Comparing Data Structures for k-Nearest Neighbours in Crowd Simulation

Jordi Vermeulen (3835634)

January 18, 2016

Abstract

The k-nearest neighbour (kNN) problem appears in many different fields of computer science, such as machine learning, databases, computer vision and computational geometry. In crowd simulation, kNN queries are typically used by the collision avoidance method to prevent unnecessary computations. Many different methods for finding these neighbours exist, but it is unclear which will work best in crowd simulation, an application which is characterised by low dimensionality and frequent change of the data. We therefore compare several different data structures that can be used to accelerate kNN queries. We find that the nanoflann implementation of a k-d tree offers the best performance of the structures tested by far on many different scenarios, processing 100,000 characters in about 60 milliseconds.

1 Introduction

In crowd simulation, we are continuously looking to simulate more characters in real time. In games, we might want to simulate large armies at interactive rates, allowing for accurate recreation of historic battles. For large events we might want to simulate many variations of fence placement or entrance and exit locations, and get the results back quickly. Efficient collision avoidance plays a major role in this, and collision avoidance methods require a character to know several of its nearest neighbours (e.g. [9], [14], [18]). Choosing a high-performance method for finding these nearest neighbours is therefore of paramount importance to simulating truly massive crowds. In this study we compare several methods for finding nearest neighbours in the context of the ECM crowd simulation framework [20].

When comparing the performance of different data structures, it is important to use suitable testing data. We therefore test our structures on a large number of scenarios, which should give a representative picture of the performance that can be expected of each structure.

The structure of the rest of this paper is as follows. In Section 2, we motivate our choice of data structures. In Section 3, we give a short description of the workings of each structure and give the theoretical bounds on time complexity, whereas in Section 4 we describe the specifics of the implementations. In Section 5 we list the sources of our test scenarios and briefly describe them. Section 6 describes the design of our software and the platforms we performed our tests on. Section 7 gives the results of our tests, as well as their interpretation. In Section 8 we provide some areas that could be further explored in the future, before giving our concluding remarks in Section 9.

2 Related work

The k-nearest neighbour problem appears in many different fields of computer science, such as machine learning, databases, computer vision and computational geometry. What many of these applications have in common, is that the data typically has high dimensionality, and that there is a clear separation between an *offline* phase in which an index is constructed, and an *online* phase in which queries are performed. In addition, many algorithms are optimised for use cases where not all data fits into main memory.

Our applications are almost the exact opposite in nature: our data has only two dimensions, fits into main memory, and most values typically change between two simulation frames. It can, therefore, be difficult to judge which methods will have the best performance in our application, as the setting in which they are usually used are so different. To our knowledge, a comparative study of different kNN methods was never performed in a setting comparable to our own.

For spatial data, the kNN problem is usually solved by building a spatial index [13, 15, 16]. These indices can broadly be divided into two categories: those that partition the space and those that partition the data. Well known instances of the former are quadtrees and k-d trees, whereas the latter includes R-trees and bounding volume hierarchies. The difference between the two is that spatial partitioning methods recursively divide the remaining space into non-overlapping areas while trying to balance the number of objects inside each subdivision, whereas data partitioning methods try to cluster the data in potentially overlapping areas, preferably based on spatial proximity.

Several motion planning packages that use data structures to accelerate nearest neighbour queries exist. For instance, the *Open Motion Planning Library* [17] supports FLANN's hierarchical clustering and a structure called the *Geometric Near-Neighbour Access Tree* [6], and the *Motion Strategy Library* [11] uses the k-d tree from the ANN library.

3 Preliminaries

In this section we provide a brief overview of the data structures we have tested. We provide theoretical bounds where available and assume a dimensionality of 2 wherever applicable.

3.1 *k*-d trees

A k-d tree [4] is a spatial partitioning structure that recursively splits the data set along one of the axes of the coordinate system. This is usually done in sequence: in the case where $k = 2$ we split alternately on x and y. Splitting continues until a maximum depth has been reached, or less than a specified number of points remains in a cell. k-d trees can be straightforwardly expanded to contain objects other than points, such as triangles or line segments. In this case it can be difficult to determine the optimal splitting plane; in our case of points in two dimensions, however, we can simply sort the points for each dimension and split along the median. A k-d tree can be constructed in $O(n\ log\ n)$ time and can be used to find the k nearest neighbours in $O(k\ log\ n)$ time.

3.2 BD-trees

The box decomposition tree [2], or BD-tree (not to be confused with the bounded deformation tree, which is also known as a BD-tree), is structurally similar to a k-d tree. It differs in two major ways: the rectangles that the space is decomposed into are *fat*, and each rectangle may have an associated *inner rectangle*. A *fat* rectangle is one with bounded ratio between the shortest and longest axis. Having fat rectangles ensures the size of the regions decreases exponentially as one traverses the tree. The *inner rectangles* allow a rectangle to be split not into two rectangles along some coordinate axis, but into an outer and an inner rectangle, where the resulting region is the set-theoretic difference between the two. A BD-tree can be constructed in $O(n\ log\ n)$ time and can be used to find the k nearest neighbours in $O(k\ log\ n)$ time.

3.3 R-trees

R-trees were created as a data partitioning structure supporting multi-dimensional data in multi-dimensional spaces [8]. They are also fully dynamic: queries may be mixed with insertions and deletions without a need for periodic rebalancing. R-trees are based on B-trees, and as such are structurally similar: the number of objects in a leaf node, as well as the number of children of an internal node, is limited by both a lower and an upper bound, and all leaves appear on the same level. Many different heuristics exist for the construction of the structure; in our case we use the R*-tree [3], because it tries to prevent overlapping regions. An R*-tree can be constructed in $O(n\ log\ n)$ time and can be used to find the k nearest neighbours in $O(k\ log\ n)$ time.

3.4 Voronoi diagrams

Voronoi diagrams are spatial subdivisions where each cell is exactly the set of points closest to a site [21]. We can then find the k nearest neighbours by an algorithm similar to Dijkstra's algorithm for single-source shortest paths, or Prim's algorithm for constructing minimum spanning trees. We add a site's neighbouring cells to a priority queue, where our priority is the distance of a cell's site to the query point. We then repeatedly extract the cell with minimum distance from the queue and add its neighbours. A Voronoi diagram can be constructed in $O(n\ log\ n)$ time, and if we use a binary heap as our priority queue, we can find the k nearest neighbours in $O(k\ log\ n)$ time.

3.5 Hierarchical *k*-means clustering

k-means clustering was originally conceived to partition a population based on a sample [12]. The algorithm works by initialising k sites[1] at some location (many seeding heuristics exist; in our application the sites are chosen randomly) and calculating for each data point which of the k sites it is closest to. Each site is then moved to the mean of all points for which it is the closest site. This process is repeated until the sites no longer move, or until a maximum number of iterations has been performed. A *hierarchical* clustering then recursively performs this algorithm on each of the clusters that has been created. The spreading of points over the clusters can be arbitrarily bad, giving us a worst-case construction time of $O(n^2)$ and a query time of $O(n)$, but if each cluster is consistently of roughly equal size, we get a construction time of $O(n\ log\ n)$ and a query time of $O(k\ log\ n)$.

[1]Note that this is a different k than the one specifying the number of nearest neighbours we want to find.

3.6 Linear search and grids

Linear search is a naive, brute-force approach to the kNN problem: when querying we simply iterate through all points in the set, keeping track of the k closest points so far. This gives a query time of $O(n)$, but it has the advantage of requiring no construction whatsoever. A grid-based approach tries to improve on this in a simple way: we place a regular grid over the entire simulation area, and keep track of the characters located in each cell. The construction time is $O(n + m)$, where m is the number of cells in the grid. The worst case query time is also $O(n+m)$, but practical performance depends heavily on both the distribution of characters and the size of the cells.

4 Data structure implementations

We made our selection of data structures based on prevalence, theoretical performance and the availability of good implementations, taking into account the low dimensionality of our application and the fact that we are working in main memory. We settled on testing BD-trees, k-d trees, R-trees, hierarchical k-means clustering and an implementation based on Voronoi diagrams. We also included linear search and a simple grid-based algorithm for reference purposes. Because there are many implementations of *k*-d trees available, we test two different versions: one from a larger library and one that is less general, but highly optimised. Furthermore, because R-trees theoretically handle updates very well, we test both a version in which we rebuild the entire tree every frame, and one in which we remove and insert every character incrementally.

The sources for these implementations are as follows:

- BD-tree: ANN [13]
- *k*-d tree: FLANN [15] and nanoflann [5]

- k-means: FLANN
- R-tree: Boost [1]
- Voronoi: Boost
- Linear search: own implementation
- Grid: own implementation

Not all of these structures were suitable for our purposes out-of-the-box. We had to modify the ANN library to support querying from multiple threads. Also, Boost only allows us to build a Voronoi diagram; it does not support nearest-neighbour queries natively. We therefore implemented the algorithm for finding the k nearest sites ourselves.

Other than Boost's R-tree implementation and the grid, none of the structures straightforwardly supported adding and removing points, so for all other structures we rebuild them completely every frame. Furthermore, we use the default parameters for creating the indices for each library. Finally, we set the cell size for the grid to 10 metres, as this is the default value used in the ECM framework.

5 Test scenarios

It is important to test our structures on relevant scenarios. We therefore used both data captured from real crowds and from simulations that have real-world applications. To also give us some insight into the differences between the structures, we introduced some artificial scenarios that highlight certain properties of the crowds. A description of all our test scenarios follows.

Our data from real crowds was obtained from the Jülich Forschungszentrum's *Institute for Advanced Simulation*. Descriptions of the various situations can be read in [10] (German). Alternatively, the source video footage can be viewed at [7]. They provide data captured from real crowds in several scenarios. We used the data from the following scenarios:

- Bidirectional flow, free choice of destination
- Bidirectional flow, ordered destination, symmetric flow rate
- Bottleneck
- Mouth hole in stadium, lower level
- Mouth hole in stadium, upper level

We chose these for the presence of a relatively large number of people, as well as the relative complexity of the situations.

We also obtained data from simulations made in the ECM framework. We used many scenarios that were developed for the planning of the Grand Depart of the Tour de France cycling competition when it took place in Utrecht [19]. These are all quite similar, as they were used to simulate different placement of things such as fences and footbridges: there is a large simulation area with only a small walkable area, where people periodically spawn at one point (simulating the arrival of trains). We also used scenarios simulating the evacuation of a building with different numbers of people present in the building. The building has several large rooms, connected to a main hall with two exits on opposite ends.

As we also wanted to obtain data on specific aspects of the performance of the different structures, we also tested several completely artificial scenarios. We were interested in how the following aspects affect the performance:

- Density: we wanted to know how the distribution of characters affected the performance of the different structures. As such, we included one scenario with 10,000 characters distributed uniformly at random, and one in which the same number of characters is spawned in clusters of higher density. Clusters are formed by picking 5 random points, spawning 75% of the characters in discs around these points, then spawning the remaining 25% uniformly at random.
- Stationary characters: if many of the characters are standing still, we would expect methods that update rather than rebuild their structures to perform better. As we

