Comparison of spatial indexes

Nathalie Andrea Barbosa Roa

Nathalie Barbosa

Comparison of Spatial Indexes

In this report three different tree data structures were tested in order to find the index structure best suited to data groups retrieval for clustering purposes. Group retrieval is considered as the process of finding all the neighborhoods in the data set. This process involves a recursive implementation of radius-neighbor queries. Time and memory measures were taken for the tree construction process and for the group retrieval process. These measures are the average value over a hundred simulations. The selected structures are: the R*Tree [1], the BallTree [4] and the KDTree [2].

I Introduction

The R*Tree, or rectangular Tree, was proposed in 1990 and has been largely used since. In clustering applications, the ClusTree algorithm uses an extension of this index to efficiently locate the right place for object insertion [3]. The key idea of this data structure is to group nearby objects and represent them with their minimum bounding rectangle in the next higher level of the tree.

BallTrees is a completely binary tree in which each node refer to a region bounded by an d-dimensional hyper-sphere [4]. A parent node is hence the smallest hyper-sphere that contains all the hyper-spheres of its children. The BallTree is the only index, among the analyzed indexes, that accepts the intersection of sibling regions.

A K-dimensional binary search tree (KDTree) partitions the data space into mutually exclusive hyper-rectangular regions. Each region is found by recursively splitting the space using axis-parallel hyperplanes. The splitting process finishes when each sub-region has a number of points less than or equal to a given threshold.

The python implementations of the R*Tree[1], the BallTree[2], the KDTree[3] and the cKDTree[4] (A cython implementation of the KDTree), were tested using three different scenarios. The computation time and memory consumption (maximum resident set size used), were selected as comparative measures.

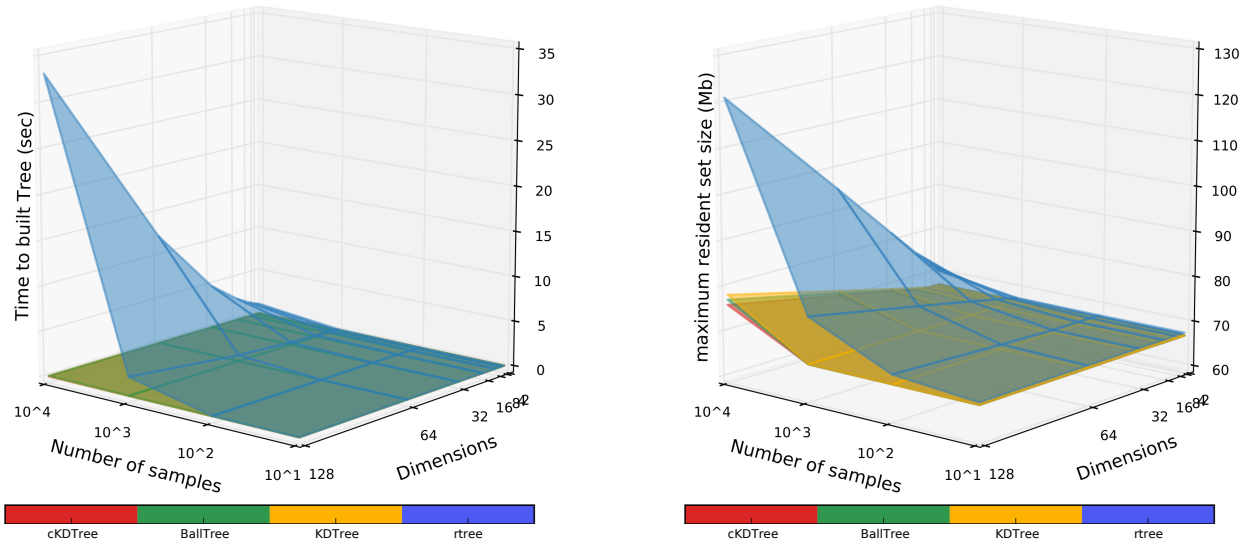
II Performance with uniform distributions

In the first scenario, data correspond to random samples extracted from a uniform distribution. The algorithms were tested for a data sets containing n data samples, where each sample is a d-dimensional vector. Tests were performed for n varying in the range $n \in \{10^1, 10^2, 10^3, 10^4\}$ and d in $d \in \{2^1, 2^2, 2^3, 2^4, 2^5\}$.

Figure 1 show the resources used for the tree building task. Computation time measured in seconds is presented in Figure 1a and the maximum resident set size measured in Mb is shown in Figure 1b. It can be seen that the R*Tree implementation performs poorly with respect to the other indexes. The actual measurements of the computation time and maximum resident set size are presented in tables 1 and 2 respectively.

The group retrieval task is computationally more expensive as can be seen in Figure 2, where the computation time for the worst case (R*Tree, $n = 18$,$d = 10^4$) passes from 33.12 seconds in the tree building task to 362.9 seconds for group retrieval (see Figure 2a). The best computation time is achieved by the cKDTree index but at cost of having the worst memory consumption (see Figure 2b). The recorded data for the group retrieval task is presented in tables 3 and 4.

