Efficient Mining and Exploration of Multiple Axis-aligned Intersecting Objects

Tilemachos Pechlivanoglou, Vincent Chu and Manos Papagelis
Lassonde School of Engineering, York University, Toronto, Canada
{tipech, vwchu, papaggel}@eecs.yorku.ca

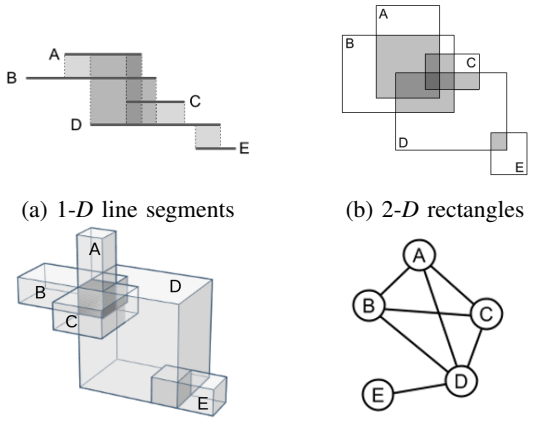
***Abstract*—Identifying and quantifying the size of multiple intersections among a large number of axis-aligned geometric objects is an essential computational geometry problem. The ability to solve this problem can effectively inform a number of spatial data mining methods and can provide support in decision making for a variety of applications. Currently, the state-of-the-art approach for addressing such intersection problems resorts to an algorithmic paradigm, collectively known as the *sweep-line* algorithm. However, its application on specific instances of the problem inherits a number of limitations. With that mind, we design and implement a novel, exact, fast and scalable yet versatile, sweep-line based algorithm, named SLIG. Our algorithm can be employed in a number of problems and applications involving the efficient computation of numerous axis-aligned object intersection problems in multiple dimensions. The key idea of our algorithm lies in constructing an auxiliary data structure when the sweep line algorithm is applied, an *intersection graph*. This graph can effectively be used to provide connectivity properties among overlapping objects, as well as to inform the much harder problem of finding the location and size of the common area defined by multiple overlapping objects. A thorough experimental evaluation on synthetic data of various characteristics and sizes, demonstrates that SLIG performs significantly faster than classic sweep-line based algorithms. SLIG is not only faster and more versatile, but also provides a suite of powerful querying capabilities. To support the reproducibility of our methods, we make source code and datasets available.**

***Index Terms*—geometric intersection problems, multiple intersections, intersection graph, sweep-line**

I. Introduction

Identifying intersections of a large number of axis-aligned geometric objects in multiple dimensions is an essential computational geometry problem [1]. The axis-aligned requirement describes objects whose shapes are aligned with the coordinate axes of the space; that includes *line segments* in 1-*D*, *rectangles* or *boxes* in 2-*D*, *cuboids* in 3-*D*, and so on (see Fig. 1a, 1b, 1c). A related, but more challenging problem, is that of identifying *multiple* intersections of such objects. The ability to solve this problem can effectively inform various data mining and querying methods, as well as various critical applications, such as spatial databases [2], VLSI physical design [3], spaciotemporal data analysis [4], computer graphics and simulation collision detection [5].

Currently, the state-of-the-art approach for addressing such intersection problems is an algorithmic paradigm that employs a conceptual sweep line to solve various problems in Euclidean space, collectively known as the *sweep-line* or *plane sweep* algorithm [6]. While this paradigm has been successfully used in a number of problems, its application on specific instances of the intersection problem can be problematic due to its lack of versatility. Classic sweep-line implementations are typically designed with definite specifications and cannot easily be adapted to different problems, such as multiple intersections or with arbitrary dimensions. With these limitations in mind, we design and propose Sweep Line Intersection Graph(SLIG), a sweep-line based algorithm that can be employed in a number of problems and applications involving the efficient computation of multiple object intersections in many dimensions. The key idea of our algorithm lies in constructing an auxiliary data structure along with the sweep line algorithm, an *intersection graph*. This graph can effectively be used to efficiently identify sets of multiple intersecting objects by utilizing the thoroughly explored graph theory problem of clique enumeration.