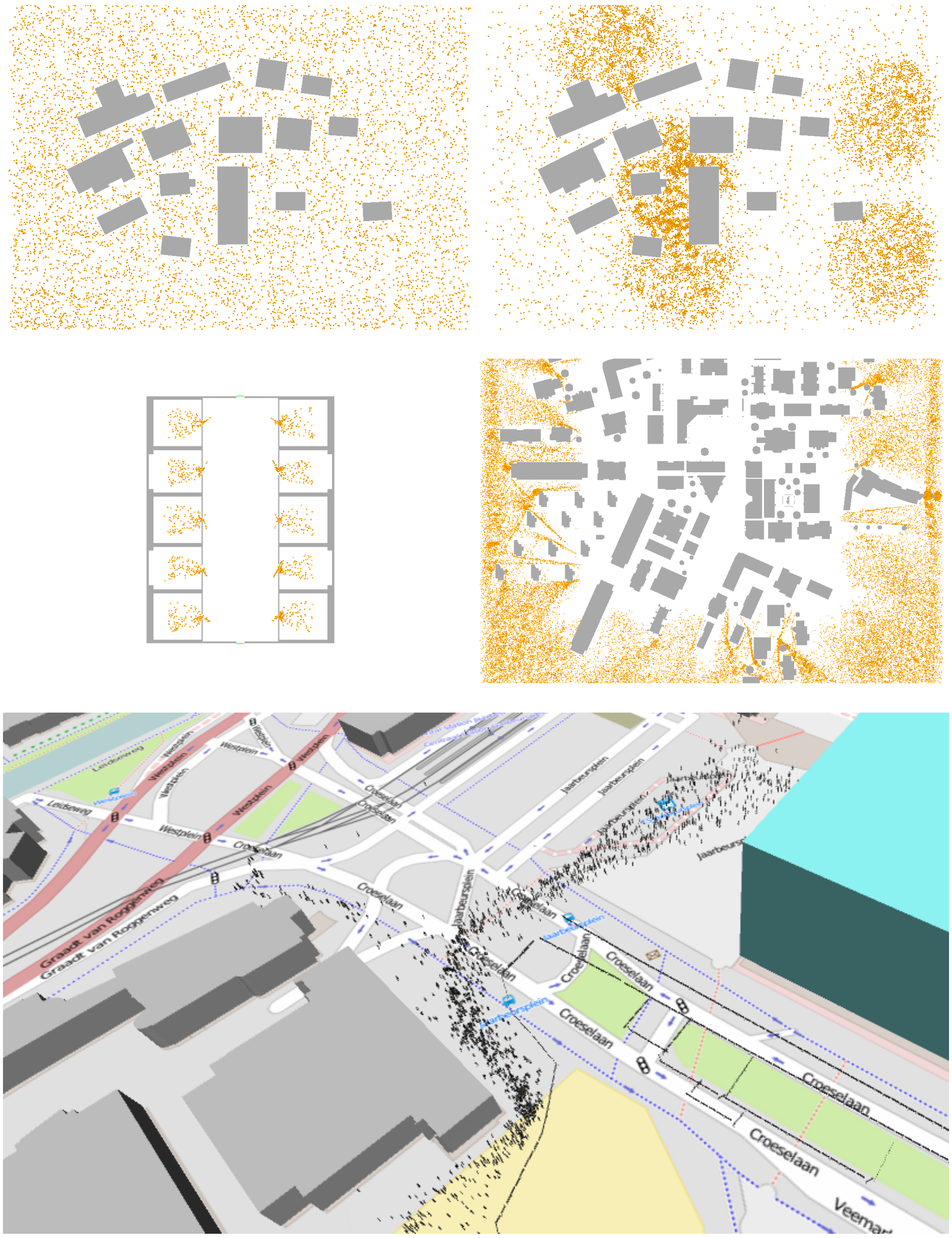


Figure 1: Some of the different test scenarios. Left to right, top to bottom: uniform density (also used for testing of stationary characters), clusters, evacuation, scaling and Tour de France.

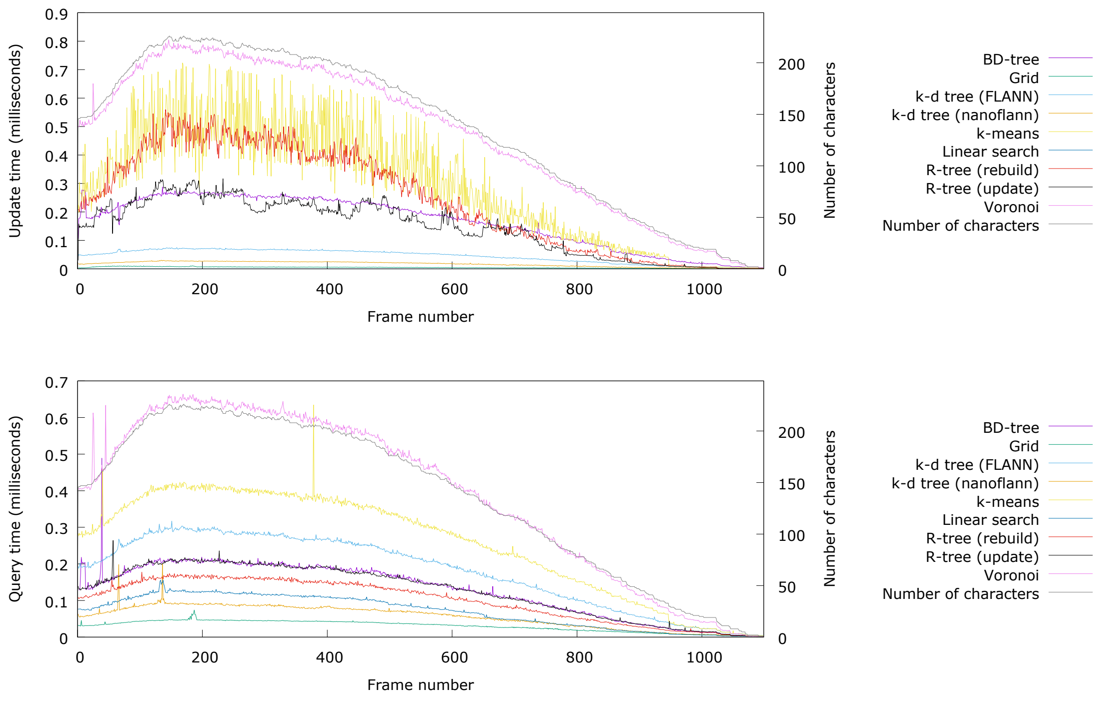


Figure 2: Update and query times for the bottleneck scenario (ao-240-400 in the tables) on the i5-4210M platform.

are interested in knowing how much better the performance will be, we included scenarios in which 25%, 50% or 75% of the characters are standing still. We implemented this by setting the goal position of the agent equal to its starting position, meaning they might still be forced to move if local agents forces compel them.

- Scaling: an important aspect of the performance of these structures is how they scale with the number of characters being simulated. To this end, we included a scenario where 100 characters are added to the simulation every frame, at the edges of the simulated area. We ran this simulation for 1000 frames, so we can see the scaling of performance from 0 to 100,000 characters.

A selection of images showing some of our test scenarios can be seen in Figure 1.

6 Testing environment

Some of the data we wanted to test our structures on was only available as trajectory data. Because we want to make a fair comparison between all our structures, we therefore decided to test all of them on this trajectory data, even when a simulation was available. All our simulations were therefore first saved as trajectory data. We then made an application that would, for each structure, read the trajectory data frame-by-frame, update the structure, then perform a kNN query for each character. For all our tests, $k = 10$; we do not vary this parameter because our collision avoidance methods generally don't need more than several of the closest neighbours. Each frame, we measure the time taken to update the structure and the time taken to perform all queries. To better approximate the real usage of these structures in the ECM simulator, we parallelised the queries using OpenMP.

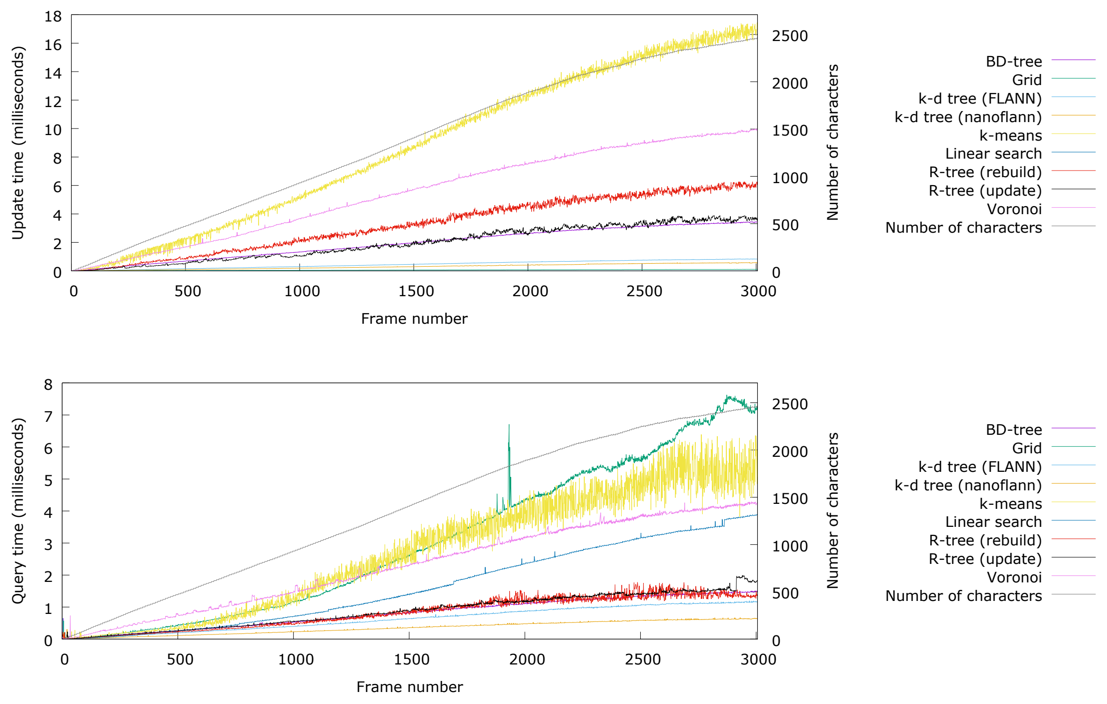


Figure 3: Update and query times for Tour de France scenario 12 on the i7-2600K platform.

All our code was implemented in the C++ programming language, as were all the libraries we used. We ran our tests on two different platforms to help prevent drawing conclusions that turn out to be platform-specific. Our first platform was a desktop computer with an Intel Core i7-2600K (4 cores, 8 threads) running at 4.3 GHz with 16 GB of DDR3 RAM, running Windows 7 64-bit and using the Microsoft C++ compiler version 19.00.23026. Our second platform was a notebook with an Intel Core i5-4210M (2 cores, 4 threads) running at 3.2 GHz with 8 GB of DDR3L RAM, running Linux Mint 17.2 and using the g++ compiler version 4.8.4. We chose to run our tests on two completely different systems because, rather than determining which hardware/software combination works best, we want to confirm that our results are consistent across platforms.

7 Results

In total, we tested 9 different methods on 64 different scenarios, measuring separately for every frame the time needed to update and query the structure. As such, the amount of data generated is quite large. We therefore present graphs only for a selection of scenarios we consider representative of the whole set. We also provide the mean and standard deviation of update and query times in Tables 1–8. To prevent a misrepresentation of the standard deviation in cases where the number of characters is not constant, the figures presented are always time needed *per character* in microseconds. This is in contrast to the graphs, which contain data on *total* update or query time, in *milli*seconds, as well as a plot of the number of characters over time. It is also worth noting that, strictly for visualisation purposes, some of the largest peaks have been filtered out of the graphs, so that the rest of the graph does not get vertically compressed. Also, as the results

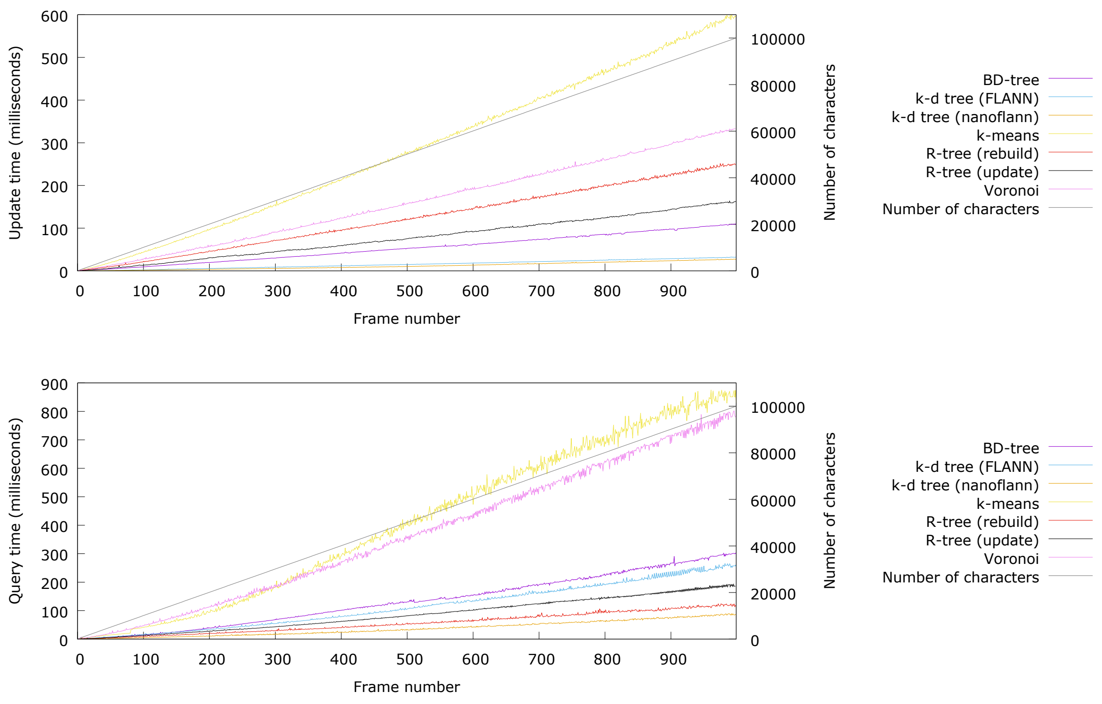


Figure 4: Update and query times for the scaling test on the i5-4210M platform.

of the two platforms generally yielded very similar graphs, we show here those that provided the clearest representation of the data (the one with less noise and less overlapping plots).