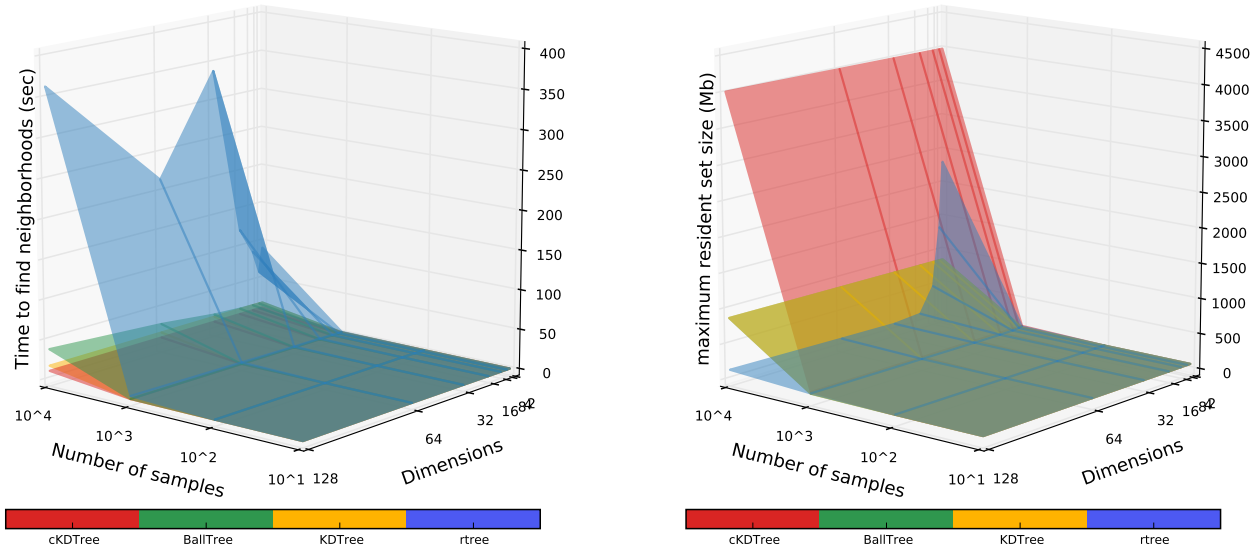
[1] `http://toblerity.org/rtree/`
[2] `http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.BallTree.html`
[3] `http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KDTree.html`
[4] `http://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.cKDTree.html`



a Computation time

b Maximum resident set size

Figure 1: Comparison between different tree indexes for the tree building task in a uniform distribution



a Computation time

b Maximum resident set size

Figure 2: Comparison between different tree indexes for the group retrieval task in a uniform distribution

Index	$log(n)$	Dimensions						
		2	4	8	16	32	64	128
R*Tree	1	12.12	13.62	14.50	16.12	17.37	23.25	31.62
	2	94.88	99.62	108.62	128.00	160.00	222.75	358.50
	3	1142.25	1236.62	1444.00	1977.12	3456.00	8483.38	26073.62
	4	13861	15552	19784	30752	60609	153557	414038
BallTree	1	1.75	1.87	1.87	1.87	1.87	1.87	1.87
	2	1.87	2.00	2.00	2.12	2.38	2.62	3.25
	3	4.00	4.25	5.25	7.50	11.87	20.37	36.37
	4	40.25	47.50	66.12	103.12	203.75	424.12	877.87
KDTree	1	1.87	1.87	1.75	1.75	1.87	1.87	1.75
	2	1.87	1.87	1.87	2.00	2.12	2.38	3.00
	3	3.50	3.75	4.50	6.00	9.00	16.62	31.62
	4	33.62	40.25	54.25	84.50	165.50	364.37	786.75
cKDTree	1	1.00	1.00	1.00	1.12	1.00	1.12	1.12
	2	1.12	1.12	1.12	1.25	1.25	1.50	1.75
	3	2.25	2.38	2.62	3.12	3.87	5.00	7.12
	4	15.12	16.38	18.75	23.88	33.75	46.25	67.88

Table 1: Tree building computation time (relative to 80μsec) for uniformly distributed data

Index	$log(n)$	Dimensions						
		2	4	8	16	32	64	128
R*Tree	1	1.01	1.01	1.01	1.01	1.01	1.01	1.01
	2	1.01	1.01	1.01	1.01	1.01	1.02	1.03
	3	1.01	1.01	1.02	1.03	1.05	1.09	1.16
	4	1.05	1.06	1.08	1.13	1.23	1.43	1.81
BallTree	1	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	3	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	4	1.00	1.00	1.00	1.01	1.03	1.07	1.15
KDTree	1	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	3	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	4	1.00	1.00	1.00	1.01	1.03	1.08	1.17
cKDTree	1	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	3	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	4	1.00	1.00	1.00	1.01	1.03	1.06	1.14