(a) 1-*D* line segments (b) 2-*D* rectangles

(c) 3-*D* cuboids (d) Intersection graph

Fig. 1: Example of axis-aligned regions in 1-*D*, 2-*D* and 3-*D*, and the corresponding intersection graph. *A*, *B*, *C* and *D* are all intersecting with each other, forming a *common region*.

In summary, the major contributions of this work are:

- We present MultiIntersect, a novel challenging problem related to geometric object intersections in multiple dimensions. This problem can appear in diverse applications and domains, but cannot be easily addressed using traditional implementations of state-of-the-art algorithms.
- We design and implement SLIG, a novel sweep-line based algorithm that can efficiently address the introduced problem. The algorithm is utilizing information coming from an auxiliary data structure, an *intersection graph*, and is *fast*, *exact*, *versatile* and *scalable*.

- We present a thorough experimental evaluation of SLIG against state-of-the-art algorithms that demonstrate that our algorithm is superior for a range of conditions.
- We make *source code* and *data* publicly available to encourage reproducibility of results.

The remainder of this paper is organized as follows: Section II introduces notation and formally defines the problems of interest in this paper. Our proposed methods, sensible baselines and the overall computational framework are presented in Section III. In Section IV we present implementation details of our algorithms. Section V presents an experimental evaluation of the methods and algorithms. After reviewing the related work in Section VI, we conclude in Section VII.

II. Problem of interest

In this section, we introduce notation and formally present the problem of interest in this paper.

A. Preliminaries

It is important to note that the methods we present generalize to multiple dimensions. Therefore, for the rest of this document, we use the term *regions* to refer to geometric objects in $\mathbb{R}^d$, where $d \geq 1$ (i.e., *line segments* in 1-*D*, *rectangles* or *boxes* in 2-*D*, *cuboids* in 3-*D*, etc.). Even though our implementations are general, we only report values for up to $d = 3$ in the experiments, as that should be enough to demonstrate the behavior of the methods in most scenarios.

Related to the problem we aim to address is the way that geometric objects are overlapping with each other. Consider for example the regions A and B in Fig. 1b. As they are overlapping with each other, they form a *common region* Z_{AB} that consists of all points in space that belong to both regions A and B. In order to generalize the concept of *common region* to more than two regions, we need to consider all the different ways that overlaps can occur. For example, in Fig. 1b, regions A, D and E have some pairwise overlaps (Z_{AD}, Z_{DE}), but they do not all overlap with each other forming a single common region (Z_{ADE}). On the other hand, regions A, B, C, D are all overlapping with each other forming Z_{ABCD}, each point of which belongs to (is covered by) all four regions. We introduce the concept of *intersection cardinality* k to refer to the type of overlapping regions that we are interested in detecting and reporting. Formally, given a set of regions $S = \{s_1, s_2, \ldots, s_n\}$ that individually may or may not overlap with each other, we can construct sets of regions $I_i \subseteq S$ where all regions in each I fully intersect with each other, forming a common region Z, while each subset I has cardinality k equal to its size $|I|$. For example, when $k = 2$ there is a pair of regions that intersect to form a common region, when $k = 3$ there is a trio of regions, when $k = 4$ there is a quadruplet, and so on. A worst-case scenario exists where $k = n$, where $n = |S|$ is the total number of objects, meaning that all regions are overlapping with each other; this, however, is a degenerate case for most real-world scenarios or applications.

TABLE I: Summary of Notations

Notation	Description
d	Number of dimensions
s_i	An axis-aligned region in $\mathbb{R}^d$
S	A set of regions $\{s_1, s_2, \ldots, s_n\}$
I	A subset of S, whose regions all intersect with each other
Z	The common overlap region of all regions in I
k	Intersection cardinality of I
l	Number of fully intersecting sets I_i in S
l_{max}	Number of maximal cliques in RIG

TABLE II: Computational Complexity of MultiIntersect

Naive	$O(d2^m)$
SweepLine	$O(d \cdot n \log n + d(l \log l)^k)$
SLIG	$O(d \cdot n \log n + n \cdot l_{max})$

B. Problem

We are now in position to formally define the problem of interest in this paper.

MultiIntersect: Given a set $S = \{s_1, s_2 \ldots, s_n\}$ of n regions in $\mathbb{R}^d, d \geq 1$, enumerate all the sets of intersecting regions $I_i \subseteq S$, and their respective common regions Z_i.

