DEGREE PROJECT IN TECHNOLOGY,
FIRST CYCLE, 15 CREDITS
STOCKHOLM, SWEDEN 2020

Analyzing the scalability of R*-tree regarding the neuron touch detection task

ANTON BRASK

FILIP BERENDT

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Analyzing the scalability of R*-tree regarding the neuron touch detection task

ANTON BRASK
FILIP BERENDT

Degree Project in Computer Science
Date: June 8, 2020
Supervisor: Alexander Kozlov
Examiner: Pawel Herman
School of Electrical Engineering and Computer Science
Swedish title: Analysering av R*-träds skalbarhet i samband med sökning av neuronkontakt

Abstract

A common task within research of neuronal morphology is neuron touch detection, that is finding the points in space where two neurites approach each other to form a synapse. In order to make efficient use of cache memory, it is important to store points that are close in space close in memory. One data structure that aims to tackle this complication is the R*-tree. In this thesis, a spatial query for touch detection was implemented and the scalability of the R*-tree was estimated on realistic neuron densities and extrapolated to explore execution times on larger volumes. It was found that touch detection on this data structure scaled much like the optimal algorithm in 3D-space and more specifically that the computing power needed to analyze a meaningful portion of the human cortex is not readily available.

Sammanfattning

En vanlig uppgift inom forskning av neuronal morfologi är att hitta var mellan olika neuroner synapser bildas. För att använda cache-minne effektivt är det viktigt att lagra punkter som är nära i rymden nära i minnet. En datastruktur som ämnar att lösa detta är R*-trädet. I denna rapport så implementerades en sökning av rummet och skalbarheten för R*-trädet uppskattades på realistiska neurondensiteter för att sedan extrapoleras och utforska körtider på större volymer. Det konstaterades att denna datastruktur skalade sig mycket som den optimala algoritmen i tredimensionell rymd och mer specifikt att datorkraften som behövs för att analysera en meningsfull del av människans hjärnbark inte är fritt tillgänglig.

Contents

Chapter 1

Introduction

Neurons have a complex shape determined by three different kinds of components: soma, axons, and dendrites. The first mentioned is the cell body, while the latter are both neurites. When put together, the morphological structure of the neuron can be described as a tree; each node has only one parent (with the exception for the root node) and usually one or two children[1]. This topological model of the neuron morphology allows efficient digitalization of the brain data and high-performance processing. A specific kind of topic within neuronal research is mapping of the connectome (i.e. finding where between two neurons synapses are located) of a network of many neurons. Being able to map the connectome efficiently would help researchers study the general rules for high-accuracy generation of arbitrary networks of neurons similar to actual networks found in organisms in nature. Since high-accuracy constructions cannot consistently be generated, this is a subject for ongoing intensive research.[3][4]

1.1 Research Question

The tree structure mentioned is great for making neuron-specific queries, as the nodes we are accessing are often close in memory, leading to efficient use of the cache memory. An obstacle arises when making spatial queries, as it often includes parts of different neurons, which have no guarantee to be closely located in memory. This risk opens up the possibility for substantially more

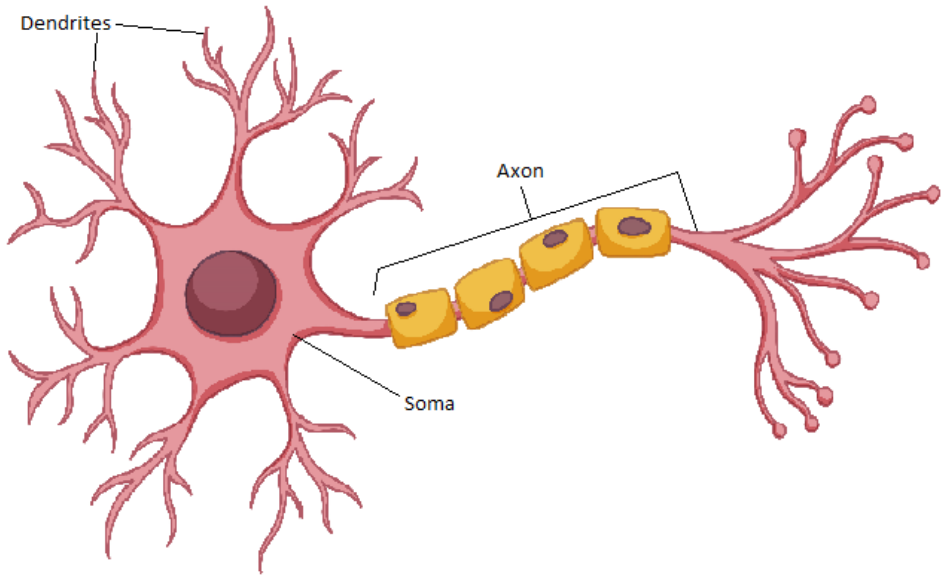

Figure 1.1: A sketch, not up to scale, among many others of how a neuron might look like.[2]

cache misses and page faults, which will affect the execution time negatively. For this topic, we are specifically interested in finding possible synapse connections between neurons in a given space. In order to minimize the amount of cache misses, an efficient data structure for spatial queries has to be used. Different data structures scale differently with the volume of the search space and the density of neurons. One previously researched data structure is k-d tree, and it was found that the limiting factor was the memory usage of the RAM[5]. A realistic query size would require a large amount of RAM commonly found in heavy duty machines, e.g. computers specifically made for scientific calculations, and not in a regular personal computer. In another study, different data structures were compared in regards to query execution time. Among those compared, they concluded that R*-tree was superior.[6] Combining the idea of the first mentioned study, with the best data structure of the second mentioned study, our problem is formulated:

How does the R*-tree data structure scale, when used for an algorithm for touch detection in neuronal morphometrics?

Also, since an aim in this area of research is to be able to analyse a realistic volume of tissue, it is interesting to speculate how such a query would look

like. This formulates an extra question:

How would a realistic problem size perform, given the scope of this study?

1.2 Scope and Limitations

It is possible to analyse the R*-tree data structure and how it behaves with different kinds of data sets and spatial queries. This thesis will focus specifically on studying how the R*-tree scales with a fixed amount of neurons and with a specific spatial query. No other data structure than the R*-tree will be tested.

Naturally, both time and computing resources put limitations on the simulations made in this study. Thus, when analyzing realistic neuron densities, all volumes are kept relatively small.