Table 2: Tree building maximum resident set size (relative to 66.762Mb) for uniformly distributed data

Index	$log(n)$	Dimensions						
		2	4	8	16	32	64	128
R*Tree	1	4.00	5.23	5.17	5.83	7.29	11.63	17.14
	2	49.83	48.77	62.66	86.66	85.66	124.57	207.80
	3	2339	1843.80	1988.77	3330.66	3424.17	6092.80	11293.66
	4	260650	171165	254612	350371	967009.14	614249	1036927
BallTree	1	1.06	1.06	1.06	1.06	1.09	1.09	1.14
	2	6.43	6.54	6.63	7.11	7.71	9.09	11.26
	3	511.94	510.60	520.17	541.06	603.00	736.14	953.86
	4	50449	49619	50422	51687	58331	81430	109406
KDTree	1	1.06	1.06	1.06	1.06	1.06	1.09	1.09
	2	6.46	6.43	6.40	6.57	6.57	6.71	7.00
	3	514.34	514.26	516.29	515.03	509.11	512.94	518.91
	4	49768	50382	50643	49987	50149	49846	49907
cKDTree	1	1.03	1.03	1.00	1.06	1.03	1.09	1.11
	2	6.14	6.17	6.14	6.14	6.29	6.54	6.89
	3	331.94	333.94	335.34	335.94	334.77	336.54	348.29
	4	30971	31098	31191	31205	31172	31185	31249

Table 3: Group retrieval computation time (relative to 350μsec) for uniformly distributed data

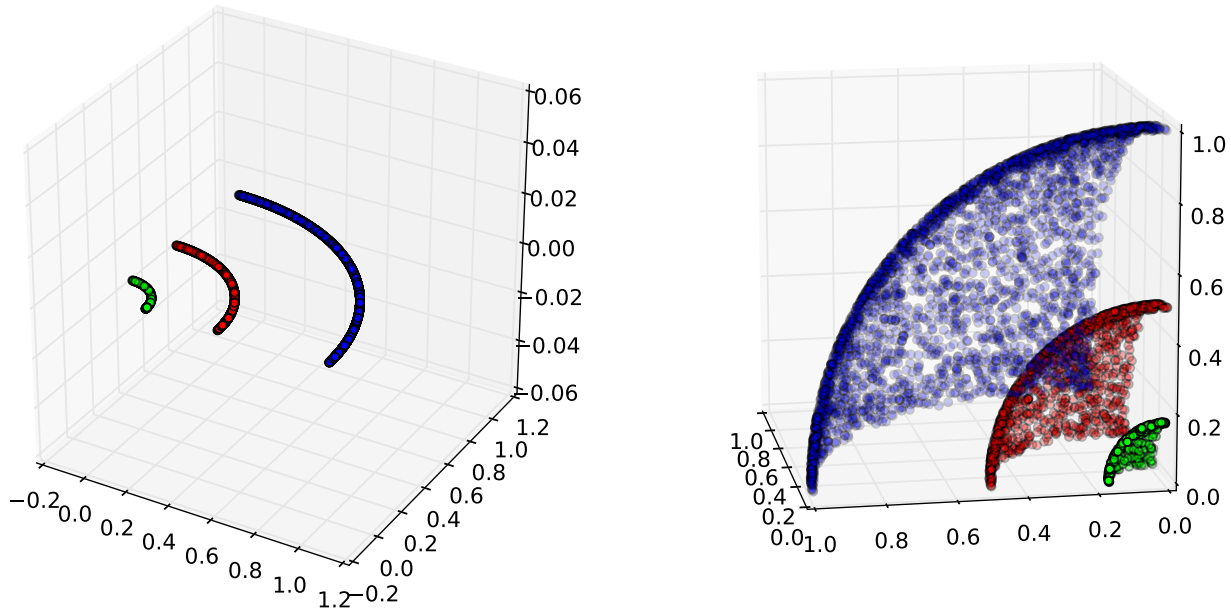
Index	$log(n)$	Dimensions						
		2	4	8	16	32	64	128
R*Tree	1	1.01	1.01	1.01	1.01	1.01	1.01	1.01
	2	1.01	1.01	1.01	1.01	1.01	1.02	1.03
	3	1.32	1.20	1.09	1.05	1.06	1.11	1.19
	4	34.63	20.14	7.27	1.88	1.36	1.58	2.07
BallTree	1	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	3	1.11	1.11	1.11	1.11	1.12	1.12	1.14
	4	12.74	12.74	12.74	12.76	12.80	12.88	13.04
KDTree	1	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	3	1.11	1.11	1.11	1.11	1.11	1.12	1.14
	4	12.74	12.74	12.74	12.76	12.81	12.89	13.05
cKDTree	1	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	3	1.47	1.47	1.47	1.47	1.48	1.49	1.51
	4	59.66	59.67	59.68	59.70	59.76	59.87	60.10

Table 4: Group retrieval maximum resident set size (relative to 66.762Mb) for uniformly distributed data