For example, given the set of regions $S = \{A, B, C, D, E\}$ of Fig. 1b, we seek to find the intersecting sets *AB, AC, AD, BC, BD, CD, DE, ABC, ABD, ACD, BCD, ABCD*.

Note that this problem is difficult to compute using the classic state-of-the-art algorithms. Although it's possible to use modified versions of the sweep-line algorithm to accommodate a specific instance of the problem, it won't be versatile enough to accommodate other versions of the problem, or an arbitrary number of dimensions. Furthermore, any possible solutions to the presented problem are inherently expensive even for 2- and 3-D cases [7], let alone for more dimensions.

III. Methodology

In this section, we describe different methods for addressing the MultiIntersect problem defined in the previous section. We present a naive implementation to address the problem and its limitations, and we follow by describing in detail of how one would utilize the state-of-the-art sweep-line paradigm instead; this is used as a sensible baseline to compare against our more sophisticated algorithm. Finally, we outline our proposed algorithm, SLIG, a sweep-line based algorithm that constructs and utilizes an intersection graph to greatly simplify and speed up the solution to the problem. As all methods described below compute *exact* results (not approximated in any way), discussion about the accuracy of the methods isn't necessary. Table II provides a summary of the computational complexities for the different methods.

A. *The* Naive *Method*

Finding whether a pair of regions in $\mathbb{R}^d$ intersect or not is straightforward, as it's only necessary to compare the corner points of both regions in all dimensions. There are, however, different methods to select which of the possible regions should be compared with each other, resulting in different numbers of unnecessary comparisons. The simplest approach is comparing every region in S with every other, finding

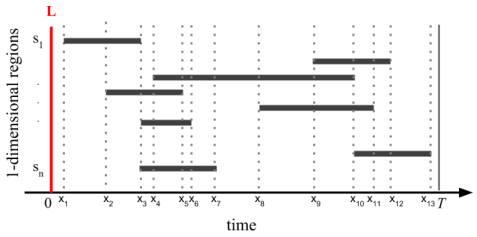


Fig. 2: Illustrative example of a sweep-line algorithm that can determine intersections of 1-D regions (line segments).

intersecting pairs, and then proceed to compare the common region defined by every intersecting pair with every other region to find triple intersections, and so on. This iterative process has to be continued until no more intersections are found or the maximum cardinality $k = n$ has been reached. As is apparent, the computational cost of such a method increases exponentially with the number of regions in S. For a specific k, the k combinations of the n regions in S are given by $\binom{n}{k}$ and enumerating all k-combinations (i.e., $k = \{2, 3, ..n\}$) would be $\sum_{2 \leq k \leq n} \binom{n}{k} = 2^n$. Moreover, as the same process has to be followed for every dimension, the computational cost of NAIVE becomes $O(d2^n)$. Unless the dataset is extremely small or there are very few intersections occurring, the cost of that computation would be prohibitively large.

B. *The* SWEEPLINE *Method*

While the NAIVE method has several shortcomings, better alternatives for addressing intersection problems exist in the literature. The sweep-line, or Bentley-Ottmann, algorithm [8] is an algorithmic paradigm that uses a conceptual sweep line to identify and report pairs of intersecting objects in Euclidean space. Given a set of d-dimensional regions (e.g., line segments, rectangles, cuboids, etc.), the first step of the algorithm is constructing a list that includes the *start* and *stop* points of all regions in each dimension and sorting them. Then, a *conceptual line*, L moves (sweeps) from left to right across the plane, examining the regions, one by one, in order. During the sweep, the *active regions* (i.e., the ones that line L is currently traversing over) are maintained. When a new region is encountered by L, it is marked as *intersecting* all the currently active regions (in the current dimension). The process consists of a single pass in each dimension. Regions that are marked as *intersecting* in all dimensions are actually *intersecting regions*. An illustrative example of the process for 1-D regions (line segments) can be seen in Fig. 2.

