Were made 100 different searchs over the Berlin graph. For each search, each algorithm runs 10 times and in each run I restart the JVM, so we can assurer that no cach is used to give an advantage to any code. Then, I take the mean time for each algorithm and store on a csv file.

The elapsed time to generate this data was 8h 52min 22s

```
In [1]: %matplotlib inline
        import matplotlib.pyplot as plt
        import matplotlib.colors as mcolors
        import pandas as pd
        import numpy as np
        import utils as ut
        import seaborn as sns
```

Loading the data generated from tests

The Vanila algorithm is a implementation made by Erick

```
In [2]: #ut.generateTimes(n_tests=10,operations=["BerlinVanilla"],save_file="../vanila_test.csv")
        df = ut.loadData()[["Distancem(m)","BerlinGraphHopperContracted","BerlinVanilla","BerlinGraphHopper","BerlinGraphasst","BerlinNeo4j"]]
        #df
```

Analysing the data

Mean time of the algorithms

```
In [3]: df.mean()
```

```
Out[3]: Distancem(m)                  27177.540
        BerlinGraphHopperContracted      33.026
        BerlinVanilla                    90.237
        BerlinGraphHopper               168.031
        BerlinGraphasst                 552.091
        BerlinNeo4j                     546.270
        dtype: float64
```

Median of the times

```
In [4]: df.median()
```

```
Out[4]: Distancem(m)                  26362.50
        BerlinGraphHopperContracted      33.10
        BerlinVanilla                    90.46
        BerlinGraphHopper               175.25
        BerlinGraphasst                 552.45
        BerlinNeo4j                     515.28
        dtype: float64
```

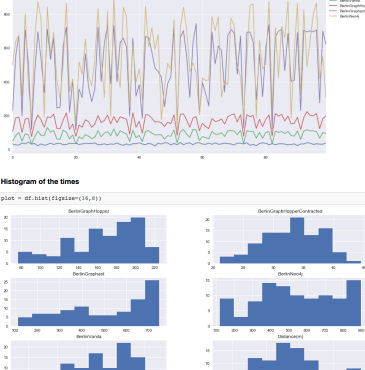
Standard Deviation

```
In [5]: df.std()
```

```
Out[5]: Distancem(m)                  10254.714928
        BerlinGraphHopperContracted       4.762620
        BerlinVanilla                    22.837541
        BerlinGraphHopper                37.321245
        BerlinGraphasst                 191.913549
        BerlinNeo4j                     218.349776
        dtype: float64
```

Time for each algorithm execute a search

```
In [6]: %config InlineBackend.figure_format = 'png'
        distances = df.columns[1:]
        #print(distances)
        plot = df.ioc[:,distances].plot(grid=True, figsize=(15,8)).legend(bbox_to_anchor=(1.0, 1))
```



Histogram of the times

```
In [7]: plot = df.hist(figsize=(16,8))
```


The data

In red, we have the algorithm that has taken more time to execute the search and, in green, the fastest one

```
In [9]: col = [c.replace("Berlin_","") for c in df.columns if c!="Distancem(m)"]
        df.rename(columns=col,inplace=True)
        df.sort_values('Distancem(m)')
        s = df.style.apply(ut.highlight_max,subset=list(col.keys()),axis=1)\
                    .apply(ut.highlight_min,subset=list(col.keys()),axis=1)
        s
```

```
Out[9]:
```