III Performance with geometrical distributions

III.1 Partial geometries

The second scenario analyzed was that of random data taken from a geometrical distribution. The distribution used in the 2-dimensional test is shown in Figure 3a and for illustrative purposes a 3-dimensional distribution is shown in Figure 3b. Distributions with $d > 3$ are not plotted but the reader can extrapolate its geometrical interpretation. The algorithm were tested again varying the number of samples and the dimensions as established in the first scenario.



a Geometrical distribution in 2D used to test

b Geometrical distribution in 3D (illustrative)

Figure 3: Representation of the geometrical distribution for 2 and 3 dimensions

The results for the tree building task are presented in tables 5 and 6. It can be seen in Table 5 that the cKDTree outperforms the other implementations, being two times faster where $n \leq 1000$ and three to almost five times faster where $n = 10000$. Its relative performance improves as the number of samples and dimensions increases. The memory consumption of the tested implementations was almost the same for the BallTree, the KDTree and the cKDTree, so arguably this factor impacts in a lesser extent the indexes performance. Figure 4 presents the discussed results graphically. As was the case for the uniform distribution the R*Tree implementation presents the poorest performance.

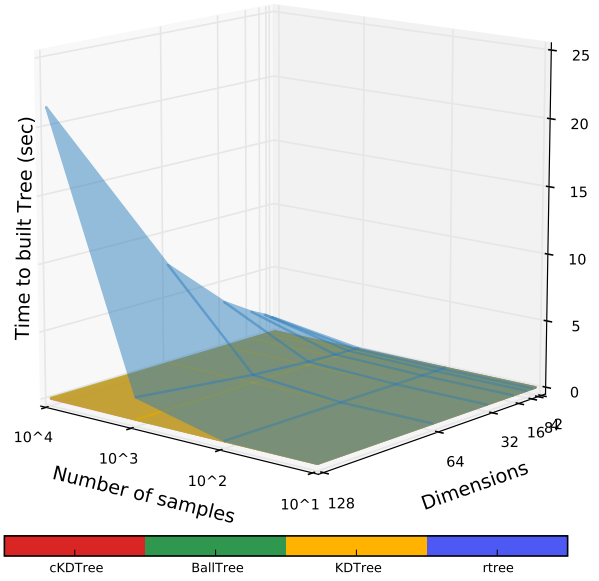
Statistics for the group retrieval task in the geometrical distribution are shown in tables 7 and 8 and plotted in Figure 5. Unlike the previous results (first scenario) in this scenario the worst memory consumption was that of R*Tree, even if the cKDTree memory consumption was only slightly better. The performance of the BallTree and the KDTree are very similar in terms of memory consumption and computation time. Once again the cKDTree presents the best performance with respect to the computation time.

Index	$log(n)$	Dimensions						
		2	4	8	16	32	64	128
R*Tree	1	15.33	15.67	16.67	18.67	21.67	28.00	41.17
	2	125.50	131.50	143.17	165.50	210.50	293.33	473.00
	3	1483.17	1640.17	1917.33	2696.67	4450.50	9861.00	27415.17
	4	17922	20325	26139	35357	58534	126822	357730
BallTree	1	2.17	2.17	2.17	2.17	2.17	2.17	2.17
	2	2.33	2.33	2.33	2.50	2.83	3.33	4.17
	3	5.00	5.33	6.83	9.50	15.67	27.00	48.67
	4	52.67	62.83	89.67	139.83	276.00	576.83	1190.00
KDTree	1	2.17	2.17	2.17	2.17	2.33	2.17	2.33
	2	2.33	2.33	2.33	2.50	2.67	3.17	3.83
	3	4.50	4.83	5.83	8.00	11.83	22.00	42.33
	4	42.17	52.50	74.00	113.83	212.00	490.00	1055.00
cKDTree	1	1.00	1.00	1.00	1.00	1.00	1.00	1.17
	2	1.17	1.17	1.17	1.33	1.50	1.83	2.33
	3	2.67	2.83	3.33	4.33	5.67	7.17	11.50
	4	19.00	20.83	26.33	37.67	70.83	137.50	255.50

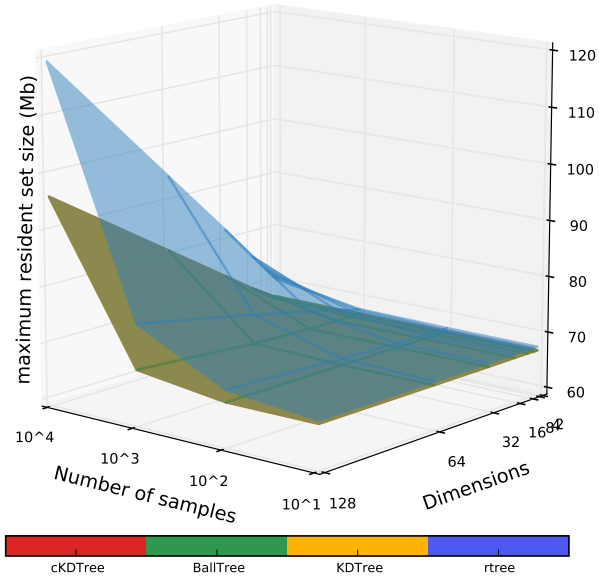
Table 5: Tree building computation time (relative to 60μsec) for geometrical distribution