In order to address the problem of interest, we had to modify the Bentley-Ottmann algorithm to produce a variant method that we call SWEEPLINE. Specifically, instead of detecting only intersecting pairs, SWEEPLINE maintains and returns all sets of intersecting regions with intersection cardinality of up to k, in a single pass. The computational cost of the original algorithm for a single dimension is $O((n + l) \log n)$, where l is the number of intersections found (in non-degenerate cases, $l \ll 2^n$). This includes the cost of sorting which is $O(n \log n)$(for one dimension) and comparisons which is $O(l \log l)$. After modifying the algorithm to support multiple intersections, the sorting cost is the same while the comparison cost becomes $l \log l$ for a specific k, and therefore to report all possible k-combinations (i.e., $k = \{2, 3, ..l\}$) it would be $\sum_{2 \leq k \leq l} l \log l = l \log l^k$. Therefore, for all dimensions, the computational cost of our modified SWEEPLINE would be in the order of $O(dn \log n + d(l \log l)^k)$. Note that since $l \ll 2^n$, this represents a significant improvement over the NAIVE methods, but it still remains significantly expensive.

C. *Sweep Line Intersection Graph (*SLIG*)*

Although SWEEPLINE employs a state-of-the-art algorithm and produces much better results than NAIVE, it still inherits a number of limitations. For instance, intermediate results (e.g., the computation of pair-wise intersecting regions), are not well utilised; they are taken into account in subsequent computations that could speed up the whole process. Towards that end, we propose SLIG, a novel method for efficiently solving the problems of interest. The method we devise is based on the following two **key observations**:

- Current best approaches to the problem depend on an expensive process of sequentially examining regions to determine if they intersect with previously visited regions or sets of regions. However, this limitation can be easily overcome by constructing a *region intersection graph*.
- The multiple intersection problem can be mapped to the *clique problem* (and variants of it) on the region intersection graph. This can suggest huge time performance savings, since it is possible to address the problem of interest just by operating on the region intersection graph, without the need to operate on the original regions or to re-apply a sweep-line algorithm multiple times.

We elaborate on these key observations in the next paragraphs.

Region Intersection Graph (RIG) is a graph where each *vertex* represents a region in S and each *edge* an intersection between two regions. In the case of 1-dimensional regions, this is known as an *interval graph*. With a region intersection graph in place, it is easy to interpret intersection queries as connectivity queries in RIG. For example, if two vertices are connected in RIG, then the corresponding regions are intersecting; or, obtaining all *neighbouring vertices* of a vertex in RIG is equivalent to finding all regions intersecting with a specific region, and so on. These graph operations are typically fast, while the construction of the *RIG* is straightforward, employing the classic sweep-line algorithm for detecting pair-wise intersections. Whenever a new region is encountered, a *new node* added to the RIG; for a pair-wise intersection, a *new edge* is added. The time complexity of this process is $O(n \log n)$ and the space complexity is $O(n)$ (due to the classic sweep-line). The space complexity of storing RIG in memory is $O(V + E)$, where $V = n$ is the set of vertices (regions) and E is the set of edges (intersections).

The clique problem refers to the computational problem of finding cliques in a graph. A *clique* is a subset of vertices of an undirected graph such that every two distinct vertices in the clique are adjacent; that is, its induced subgraph is complete. To make a connection to our problem, it is well known that

given a set of 1-dimensional line segments, where they are all pair-wise intersecting, they would have a common point or range, a *common region*, where they all overlap. It can be easily shown that the same holds true for axis-aligned objects in more dimensions, by examining each dimension separately. Effectively, these intersecting regions directly correspond to a clique in the intersection graph (see also result in [9, Lemma 3.3]). This means that, once a clique has been identified in the intersection graph, it is possible to quickly identify and report information about the common region of the multiple intersecting regions. Thus, we can provide a solution to this problem by implementing an out-of-the-box state-of-the-art algorithm that enumerates all cliques. The Bron-Kerbosch algorithm [10] has a worst-case computation cost of $O(3^{n/3})$, which is better than the NAIVE method and better than the *Sweep-line* method for very large values of l. However, we can take advantage of significant prior research on the concept of *maximal cliques*. A *maximal clique* is a clique that cannot be extended by including an additional adjacent vertex (i.e., it isn't part of a larger clique). A large number of highly optimized algorithms exist that can enumerate all cliques, while their computation cost is only bounded by the much smaller number of maximal cliques only. Specifically, the algorithm by Tsukiyama et al [11] has a cost of $O(n \cdot l_{max})$, where l_{max} is the number of maximal cliques. Notice that the dimensions cost multiplier d is only included in the sorting; this is because that cost would only incur during the construction of the intersection graph, but never again, suggesting large computation time savings. Therefore, the computational cost of SLIG would be $O(dn \log n + n \cdot l_{max})$.