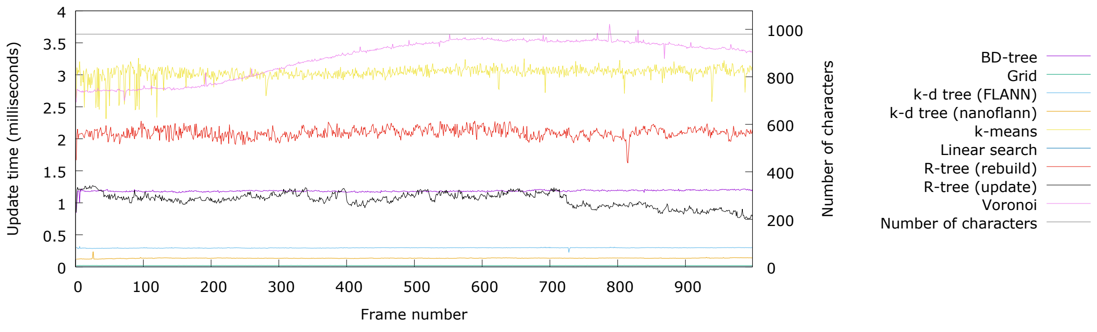
7.1 Bottleneck scenario

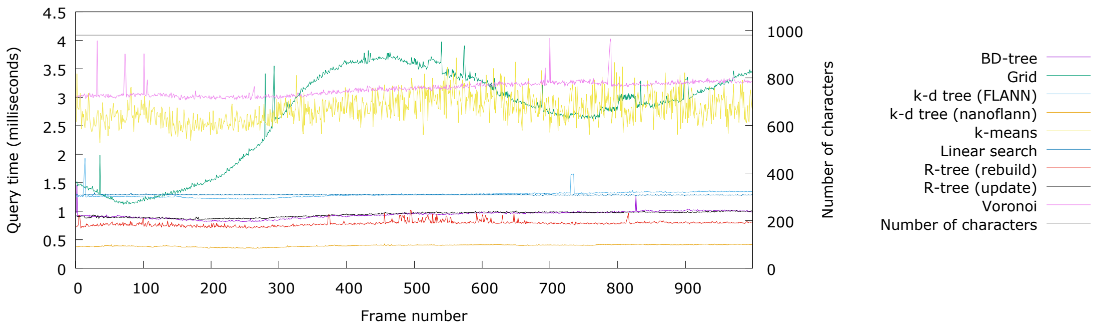
In Figure 2, the update and query performance of all structures on the bottleneck scenario from the Jülich data can be seen (ao-240-400 in the tables). There is clearly a strong correlation between the number of characters and computation time for both updating and querying the structures. The Voronoi diagram-based implementation is clearly the slowest: it incurs more overhead than the other methods, as it needs auxiliary data structures to map characters to Voronoi cells and vice versa. The k-means method has wildly varying update times, which we think is caused by either good or poor outcomes of the random initialisation of the k sites. More surprising is the large variance in the update of the R-tree with rebuilding. This implementation uses a packing algorithm to generate the R-tree, which is apparently quite sensitive to the exact layout of the points. The low number of characters and the uniform density make this scenario particularly well-suited for a grid-based approach, and indeed this method clearly outperforms the others in this case.

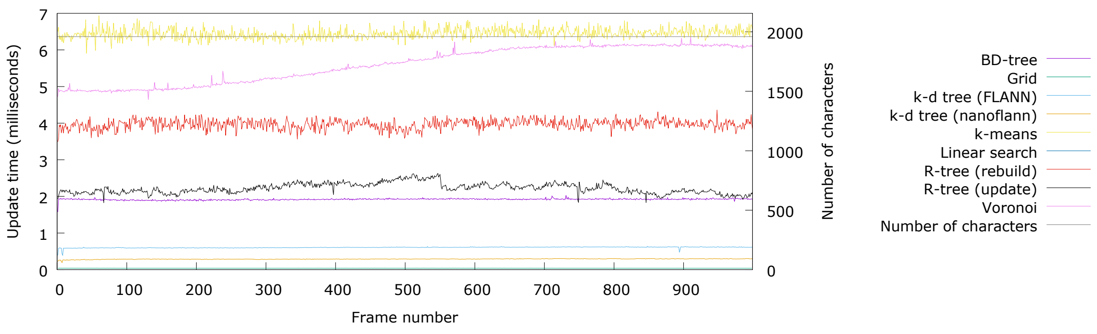
7.2 Evacuation scenario

Figure 5 shows the update and query performance on evacuation scenarios 1 and 3 side-by-side. The performance of the grid-based method is the first thing that catches the eye: the query time quickly rises as people begin to crowd the doors of the building, then drops as people make it outside. There is also a distinct rise in update time for the Voronoi diagram-based method: apparently, the algorithm used to create the Voronoi diagram is somewhat sensitive to the distribution of the sites.

In evacuation scenario 1, we also see a slight







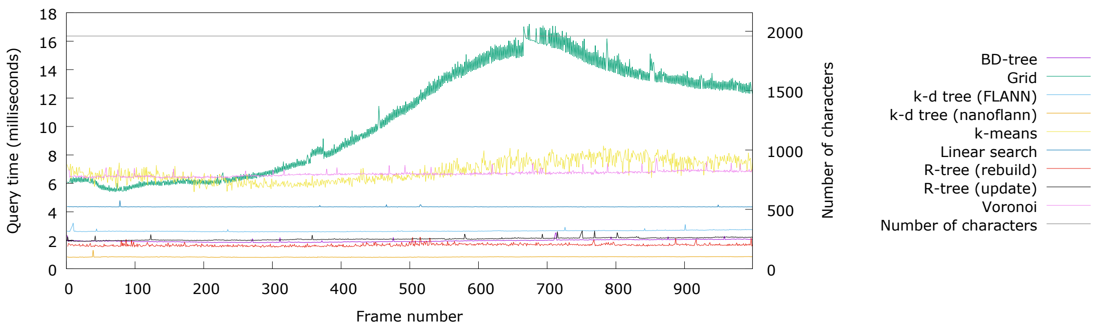


Figure 5: Update and query times for evacuation scenario 1 (top two) and 3 (bottom two) on the i5-4210M platform.

drop in update time for the R-tree method, as a growing portion of people have successfully evacuated the building and have stopped moving. We don't see this in scenario 3, because the fraction of people that reach their destination by the end of the simulation is much smaller, not only because there are simply more people, but also because the congestion at the doorways is much more severe.

7.3 Tour de France scenario

In Figure 3, we clearly see that the query performance of the grid-based method starts to show some non-linear behaviour: most of the simulated area is empty, and all characters follow a relatively narrow corridor, meaning most of the cells explored when searching for neighbours will be empty. However, when we consider the extremely low update times, the method still performs better than the k-means and Voronoi methods.

7.4 Stationary characters test

The three tests with varying degrees of stationary characters perform exactly as expected: the implementation that updates an R-tree rather than rebuilding it has progressively better update performance as more characters are stationary, while other structures are unaffected. However, even with 75% of the characters standing still, the R-tree fails to outperform either of the k-d tree implementations.

7.5 Density test

Figure 6 shows that at 10,000 characters linear search and the grid-based method become too slow to still be viable options. With a uniformly random distribution of characters the grid still performs relatively well, but over time the distribution of characters naturally becomes more clustered, causing a large performance hit for queries. None of the other methods seem to be particularly affected by changes in density.

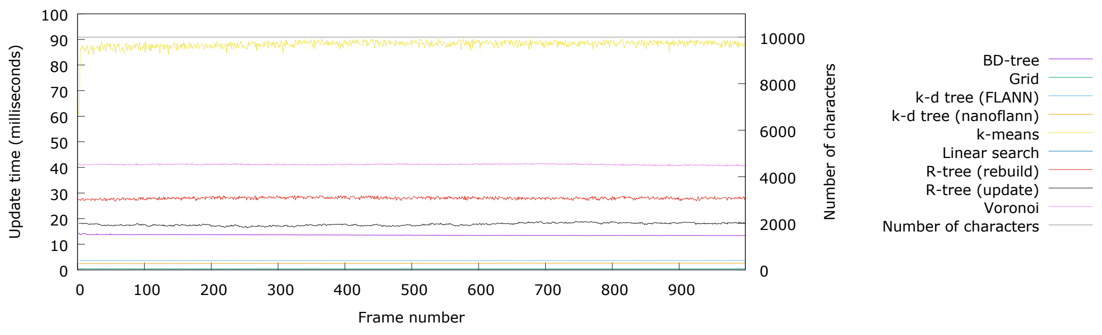
7.6 Scaling test

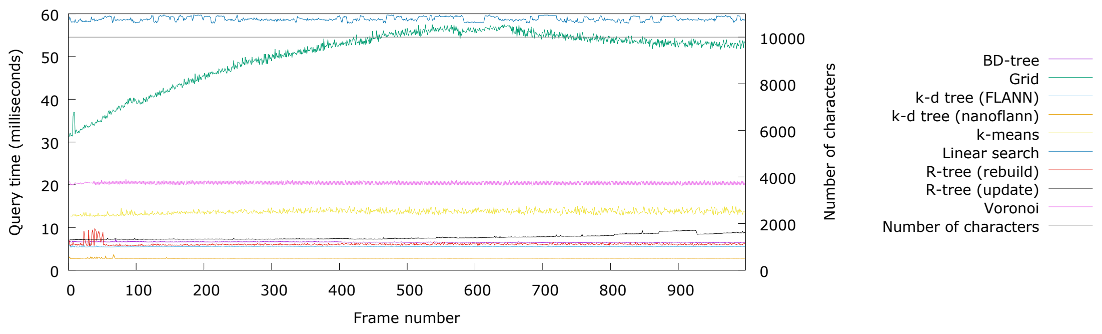
Figure 4 shows the change in performance as the number of characters steadily increases. In these plots, the grid and linear search are omitted, as they become so slow that the rest of the graph becomes unreadable. The k-means method most clearly shows a non-linear response to an increase in the number of characters. Furthermore, the difference between rebuilding and updating the R-tree becomes more pronounced: rebuilding the entire tree takes more time, but yields better query performance. Interestingly enough, the two roughly balance out, giving similar performance for updating *and* querying the structure.

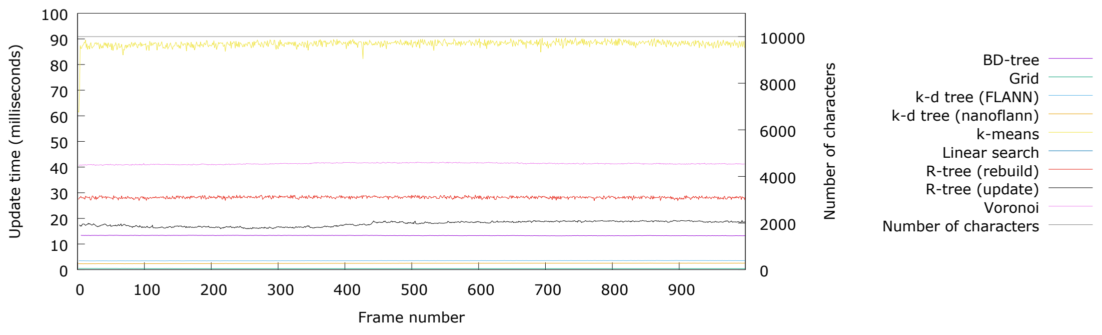
7.7 General remarks

Looking at the tables of means and standard deviations, the first thing we notice is that there are quite many outliers. These are all caused by tests occasionally containing frames in which computation took many times longer than usual. The fact that they are almost exclusively present in the query time measurements suggests that the parallel execution of multiple queries plays a role in this behaviour. It is unclear however whether these spikes are caused by inter-thread disruption (e.g. one threads invalidates another's cache) or by external interruption (e.g. the operating system momentarily allocates processor time to an unrelated process). We also notice that these spikes are much more common with the BD-tree than with other structures. This might be because the BD-tree is consistently tested first: many of the spikes occur in the first few frames for this structure.