Index	$log(n)$	Dimensions						
		2	4	8	16	32	64	128
R*Tree	1	1.01	1.01	1.01	1.01	1.01	1.01	1.01
	2	1.01	1.01	1.01	1.01	1.01	1.02	1.03
	3	1.01	1.02	1.02	1.03	1.05	1.09	1.15
	4	1.05	1.06	1.09	1.13	1.23	1.42	1.79
BallTree	1	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	3	1.00	1.00	1.00	1.00	1.00	1.01	1.03
	4	1.00	1.00	1.01	1.04	1.10	1.21	1.44
KDTree	1	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	3	1.00	1.00	1.00	1.00	1.00	1.01	1.03
	4	1.00	1.00	1.01	1.04	1.10	1.21	1.44
cKDTree	1	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	3	1.00	1.00	1.00	1.00	1.00	1.01	1.03
	4	1.00	1.00	1.01	1.04	1.10	1.21	1.44

Table 6: Tree building maximum resident set size (relative to 66.68Mb) for geometrical distribution

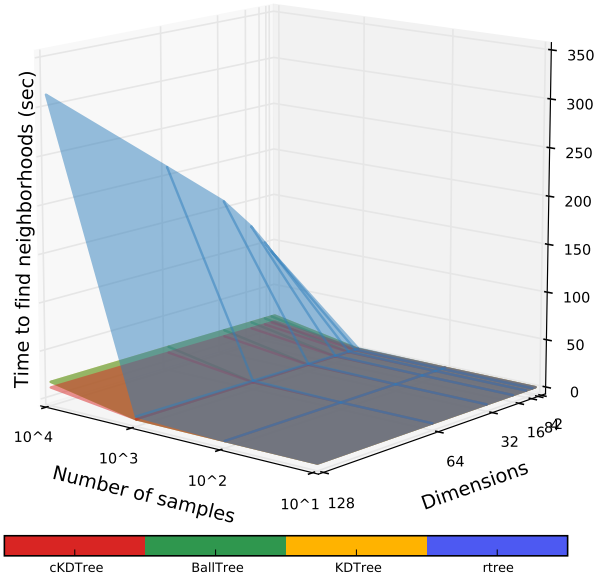


a Computation time

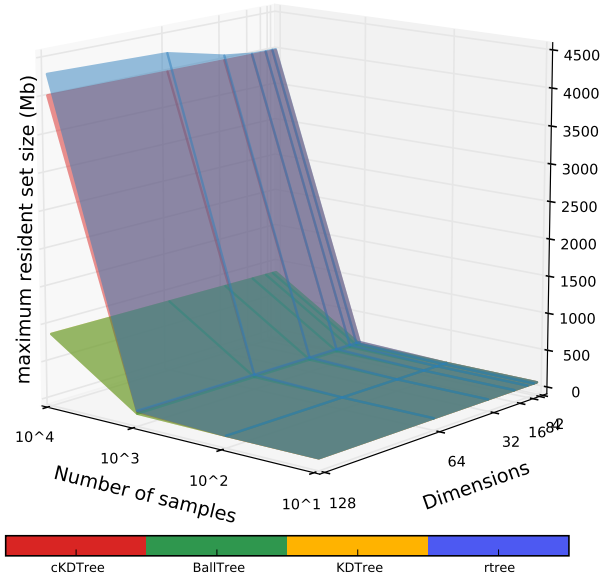


b Maximum resident set size

Figure 4: Comparison between different tree indexes for the tree building task in a geometrical distribution



a Computation time



b Maximum resident set size

Figure 5: Comparison between different tree indexes for the group retrieval task in a geometrical distribution

Index	$log(n)$	Dimensions						
		2	4	8	16	32	64	128
R*Tree	1	3.38	3.91	4.62	5.65	7.18	9.74	15.79
	2	35.24	40.06	45.24	66.68	98.44	146.00	228.00
	3	2199.97	2259.26	2393.71	2758.03	3645.88	5432.44	8878.50
	4	250598	265346	300324	364142	467381	619182	920223
BallTree	1	1.00	1.03	1.00	1.03	1.00	1.06	1.09
	2	6.71	6.59	6.74	6.71	6.82	7.03	7.44
	3	531.32	527.71	531.82	527.76	537.00	539.74	535.24
	4	51958	52095	52212.85	51986	52660	52389	52093
KDTree	1	1.00	1.00	1.00	1.00	1.03	1.06	1.06
	2	6.68	6.76	6.68	6.68	6.74	7.00	7.21
	3	533.44	540.38	530.12	537.88	528.24	536.47	543.06
	4	52054	52308	52424	52653	51969	52864	51874
cKDTree	1	1.06	1.03	1.03	1.03	1.06	1.06	1.12
	2	6.41	6.38	6.62	6.53	6.71	7.00	7.47
	3	348.56	346.62	351.38	354.29	354.50	354.21	361.38
	4	32318	32319	32523	32783	33095	33906	34722

