Project 2 Report

Sara Djambazovska

The aim of this Project 2 was to implement Locality Sensitive Hashing scheme to optimize the similarity checks between movies. I compared 3 different LSH implementation together with the baseline – the Exact Near Neighbor operator.

The Exact NN operator (given the Jaccard distance threshold which I set to 0.3 in the tests) checks simply the cartesian product between the query and the original corpus data, and then does a pairwise Jaccard similarity check between each movie and its keywords. Of course, it performs best in terms of accuracy and precision, but is very slow with O(N*M) computations, where N and M are the tuples in the first and second set respectively, causing this operator to even time out on the cluster for the large datasets.

The LSH operators avoid the cartesian product by making use of MinHash, computing it once for each element for each dataset, and then do the join on them (first separate into partitions for the Balanced version). The Broadcast version implementation further improves time performance as it avoids unnecessary shuffles of join operation, by broadcasting a map of values in the corpus dataset.

Execution time comparison

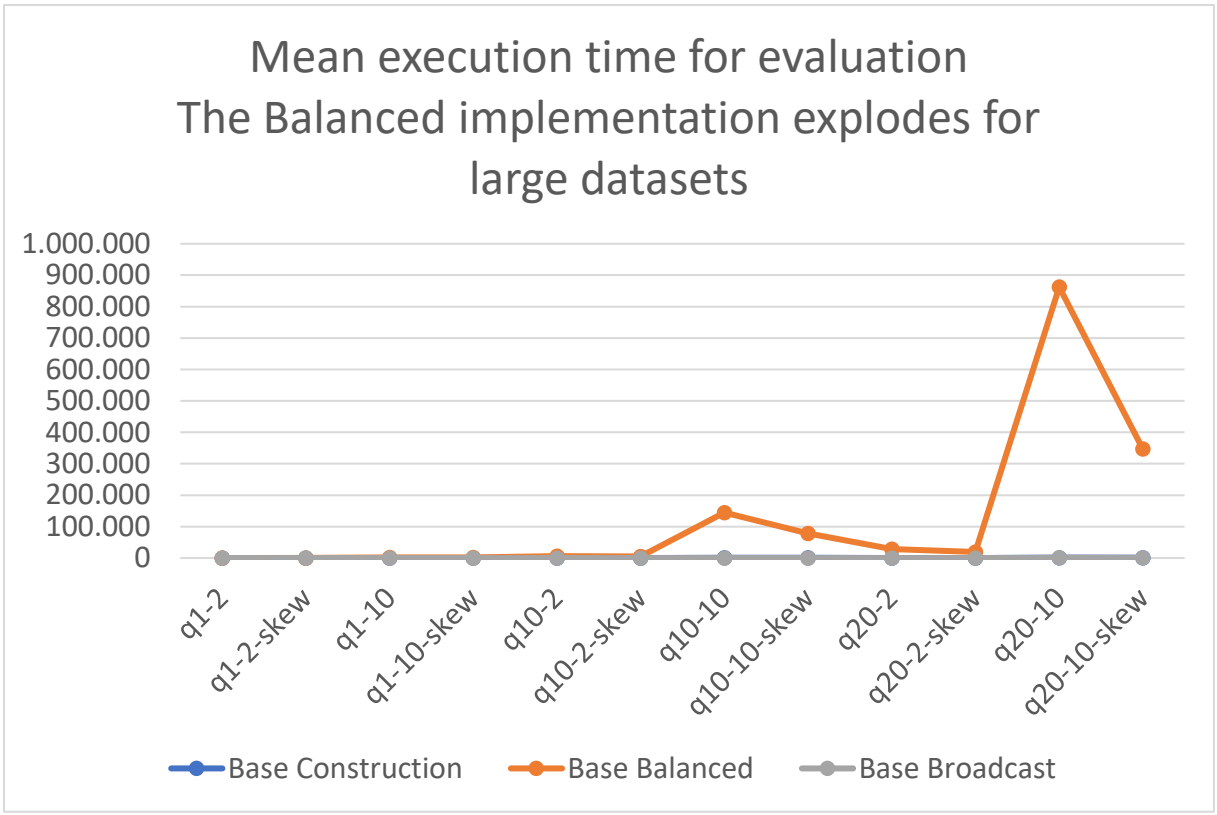
Query	Base Construction		Base Balanced		Base Broadcast		Exact NN
	Mean	***Std***	***Mean***	***Std***	***Mean***	***Std***	***Mean***
Query1-2	219	303.94	407	166.14	41	8.1	28573
Query1-2-skew	101	20.9	383	77.5	44	5.6	43286
Query1-10	164	22.9	1939	479.6	67	13.6	143039
Query1-10-skew	144	18.1	1638	222.8	66	9.7	155372
Query10-2	244	21.8	5610	1123,1	169	10.3	4327050
Query10-2-skew	260	35.3	5312	1497.9	147	16.9	
Query10-10	757	68.6	144286	33774.1	404	40.7	
Query10-10-skew	543	48.7	78203	19681.3	365	24.7	
Query20-2	440	28.1	28960	8266.5	305	34.1	