Another point of interest is that there are some differences between the two platforms. The FLANN implementation of the k-d tree has decidedly poorer query performance on the i5-4210M platform than on the i7-2600K platform. This could be because the compiler fails to make an optimisation. Furthermore, even though the i5 platform has a lower clock speed, the update performance is consistently better than on the i7 platform. Again, we suspect this is due to an optimisation one platform is fail-







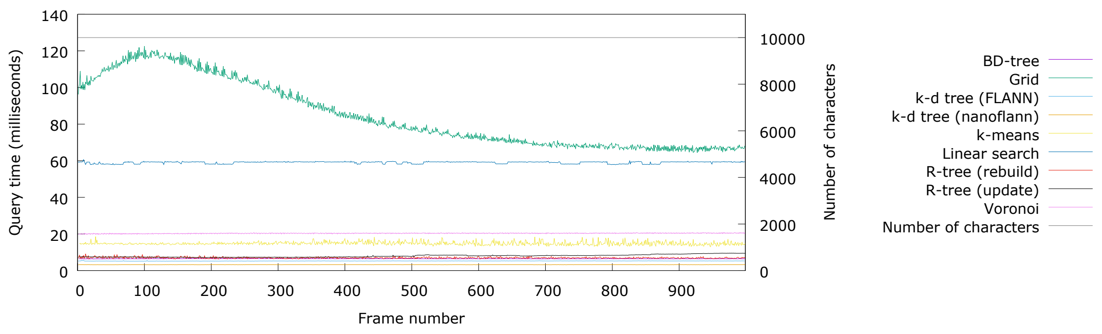


Figure 6: Update and query times for the uniform density (top two) and the clusters (bottom two) test on the i7-2600K platform.

ing to make. This difference has little influence on the relative performance between the data structures.

Overall, the nanoflann implementation of the k-d tree clearly gives the best performance: it is only outperformed in situations with very few characters. On the i7-2600K system it can process 100,000 characters in about 60 milliseconds per frame.

8 Future work

When querying a character for its nearest neighbours, we are generally not interested in characters that are far away: they are unlikely to have any noticeable influence on the character's behaviour. Employing a fixed-radius nearest neighbour query might then yield better querying performance when there are often few characters within the desired range. The data structures we describe can straightforwardly support this operation (and indeed, most libraries we used already do), so if this is desirable our code can easily be modified to support this type of query.

Another observation is that each character only ever moves a small distance per simulation frame. We could potentially exploit this by only updating the query structure once every few frames, as we expect the results will not differ much. When combined with a fixed-radius search, we could even guarantee finding a superset of the exact result by increasing the search radius by twice the maximum distance any character can move between updates. This modification could potentially give a great boost to performance, but research is needed to determine the number of frames we could delay the update by.

The ECM framework supports environments with multiple layers. Currently, the data structures used to accelerate kNN queries are built separately for each layer. It might be beneficial on the whole to devise a data structure that properly deals with such environments. Furthermore, the neighbours are currently filtered based on visibility after a query is performed; it might be more efficient to take this into account during the query in some way.

Finally, the Voronoi diagram-based implementation currently does not perform particularly well. This can largely be attributed to our unoptimised method of performing the actual kNN queries. Large performance gains might be possible by applying a proper point location algorithm to accelerate our queries. Additionally, higher-order Voronoi diagrams might be used to generate cells that have the same k nearest neighbours, removing the need to look at any of the neighbouring cells.

9 Conclusion

Having tested nine different implementations of structures for finding the k nearest neighbours of characters in a simulated crowd on a variety of scenarios, we can conclude that the nanoflann implementation of the k-d tree is the fastest by quite a large margin, even for a moderate number of characters. A grid-based approach can be efficient for moderate numbers of characters (up to about 1000), mostly because updating the structure is very cheap, but this method is very sensitive to the spatial distribution of characters. The implementations of BD-trees and R-trees offer roughly equal performance, but neither can really compete with the nanoflann k-d tree. The k-means and Voronoi diagram-based methods are significantly slower than the other spatial indices, but the performance of the latter might be improved by implementation of a point location algorithm and by employing higher-order Voronoi diagrams. In the end we can index 100,000 characters and find the 10 nearest neighbours for each of them in about 60 milliseconds per frame on an Intel Core i7-2600K.

References

[1] Boost C++ Libraries. `http://www.boost.org`.

[2] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. An optimal algorithm for

approximate nearest neighbor searching fixed dimensions. *Journal of the ACM (JACM)*, 45(6):891–923, 1998.

[3] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The r*-tree: an efficient and robust access method for points and rectangles. In *SIGMOD '90 Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, pages 322–331. ACM, 1990.

[4] Jon L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[5] Jose Luis Blanco-Claraco. nanoflann. `https://github.com/jlblancoc/nanoflann` (accessed 17-01-2016).

[6] Sergey Brin. Near neighbor search in large metric spaces. In *VLDB '95 Proceedings of the 21st International Conference on Very Large Data Bases*, pages 574–584.

[7] Jülich Forschungszentrum Institute for Advanced Simulation. Pedestrian Dynamics Database. `http://www.fz-juelich.de/ias/jsc/EN/Research/ModellingSimulation/CivilSecurityTraffic/PedestrianDynamics/Activities/database/databaseNode.html` (accessed 17-01-2016).

[8] Antonin Guttman. R-trees: a dynamic index structure for spatial searching. In *SIGMOD '84 Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, pages 47–57. ACM, 1984.

[9] Stephen J. Guy, Jatin Chhugani, Changkyu Kim, Nadathur Satish, Ming Lin, Dinesh Manocha, and Pradeep Dubey. Clearpath: highly parallel collision avoidance for multi-agent simulation. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 177–187. ACM, 2009.

[10] Christian Keip and Kevin Ries. Dokumentation von Versuchen zur Personenstromdynamik, 2009. `http://ped.fz-juelich.de/experiments/2009.05.12_Duesseldorf_Messe_Hermes/docu/VersuchsdokumentationHERMES.pdf`.

[11] Steve LaValle et al. Motion Strategy Library. `http://msl.cs.uiuc.edu/msl/`.

[12] James MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA., 1967.

[13] David M. Mount and Sunil Arya. ANN: A Library for Approximate Nearest Neighbor Searching. `http://www.cs.umd.edu/~mount/ANN/`.

[14] Mehdi Moussaïd, Dirk Helbing, and Guy Theraulaz. How simple rules determine pedestrian behavior and crowd disasters. *Proceedings of the National Academy of Sciences*, 108(17):6884–6888, 2011.

[15] Marius Muja and David G. Lowe. FLANN - Fast Library for Approximate Nearest Neighbors. `http://www.cs.ubc.ca/research/flann/`.

[16] Nick Roussopoulos, Stephen Kelley, and Frédéric Vincent. Nearest Neighbor Queries. In *SIGMOD '95 Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, volume 24, pages 71–79. ACM, 1995.

[17] Ioan A. Şucan, Mark Moll, and Lydia E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, December 2012. `http://ompl.kavrakilab.org`.

[18] Jur van den Berg, Ming Lin, and Dinesh Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. In *IEEE International Conference*

on Robotics and Automation, 2008. ICRA 2008., pages 1928–1935. IEEE, 2008.

[19] Marijn van der Zwan. The Impact of Density Measurement on the Fundamental Diagram, 2015. To be published.

[20] Wouter G. van Toll, Norman S. Jaklin, and Roland Geraerts. Towards Believable Crowds: A Generic Multi-Level Framework for Agent Navigation. 2015.

[21] Georgy Voronoi. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. premier mémoire. sur quelques propriétés des formes quadratiques positives parfaites. *Journal für die reine und angewandte Mathematik*, 133:97–178, 1908.

	BD-tree	Grid	k-d tree (FLANN)	k-d tree (nanoflann)	k-means	Linear search	R-tree (Rebuild)	R-tree (Update)	Voronoi
ao-240-400	1.21	0.07	0.45	0.19	2.73	0.00	1.37	0.90	3.63
ao-300-400	1.28	0.07	0.42	0.17	2.67	0.00	1.36	0.89	3.59
ao-360-400	1.29	0.07	0.39	0.17	2.65	0.00	1.36	0.89	3.57
ao-440-400	1.28	0.07	0.40	0.17	2.51	0.00	1.29	7.05	3.61
ao-500-400	1.29	0.08	0.42	0.17	2.47	0.00	1.27	0.81	3.49
bo-360-050-050	1.31	0.12	0.63	0.21	0.68	0.01	0.34	0.32	3.11
bo-360-075-075	1.31	0.11	0.61	0.21	0.66	0.01	0.33	0.27	3.04
bo-360-090-090	1.31	0.10	0.45	0.17	1.54	0.00	0.48	0.39	3.29
bo-360-120-120	1.30	0.09	0.43	0.16	1.75	0.00	0.62	0.49	3.32
bo-360-160-160	1.31	0.08	0.43	0.17	1.94	0.00	0.73	0.60	3.34
bot-300-050-050	1.27	0.11	0.59	0.21	0.64	0.01	0.34	0.28	3.10
bot-300-065-065	1.30	0.11	0.58	0.21	0.84	0.01	0.41	0.32	3.15
bot-300-075-075	1.31	0.10	0.57	0.20	1.33	0.01	0.41	0.32	3.22
bot-300-085-085	1.31	0.10	0.48	0.17	1.58	0.00	0.46	0.39	3.33
bot-300-100-100	1.31	0.09	0.44	0.16	1.79	0.00	0.52	0.47	3.39
bot-360-050-050	1.31	0.11	0.61	0.21	0.66	0.01	0.33	0.27	3.03
bot-360-075-075	1.33	0.11	0.59	0.21	1.14	0.01	0.41	0.33	3.18
bot-360-090-090	1.36	0.10	0.58	0.20	1.73	0.01	0.48	0.39	3.28
bot-360-120-120v2	1.36	0.09	0.46	0.17	1.86	0.00	0.56	0.50	3.42
bot-360-120-120	1.34	0.09	0.47	0.17	1.89	0.00	0.59	0.52	3.38
bot-360-160-160	1.35	0.08	0.43	0.17	2.04	0.00	0.70	0.65	3.39
bot-360-200-200	1.32	0.08	0.41	0.16	2.27	0.00	0.83	0.81	3.48
bot-360-250-250	1.31	0.08	0.44	0.17	2.29	0.00	0.86	0.84	3.40
mo11_MB	1.31	0.06	0.38	0.16	2.34	0.00	1.06	0.70	3.30
mo12_MB	1.31	0.06	0.37	0.16	2.52	0.00	1.13	0.77	3.35
mo13_MB	1.30	0.07	0.40	0.16	2.02	0.00	0.78	0.70	3.20
mo15_MB	1.33	0.07	0.44	0.18	2.48	0.00	1.09	0.80	3.33
mo16_MB	1.34	0.07	0.59	0.21	2.52	0.01	1.07	0.79	3.26
mo21_MB	1.27	0.07	0.45	0.17	1.55	0.00	0.61	0.51	3.05
mo22_MB	1.33	0.08	0.58	0.22	1.73	0.01	0.57	0.43	3.06
mo24_MB	1.29	0.06	0.39	0.16	2.07	0.00	0.79	0.65	3.29
mu11v2_MB	1.29	0.06	0.43	0.17	2.53	0.00	1.25	0.71	3.50
mu12_MB	1.31	0.07	0.43	0.17	2.21	0.00	1.01	0.62	3.35
mu14_MB	1.33	0.07	0.46	0.18	2.41	0.00	1.19	0.81	3.49
mu15_MB	1.34	0.07	0.52	0.20	2.52	0.00	1.22	0.73	3.46
mu21v2_MB	1.31	0.06	0.40	0.16	2.25	0.00	0.99	0.67	3.39
mu21_MB	1.29	0.07	0.42	0.17	2.07	0.00	0.88	0.65	3.34
mu22_MB	1.31	0.07	0.46	0.17	1.74	0.00	0.71	0.52	3.18
tour de france 1	1.39	0.04	0.33	0.19	5.35	0.00	2.02	1.23	3.55
tour de france 2	1.42	0.04	0.33	0.20	5.60	0.00	2.08	1.28	3.54
tour de france 3	1.42	0.04	0.33	0.19	5.79	0.00	2.14	1.32	3.55
tour de france 4	1.41	0.04	0.33	0.20	5.93	0.00	2.17	1.33	3.55
tour de france 5	1.42	0.04	0.32	0.19	5.36	0.00	2.04	1.26	3.51
tour de france 6	1.42	0.04	0.32	0.19	5.32	0.00	2.03	1.25	3.50
tour de france 7	1.45	0.04	0.32	0.19	4.53	0.00	1.84	1.11	3.49
tour de france 8	1.46	0.04	0.33	0.18	4.33	0.00	1.76	1.04	3.47
tour de france 9	1.44	0.04	0.33	0.19	4.90	0.00	1.94	1.23	3.47
tour de france 10	1.45	0.04	0.33	0.19	4.74	0.00	1.90	1.16	3.58
tour de france 11	1.43	0.04	0.33	0.20	5.07	0.00	1.97	1.27	3.49
tour de france 12	1.42	0.04	0.33	0.21	5.73	0.00	2.19	1.33	3.88
tour de france 13	1.41	0.04	0.33	0.20	5.74	0.00	2.22	1.34	3.88
tour de france 14	1.40	0.04	0.33	0.21	6.33	0.00	2.41	1.43	3.95
tour de france 15	1.41	0.04	0.33	0.20	6.16	0.00	2.33	1.37	3.91
tour de france 16	1.40	0.04	0.33	0.21	6.34	0.00	2.38	1.43	3.94
tour de france 17	1.40	0.04	0.33	0.21	6.30	0.00	2.35	1.39	3.92
evacuation 1	1.32	0.04	0.32	0.19	5.57	0.00	2.25	1.17	4.91
evacuation 2	1.30	0.04	0.32	0.20	6.12	0.00	2.36	1.33	4.75
evacuation 3	1.29	0.04	0.32	0.20	6.43	0.00	2.49	1.43	4.61
uniform density	1.36	0.04	0.36	0.26	8.81	0.00	2.80	1.78	4.12
clusters	1.33	0.04	0.35	0.25	8.81	0.00	2.82	1.79	4.14
stationary 25%	1.48	0.04	0.39	0.25	8.86	0.00	2.90	1.29	4.34
stationary 50%	1.46	0.04	0.38	0.25	8.78	0.00	2.94	0.88	4.34
stationary 75%	1.45	0.04	0.39	0.26	8.73	0.00	2.92	0.47	4.36
scaling	1.30	0.04	0.42	0.32	8.62	0.00	2.89	1.82	3.68

Table 1: The average time needed per character to update the structure on the i7-2600K system, in microseconds.