The benchmarks will be executed purely sequentially. Any possibilities for parallelism will not be considered at all for the methods.

1.3 Thesis Outline

The background presents the relevant data structures and algorithms as well as the libraries used in this thesis. The method chapter explains in detail how all measurements were made and the motivations for choices made. The measurements are then presented in the results chapter with corresponding commentary. Subsequently, the results are discussed in the discussion chapter and a conclusion is presented.

Chapter 2

Background

2.1 Neurons and Touch Detection

As mentioned in the introduction, neurons form a vast and complicated network by connecting to each other through synapses. From previous studies, it is known that synapses typically form at an average distance of 5 microns[7][8]. Taking this into consideration, one can find different points for a synapse to potentially form in between two neurites of different neurons, when analyzing a digital representation of a neuronal network (figure 2.1). This specific query is what is called 'touch detection'. It is possible for autaptic synapses to form. These are synapses that form between a single neuron's different neurites. Autaptic are considered to be a quite rare occurance.[9] Because of this rarity, and since autaptic synapses aren't problematic with regards to memory management, a simplification made in this study is that synapses only form between different neurons.

2.1.1 Brain Cortex

The human neocortex accounts for nearly 78% of the total brain volume of approximately 1200 cm^3 and has been estimated to have a neuron density of 24186 neurons/mm^3 when observing all cortical layers. This neuron density is much lower than the estimated neuron density of the mouse cortex (120315 neurons/mm^3) but on the other hand the total volume of the mouse brain is

much smaller.[10] Differences in neuron density and actual size of the brain presents two different numerically difficult touch detection tasks.

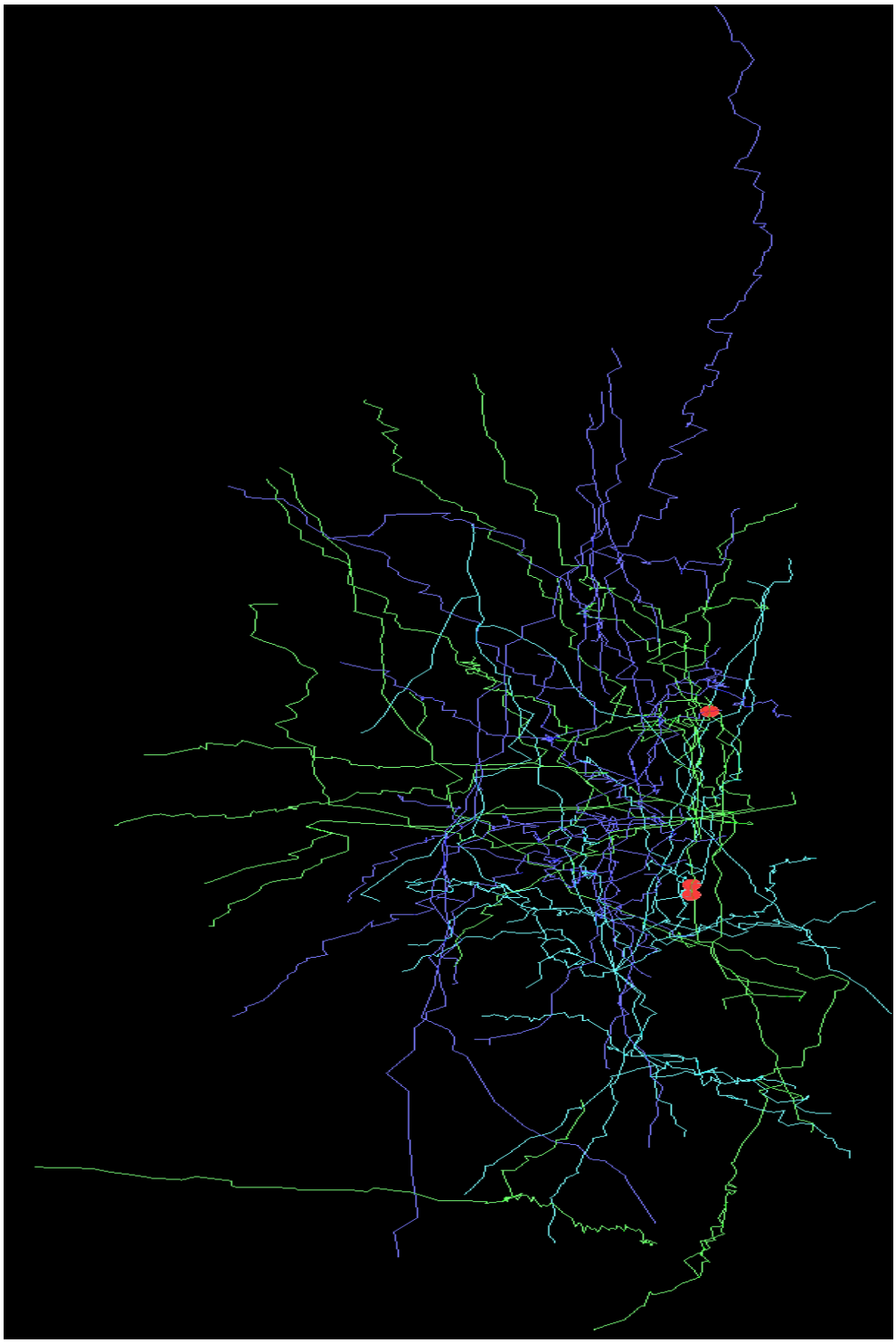

Figure 2.1: Three neurons with three 'touches' (red circles) detected, with a maximum distance of 1 micron (C010398B-P3, C170898A-P3, C031097B-P3 found at neuromorpho.org)

2.2 Data Structures

There are plenty of different data structures for storing spatial data and the common theme is to make data points that are close in space close in memory. The data structure that is analysed in this thesis is R*-tree which is a variant of the R-tree. There are also data structures such as e.g KD-tree, MVR-tree and TPR-tree.[6][5]

2.2.1 KD-tree

The KD-tree is a binary tree composed of nodes containing a k-dimensional point, and two pointers to its child nodes. Naturally, any node not having any pointers to children initialized are considered leaf nodes. Every non-leaf node references a k-dimensional space, which is split across a hyperplane perpendicular to a single dimensional axis, generating two distinct 'half-spaces'. Points in the right half-space are represented solely by the right subtree, while points in the left half-space are represented by the left subtree. The direction which defines the splitting hyperplane is determined by the depth of the node which the hyperplane splits. The following regards a three-dimensional tree. The root node, which has a depth of 0, could be split by the plane perpendicular to the x-axis. Then its children, which have a depth of 1, could be split by the plane perpendicular to the y-axis. Finally, their children, which has a depth of 2, could be split by the plane perpendicular to the z-axis. Since the tree is three-dimensional, this pattern is repeated in a modular cycle for any deeper nodes.[11]