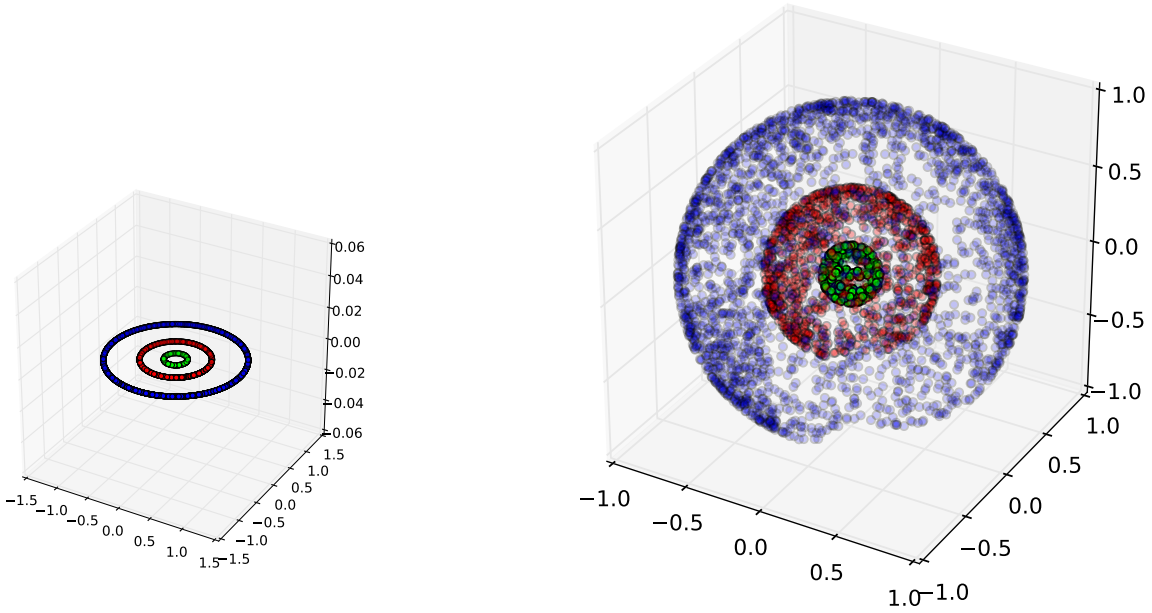
Table 7: Group retrieval computation time (relative to 340μsec) for geometrical distribution

Index	$log(n)$	Dimensions						
		2	4	8	16	32	64	128
R*Tree	1	1.01	1.01	1.01	1.01	1.01	1.01	1.01
	2	1.01	1.01	1.01	1.01	1.01	1.02	1.03
	3	1.52	1.53	1.54	1.55	1.56	1.62	1.72
	4	59.92	59.92	59.90	59.98	60.78	63.63	64.32
BallTree	1	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	3	1.11	1.11	1.11	1.11	1.12	1.12	1.14
	4	12.75	12.75	12.76	12.77	12.81	12.89	13.05
KDTree	1	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	3	1.11	1.11	1.11	1.11	1.12	1.12	1.14
	4	12.75	12.75	12.76	12.78	12.82	12.90	13.07
cKDTree	1	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	3	1.47	1.47	1.47	1.47	1.48	1.49	1.51
	4	59.72	59.73	59.73	59.76	59.82	59.94	60.16

Table 8: Group retrieval maximum resident set size (relative to 66.70Mb) for geometrical distribution

III.2 Spherical distribution

The final scenario test the indexes performance (computation time and maximum resident set size) for the tree building and group retrieval tasks in a spherical distribution. The 2-dimensional version of this distribution, formed by concentric circles, is shown in Figure 6a. In higher dimensions this distribution represents samples located in the surface of concentric hyper-spheres. For illustrative purposes a 3-dimensional spherical distribution is depicted in Figure 6b.



a Spherical distribution in 2D used to test

b Spherical distribution in 3D (illustrative)

Figure 6: Representation of the spherical distribution for 2 and 3 dimensions

Performance statistics for the tree building task are shown in tables 9 and 10 and its graphical representation depicted in Figure 7. In general all the indexes perform better in the tree building task for this distribution that for the uniform distribution and the performances were similar to those in the geometrical distribution. On the contrary, for the group retrieval task, the indexes did better is this distribution that in the other two as can be seen in Figure 8. The group retrieval measurements are shown in tables 11 and 12.