	BD-tree	Grid	k-d tree (FLANN)	k-d tree (nanoflann)	k-means	Linear search	R-tree (Rebuild)	R-tree (Update)	Voronoi
ao-240-400	0.33	0.02	0.38	0.12	1.18	0.02	0.72	0.38	1.05
ao-300-400	0.07	0.02	0.32	0.09	1.15	0.01	0.69	0.40	0.45
ao-360-400	0.22	0.02	0.22	0.06	1.23	0.01	0.74	1.46	0.77
ao-440-400	0.06	0.01	0.19	0.05	1.10	0.01	0.68	46.69	0.41
ao-500-400	0.07	0.02	0.27	0.07	1.09	0.02	0.70	0.38	0.51
bo-360-050-050	0.18	0.06	0.65	0.17	0.78	0.02	0.21	0.14	0.49
bo-360-075-075	0.22	0.06	0.57	0.15	0.68	0.03	0.18	0.14	0.43
bo-360-090-090	0.15	0.04	0.35	0.11	0.60	0.02	0.16	0.11	0.34
bo-360-120-120	0.15	0.04	0.30	0.10	0.66	0.01	0.19	0.17	0.54
bo-360-160-160	0.10	0.03	0.32	0.10	0.69	0.02	0.23	0.23	0.41
bot-300-050-050	0.18	0.06	0.46	0.13	0.54	0.02	0.15	0.14	0.48
bot-300-065-065	0.13	0.04	0.49	0.16	0.77	0.02	0.15	0.13	0.50
bot-300-075-075	0.17	0.04	0.53	0.15	0.80	0.03	0.18	0.14	0.47
bot-300-085-085	0.12	0.04	0.38	0.11	0.67	0.01	0.15	0.14	0.39
bot-300-100-100	0.12	0.04	0.32	0.10	0.68	0.01	0.15	0.17	0.42
bot-360-050-050	0.16	0.04	0.46	0.13	0.60	0.02	0.16	0.13	0.43
bot-360-075-075	0.21	0.05	0.61	0.20	0.93	0.03	0.17	0.13	0.42
bot-360-090-090	0.24	0.05	0.65	0.18	0.80	0.03	0.18	0.14	0.45
bot-360-120-120v2	0.37	0.05	0.45	0.13	0.76	0.02	0.19	0.17	0.38
bot-360-120-120	0.21	0.04	0.47	0.13	0.73	0.03	0.20	0.19	0.41
bot-360-160-160	0.27	0.04	0.37	0.10	0.78	0.02	0.27	0.29	0.45
bot-360-200-200	0.08	0.04	0.33	0.10	0.78	0.02	0.29	0.35	0.34
bot-360-250-250	0.09	0.04	0.43	0.11	0.83	0.02	0.31	0.41	0.39
mo11_MB	0.07	0.01	0.16	0.04	0.92	0.01	0.52	0.28	0.32
mo12_MB	0.07	0.02	0.23	0.06	1.00	0.01	0.51	0.30	0.32
mo13_MB	0.08	0.02	0.21	0.06	0.77	0.01	0.34	0.33	0.35
mo15_MB	0.14	0.03	0.49	0.13	1.04	0.02	0.52	0.35	0.36
mo16_MB	0.16	0.05	0.81	0.21	1.08	0.04	0.50	0.36	0.40
mo21_MB	0.14	0.02	0.18	0.05	0.96	0.01	0.40	0.37	0.46
mo22_MB	0.22	0.04	0.63	0.18	0.85	0.02	0.24	0.20	0.51
mo24_MB	0.07	0.02	0.24	0.07	0.84	0.01	0.33	0.24	0.31
mu11v2_MB	0.10	0.03	0.32	0.08	1.02	0.01	0.60	0.28	0.42
mu12_MB	0.13	0.03	0.34	0.11	0.97	0.02	0.51	0.27	0.46
mu14_MB	0.13	0.04	0.48	0.12	1.11	0.02	0.64	0.85	0.47
mu15_MB	0.18	0.04	0.79	0.16	1.15	0.03	0.61	0.33	0.65
mu21v2_MB	0.20	0.03	0.27	0.08	0.89	0.01	0.43	0.26	0.60
mu21_MB	0.18	0.02	0.28	0.08	1.06	0.01	0.45	0.33	0.64
mu22_MB	0.25	0.02	0.33	0.10	1.53	0.01	0.38	0.27	0.72
tour de france 1	0.12	0.00	0.05	0.02	0.96	0.00	0.34	0.18	0.96
tour de france 2	0.04	0.00	0.03	0.02	0.97	0.00	0.34	0.18	0.12
tour de france 3	0.04	0.00	0.03	0.02	0.97	0.00	0.33	0.19	0.11
tour de france 4	0.03	0.00	0.02	0.02	0.95	0.00	0.32	0.16	0.13
tour de france 5	0.06	0.00	0.06	0.02	1.16	0.00	0.40	0.22	0.17
tour de france 6	0.04	0.00	0.03	0.03	1.16	0.00	0.42	0.21	0.16
tour de france 7	0.06	0.01	0.07	0.03	0.92	0.00	0.39	0.20	0.21
tour de france 8	0.06	0.00	0.09	0.03	0.95	0.00	0.40	0.20	0.16
tour de france 9	0.05	0.00	0.16	0.03	0.96	0.00	0.37	0.21	0.14
tour de france 10	0.05	0.02	0.05	0.02	0.93	0.00	0.38	0.20	7.02
tour de france 11	0.04	0.01	0.05	0.02	0.96	0.00	0.38	0.21	0.14
tour de france 12	0.05	0.00	0.03	0.02	1.29	0.00	0.42	0.24	0.25
tour de france 13	0.04	0.00	0.04	0.02	1.25	0.00	0.44	0.22	0.23
tour de france 14	0.08	0.00	0.09	0.02	1.11	0.00	0.35	0.20	0.18
tour de france 15	0.11	0.00	0.02	0.02	1.16	0.00	0.33	0.19	0.19
tour de france 16	0.03	0.00	0.02	0.02	1.12	0.00	0.35	0.21	0.17
tour de france 17	0.03	0.00	0.02	0.02	1.07	0.00	0.33	0.18	0.16
evacuation 1	0.07	0.00	0.01	0.01	0.21	0.00	0.11	0.13	0.66
evacuation 2	0.02	0.00	0.01	0.02	0.16	0.00	0.10	0.09	0.59
evacuation 3	0.02	0.00	0.00	0.00	0.18	0.00	0.10	0.07	0.55
uniform density	0.13	0.00	0.00	0.01	0.18	0.00	0.04	0.05	0.02
clusters	0.01	0.00	0.00	0.01	0.17	0.00	0.04	0.10	0.06
stationary 25%	0.18	0.00	0.01	0.00	0.36	0.00	0.22	0.03	0.26
stationary 50%	0.04	0.00	0.03	0.01	0.18	0.00	0.29	0.02	0.18
stationary 75%	0.03	0.00	0.03	0.16	0.18	0.00	0.08	0.01	0.10
scaling	0.15	0.01	0.03	0.04	0.62	0.00	4.30	0.15	0.42

Table 2: The standard deviation of the time needed per character to update the structure on the i7-2600K system, in microseconds.

	BD-tree	Grid	k-d tree (FLANN)	k-d tree (nanoflann)	k-means	Linear search	R-tree (Rebuild)	R-tree (Update)	Voronoi
ao-240-400	0.91	0.18	0.83	0.36	0.86	0.62	1.04	1.09	1.65
ao-300-400	0.76	0.29	0.55	0.26	0.74	0.31	0.67	0.89	1.62
ao-360-400	1.64	0.90	1.07	0.28	0.74	0.31	0.75	1.08	2.16
ao-440-400	0.74	0.29	0.56	0.27	0.74	0.39	0.49	36.38	1.56
ao-500-400	0.76	0.31	0.47	0.28	0.93	0.30	0.49	0.95	1.90
bo-360-050-050	13.32	17.01	15.88	0.45	1.14	15.63	14.40	88.44	17.37
bo-360-075-075	8.72	8.68	15.75	0.48	0.57	0.35	0.64	81.96	1.22
bo-360-090-090	0.59	0.34	0.56	0.30	0.88	0.33	5.57	17.70	1.57
bo-360-120-120	0.64	0.27	0.52	0.30	3.03	0.33	0.69	1.16	2.71
bo-360-160-160	0.62	0.26	0.47	0.26	1.55	0.30	0.54	2.18	2.23
bot-300-050-050	19.66	8.36	16.35	2.45	3.25	0.42	0.59	5.71	1.21
bot-300-065-065	0.62	0.47	0.56	0.29	0.62	0.44	0.73	0.52	2.62
bot-300-075-075	0.74	0.84	1.21	0.56	0.89	0.42	0.58	0.50	1.66
bot-300-085-085	2.77	0.27	0.59	0.32	0.83	0.39	0.56	0.53	4.55
bot-300-100-100	0.58	0.21	0.57	0.31	0.89	0.39	0.60	0.51	1.54
bot-360-050-050	0.59	0.47	0.59	0.38	0.60	0.39	0.66	0.50	1.23
bot-360-075-075	9.08	0.30	0.64	0.45	0.68	0.36	0.57	0.51	1.30
bot-360-090-090	0.59	0.48	0.90	0.34	3.80	0.44	0.57	0.56	1.47
bot-360-120-120v2	0.78	0.38	0.64	0.32	2.87	0.45	0.59	0.56	1.45
bot-360-120-120	8.18	0.29	0.47	0.31	0.85	0.40	0.55	0.53	2.06
bot-360-160-160	2.30	0.31	0.57	0.31	0.78	1.70	0.62	0.52	2.16
bot-360-200-200	0.66	0.21	0.43	0.27	1.66	0.36	0.51	0.54	1.57
bot-360-250-250	1.02	0.31	0.52	0.29	1.14	0.34	0.56	1.80	1.93
mo11_MB	1.15	0.17	0.51	0.25	0.95	0.29	0.75	0.63	1.71
mo12_MB	0.66	0.17	0.44	0.32	0.76	0.37	0.49	0.62	1.63
mo13_MB	0.58	0.21	0.45	0.25	0.75	0.29	0.54	1.03	1.46
mo15_MB	0.60	0.24	0.88	0.28	1.06	0.30	0.48	0.82	1.75
mo16_MB	0.62	0.25	0.65	0.28	0.81	0.30	0.81	0.70	1.51
mo21_MB	0.78	0.20	0.43	0.25	0.67	0.29	0.49	0.85	1.43
mo22_MB	0.95	0.32	0.51	0.30	0.75	0.26	0.59	0.90	1.54
mo24_MB	1.06	0.18	0.43	0.62	0.86	0.27	0.48	0.56	1.51
mu11v2_MB	0.90	0.33	0.43	0.28	0.85	0.29	0.55	0.72	1.63
mu12_MB	0.81	0.19	0.44	0.25	0.76	0.54	7.75	0.84	1.94
mu14_MB	0.63	0.21	0.44	0.26	1.13	0.32	0.87	1.00	1.47
mu15_MB	0.69	0.21	0.53	0.29	1.14	0.43	0.64	0.67	1.64
mu21v2_MB	0.69	0.54	0.45	0.25	0.90	0.28	0.61	1.17	1.58
mu21_MB	0.58	0.19	0.76	0.28	0.75	0.27	0.48	1.75	1.59
mu22_MB	0.58	0.19	0.45	0.27	0.80	0.33	0.48	2.26	1.47
tour de france 1	0.62	1.46	2.77	0.25	1.26	0.82	0.48	0.75	3.69
tour de france 2	0.80	2.92	0.53	0.25	1.34	1.38	0.49	1.14	1.96
tour de france 3	0.60	2.56	0.82	0.24	3.14	0.99	0.81	1.25	1.80
tour de france 4	0.63	1.73	1.15	0.55	2.50	1.12	0.74	1.18	1.81
tour de france 5	1.62	2.14	0.78	1.27	2.29	1.86	1.45	1.33	1.72
tour de france 6	0.57	1.38	0.42	1.21	1.30	1.66	0.94	1.24	2.04
tour de france 7	1.46	2.05	1.64	1.78	1.68	1.71	1.76	3.17	2.56
tour de france 8	2.01	1.02	1.30	2.27	1.95	2.29	1.92	7.95	3.04
tour de france 9	1.40	1.84	1.43	2.33	0.98	2.53	0.92	1.22	5.12
tour de france 10	1.36	1.94	3.13	0.23	2.80	1.71	1.40	1.64	1.93
tour de france 11	1.14	1.15	0.78	1.25	1.57	0.70	3.60	1.54	5.08
tour de france 12	0.58	3.28	0.48	1.94	1.94	1.01	2.01	2.10	1.65
tour de france 13	0.58	2.47	2.02	0.78	1.67	2.39	0.57	2.04	3.03
tour de france 14	0.60	5.51	0.74	0.46	2.04	1.53	0.87	1.42	2.47
tour de france 15	0.87	3.43	0.85	0.28	2.24	1.41	1.94	1.13	1.70
tour de france 16	0.81	3.16	1.06	0.27	2.73	1.57	1.68	1.37	1.99
tour de france 17	0.70	2.47	0.74	0.67	1.88	1.46	0.66	0.85	1.80
evacuation 1	0.89	1.80	0.58	0.26	1.33	0.86	0.53	0.60	1.75
evacuation 2	0.63	2.12	0.45	0.25	1.61	1.03	0.59	0.63	1.63
evacuation 3	0.63	2.82	0.46	0.25	1.69	1.29	0.56	0.62	1.64
uniform density	0.66	5.05	0.56	0.28	1.38	5.88	0.61	0.78	2.04
clusters	0.65	8.49	0.53	0.32	1.51	5.93	0.71	0.80	2.04
stationary 25%	0.85	1.28	0.62	0.29	2.50	6.09	0.64	0.73	2.20
stationary 50%	0.75	1.04	0.55	0.29	1.35	6.03	0.63	0.73	2.09
stationary 75%	0.80	0.41	0.55	0.28	2.41	6.26	0.60	0.73	2.08
scaling	1.03	8.42	0.68	0.35	4.34	64.59	0.92	1.15	3.59