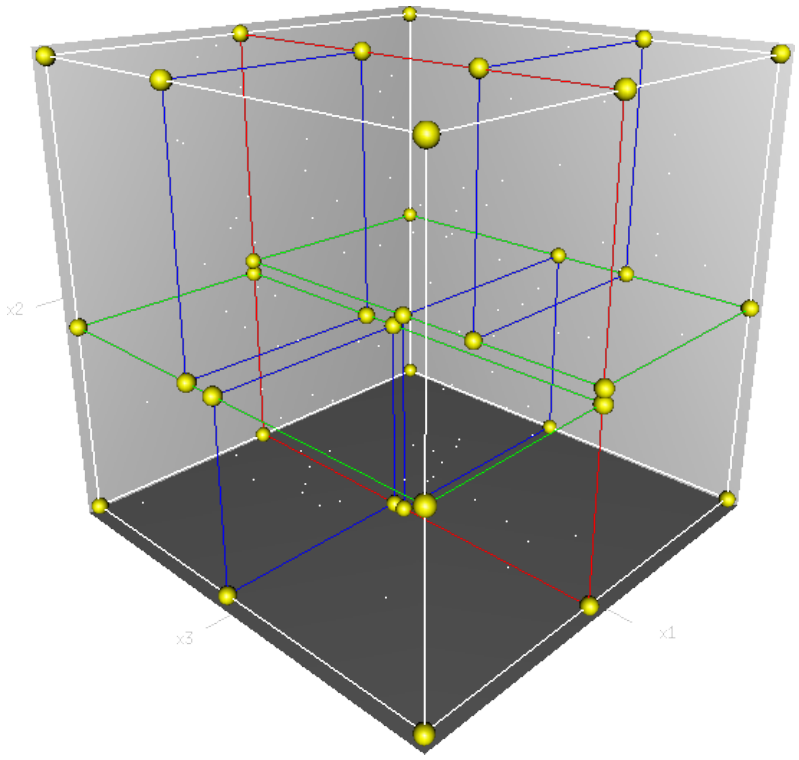

Figure 2.2: A 3-dimensional KD-tree illustrated by boxes split with regards to rotational dimensions.[12]

2.2.2 R-tree

The R-tree is a tree based data structure that indexes spatial data points with tree nodes consisting of minimum bounding boxes that encapsulates a part of space. These boxes reference either smaller minimum bounding boxes or individual data points within the boxes. Data points that are close to each other in space are referenced by a minimum bounding box in the next level of the tree. This spatial tree hierarchy makes it possible to exploit the spatial locality of the data and store data points close in memory. In order to search the tree one has to only consider the boxes which intersect the desired search volume and can thus exclude many branches of the tree, consequently avoiding whole pages of memory. When processing nodes close in space less page faults and cache misses will be generated, resulting in efficient use of memory.

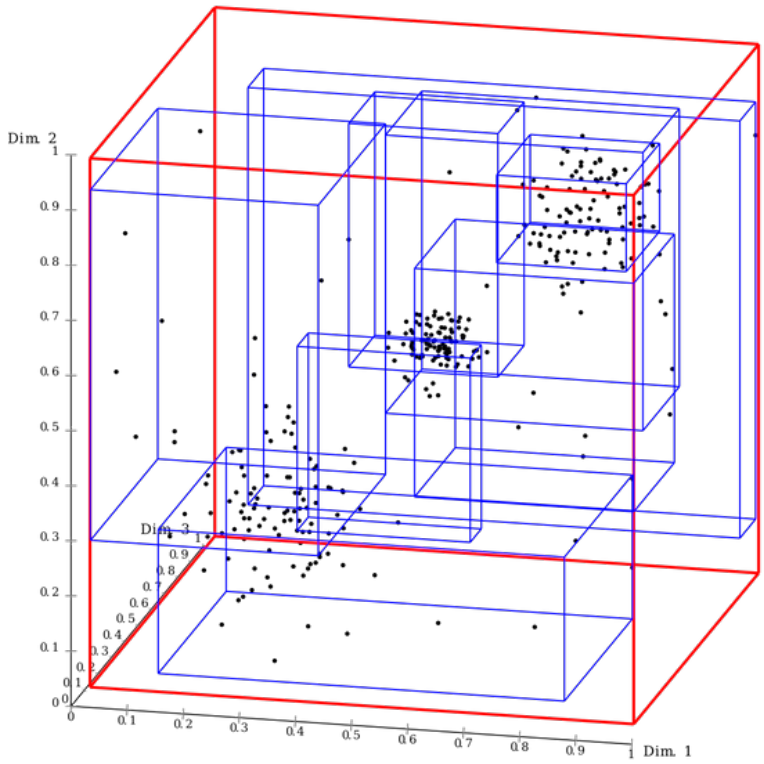

Figure 2.3: A 3-dimensional R-tree illustrated by bounding boxes.[13]

A great flaw with this data structure is the construction of the tree. There is no guarantee that any two minimum bounding boxes won't overlap and thus a decision on which box to put a point in has to be made. When inserting and choosing between overlapping boxes the box which would need to be extended the least is always favoured. On node overflow, that is when the amount of points referenced by a box overflows, a split of the box into two boxes has to be made. This split is usually chosen by one of two heuristics; quadratic split or linear split. This flaw makes the quality of the construction sensitive to the order of the data points inserted into the structure. A bad construction results in slower queries, which is highly undesired when making many complex queries on the same structure.[14]

2.2.3 R*-tree

The R*-tree is a variant of the R-tree and differs only when constructing the tree. In order to construct a fast performing R-tree, the amount of overlap and the resulting size of the boxes has to be minimized. Less overlap between boxes will result in less branches having to be searched when querying the tree. The R*-tree aims to solve these issues by using improved heuristics to handle the splits on node overflow and a reinsertion strategy that tries to reinsert some of the points contained in the overflowing box. This reinsertion strategy minimizes the dependency of the order of the data being inserted during the construction of the tree, resulting in an almost optimally constructed tree. While this strategy will likely result in many overheads during construction, the final construct will show close to optimal query performance, which is likely much better than what a R-tree construct would show.[14]