IV. ALGORITHMS

In order to evaluate the performance of the different methods, we implemented the NAIVE algorithm, a modified version of the SWEEPLINE algorithm that is able to accommodate the problem of interest, and our proposed SLIG algorithm. The NAIVE algorithm simply consists of a recursive loop where each object is compared with each other one and, if the two are intersecting, the intersection is compared with every other, etc. For the SWEEPLINE and SLIG, the implementation follows the methods outlined in Section III, and the details are provided in Alg. 1 and Alg. 2, respectively. The `intersects()` function is a simple geometric comparison in all dimensions.

Reproducibility: The source code for these algorithms is publicly available at: https://github.com/tipech/overlap-graph.

V. EXPERIMENTS

In this Section, we experimentally evaluate the performance of our proposed method SLIG against SWEEP-LINE's sensible baseline method. As the NAIVE method is not sophisticated enough and shows considerably inadequate performance we don't consider it in the experimental evaluation. We also examine the effect of some critical dataset parameters on the intersection graph's density and distribution. Before presenting the results, we provide details of the computational environment and the data sets employed.

Algorithm 1: SWEEPLINE

Input: Set S of regions
Output: Set O of intersecting sets of regions, grouped by intersection cardinality k in the form $O = \{k_2 : [\{s_1, s_2\}, \ldots], k_3 : [\{s_1, s_2, s_3\}, \ldots], \ldots\}$

```
Points ← sort(x^0, x^1 ∀ s_i, d ← 1), O ← [ ], k ← 2
LastIntersects ← [ [region] ∀ region in S ]

while O[k-1] not empty do
    Intersects ← GetKIntersects(k, LastIntersects)
    O[k] ← Intersects
    LastIntersects ← Intersects
    k ← k + 1

Function GetKIntersects(k, LastIntersects)
    Actives ← [], Intersects ← {}
    for point in Points do
        if point.type = start then
            Intersects[point.region] ← []
            for activeIntersect in Actives do
                if activeIntersect.intersects(point.region) then
                    intersection ← activeIntersect
                    intersection.append(point.region)
                    Intersects[point.region].append(intersection)
            for intersection in LastIntersects[point.regions] do
                Actives.append(intersection)
        else
            for intersection in LastIntersects[point.regions] do
                Actives.remove(intersection)
```

*1) **Environment**:* All experiments are conducted on a PC with 8x Intel(R) Core™ i7-7700 CPU @ 3.60GHz and 64GB memory using Python 3.7. For each algorithm or parameter effect evaluation, we execute the algorithm ten (10) independent times and report the average execution time or other results.

*2) **Data**:* In order to evaluate the behavior of the algorithms under certain conditions, we had to resort to synthetic data. A data generator was implemented that produces data sets with specific characteristics thanks to a controlled number of parameters. We define a d-dimensional space where each dimension has size T, effectively ranging in $[0, T]$; unless otherwise noted $T = 1000$. Within that space, we uniformly generate n regions at random. The size of each dimension (side) of a region is randomly selected from the uniform range $t : [0, t_{max}]$, where $t_{max} = r \cdot T$ and $r \in [0, 1]$ represents a ratio of the total length T. For example, if $r = 0.01$ and $T = 1000$, then the size for each dimension of a region would be bound by $0 \leq t \leq 10$. Therefore, the configurable parameters of the data generator are *number of dimensions* d, *number of objects* n and *ratio* r. For experimental evaluation purposes, various datasets were created, ranging from 10^1 to 10^5 regions and resulting in 10^2 to 10^8 intersections.

*3) **Experiments**:* We aim to evaluate the following aspects:

- **Effect of Parameters n and r on RIG** How does the number of regions n and the size of regions (as defined by the ratio r) affect the properties of the Region Intersection Graph (RIG)? What is the size (number of edges) of the RIG obtained, in different dimensions?