Table 3: The average time needed per character to query the structure on the i7-2600K system, in microseconds.

	BD-tree	Grid	k-d tree	k-d tree	k-means	Linear	R-tree	R-tree	Voronoi
			(FLANN)	(nanoflann)		search	(Rebuild)	(Update)	
ao-240-400	4.70	0.17	4.38	3.33	1.96	4.78	17.20	8.97	2.24
ao-300-400	4.23	3.15	3.22	0.17	0.32	0.14	2.40	4.87	3.55
ao-360-400	26.79	18.07	18.18	0.71	0.28	0.21	3.16	11.64	19.44
ao-440-400	3.65	3.03	2.90	0.25	0.52	1.55	0.25	151.36	3.15
ao-500-400	3.17	2.77	0.34	0.24	3.09	0.13	0.23	0.31	5.36
bo-360-050-050	320.29	500.39	463.49	1.24	9.47	466.93	431.57	334.37	499.26
bo-360-075-075	233.28	268.94	504.59	2.58	2.61	3.28	3.26	514.94	2.74
bo-360-090-090	2.18	2.74	3.04	0.63	2.76	3.28	47.71	117.87	6.58
bo-360-120-120	3.30	2.54	2.32	1.52	16.42	3.05	5.14	5.50	14.70
bo-360-160-160	1.05	1.32	0.88	0.25	10.80	1.13	1.59	23.13	10.74
bot-300-050-050	595.63	258.12	442.88	22.54	25.41	3.27	3.18	78.15	2.67
bot-300-065-065	2.32	6.09	3.05	0.48	2.25	3.73	4.21	0.32	23.62
bot-300-075-075	4.79	13.42	10.21	2.84	3.09	4.37	3.37	0.18	8.47
bot-300-085-085	20.77	2.04	2.99	1.95	2.26	3.66	2.94	0.78	29.41
bot-300-100-100	1.27	0.33	3.56	1.97	3.11	4.00	3.52	0.20	5.54
bot-360-050-050	1.36	3.48	2.71	1.93	2.46	3.75	4.55	0.25	4.53
bot-360-075-075	63.98	1.43	2.84	2.75	0.95	2.14	1.76	0.19	1.25
bot-360-090-090	0.97	4.69	11.47	2.13	25.74	5.58	1.92	1.38	4.88
bot-360-120-120v2	3.85	5.18	5.53	2.17	16.27	4.27	3.15	0.95	1.57
bot-360-120-120	90.87	2.30	0.63	1.80	2.56	5.13	1.67	0.37	18.91
bot-360-160-160	15.97	2.96	4.73	1.86	1.31	12.98	2.86	0.18	9.15
bot-360-200-200	1.86	0.80	0.29	0.88	9.87	2.61	0.85	0.14	1.83
bot-360-250-250	5.41	2.64	2.44	0.96	5.14	1.65	1.49	5.70	5.96
mo11_MB	8.17	0.18	1.98	0.20	4.21	0.15	4.18	3.96	3.68
mo12_MB	2.63	0.25	0.93	1.54	0.60	1.20	0.20	2.57	2.13
mo13_MB	0.27	1.24	0.60	0.22	0.69	0.73	1.68	16.77	0.38
mo15_MB	0.43	1.14	7.12	0.57	5.77	0.17	0.14	4.48	2.06
mo16_MB	0.68	0.40	2.62	0.47	0.53	0.28	4.56	5.61	1.12
mo21_MB	4.47	0.24	0.37	0.26	0.54	0.93	0.62	15.14	2.06
mo22_MB	6.71	1.29	0.74	0.73	0.56	0.37	2.32	12.07	4.13
mo24_MB	8.36	0.55	0.30	6.14	2.54	0.26	0.62	2.88	0.61
mu11v2_MB	2.57	2.53	0.20	0.84	1.71	0.14	1.34	8.62	1.22
mu12_MB	3.53	0.24	0.58	0.43	0.49	4.63	71.28	10.48	5.92
mu14_MB	2.37	0.31	0.19	0.45	4.66	0.62	4.55	11.70	0.38
mu15_MB	1.98	0.28	1.28	0.85	5.49	5.47	4.79	4.77	2.07
mu21v2_MB	4.66	17.53	0.35	0.15	3.05	0.16	2.18	30.35	1.16
mu21_MB	0.76	0.37	4.78	0.83	0.55	0.20	0.33	49.66	1.09
mu22_MB	1.15	0.16	0.62	0.50	2.28	1.46	0.51	61.20	1.03
tour de france 1	0.76	15.08	128.10	0.50	1.61	1.70	0.05	4.73	97.17
tour de france 2	8.19	90.11	4.09	0.03	0.65	23.26	0.09	9.81	8.75
tour de france 3	2.16	59.65	25.56	0.01	79.37	1.51	9.81	12.59	4.95
tour de france 4	2.11	3.48	34.30	10.17	56.84	2.45	4.63	10.39	2.45
tour de france 5	20.06	17.39	5.16	19.52	19.61	21.14	20.04	10.78	2.19
tour de france 6	0.04	0.55	0.03	44.88	1.45	16.10	9.23	22.52	8.99
tour de france 7	27.35	29.16	31.98	37.28	17.61	30.75	35.37	188.71	30.45
tour de france 8	41.97	4.76	19.46	52.55	25.76	45.47	40.48	300.84	43.97
tour de france 9	22.64	20.83	35.96	118.88	0.17	79.50	16.12	20.12	170.18
tour de france 10	22.33	42.96	149.06	0.03	117.29	26.37	23.89	17.94	16.00
tour de france 11	13.99	2.05	7.55	26.16	20.80	0.59	172.74	17.44	185.05
tour de france 12	0.05	73.09	1.75	69.20	6.48	1.64	66.04	48.35	0.46
tour de france 13	0.03	27.14	67.15	18.00	0.59	68.98	0.11	78.75	72.05
tour de france 14	0.06	73.67	4.90	4.14	4.01	2.55	10.01	57.59	48.69
tour de france 15	7.93	113.92	18.41	1.37	26.83	8.02	71.30	25.15	1.42
tour de france 16	11.25	10.16	29.50	0.05	46.82	6.57	60.14	32.24	10.15
tour de france 17	2.01	5.88	6.26	10.09	4.36	2.83	2.90	8.90	1.49
evacuation 1	2.87	1.31	0.58	0.49	0.24	0.98	0.39	0.38	0.77
evacuation 2	0.47	0.91	0.32	0.12	0.24	0.26	1.05	0.14	0.26
evacuation 3	0.35	0.86	0.27	0.11	0.36	0.17	0.15	0.04	0.12
uniform density	0.02	0.65	0.03	0.03	0.10	0.06	0.07	0.09	0.06
clusters	0.01	1.81	0.05	0.04	0.16	0.47	0.06	0.07	0.02
stationary 25%	0.44	1.38	0.37	0.03	0.45	0.48	0.28	0.05	0.64
stationary 50%	0.33	0.43	0.03	0.09	0.11	0.23	0.37	0.01	0.08
stationary 75%	0.36	0.17	0.07	0.03	0.11	1.07	0.09	0.01	0.11
scaling	0.42	3.84	0.45	1.08	1.75	63.23	0.42	3.76	1.19

Table 4: The standard deviation of the time needed per character to query the structure on the i7-2600K system, in microseconds.