2.3 Algorithms and software

The optimal algorithm for finding pairs of points within a distance to each other in three dimensional space is a recursive divide and conquer algorithm that is *O(n*log(n))*, with *n* being the amount of points. This algorithm recursively divides the space on one axis and then on another axis, calculating distances between points only around the splitting plane and when reaching a sufficiently deep recursion depth. The plane to split on would for example be the y-z plane passing through the middle point on the x-axis. This creates two sub problems on either side of the splitting plane. The points that are close enough to the plane are projected onto the plane so that neighbours close enough to each other on opposite sides of the plane can be found. A similar algorithm is used for searching the two dimensional plane created by the projection.[15]

The R*-tree provides means for efficiently querying the data structure as one can descend the tree only choosing the points of interest. A query can choose whether or not to evaluate a node based on the space that is encapsulated by the minimum bounding box. This makes it possible for e.g a range query to exclude many data points from the search and as data points close in space are contained on the same pages, entire pages are avoided and I/O overhead is minimized. Consequently, queries for touch detection can choose to only evaluate nodes that are closely located in space and will thus only access

data closely located in memory. An algorithm utilizing the tree structure will thus be much more memory efficient when unnecessary memory overheads are avoided.

'libspatialindex' is a library offering implementations of different spatial data structures such as R-tree, R*-tree, MVR-tree and TVR-tree. Query capabilities such as nearest neighbour search and basic range search can be easily accessed through a C API and some customized queries can be constructed through provided functions.[16] 'boost' is another library that offers among other things an implementation of the R-tree with the choice of R*-tree indexing. A wide range of different spatial queries are provided, for example intersection, disjoint and covered-by queries.[17]

2.4 Related Work

2.4.1 Closely related work

There are several examples of studies on spatial data structures but not many with regards to the touch detection problem. Westlin/Mårtensson analysed the scalability of the KD-tree when applied to the touch detection problem. As pointed out in the introduction, they concluded that memory usage was the limiting factor as they ran into stack overflow issues when performing their measurements. Their findings could be useful when compared to the findings of this thesis as it would be possible to conclude which of the two data structures scales better.[5]

Another study by Norelius/Tacchi focused on comparing the performance of different data structures for spatial queries. They compared three different types of spatial data structures; R-tree, R*-tree, and Quadtree. They concluded that R*-tree scaled better than both other trees when the number of entries was increased.[6] Their conclusion was what led this thesis to analyse the R*-tree as no other thesis had yet explored that data structure with regards to the touch detection problem.

2.4.2 Speeding up range queries for brain simulations

In this study, a new way to index spatial data points is presented and compared to different versions of the R-tree when executing a range query. The new indexing method, chosen to be called FLAT, was tested along with the Hilbert R-tree, the STR R-tree and the PR-tree. They identified the problem of increased overlap between minimum bounding rectangles when performing queries on very dense data sets and thus developed FLAT. Benchmarks were made in the form of range queries on the different data structures and measurements of page reads and data retrieved was collected. Upon comparison of their benchmarks it was found that the FLAT method scaled much better in terms of executed page reads and thus the IO overhead was minimized. Their conclusion was that their new method FLAT made range queries on dense data sets much faster than any of the compared data structures but with higher construction costs. FLAT is not in the public domain and thus not considered in this thesis.[18]

Chapter 3

Methods

3.1 Reconstructions of Neurons

Publicly available reconstructions of neurons from the online archive neuromorpho.org was used[19]. The choice of which neurons to use is arbitrary, but for consistency all neurons came from the same animal and cortical layer. The neurons chosen were pyramidal neurons from a rat available in the Markram archive on Neuromorpho. The pyramidal cells are the main type of neurons in the cortex. The neurons in the Markram archive has been corrected as much as possible, with regards to reconstruction errors such as tissue shrinkage, which makes them especially usable for simulations.[20]

3.2 Implementation

The process of benchmarking R*-tree begins with adding all the neurons to the data structure. As the implementation of R*-tree provided by libspatialindex is used, it provides abstracted means of creating and inserting points to the structure. There is only a fixed amount of neuron reconstructions to use. Therefore, multiple copies of each neuron is added to the tree in order to reach the desired neuron density. Each neuron inserted is rotated randomly and placed with a random offset within allowed bounds.

In order to query the data structure for pairs of points a spatial query for touch detection was needed. The library libspatialindex was chosen for this thesis as it contains ready to use implementations of the data structure and some customizability. As there was no implementation provided by libspatialindex, or any other library for that matter, to query for pairs of points within a distance from each other the library had to be modified so that a spatial query for touch detection could be made (Appendix A). The spatial query implemented for this thesis simply descends the tree and only compares nodes that are within the touch detection distance. This eliminates much of the branching when searching the tree and only necessary comparisons of leaf nodes will be made. This implementation was chosen as it most truthfully shows the scalability of the data structure, as opposed to implementing the optimal algorithm for three dimensional closest pair which simply isn't feasible to use on large data sets where memory efficiency is important.

The construction of a R*-tree can be tweaked by altering certain variables, e.g leaf node capacity. Trees that are customized differently will more than likely show varying performance on the same data sets. As it is by no means an easy task to determine what would be the optimal construction parameters, all variables are set to their respective default values as provided by the library.

3.3 Benchmarks

There are two possible variables to vary upon, for benchmarking the scalability of the data structure. The variables for each test run are the neuron density of the search space and the maximum allowed distance between two points of different neurons for a connection to possibly exist.
In order to reach realistic values for the neuron density in the range of 50000-200000 neurons/mm^3[10] the volume of the search space is limited to a cube of 0.001mm^3 and the amount of neurons is iteratively set between 50 and 200, with fixed touch detection distances of 1, 3, and 5 microns for each run. The measurements for iterating upon touch detection distance is made with the same volume of space, with fixed neuron densities of 50000, 100000, and 150000 neurons/mm^3, while iterating upon the touch detection distance in the range of 0.5-5.0 microns. Each test with a set neuron density and touch detection distance is run 3 times, to counter exceptionally good or bad outliers.

In order to assess the scalability in a more realistic environment tests are also made with varied volume where all variables are set to match the human cortex. The value for touch detection distance was chosen to be 3 microns and the neuron density was set to 25000 neurons/mm^3. 3 microns might appear to be arbitrary but it was chosen as it was the middle value used for the measurements on neuron densities. The volume was then varied from 0.001mm^3 to 0.015mm^3, where the upper limit was set due to the calculations becoming too time consuming.