Algorithm 2: SLIG

Input: Set S of regions
Output: Set O of intersecting sets of regions, grouped by intersection cardinality k in the form $O = \{k_2 : [\{S_1, S_2\}, \ldots], k_3 : [\{S_1, S_2, S_3\}, \ldots], \ldots\}$

```
Points ← sort(x^0, x^1 ∀ s_i, d ← 1), O ← [ ]
GetIntersectionGraph(S)
GenerateKCliques(CliqueList,Graph)

for clique in CliqueList do
    k ← len(clique)
    for i in [2,k-1] do
        O[i].append(all_combinations(clique, i))

Function GetIntersectionGraph(S)
    Actives ← [], Graph ← []
    for point in Points do
        if point.type = start then
            for activeRegion in Actives do
                if activeRegion.intersects(point.region) then
                    Graph.addEdge(activeRegion, point.region)
            activeRegion.append(point.region)
        else
            activeRegion.remove(point.region)
```

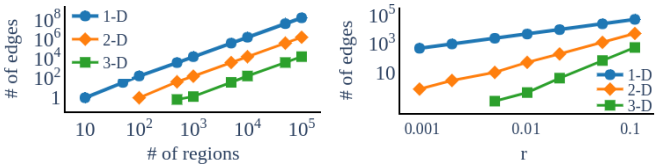
- **SLIG Comparative Performance** How does our proposed SLIG method compare to the SWEEP-LINE method for the MULTIINTERSECT problem?
- **Effect of RIG Topology on SLIG** How does the structure of the RIG influence the performance of SLIG?

A. Effect of Parameters n and r on RIG

In principle, the number of intersections in the data set and therefore edges in the RIG depend on two parameters: the number of regions n in the data set and the size of these regions, as determined by the ratio r. In this experiment, we aim to examine the effect of these parameters to the RIG. For the first experiment, we set the ratio to be $r = 0.01$ and vary the number of regions n (Fig. 3a). For the second experiment, we fix $n = 1000$ and vary the values of the ratio r (Fig. 3b). In all instances, since the generated regions are generated uniformly at random with a region size $t : 0 \leq t \leq t_{max}$ in each dimension, the average region size in each dimension corresponds to $\frac{t_{max}}{2}$. As can be seen in these figures, the size of the RIG increases linearly with both n and r ($O(n \cdot r)$). That is to be expected, as in both cases the probability of any two regions intersecting is increasing with the relative size of the region in the d-dimensional space. Similar trends are demonstrated for all dimensions reported $d = \{1, 2, 3\}$. However, as the available volume increases with more dimensions, intersections are rarer, and thus the RIG becomes sparser.

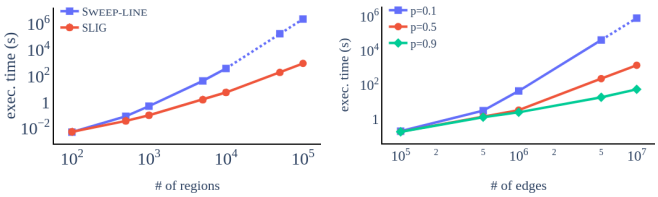
B. SLIG Comparative Performance

We evaluated the time performance of SLIG against the SWEEP-LINE method, as a function of the number of regions n in the dataset. Based on the parameter sensitivity experiments presented earlier, we set the parameter values as follows: $d = 2$, $r = 0.01$. Fig. 4a displays the results. It is apparent from



(a) #edges × #regions (b) #edges × region size

Fig. 3: Effect of dataset parameters on RIG.



(a) SLIG vs. SWEEP-LINE. (b) RIG Topology Effect

Fig. 4: Comparative performance of SLIG

the results that our SLIG algorithm is **multiple orders of time faster** than SWEEP-LINE. Note that the SWEEP-LINE algorithm would require an estimated *1 day* to process the $5{\cdot}10^4$ regions and *11 days* for processing 10^5 regions, therefore only estimated times are reported (dashed lines).