	BD-tree	Grid	k-d tree (FLANN)	k-d tree (nanoflann)	k-means	Linear search	R-tree (Rebuild)	R-tree (Update)	Voronoi
ao-240-400	1.18	0.04	0.38	0.14	1.84	0.00	1.40	0.88	3.41
ao-300-400	1.17	0.04	0.38	0.14	1.81	0.00	1.40	0.84	3.37
ao-360-400	1.20	0.04	0.36	0.13	1.76	0.00	1.41	0.84	3.36
ao-440-400	1.24	0.04	0.39	0.13	1.74	0.00	1.35	0.80	3.42
ao-500-400	1.18	0.04	0.38	0.14	1.71	0.00	1.32	0.79	3.31
bo-360-050-050	1.14	0.07	0.52	0.17	0.57	0.01	0.31	0.26	2.89
bo-360-075-075	1.05	0.06	0.49	0.18	0.54	0.00	0.26	0.22	3.17
bo-360-090-090	1.23	0.06	0.39	0.13	1.11	0.00	0.52	0.34	3.30
bo-360-120-120	1.23	0.05	0.41	0.13	1.26	0.00	0.70	0.45	3.31
bo-360-160-160	1.23	0.05	0.40	0.13	1.37	0.00	0.82	0.57	3.25
bot-300-050-050	1.06	0.07	0.49	0.17	0.54	0.00	0.32	0.23	3.45
bot-300-065-065	1.12	0.08	0.47	0.17	0.66	0.00	0.39	0.28	3.25
bot-300-075-075	1.06	0.07	0.47	0.16	0.99	0.00	0.42	0.28	3.31
bot-300-085-085	1.06	0.06	0.40	0.14	1.11	0.00	0.49	0.35	3.32
bot-300-100-100	1.16	0.06	0.39	0.13	1.26	0.00	0.61	0.45	3.35
bot-360-050-050	1.08	0.07	0.51	0.17	0.57	0.00	0.29	0.22	2.81
bot-360-075-075	1.08	0.07	0.52	0.17	0.97	0.00	0.39	0.28	3.29
bot-360-090-090	1.13	0.06	0.47	0.19	1.25	0.00	0.50	0.35	3.53
bot-360-120-120v2	1.35	0.06	0.42	0.14	1.30	0.00	0.63	0.45	3.63
bot-360-120-120	1.18	0.05	0.41	0.14	1.32	0.00	0.67	0.49	3.29
bot-360-160-160	1.19	0.05	0.39	0.13	1.41	0.00	0.81	0.61	3.28
bot-360-200-200	1.19	0.05	0.37	0.13	1.54	0.00	0.96	0.77	3.35
bot-360-250-250	1.24	0.05	0.42	0.14	1.58	0.00	0.96	0.80	3.36
mo11_MB	1.18	0.03	0.35	0.13	1.60	0.00	1.13	0.69	3.17
mo12_MB	1.20	0.03	0.36	0.13	1.71	0.00	1.20	0.79	3.18
mo13_MB	1.16	0.04	0.36	0.13	1.39	0.00	0.85	0.67	3.09
mo15_MB	1.19	0.03	0.40	0.15	1.68	0.00	1.14	0.75	3.23
mo16_MB	1.19	0.04	0.50	0.17	1.78	0.00	1.10	0.75	3.31
mo21_MB	1.10	0.04	0.39	0.13	1.10	0.00	0.75	0.48	2.83
mo22_MB	1.12	0.04	0.49	0.17	1.26	0.00	0.59	0.38	3.21
mo24_MB	1.16	0.03	0.35	0.12	1.42	0.00	0.88	0.64	3.16
mu11v2_MB	1.19	0.03	0.35	0.13	1.70	0.00	1.26	0.70	3.23
mu12_MB	1.20	0.04	0.40	0.14	1.52	0.00	1.03	0.61	3.16
mu14_MB	1.18	0.04	0.40	0.14	1.65	0.00	1.19	0.75	3.32
mu15_MB	1.18	0.04	0.44	0.16	1.69	0.00	1.22	0.67	3.32
mu21v2_MB	1.18	0.03	0.36	0.12	1.50	0.00	1.04	0.65	3.19
mu21_MB	1.17	0.04	0.38	0.13	1.42	0.00	0.95	0.63	3.24
mu22_MB	1.15	0.04	0.40	0.13	1.22	0.00	0.77	0.49	3.13
tour de france 1	1.29	0.02	0.33	0.15	3.05	0.00	2.00	1.23	2.71
tour de france 2	1.21	0.02	0.33	0.15	3.12	0.00	2.00	1.16	2.63
tour de france 3	1.17	0.02	0.33	0.15	3.17	0.00	1.99	1.16	2.58
tour de france 4	1.14	0.02	0.34	0.15	3.22	0.00	1.98	1.13	2.59
tour de france 5	1.21	0.02	0.32	0.15	3.04	0.00	1.98	1.12	2.74
tour de france 6	1.21	0.02	0.32	0.15	3.03	0.00	1.98	1.16	2.70
tour de france 7	1.36	0.02	0.35	0.14	2.86	0.00	2.05	1.11	3.03
tour de france 8	1.33	0.02	0.33	0.14	2.82	0.00	1.97	1.07	3.15
tour de france 9	1.38	0.02	0.37	0.15	2.94	0.00	2.05	1.23	2.87
tour de france 10	1.31	0.02	0.33	0.15	2.88	0.00	2.07	1.16	2.93
tour de france 11	1.32	0.02	0.33	0.15	2.96	0.00	2.02	1.23	2.81
tour de france 12	1.15	0.02	0.33	0.16	3.15	0.00	2.03	1.14	2.79
tour de france 13	1.14	0.02	0.33	0.16	3.16	0.00	2.04	1.15	2.77
tour de france 14	1.04	0.02	0.31	0.15	3.31	0.00	2.08	1.10	2.63
tour de france 15	1.07	0.02	0.33	0.16	3.27	0.00	2.05	1.09	2.64
tour de france 16	1.02	0.02	0.30	0.16	3.31	0.00	2.05	1.08	2.62
tour de france 17	1.05	0.02	0.32	0.16	3.30	0.00	2.04	1.06	2.61
evacuation 1	1.21	0.02	0.30	0.14	3.10	0.00	2.14	1.07	3.34
evacuation 2	1.10	0.02	0.31	0.14	3.23	0.00	2.04	1.18	3.03
evacuation 3	0.98	0.02	0.31	0.15	3.30	0.00	2.03	1.14	2.86
uniform density	0.89	0.02	0.26	0.15	4.41	0.00	2.20	1.27	2.71
clusters	0.88	0.02	0.25	0.15	4.36	0.00	2.22	1.32	2.81
stationary 25%	0.98	0.02	0.27	0.16	4.34	0.00	2.18	0.94	2.78
stationary 50%	0.96	0.02	0.26	0.15	4.31	0.00	2.18	0.64	2.71
stationary 75%	0.94	0.02	0.26	0.15	4.29	0.00	2.18	0.34	2.73
scaling	1.04	0.02	0.30	0.21	5.32	0.00	2.38	1.49	3.09

Table 5: The average time needed per character to update the structure on the i5-4210M system, in microseconds.

	BD-tree	Grid	k-d tree (FLANN)	k-d tree (nanoflann)	k-means	Linear search	R-tree (Rebuild)	R-tree (Update)	Voronoi
ao-240-400	0.23	0.01	0.26	0.09	0.76	0.00	0.71	0.39	0.53
ao-300-400	0.10	0.01	0.24	0.08	0.74	0.00	0.71	0.38	0.22
ao-360-400	0.60	0.01	0.17	0.06	0.77	0.00	0.85	0.38	0.64
ao-440-400	0.67	0.01	0.14	0.05	0.72	0.00	0.71	0.57	0.21
ao-500-400	0.27	0.01	0.21	0.08	0.85	0.00	0.84	0.39	0.27
bo-360-050-050	2.14	0.10	0.93	0.17	0.76	0.01	0.21	0.18	1.00
bo-360-075-075	0.17	0.03	0.41	0.18	0.55	0.01	0.15	0.17	0.60
bo-360-090-090	3.37	0.03	0.26	0.09	0.43	0.00	0.20	0.12	1.58
bo-360-120-120	0.20	0.03	0.25	0.10	0.44	0.00	0.26	0.18	2.56
bo-360-160-160	0.11	0.02	0.24	0.08	0.46	0.00	0.31	0.63	1.23
bot-300-050-050	0.13	0.04	0.35	0.13	0.47	0.01	0.16	0.17	3.74
bot-300-065-065	1.83	0.38	0.39	0.14	0.58	0.01	0.17	0.14	0.57
bot-300-075-075	0.13	0.04	0.41	0.15	0.61	0.01	0.20	0.16	0.58
bot-300-085-085	0.12	0.04	0.30	0.11	0.46	0.01	0.19	0.15	0.49
bot-300-100-100	0.16	0.03	0.26	0.10	0.65	0.00	0.21	1.11	0.52
bot-360-050-050	1.02	0.06	0.84	0.15	1.05	0.01	0.22	0.15	0.95
bot-360-075-075	0.10	0.03	1.09	0.33	3.32	0.01	0.16	0.15	1.33
bot-360-090-090	0.13	0.03	0.48	1.07	0.60	0.01	0.18	0.16	4.59
bot-360-120-120v2	1.31	0.03	0.36	0.11	0.54	0.01	0.22	0.17	9.06
bot-360-120-120	0.31	0.03	0.34	0.13	0.50	0.01	0.25	0.22	0.49
bot-360-160-160	0.14	0.03	0.27	0.10	0.51	0.00	0.69	0.30	0.54
bot-360-200-200	0.11	0.03	0.25	0.09	0.55	0.00	0.76	0.34	1.29
bot-360-250-250	0.12	0.02	0.31	0.10	0.55	0.01	0.38	0.63	2.92
mo11_MB	0.11	0.01	0.12	0.05	0.60	0.00	0.55	0.29	0.15
mo12_MB	0.10	0.01	0.15	0.07	0.66	0.00	0.54	0.33	0.19
mo13_MB	0.12	0.01	0.16	0.16	0.51	0.00	0.55	0.35	0.20
mo15_MB	0.16	0.01	0.34	0.25	0.68	0.01	0.56	0.35	0.42
mo16_MB	0.15	0.02	0.57	0.18	0.84	0.01	0.54	0.39	0.83
mo21_MB	0.15	0.01	0.14	0.05	0.71	0.00	4.80	0.38	0.45
mo22_MB	0.14	0.02	0.48	0.18	0.64	0.01	0.27	0.20	0.76
mo24_MB	0.25	0.01	0.17	0.06	0.54	0.00	0.45	0.28	0.21
mu11v2_MB	0.26	0.01	0.18	0.06	0.74	0.00	0.59	0.29	0.23
mu12_MB	0.13	0.01	0.23	0.09	0.65	0.00	0.54	0.29	0.27
mu14_MB	0.13	0.01	0.33	0.11	0.73	0.01	0.71	0.39	0.39
mu15_MB	0.11	0.01	0.45	0.27	0.77	0.01	0.68	0.33	0.41
mu21v2_MB	0.35	0.01	0.17	0.07	0.66	0.00	0.49	0.27	0.42
mu21_MB	0.20	0.01	0.24	0.08	0.58	0.00	0.49	0.37	0.46
mu22_MB	0.22	0.01	0.23	0.08	0.67	0.00	0.48	0.30	0.68
tour de france 1	0.75	0.00	0.03	0.21	0.57	0.00	0.29	0.18	0.57
tour de france 2	0.08	0.00	0.02	0.01	0.42	0.00	0.27	0.16	0.25
tour de france 3	0.47	0.00	0.02	0.04	0.41	0.00	0.26	0.15	0.32
tour de france 4	0.12	0.00	0.25	0.01	0.46	0.00	0.24	0.24	0.41
tour de france 5	0.09	0.00	0.03	0.01	0.51	0.00	0.47	0.18	0.79
tour de france 6	0.12	0.00	0.02	0.01	0.51	0.00	0.35	0.41	0.28
tour de france 7	0.05	0.00	0.05	0.02	0.53	0.00	0.42	0.21	0.15
tour de france 8	0.14	0.00	0.07	0.05	0.78	0.00	0.44	0.21	0.21
tour de france 9	0.57	0.00	0.04	0.02	0.50	0.00	0.36	0.22	0.97
tour de france 10	0.05	0.00	0.06	0.05	0.50	0.00	0.39	0.22	0.17
tour de france 11	0.08	0.01	0.05	0.28	0.56	0.00	0.33	0.21	0.21
tour de france 12	0.15	0.00	0.03	0.01	0.53	0.00	0.37	0.19	0.29
tour de france 13	0.16	0.00	0.02	0.02	0.66	0.00	0.34	0.17	0.30
tour de france 14	0.28	0.00	0.06	0.01	0.41	0.00	0.27	0.14	0.30
tour de france 15	0.16	0.00	0.11	0.02	0.42	0.00	0.25	0.13	0.28
tour de france 16	0.16	0.00	0.04	0.01	0.41	0.00	0.37	0.13	0.28
tour de france 17	0.17	0.00	0.03	0.17	0.39	0.00	0.24	0.13	0.28
evacuation 1	0.04	0.00	0.01	0.04	0.14	0.00	0.09	0.12	0.33
evacuation 2	0.07	0.00	0.01	0.00	0.12	0.00	0.07	0.07	0.29
evacuation 3	0.03	0.00	0.01	0.02	0.11	0.00	0.08	0.08	0.26
uniform density	0.01	0.00	0.01	0.00	0.07	0.00	0.04	0.07	0.06
clusters	0.01	0.00	0.01	0.01	0.06	0.00	0.04	0.11	0.06
stationary 25%	0.01	0.00	0.01	0.00	0.07	0.00	0.03	0.03	0.02
stationary 50%	0.01	0.00	0.00	0.00	0.07	0.00	0.04	0.02	0.02
stationary 75%	0.01	0.00	0.00	0.00	0.07	0.00	0.05	0.01	0.06
scaling	0.07	0.00	0.03	0.04	0.61	0.00	0.16	0.11	0.24

Table 6: The standard deviation of the time needed per character to update the structure on the i5-4210M system, in microseconds.