3.4 Environment for Measurements

All measurements in this thesis were made on a system with an Intel Core i7-4770K processor @ 3.50GHz and 16 gigabytes of RAM. While this will affect the execution time the scalability of the data structure can still be estimated and compared to more powerful systems.

3.5 Regression

To help assert the scalability of the data structure, the benchmark results from iterating densities is compared to curves generated from the same results by the least squares method, using different curve structures. The curve structures being used are $an^2log(n)+C$, $anlog(n)+C$, $alog(n)+C$, and an^2+bn+C. The second structure mentioned is chosen based on the theoretical time complexity for the optimal algorithm mentioned in 2.3. The other structures are chosen based on the fact that they're common time complexities for algorithms, and that they are slight deviations of the theoretical one. The r^2-values for each regression curve is then presented for discussion.

Chapter 4

Results

The results presented was produced from three individual test runs for each set of variables. The performance of each run is represented by a red dot. The mean performance for each set of parameters is represented by a blue bar.

4.1 Neuron Density

Benchmarks for different neuron densities were made in the range 50000-200000 neurons/mm^3, with three different touch detection distances. Each iteration increases the amount by 10000 (figure 4.4, 4.5, 4.6).

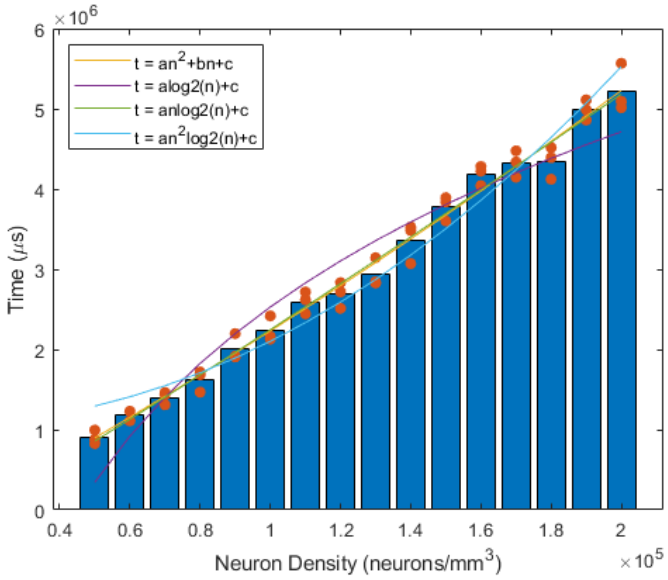

Figure 4.1: Results for varying neuron density with touch detection distance **1 micron**

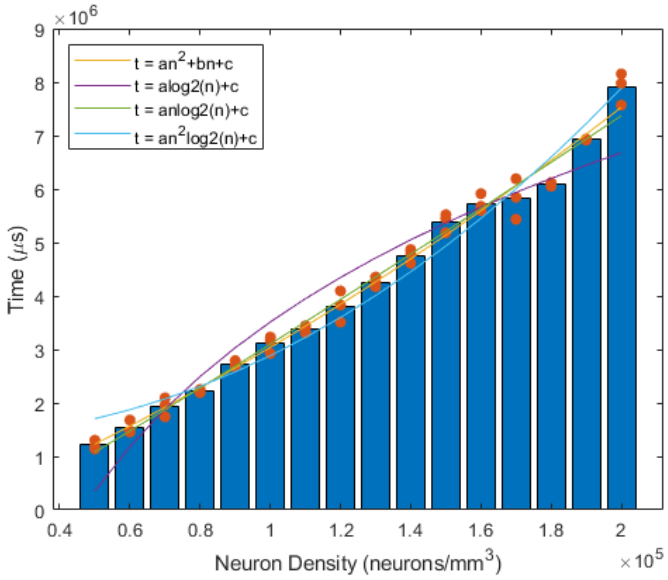

Figure 4.2: Results for varying neuron density with touch detection distance **3 microns**

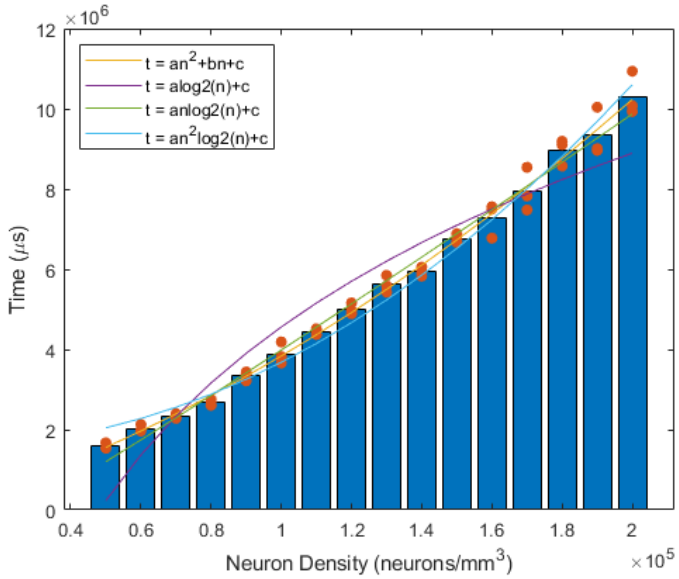

Figure 4.3: Results for varying neuron density with touch detection distance **5 microns**

4.2 Detection Distance

Benchmarks for different maximum touch distances were made in the range of 0.5-5.0 microns, with three different neuron densities. Each iteration increases the distance with 0.5 microns (figure 4.1). The upper limit was determined by the limit put by previous studies[7][8].