C. Effect of RIG Topology on SLIG

Besides parameters n and r, the distribution of regions in space can have an important effect on the characteristics and topology of the RIG, and therefore the performance of SLIG. In order to evaluate the impact of RIG's topology, we designed experiments with networks generated using the Watts—Strogatz small-world model [12]. This model employs a re-wiring probability $p \in [0, 1]$ that can control the overall clustering of the network, for a specific number of nodes and edges. Fig. 4b reports the time performance of SLIG in networks with the same number of nodes $n = 10^5$, but different numbers of edges $[10^5, 2x10^5, 5x10^5, 10^6, 2x10^6, 5x10^6, 10^7]$), and for varying values of $p = \{0.1, 0.5, 0.9\}$. It can be seen that as p decreases (i.e., higher clustering), the algorithm requires significantly more processing time, since more, larger cliques form.

VI. RELATED WORK

Here we present a more comprehensive coverage of topics related to our research, specifically the computational geometry problems of *object intersection*, *spatial data structures*, as well as the problem of *clique enumeration* from graph theory. A great number of data structures and algorithms have been developed that deal with identifying intersecting objects [1], [13], and specifically axis-aligned rectangles in $\mathbb{R}^d$ [14]. The objective is often to identify and report pairs of intersecting objects with speed and accuracy, usually for collision detection in simulations [5] or object placement problems [15]. In our research, we deal with the task of identifying multiple object intersections. Some methods exist for problems conceptually similar but fundamentally different from ours, such as a proposed technique to pre-process data

in order to quickly find intersecting pairs that overlap a query rectangle, i.e. triple intersections ($k = 3$) [16]. The existing state-of-the-art methods used for this problem belong to one of two families of algorithms: either a *sweep-line* (also known as *plane sweep*) or a *divide-and-conquer* algorithm, which have been shown to be computationally equivalent [17]. These are commonly used to identify pair-wise intersections while constructing data structures that can accommodate spatial access queries [18]. While these algorithms have been extensively studied, improved and optimized, [19], they can be problematic when applied to identify more than just pairs of intersections, or arbitrary numbers of dimensions. Applications involving spatial access queries commonly utilize hierarchical data structures for indexing information and significantly reducing computation costs. Tree-based data structures, such as interval trees in 1-D problems, and *bounding volume hierarchy trees*, or *R-trees* and their variants [20] for more than one dimensions, are very effective for answering access queries, which effectively require the traversal of the tree from the root to the leaves. However, they are inadequate to answer queries related to the intersections between groups of objects, since they require traversing along the leaf nodes of the tree. A data structure more capable of answering these types of queries is an intersection graph (a.k.a. an *interval graph* for 1-D regions) [21]. In intersection graphs, each *vertex* represents a single rectangle or interval and each *edge* represents an intersection between two of these objects. As this data structure maintains information on the intersections between individual objects, it is possible to use it in order to answer queries involving groups of objects [22].

Our work also relates to the problem of clique enumeration [23], a widely studied, NP-hard combinatorial problem with significance in real applications [10]. In relation to our problem, we explained how existing algorithms can be directly employed to report sets of intersecting regions. As such, clique enumeration algorithms are orthogonal to our methods and improvements can be adapted as needed.

VII. Conclusions

We have introduced MultiIntersect, a novel and computationally challenging problem that arises in the context of *identifying* multiple intersections of a large number of axis-aligned multi-dimension geometric objects (*regions*). To address this problem we designed and implemented an efficient algorithm, named SLIG. SLIG is based on the sweep-line method and operates with the help of an auxiliary graph-based data structure, a *region intersection graph* (RIG). The RIG provides fast access to information regarding whether regions intersect or not, which is otherwise difficult to obtain directly from the data. As a result, our proposed method SLIG, is able to address the problem of interest with grater performance and versatility than sensible state-of-the-art approaches. Extensive experiments were performed to demonstrate the effectiveness of our algorithm in a wide range of conditions, demonstrating it is scaleable to very large amounts of regions. We are confident that the novel problem and method presented will prove useful and find interesting application in a number of real-world applications and solutions. To encourage reproducibility we provide details of our data generator and methods' pseudo-code, while we make the source code publicly available.

Acknowledgments: This work was funded by the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

[1] Bentley and Wood, "An optimal worst case algorithm for reporting intersections of rectangles," *IEEE Transactions on Computers*, vol. C-29, no. 7, pp. 571–577, July 1980.

