29, 1977. doi:10.1007/BF00263763.
- 21 George S. Lueker. A data structure for orthogonal range queries. In *19th Annual Symposium on Foundations of Computer Science, Ann Arbor, Michigan, USA, 16-18 October 1978*, pages 28–34. IEEE Computer Society, 1978. doi:10.1109/SFCS.1978.1.
- 22 Yuzuru Okajima and Kouichi Maruyama. Faster linear-space orthogonal range searching in arbitrary dimensions. In Ulrik Brandes and David Eppstein, editors, *Proceedings of the Seventeenth Workshop on Algorithm Engineering and Experiments, ALENEX 2015, San Diego, CA, USA, January 5, 2015*, pages 82–93. SIAM, 2015. doi:10.1137/1.9781611973754.8.
- 23 Mark H. Overmars and Chee-Keng Yap. New upper bounds in klee’s measure problem. *SIAM J. Comput.*, 20(6):1034–1045, 1991. doi:10.1137/0220065.
- 24 Chung Keung Poon. Dynamic orthogonal range queries in OLAP. *Theor. Comput. Sci.*, 296(3):487–510, 2003. doi:10.1016/S0304-3975(02)00741-7.
- 25 Virginia Vassilevska Williams, Joshua R. Wang, Richard Ryan Williams, and Huacheng Yu. Finding four-node subgraphs in triangle time. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1671–1680. SIAM, 2015. doi:10.1137/1.9781611973730.111.
- 26 Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix and triangle problems. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 645–654. IEEE Computer Society, 2010. doi:10.1109/FOCS.2010.67.
- 27 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005. doi:10.1016/j.tcs.2005.09.023.
- 28 Virginia Vassilevska Williams. Hardness of easy problems: Basing hardness on popular conjectures such as the strong exponential time hypothesis (invited talk). In Thore Husfeldt and Iyad A. Kanj, editors, *10th International Symposium on Parameterized and Exact Computation, IPEC 2015, September 16-18, 2015, Patras, Greece*, volume 43 of *LIPICs*, pages 17–29. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPICs.IPEC.2015.17.
- 29 Raphael Yuster and Uri Zwick. Detecting short directed cycles using rectangular matrix multiplication and dynamic programming. In J. Ian Munro, editor, *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004, New Orleans, Louisiana, USA, January 11-14, 2004*, pages 254–260. SIAM, 2004. URL: <http://dl.acm.org/citation.cfm?id=982792.982828>.