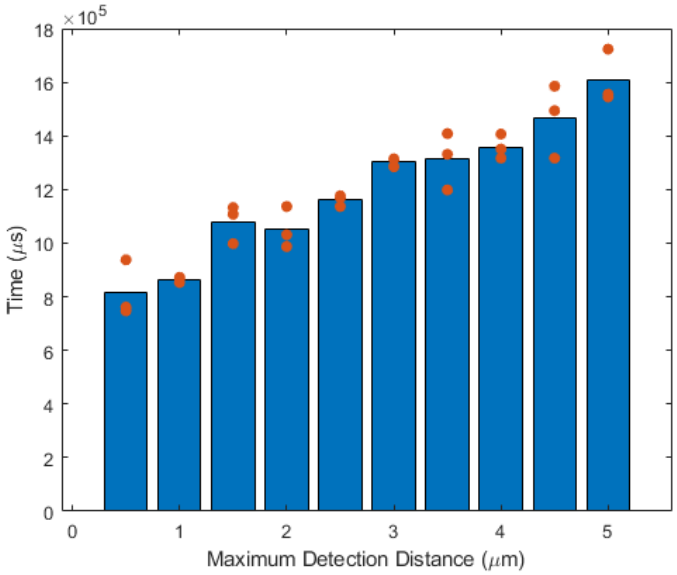

Figure 4.4: Results for varying lengths of touch detection distance with **50000 neurons/mm^3**

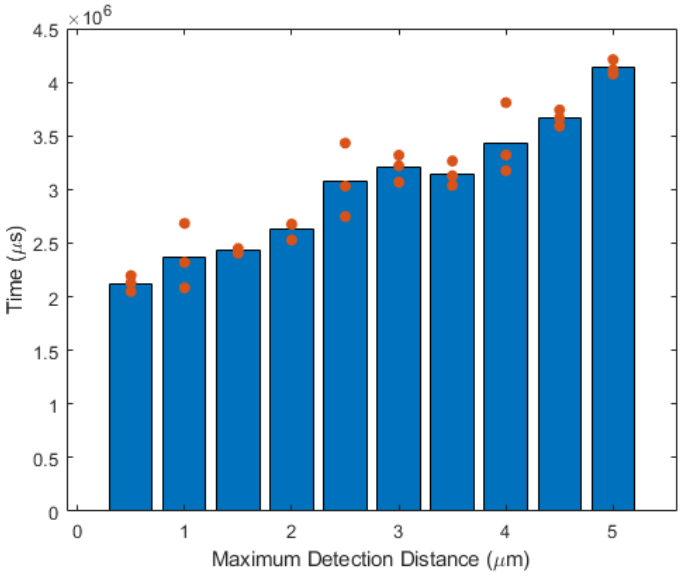

Figure 4.5: Results for varying lengths of touch detection distance with **100000 neurons/mm^3**

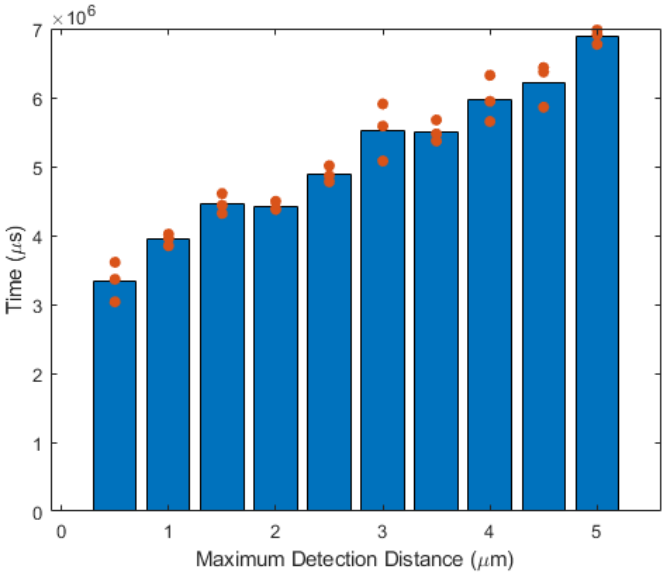

Figure 4.6: Results for varying lengths of touch detection distance with **150000 neurons/mm^3**

4.3 Regression

Curve Structures	1 µm	3 µm	5 µm
$an^2log(n)+C$	0.9738	0.9780	0.9907
$anlog(n)+C$	0.9937	0.9898	0.9931
$alog(n)+C$	0.9476	0.9365	0.9268
an^2+bn+C	0.9940	0.9917	0.9984

Table 4.1: The r^2-values for each curve structure generated by the least squares method. Higher values indicate a better matching curve.

4.4 Human Cortex

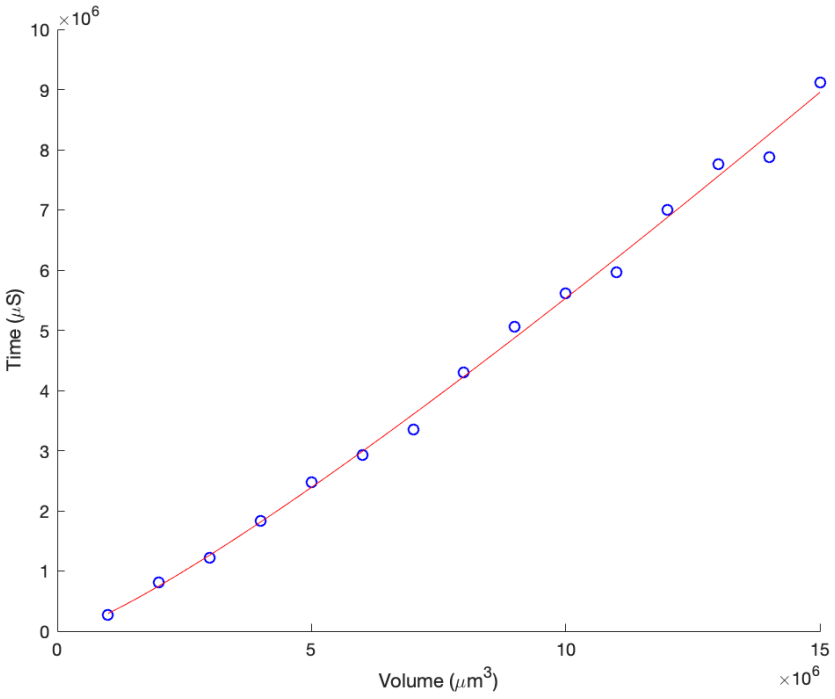

Figure 4.7: Results for varying volume (averages of 3 runs) with fixed density of 25000 neurons/mm^3 and touch detection distance 3 microns, fitted with a curve.

Chapter 5

Discussion

5.1 Iterating Neuron Density

The charts for varying neuron density (figure 4.1 - 4.3) shows some usable results. The scalability looks almost linear when increasing the density with a touch detection distance of 1 micron but shows more *O(n*log(n))* behaviour with a distance of 5 microns. This is much expected as a 5 times larger touch detection distance will result in a much larger volume around the points where other points would be accepted. After fitting the data points with the best fit curve, as indicated in table 4.1, it is clear that the time complexity resembles that of the optimal algorithm. Although the 2nd degree polynomial seems to fit the data better, it is excluded as it simply isn't realistic to have quadratic time complexity when only a small fraction of all points are compared for every point.