Index	$log(n)$	Dimensions						
		2	4	8	16	32	64	128
R*Tree	1	15.50	16.00	16.83	18.50	21.83	28.50	42.50
	2	127.83	134.00	147.83	168.83	216.33	298.83	487.17
	3	1524.67	1640.83	1912.83	2613.17	4548.17	10029.67	27845.00
	4	18678	21197	26249	36314	59633	1302993	374534
BallTree	1	2.17	2.17	2.17	2.17	2.17	2.17	2.17
	2	2.33	2.33	2.33	2.50	2.83	3.33	4.17
	3	5.17	5.50	6.83	9.67	15.67	26.67	49.50
	4	56.00	62.67	90.00	139.00	275.17	576.17	1180.50
KDTree	1	2.17	2.17	2.17	2.17	2.17	2.17	2.17
	2	2.33	2.33	2.33	2.50	2.67	3.00	3.83
	3	4.50	4.83	5.83	7.67	11.83	22.33	43.33
	4	45.67	52.50	72.83	113.50	216.33	492.67	1060.67
cKDTree	1	1.00	1.00	1.00	1.00	1.00	1.17	1.17
	2	1.17	1.17	1.17	1.33	1.33	1.67	2.00
	3	2.67	3.00	3.00	3.83	5.00	6.33	9.33
	4	19.67	21.83	31.00	31.83	44.67	62.50	91.67

Table 9: Tree building computation time (relative to 60μsec) for spherical distribution

Index	$log(n)$	Dimensions						
		2	4	8	16	32	64	128
R*Tree	1	1.01	1.01	1.01	1.01	1.01	1.01	1.01
	2	1.01	1.01	1.01	1.01	1.01	1.02	1.03
	3	1.01	1.02	1.02	1.03	1.05	1.09	1.16
	4	1.05	1.06	1.08	1.13	1.23	1.42	1.80
BallTree	1	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	3	1.00	1.00	1.00	1.00	1.00	1.01	1.03
	4	1.00	1.00	1.01	1.04	1.10	1.21	1.44
KDTree	1	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	3	1.00	1.00	1.00	1.00	1.00	1.01	1.03
	4	1.00	1.00	1.01	1.04	1.10	1.21	1.44
cKDTree	1	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	3	1.00	1.00	1.00	1.00	1.00	1.01	1.03
	4	1.00	1.00	1.01	1.04	1.10	1.21	1.44

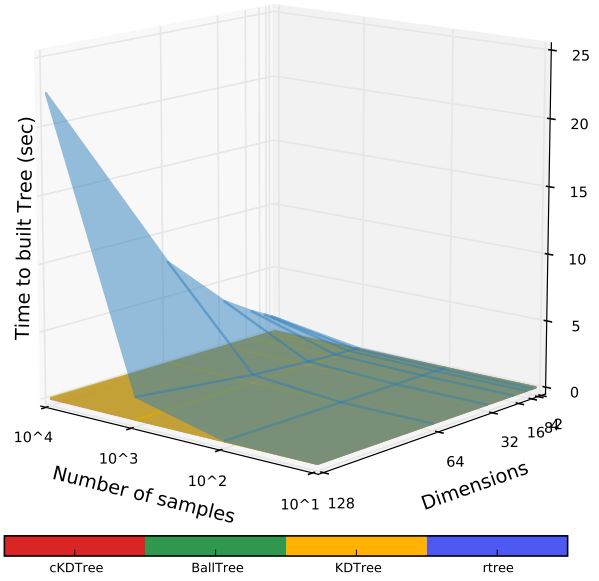
Table 10: Tree building maximum resident set size (relative to 66.64Mb) for spherical distribution

Index	$log(n)$	Dimensions						
		2	4	8	16	32	64	128
R*Tree	1	3.55	3.80	3.80	4.58	6.07	9.10	15.10
	2	32.60	40.12	49.67	65.12	90.73	125.00	191.50
	3	1920.95	1955.17	2094.47	2451.57	3199.20	4710.50	7680.00
	4	219311	236073	267983	317122	402437	528211	804337
BallTree	1	1.05	1.15	1.20	1.28	1.60	2.17	3.00
	2	5.75	5.60	5.62	5.77	5.72	5.85	5.95
	3	458.25	450.72	455.75	450.77	456.17	451.57	463.32
	4	44230	44562	44349	44420	44831.72	44533	44381
KDTree	1	1.00	1.15	1.17	1.28	1.50	2.25	3.00
	2	5.58	5.77	5.62	5.70	5.70	5.88	6.10
	3	448.47	454.30	459.42	456.32	459.35	455.32	457.50
	4	44812	44817	44637	44389	44525	44722	44339
cKDTree	1	1.10	1.20	1.30	1.38	1.60	1.92	2.45
	2	5.47	5.53	5.45	5.45	5.65	5.67	6.00
	3	300.75	298.70	301.20	301.38	311.93	299.07	301.97
	4	27416	27553	28353	27688	27607	27688	27730