	Distancem(m)	GraphHopperContracted	Vanila	GraphHopper	Graphast	Neo4j
0	22525	33	49.7	106	226.7	502
1	31693	37.6	82.9	166.7	553.3	638.2
2	34659	35.5	100	191.3	673.8	677.2
3	13117	36.4	49.5	101.4	198.3	317.7
4	37356	31	93	180.2	706.7	803.7
5	14173	34.9	87.4	184.1	312.9	388
6	15503	31.5	32.8	76.5	119.4	175.7
7	30378	38.1	75.2	159.8	454.6	652.2
8	44594	32	109.3	203.7	696.5	875.2
9	33695	33.7	83.9	175.8	536.2	772.5
10	18268	35.2	83.4	132.3	333.8	194.5
11	36637	39.1	129.9	192.7	712.2	844.6
12	17353	37.8	102.3	189.4	533.1	554.1
13	37430	35	111.5	208.8	660.5	891.5
14	23796	35.3	62	124.6	245.7	409.7
15	12475	30.8	61.5	123.3	249.1	291.1
16	29693	34.5	114	174.4	511.2	510.5
17	45298	39.3	101.5	216.4	723.7	885.4
18	11777	37.2	62.3	118.9	220.2	341
19	21014	32.4	86.2	150.3	405.7	418.5
20	6112	36.6	57.1	77.8	114.4	155.7
21	27096	32.1	75.5	145.4	357.4	518.5
22	24849	33.1	83.7	131.8	307.9	443.9
23	24951	35.5	87.6	176.2	566.2	552.9
24	13391	33.7	78.9	164.8	370.6	431.8
25	13057	35	65.1	130.9	263.5	377.7
26	27302	39.2	72.9	151.5	356.8	534.9
27	47253	37.8	110.3	201.2	723.1	934.1
28	29434	34.7	75.8	159.2	402.4	541.3
29	20708	33.5	97	172	529	308.4
30	34775	38.1	104.5	203.7	738.9	816.9
31	17140	33.1	82	151.8	388.5	192.7
32	42641	36.9	121.6	195.5	733.6	881.3
33	19535	34.8	79.1	156.8	400	492.4
34	21494	39.6	111.4	193.9	603.5	367.3
35	36935	33.6	100.5	186.1	658.9	840.4
36	35668	33.2	92.4	179.2	605.9	789.4
37	20847	33.2	67.2	91.5	196.8	351.5
38	29358	34.8	113.3	217.3	557.6	618.3
39	25247	33.7	69.8	131.2	315.2	397.8
40	14814	32.3	83.1	166.1	383.1	476.7
41	20513	32.3	46	96.4	184.8	292.3
42	47990	38.8	107	206.8	666.5	847.6
43	29797	37.2	111.7	203.6	625.1	607.4
44	34017	31.8	96.2	184.7	575.6	823
45	31840	37.1	85.5	162.4	482.1	592.5
46	31207	38.1	87.2	165.4	460.6	682.9
47	16018	32.1	86.8	158.3	426.7	176.7
48	11931	35.2	62.3	136	252.9	371.2
49	16088	35.4	81.9	161.4	380.7	529.4
50	28648	37.1	78.6	153.2	430.4	473.3
51	32821	35.6	114.8	205.1	720.2	766
52	9789	39	40	83.2	153.6	118.3
53	26290	31.1	105	208.7	664.1	720.2
54	48253	35	118.6	210.6	702.7	843.7
55	31726	36.4	90.3	174.7	551.6	711.3
56	25255	34.8	71.1	131.6	307.4	459.1
57	27971	34.6	119.2	186.9	603.1	911.9
58	25203	39.3	115.5	195	746.1	454.9
59	7736	38.5	53.3	111	208	208.2
60	27958	32.3	75.9	155.5	396.9	424.1
61	25782	32.9	70.1	135.4	330.8	407.8
62	23789	37.7	63.9	123.8	257.5	411.2
63	26435	33.9	107.8	183.6	696.3	787.2
64	31866	38.2	90.9	165.4	505.8	703.4
65	35000	31	90.1	169.7	612.4	784.9
66	21642	32.8	99.5	180.3	504.7	339.7
67	14764	37.9	76.8	150.5	338.4	395.5
68	23241	34	111.4	192.9	575.2	382.6
69	36869	32.8	111.4	205.5	711	776.8
70	37931	38.3	109.4	209.9	669.2	731.1
71	15412	37.9	88.3	163	445.5	554.9
72	23997	35.7	112.6	205.7	577.4	430.2
73	16604	32.9	96.2	79.5	127.3	180.1
74	28491	35.2	110.7	206.5	660.8	539.9
75	47025	34.8	107.1	193.6	695.8	851.7
76	43397	32.5	106.7	201.8	636.2	961.9
77	24383	33.9	87.5	124.4	296.4	187.2
78	18859	35.6	101.2	179	564.4	527.6
79	25543	39.2	66	136	312.8	475.3
80	17094	37.2	101.8	180.3	605.7	506.8
81	45283	38.2	105.6	204.9	739	979.2
82	9205	34.6	54	80.6	123.6	233.7
83	40793	38.2	107.7	212.9	663.7	722.1
84	25670	33.6	102	186.6	595.8	426
85	27216	33.8	106	207.3	710.8	729.4
86	30759	32.4	94	169.3	467.2	661.2
87	43990	37.4	107.6	198.4	687.7	977.2
88	15963	35.4	82.5	150.6	403.9	186.3
89	28501	33.2	78.5	155.2	409.2	549.2
90	44059	33.6	109.3	190.9	693.6	864.4
91	22506	33.2	60.1	118.5	229.8	322.2
92	39560	32.7	108.9	189.3	705.8	828.3
93	25801	32.4	116.3	224.7	695.8	489.7
94	30717	37.5	110.9	201	701.9	643.7
95	34323	29.9	110.3	199.7	696.8	804.1
96	41318	33.8	106	213	707.7	965
97	24728	32.1	67	122.9	276.4	418.4
98	28056	31.5	100	183.4	701.2	550.8
99	21620	39.1	95.4	199.2	600	302.4