5.2 Iterating Touch Detection Distance

Our measurements for different values on touch detection distance (figure 4.4 - 4.6) clearly show increased execution time with increased distance, as one could expect. Upon comparing the three charts it is also visible that increasing the density makes the query scale worse as there are much more points to compare to when increasing the detection distance. It is also apparent that

increasing detection distance does not directly follow a cubic relationship as the optimal algorithm would do in theory. These results can be explained by the structure of the tree; leaf nodes contain several hundred data points and as the detection distance is increased the amount of comparisons made are only increased when more, previously not included, nodes are within the touch detection distance. That would explain the very similar results of slightly differing distances (e.g. the 1.5 and 2.0 micron tests for high densities). This could make the cubic relationship hard to capture with a low variation of touch detection distance, and could indicate a possible inefficiency as unnecessary comparisons are done.

5.3 Increasing Volume of Search Space

The results yielded from varying the volume of the search space (figure 4.7) while keeping the neuron density fixed aimed to show the scalability under circumstances that mimicked the human cortex. The curve fitted to the observed data points was chosen to have *O(n*log(n))* time complexity due to it having the best fit. What is shown in these results is that for a mere 10 μm^3 over 5 seconds of processing time is needed on the machine these tests were made on. The volumes used in this test are nowhere near a sizeable portion of the human cortex and extrapolating the yielded curve estimates that for a whole cm^3, more than 20 days of execution time would be required. If one would like to search the whole human cortex which is vastly larger, that number climbs to over 27000 days, which is nearly 74 years of execution time.

5.3.1 Speculation for Faster Execution

Even with our speculative time complexity, which is often considered 'good' for computation, the sheer size of the problem makes it unfeasible for purely sequential execution. Assuming that the query is completely parallelizable, one could consider the option to analyse an entire human cortex on a supercomputer with a vast array of processing units. Beskow is the supercomputer located at KTH.[21] Its prime purpose is to greatly reduce execution time on parallelizable programs, with its vast number of CPUs. We're assuming the entirety of the algorithm can be parallelized, due to the fact that we're searching for pairing points to each point in the data structure. We're also ignoring

any overheads and difficulties introduced by parallelizing the algorithm. Our computer has 8 cores, but since the program is purely sequential, only a single core is used. Since Beskow has 67456 cores, one could make some (extremely) rough calculations on how long Beskow would take to complete the task, if it was parallelized. Assuming that all cores on Beskow are equal, and equal to our cores, the execution time is reduced approximately 67456-fold, resulting in a execution time a little shy of half a day. Half a day execution time is still 'slow', but it is at least much more realistic than 74 years, and still more realistic than the approximately 9 years that we would achieve by the same rules on our own computer. Note that there are hundreds other supercomputers available in the world, that are considered better than Beskow[22]. Their performance might be even better.

5.4 Errors and Improvements

The measurements made in this thesis were made entirely in main memory meaning that the created R*-tree was in main memory and not on disk. This could mean that even though are a lot of cache misses the resulting penalty would be relatively small and hard to capture in the measurements. If the memory needed to store the tree is in the same order of magnitude as the different caches the penalty of misses would hardly be noticeable as much of the tree would likely be cached. Having a larger R*-tree stored on disk could drastically change the results as potential page faults would take more time to process. This would generate much more realistic results as data bases containing neuronal morphology are usually very large and cannot be stored in main memory.

The choice of neurons for this thesis was arbitrary and could have a large effect on the results. 10 different neurons was chosen for all measurements, which isn't anywhere close to the real value of unique neurons in the chosen cortical layer. Increasing both the amount of different neurons and improving the selection by choosing the most common neurons could generate more realistic results.

It is also possible that the simplifications made in this study affected the yielded results. Due to limitations on computing resources, all tests were made on a very small volume which could have caused parts of some neurons to be

placed outside the given volume. This could create a misleading density of data points inside the volume. While this is entirely dependent on randomization, we do not know to what extent this affects the results.

5.5 Comparison With Previous Work

The study Westlin and Mårtensson performed, despite the fact that we found major flaws regarding their unrealistic neuron density and their graphs being unreliable, pointed at that a realistic query size was prohibitively large, regarding both execution time and memory. Our study seems to point at the same conclusion.[5]

5.6 Future Work

Possible extensions of this thesis would be to either further analyze the R*-tree in depth or apply the touch detection task to other spatial data structures. The construction of the R*-tree can be customized and tweaked with different parameters e.g leaf node capacity, index node capacity and the factor of elements that get reinserted. This poses a optimization problem and could possibly be a subject for future work.

The previously mentioned data structure FLAT was proven to be more efficient than several different versions of the R-tree and would be the single most interesting data structure to analyse, but is sadly not in the public domain. Comparing the performance of different spatial data structures when used for the touch detection problem could be useful as one could conclude what kind of indexing is most suitable for neuronal morphology.

Since the problem size becomes unfeasibly large for any realistic query with our sequential execution, future work could look at the possibility of parallelizing the query, in the hopes of greatly reducing the execution time using multiple processing units.

Chapter 6

Conclusions

We can conclude that the touch detection task with the R*-tree data structure scales much like the optimal algorithm would do in theory, but with some minor possible inefficiencies as indicated in the above discussion. More work would be required to properly assess if those inefficiencies could be addressed with different construction parameters or if that simply is a fault with the R*-tree. It is also apparent that the realistic computing power needed to perform touch detection on a significant portion of the human cortex is far greater than what is available on an ordinary personal computer, but it is plausible for a supercomputer to do utilising parallelism.

Bibliography

[1] Cannon RC et al. “An on-line archive of reconstructed hippocampal neurons”. In: *J Neurosci Methods* 84(1-2) (1998), pp. 49–54.

[2] brgfx. Visited: 2020-04-29, Altered by Filip Berendt by adding tags. URL: https://www.freepik.com/free-vector/stem-cell-diagram-white-background_2480958.htm#page=1&query=stem-cell-diagram-white&position=4.

[3] S. Lawrence Zipursky Joshua R.Sanes. “Chemoaffinity Revisited: Dscams, Pro- tocadherins, and Neural Circuit Assembly”. In: (2010).