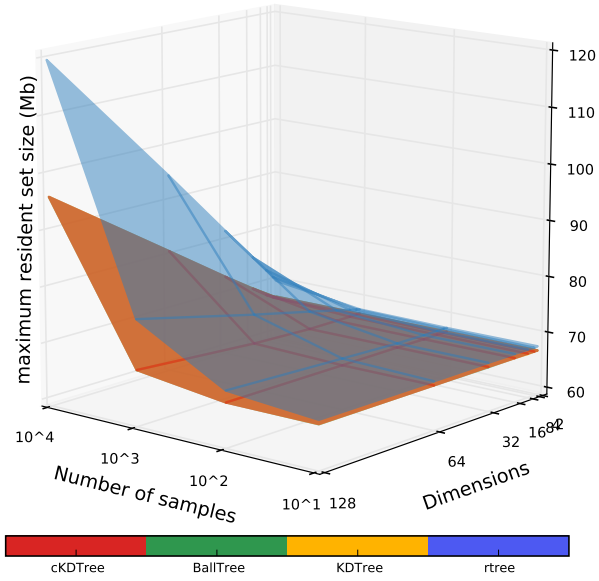
Table 11: Group retrieval computation time (relative to 400μsec) for spherical distribution

Index	$log(n)$	Dimensions						
		2	4	8	16	32	64	128
R*Tree	1	1.01	1.01	1.01	1.01	1.01	1.01	1.01
	2	1.01	1.01	1.01	1.01	1.01	1.02	1.03
	3	1.53	1.53	1.54	1.55	1.57	1.62	1.72
	4	59.97	59.99	59.97	60.02	60.87	63.45	64.32
BallTree	1	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	3	1.11	1.11	1.11	1.11	1.12	1.12	1.14
	4	12.76	12.76	12.77	12.78	12.83	12.90	13.06
KDTree	1	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	3	1.11	1.11	1.11	1.11	1.12	1.12	1.14
	4	12.76	12.76	12.77	12.79	12.83	12.91	13.08
cKDTree	1	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	2	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	3	1.47	1.47	1.47	1.47	1.48	1.49	1.51
	4	59.77	59.78	59.79	59.81	59.87	59.98	60.21

Table 12: Group retrieval maximum resident set size (relative to 66.64Mb) for spherical distribution

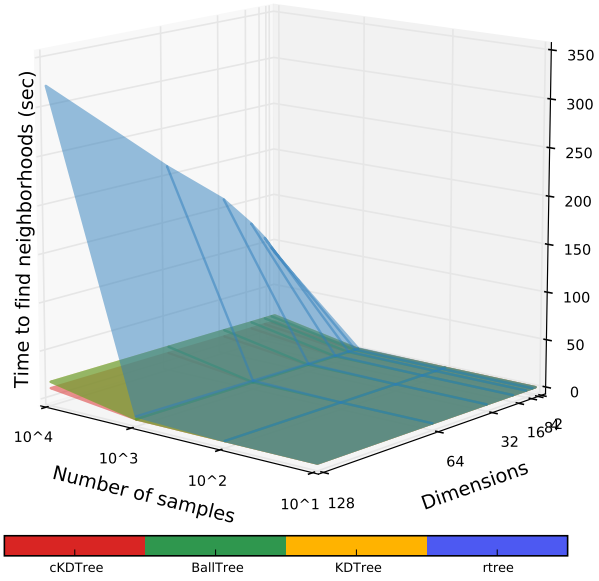


a Computation time

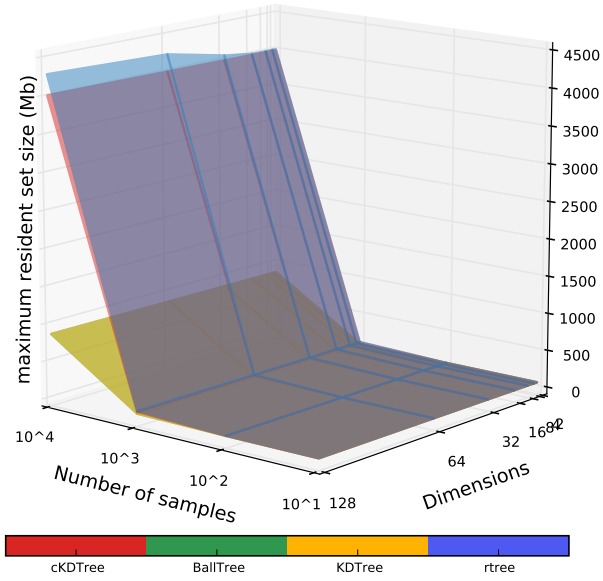


b Maximum resident set size

Figure 7: Comparison between different tree indexes for the tree building task in a spherical distribution



a Computation time



b Maximum resident set size

Figure 8: Comparison between different tree indexes for the group retrieval task in a spherical distribution

References

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

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

[3] Philipp Kranen, Ira Assent, Corinna Baldauf, and Thomas Seidl. The ClusTree: indexing micro-clusters for any stream mining. *Knowledge and information systems*, 29(2):249–272, 2011.

[4] Stephen M. Omohundro. Five balltree construction algorithms. Technical report, International Computer Science Institute Berkeley, 1989.