[2] J. M. Patel and D. J. DeWitt, "Partition based spatial-merge join," *SIGMOD Rec.*, vol. 25, no. 2, pp. 259–270, Jun. 1996.

[3] J. Fang, J. Wong, K. Zhang, and P. Tang, "A new fast constraint graph generation algorithm for vlsi layout compaction," *1991., IEEE International Sympoisum on Circuits and Systems*, 1991.

[4] T. Pechlivanoglou and M. Papagelis, "Fast and accurate mining of node importance in trajectory networks," *2018 IEEE International Conference on Big Data (Big Data)*, 2018.

[5] T. Tang, E. L. Bohez, and P. Koomsap, "The sweep plane algorithm for global collision detection with workpiece geometry update for five-axis nc machining," *Computer-Aided Design*, vol. 39, no. 11, p. 1012, 2007.

[6] M. I. Shamos and D. Hoey, "Geometric intersection problems," in *17th symposium on foundations of computer science*. IEEE, 1976, p. 208.

[7] M. de Berg, J. Gudmundsson, and A. D. Mehrabi, "Finding pairwise intersections inside a query range," *Algorithmica*, vol. 80, no. 11, pp. 3253–3269, Nov 2018.

[8] Bentley and Ottmann, "Algorithms for reporting and counting geometric intersections," *IEEE Transactions on Computers*, vol. C-28, no. 9, pp. 643–647, Sep. 1979.

[9] R. Bar-Yehuda, M. M. Halldrsson, J. S. Naor, H. Shachnai, and I. Shapira, "Scheduling split intervals," *SIAM Journal on Computing*, vol. 36, no. 1, pp. 1–15, 2006.

[10] C. Bron and J. Kerbosch, "Algorithm 457: finding all cliques of an undirected graph," *Communications of the ACM*, vol. 16, no. 9, pp. 575–577, 1973.

[11] S. Tsukiyama, M. Ide, H. Ariyoshi, and I. Shirakawa, "A new algorithm for generating all the maximal independent sets," *SIAM Journal on Computing*, vol. 6, no. 3, pp. 505–517, 1977.

[12] D. J. Watts and S. H. Strogatz, "Collective dynamics of small-world networks," *Nature*, vol. 393, pp. 440–442, 2011.

[13] F. Dévai and L. Neumann, "A rectangle-intersection algorithm with limited resource requirements," in *10th IEEE International Conference on Computer and Information Technology*, June 2010, pp. 2335–2340.

[14] T. M. Chan, "A note on maximum independent sets in rectangle intersection graphs," *Information Processing Letters*, vol. 89, no. 1, pp. 19–23, 2004.

[15] P. K. Agarwal, M. V. Kreveld, and S. Suri, "Label placement by maximum independent set in rectangles," *Computational Geometry*, vol. 11, no. 3-4, pp. 209–218, 1998.

[16] E. Oh and H. Ahn, "Finding pairwise intersections of rectangles in a query rectangle," *CoRR*, vol. abs/1801.07362, 2018.

[17] R. H. Güting and W. Schilling, "A practical divide-and-conquer algorithm for the rectangle intersection problem," *Information Sciences*, vol. 42, no. 2, pp. 95–112, 1987.

[18] F. Zhang, X.-Z. Qiao, and Z.-Y. Liu, "A parallel smith-waterman algorithm based on divide and conquer," in *International Conference on Algorithms and Architectures for Parallel Processing*, 2002, p. 162.

[19] M. McKenney and T. McGuire, "A parallel plane sweep algorithm for multi-core systems," in *17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 2009, p. 392.

[20] G. Bergen, "Efficient collision detection of complex deformable models using aabb trees," *Journal of Graphics Tools*, vol. 2, no. 4, p. 1, 1997.

[21] C. S. Rim and K. Nakajima, "On rectangle intersection and overlap graphs," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 42, no. 9, pp. 549–553, Sept 1995.

[22] H. Imai and T. Asano, "Finding the connected components and a maximum clique of an intersection graph of rectangles in the plane," *Journal of Algorithms*, vol. 4, no. 4, pp. 310–323, 1983.

[23] E. Balas and C. S. Yu, "Finding a maximum clique in an arbitrary graph," *SIAM Journal on Computing*, vol. 15, no. 4, pp. 1054–1068, 1986.