[4] de Garis H. Shuo C. Goertzel B. Ruiting L. “A world survey of artificial brain projects, Part I: Large-scale brain simulations”. In: *Neurocomputing 74.Issue 1-3* (2010), pp. 3–29.

[5] Mårtensson Westlin. “An estimation of scalability when using a k-d tree as the data structure for neuron touch detection”. In: (2018).

[6] Tacchi Norelius. “Evaluating data structures for range queries in brain simulations”. In: (2018).

[7] Reimann. Michael King. James Muller. Eilif Ramaswamy. Markram. “An algorithm to predict the connectome of neural microcircuits”. In: (2015).

[8] Hill. Sean Wang. Riachi. Schürmann. Markram. “Statistical connectivity provides a sufficient foundation for specific functional connectivity in neocortical neural microcircuit”. In: (2012).

[9] J. M. Bekkers. “Synaptic Transmission: Functional Autapses in the Cortex”. In: (2003). Visited: 2020-05-15, URL: https://www.sciencedirect.com/science/article/pii/S0960982203003634.

[10] Javier DeFelipe. Lidia Alonso-Nanclares. Jon I. Arellano. “Microstructure of the neocortex: Comparative aspects”. In: *Journal of Neurocytology 31* (2002), p. 302.

[11] J. L. Bentley. “Multidimensional binary search trees used for associative searching”. In: (1975). Visited: 2020-04-28, URL: https://dl.acm.org/doi/10.1145/361002.361007.

[12] Chire. *KDTree-Visualization*. Visited: 2020-04-19, URL: https://commons.wikimedia.org/wiki/File:3dtree.png.

[13] Chire. *RTree-Visualization*. Visited: 2020-04-19, URL: https://commons.wikimedia.org/wiki/File:RTree-Visualization-3D.svg.

[14] Norbert Beckmann. Hans-Peterbegel Ralf Schneider. Bernhard. Seeger. “The R*-tree: An Efficient and Robust AccessMethod for Points and Rectangles”. In: (1990), pp. 322–325.

[15] UC Santa Barbara Subhash Suri. *Closest Pair Problem*. Visited: 2020-03-26, URL: https://sites.cs.ucsb.edu/ suri/cs235/ClosestPair.pdf.

[16] Marios Hadjieleftheriou. *libspatialindex*. Visited: 2020-03-26 URL: https://libspatialindex.org/.

[17] Rene Rivera Beman Dawes David Abrahams. *boost*. Visited: 2020-03-26 URL: https://www.boost.org/.

[18] Farhan Tauheed. Laurynas Biveinis. Thomas Heinis. Felix Schürmann. Henry Markram. Anastasia Ailamaki. “Speeding Up Range Queries For Brain Simulations”. In: (2011).

[19] George Mason University. *NeuroMorpho*. Visited: 2020-03-26, URL: neuromorpho.org.

[20] Markram et al. “Reconstruction and Simulation of Neocortical Microcircuitry”. In: *Cell, Volume 163 Issue 2* (2015). Visited: 2020-05-15, URL: https://www.cell.com/cell/fulltext/S0092-8674(15)01191-5.

[21] PDC. Visited: 2020-05-15. URL: https://www.pdc.kth.se/hpc-services/computing-systems/beskow-1.737436.

[22] E. Strohmaier et al. Visited: 2020-06-04. URL: https://www.top500.org/list/2019/11/?page=1.

Appendix A

Spatial Query

The main part of the source code for the spatial query added to libspatialindex.

```
// The public function exposed in the class
void SpatialIndex::RTree::RTree::query_pairs(IVisitor&
   vis, double& r) {
   NodePtr n1 = readNode(m_rootID);
   int index = 0;
   // Search all combinations of boxes below root node.
   for (uint32_t cChild1 = 0; cChild1 < n1->m_children;
      ++cChild1) {
      for (uint32_t cChild2 = index; cChild2 <
         n1->m_children; ++cChild2) {
         query_pairs(n1->m_pIdentifier[cChild1],
            n1->m_pIdentifier[cChild2], vis, r);
      }
      index++;
   }
}

// Helper function
void SpatialIndex::RTree::RTree::query_pairs(id_type
   id1, id_type id2, IVisitor& vis, double& r) {
   // Read the two nodes to be compared
   NodePtr n1 = readNode(id1);
   NodePtr n2 = readNode(id2);
   vis.visitNode(*n1);
```

```
vis.visitNode(*n2);
int index = 0;

for (uint32_t cChild1 = 0; cChild1 < n1->m_children;
   ++cChild1) {
   for (uint32_t cChild2 = index; cChild2 <
      n2->m_children; ++cChild2) {

      // break away all branches too far away
      if
         ((*(n1->m_ptrMBR[cChild1])).getMinimumDistance(
         *(n2->m_ptrMBR[cChild2])) >= r) {
         continue;
      }

      if (n1->m_level == 0 && n2-> m_level == 0) {
         // Compare two leaf nodes
         std::vector<const IData*> v;
         Data e1(n1->m_pDataLength[cChild1],
            n1->m_pData[cChild1],
            *(n1->m_ptrMBR[cChild1]),
            n1->m_pIdentifier[cChild1]);
         Data e2(n2->m_pDataLength[cChild2],
            n2->m_pData[cChild2],
            *(n2->m_ptrMBR[cChild2]),
            n2->m_pIdentifier[cChild2]);
         v.push_back(&e1);
         v.push_back(&e2);
         vis.visitData(v);
      } else if (n1->m_level == 0) {
         // If n1 is a leaf node then further descend
            n2
         query_pairs(id1, n2->m_pIdentifier[cChild2],
            vis, r);
      } else if (n2->m_level == 0) {
         // If n2 is a leaf node then further descend
            n1
         query_pairs(n1->m_pIdentifier[cChild1], id2,
            vis, r);

         // break because of n2 being a leaf node
         break;
```

```
            } else {
               query_pairs(n1->m_pIdentifier[cChild1],
                  n2->m_pIdentifier[cChild2], vis, r);
            }
         }
         // Stop duplicate search if same node
         if (id1 == id2) {
            index++;
         }

         // break if n1 is at bottom of the tree and we're
            not comparing two leaf nodes.
         if (n1->m_level == 0 && n2->m_level != 0) {
            break;
         }
      }
   }
```

TRITA -EECS-EX-2020:337