	BD-tree	Grid	k-d tree (FLANN)	k-d tree (nanoflann)	k-means	Linear search	R-tree (Rebuild)	R-tree (Update)	Voronoi
ao-240-400	1.03	0.29	1.37	0.43	1.88	0.51	0.75	0.97	2.74
ao-300-400	1.01	0.44	1.40	1.96	1.85	0.52	0.75	0.93	2.75
ao-360-400	1.32	0.32	1.40	0.46	1.91	0.52	0.96	0.92	2.83
ao-440-400	1.08	0.33	1.42	0.54	1.82	0.51	0.80	0.94	2.78
ao-500-400	0.95	0.31	1.38	0.51	1.80	0.50	0.92	1.11	2.70
bo-360-050-050	13.17	0.53	1.60	0.57	1.77	0.55	0.96	1.04	3.22
bo-360-075-075	1.63	0.51	1.85	0.96	1.48	0.54	1.22	3.81	2.10
bo-360-090-090	1.81	0.47	1.34	0.51	1.94	0.47	0.70	0.92	2.61
bo-360-120-120	1.81	0.35	1.33	0.43	1.90	0.44	0.96	0.90	2.66
bo-360-160-160	0.91	0.35	1.40	0.45	1.93	0.57	0.91	0.93	2.80
bot-300-050-050	1.18	0.47	1.73	0.50	2.69	0.43	0.91	1.38	2.04
bot-300-065-065	0.92	1.53	1.87	0.63	1.60	0.44	1.33	1.04	2.28
bot-300-075-075	2.46	0.91	1.44	0.66	2.35	0.76	0.79	0.99	2.58
bot-300-085-085	0.91	0.44	1.34	0.52	1.88	0.47	0.88	0.93	3.24
bot-300-100-100	7.34	0.36	1.39	0.48	1.89	0.44	0.76	0.92	2.64
bot-360-050-050	12.23	1.17	1.63	0.54	1.50	1.00	0.76	1.10	2.07
bot-360-075-075	0.95	0.50	1.48	3.57	1.76	0.47	0.92	1.02	2.86
bot-360-090-090	2.74	0.43	1.39	0.53	1.96	0.48	0.72	0.97	2.59
bot-360-120-120v2	5.46	0.36	1.39	0.45	1.93	0.47	1.07	0.93	2.79
bot-360-120-120	36.67	0.81	1.38	0.58	2.58	0.51	0.82	1.10	3.04
bot-360-160-160	3.00	0.31	1.51	0.51	2.17	0.48	0.73	0.88	2.92
bot-360-200-200	2.66	0.33	1.58	0.46	2.00	0.49	0.88	0.91	2.99
bot-360-250-250	0.95	0.36	1.42	0.47	1.98	0.50	0.85	0.92	2.83
mo11_MB	0.90	0.28	1.30	0.46	1.85	0.46	0.74	0.86	2.83
mo12_MB	0.89	0.28	1.29	0.40	1.83	0.46	0.70	0.85	2.80
mo13_MB	0.92	0.32	1.63	0.43	1.98	0.47	0.72	0.83	2.75
mo15_MB	0.94	0.32	1.40	0.44	2.02	0.54	0.74	0.96	2.75
mo16_MB	1.04	0.44	1.55	0.50	2.03	0.55	0.83	0.95	2.77
mo21_MB	0.91	0.43	1.35	0.56	1.73	0.43	0.76	0.88	2.35
mo22_MB	0.91	0.44	1.45	0.47	1.91	0.46	0.77	0.91	2.61
mo24_MB	0.86	0.27	1.32	0.51	1.84	0.43	0.72	0.85	2.76
mu11v2_MB	0.94	0.31	1.33	0.40	1.85	0.51	0.73	1.00	2.88
mu12_MB	0.92	0.31	1.31	0.41	1.82	0.46	0.71	0.91	2.64
mu14_MB	1.49	0.36	1.37	0.43	1.86	0.55	0.79	0.88	2.72
mu15_MB	0.96	0.36	1.47	0.47	1.93	0.57	0.89	0.92	2.70
mu21v2_MB	0.93	0.30	1.31	0.39	1.89	0.45	0.74	0.84	2.73
mu21_MB	1.02	0.31	1.33	0.40	1.83	0.44	0.70	0.86	2.76
mu22_MB	0.84	0.36	1.28	0.40	1.71	0.41	0.68	0.86	2.48
tour de france 1	1.14	1.80	1.44	0.41	2.83	1.34	0.79	1.02	3.31
tour de france 2	1.10	2.23	1.40	0.44	2.90	1.48	0.81	1.04	3.32
tour de france 3	1.10	2.40	1.40	0.43	3.09	1.64	0.91	1.07	3.40
tour de france 4	1.12	2.75	1.44	0.45	3.25	1.73	0.88	1.08	3.68
tour de france 5	1.04	2.07	1.37	0.43	2.81	1.40	0.80	1.03	3.31
tour de france 6	1.04	2.34	1.37	0.46	2.78	1.39	0.80	1.02	3.27
tour de france 7	1.01	1.23	1.33	0.40	2.15	0.92	0.75	0.95	3.14
tour de france 8	0.95	1.06	1.35	0.40	2.10	0.87	0.74	0.92	3.16
tour de france 9	1.03	1.58	1.37	0.42	2.34	1.13	0.80	0.96	3.19
tour de france 10	0.96	1.38	1.34	0.41	2.21	1.01	0.76	0.95	3.14
tour de france 11	1.06	1.97	1.38	0.41	2.48	1.11	0.78	0.99	3.51
tour de france 12	1.08	3.07	1.43	0.47	3.51	1.81	0.97	1.09	3.41
tour de france 13	1.11	2.99	1.42	0.44	3.50	1.84	0.92	1.11	3.40
tour de france 14	1.15	9.35	1.49	0.46	3.87	2.70	0.92	1.14	3.61
tour de france 15	1.10	3.52	1.43	0.45	3.51	2.27	0.87	1.11	3.54
tour de france 16	1.14	4.97	1.46	0.46	3.71	2.64	0.87	1.13	3.59
tour de france 17	1.13	3.94	1.44	0.46	3.56	2.55	0.88	1.10	3.61
evacuation 1	0.97	2.76	1.34	0.42	2.90	1.33	0.82	0.97	3.26
evacuation 2	1.01	3.99	1.36	0.42	3.39	1.87	0.85	1.05	3.31
evacuation 3	1.01	5.45	1.37	0.43	3.54	2.23	0.86	1.08	3.44
uniform density	1.37	11.25	1.69	0.50	4.45	20.83	0.98	1.38	4.88
clusters	1.38	20.16	1.65	0.51	4.87	20.87	1.08	1.46	5.17
stationary 25%	1.48	1.94	1.70	0.51	4.76	20.84	0.95	1.33	5.22
stationary 50%	1.36	1.87	1.61	0.49	4.41	20.79	0.96	1.33	4.77
stationary 75%	1.39	1.76	1.61	0.49	4.36	20.80	0.96	1.34	4.76
scaling	2.42	19.63	2.09	0.67	7.10	220.75	1.07	1.60	6.72

Table 7: The average time needed per character to query the structure on the i5-4210M system, in microseconds.

	BD-tree	Grid	k-d tree	k-d tree	k-means	Linear	R-tree	R-tree	Voronoi
			(FLANN)	(nanoflann)		search	(Rebuild)	(Update)	
ao-240-400	1.96	0.43	0.75	0.41	0.69	0.43	0.49	1.36	0.79
ao-300-400	1.67	4.01	1.13	4.65	0.79	0.58	0.60	0.60	1.03
ao-360-400	6.75	0.91	1.10	0.92	1.61	0.78	2.12	0.68	1.58
ao-440-400	2.09	0.76	1.60	1.58	1.06	0.86	1.05	1.08	1.55
ao-500-400	0.90	0.48	0.80	1.10	0.77	0.51	1.90	1.98	1.21
bo-360-050-050	230.93	2.72	4.55	3.07	7.34	3.73	5.39	4.86	13.35
bo-360-075-075	19.12	2.55	14.32	11.08	2.92	4.47	7.68	71.66	3.90
bo-360-090-090	18.98	3.39	2.10	2.11	3.12	2.29	1.72	3.28	4.28
bo-360-120-120	31.25	1.13	1.83	0.99	2.37	1.21	3.46	2.37	2.08
bo-360-160-160	1.82	1.06	2.02	1.11	2.36	2.52	3.03	2.19	4.58
bot-300-050-050	7.31	2.24	8.13	2.37	20.58	2.17	4.79	8.73	0.52
bot-300-065-065	3.08	26.05	7.78	3.58	2.54	1.94	9.73	3.41	3.29
bot-300-075-075	45.51	8.07	2.28	3.69	7.60	7.83	2.99	2.81	6.38
bot-300-085-085	2.09	1.92	1.73	2.00	2.45	1.88	4.04	2.07	7.74
bot-300-100-100	149.88	1.21	2.78	1.41	1.71	1.16	1.94	2.45	2.06
bot-360-050-050	197.75	12.61	5.54	2.66	2.77	10.10	2.26	4.51	3.41
bot-360-075-075	2.87	2.07	4.75	91.32	2.48	2.14	4.89	2.97	9.47
bot-360-090-090	45.66	1.13	1.37	1.96	3.07	2.28	1.59	2.22	3.37
bot-360-120-120v2	101.02	1.05	1.71	1.35	2.39	1.92	4.49	1.61	4.99
bot-360-120-120	272.97	4.64	1.70	2.88	7.61	1.72	2.19	3.69	5.67
bot-360-160-160	71.38	0.63	3.45	2.07	4.90	1.12	0.89	1.75	4.79
bot-360-200-200	49.93	1.43	2.93	1.02	2.56	0.93	2.68	0.98	4.24
bot-360-250-250	1.50	0.99	1.74	0.90	2.23	0.93	1.80	1.22	3.19
mo11_MB	0.53	0.37	0.35	1.24	0.51	0.35	1.16	0.52	0.66
mo12_MB	0.51	0.49	0.44	0.60	0.41	0.30	0.31	0.45	0.77
mo13_MB	1.06	1.10	3.24	0.68	2.75	1.19	0.94	0.62	1.59
mo15_MB	0.71	0.47	0.99	0.53	2.11	1.00	0.52	1.18	1.06
mo16_MB	1.81	0.84	1.48	0.64	1.33	0.44	0.86	0.52	1.48
mo21_MB	2.06	1.04	0.71	1.74	0.77	1.20	1.91	2.97	0.72
mo22_MB	0.91	0.95	1.59	0.71	1.26	0.65	1.66	1.66	1.61
mo24_MB	0.45	0.37	1.14	5.71	0.76	0.49	1.05	1.17	1.38
mu11v2_MB	1.30	0.85	0.81	0.24	0.67	1.06	0.56	1.82	1.69
mu12_MB	1.43	0.45	0.64	0.45	0.83	0.43	0.53	1.58	0.83
mu14_MB	5.16	0.87	0.95	0.66	0.76	1.15	1.28	0.51	1.40
mu15_MB	1.00	0.62	1.35	0.66	1.00	1.15	1.67	0.70	1.41
mu21v2_MB	1.39	0.73	0.66	0.44	1.31	0.64	1.14	1.02	1.44
mu21_MB	4.16	0.41	0.88	0.39	0.61	0.51	0.47	1.23	0.82
mu22_MB	1.03	0.95	0.69	0.55	1.01	0.52	0.64	1.51	1.14
tour de france 1	0.83	0.54	2.02	0.17	2.80	0.53	0.64	0.44	0.63
tour de france 2	0.53	0.72	0.54	0.92	0.70	0.49	0.77	0.45	0.72
tour de france 3	0.44	0.80	0.56	0.27	2.11	0.84	1.54	0.59	0.91
tour de france 4	0.66	1.01	0.94	0.50	2.16	0.88	1.14	0.75	4.07
tour de france 5	0.89	0.91	0.47	0.57	0.92	0.71	0.42	0.69	0.92
tour de france 6	0.30	1.06	0.73	0.93	0.90	0.68	0.44	0.35	0.85
tour de france 7	2.42	0.77	0.84	0.45	1.61	0.44	0.68	0.82	1.30
tour de france 8	0.63	0.97	1.48	0.47	2.58	0.62	1.01	0.46	3.42
tour de france 9	1.12	0.80	1.14	0.60	1.04	1.38	1.07	0.59	0.73
tour de france 10	0.16	0.56	0.60	0.28	0.84	0.63	0.46	0.73	0.88
tour de france 11	0.93	0.74	0.21	0.06	0.67	0.43	0.49	0.81	0.85
tour de france 12	0.61	1.48	0.82	0.81	1.43	0.94	1.45	0.48	1.26
tour de france 13	1.59	1.43	0.92	0.44	1.20	0.93	0.82	1.32	1.33
tour de france 14	0.33	17.99	1.19	0.40	1.08	1.48	0.85	0.39	1.00
tour de france 15	0.14	1.21	0.53	0.26	1.27	0.87	0.74	0.54	1.22
tour de france 16	0.82	2.34	0.66	0.34	1.00	1.50	0.24	0.50	0.83
tour de france 17	0.47	1.20	0.48	0.61	1.31	1.36	1.04	0.38	1.77
evacuation 1	0.23	0.86	0.23	0.15	0.41	0.19	0.21	0.11	0.36
evacuation 2	0.16	1.28	0.14	0.12	0.44	0.11	0.14	0.15	0.26
evacuation 3	0.11	1.92	0.14	0.10	0.43	0.08	0.12	0.13	0.22
uniform density	0.06	1.51	0.24	0.03	0.11	0.14	0.05	0.07	0.11
clusters	0.03	3.36	0.03	0.03	0.19	0.10	0.08	0.15	0.08
stationary 25%	0.03	0.47	0.04	0.03	0.10	0.09	0.03	0.04	0.05
stationary 50%	0.04	0.12	0.05	0.02	0.08	0.14	0.05	0.03	0.10
stationary 75%	0.05	0.07	0.05	0.03	0.05	0.14	0.05	0.03	0.10
scaling	0.68	9.69	0.34	0.16	1.98	157.39	0.14	0.24	1.31

Table 8: The standard deviation of the time needed per character to query the structure on the i5-4210M system, in microseconds.