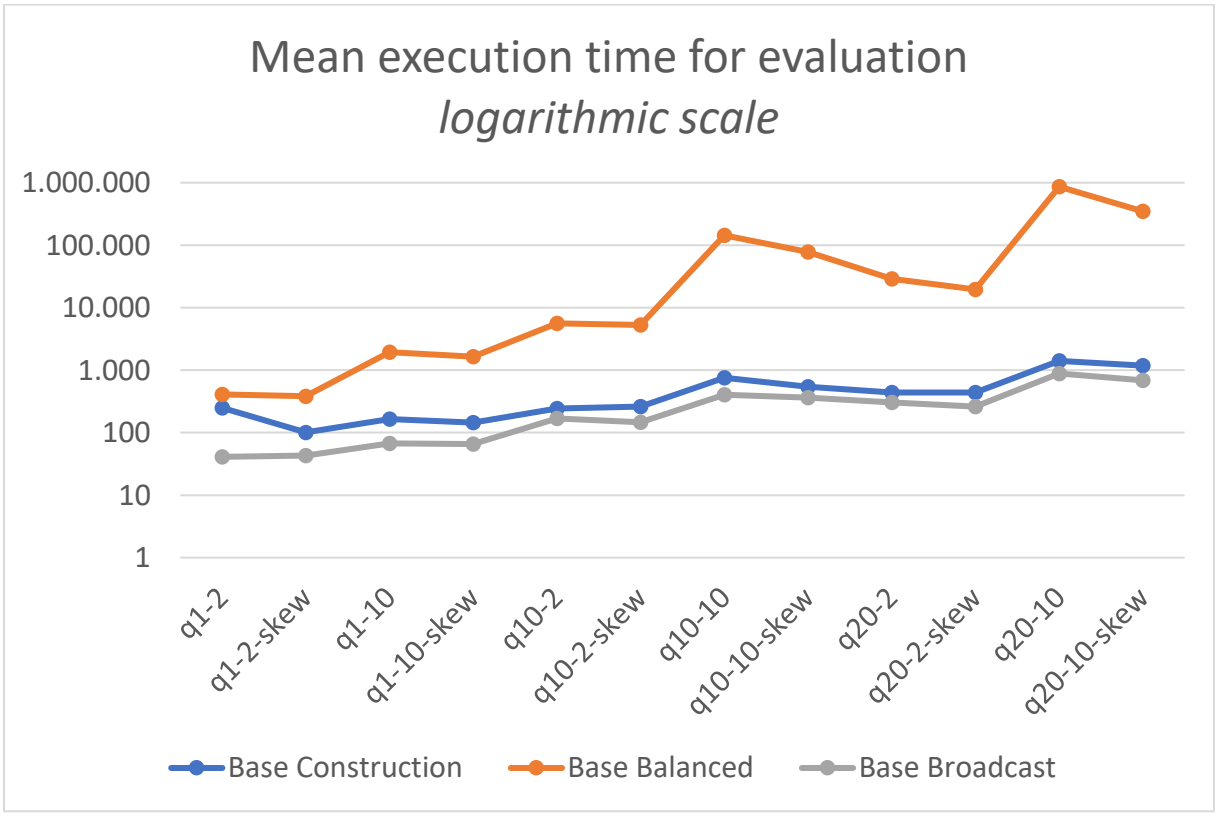
Query20-2-skew	440	47.9	19538	5174.4	260	31.3
Query20-10	1406	59.5	862216	401161.2	884	65.0
Query20-10-skew	1176	103.5	347362	96742.2	683	35.5

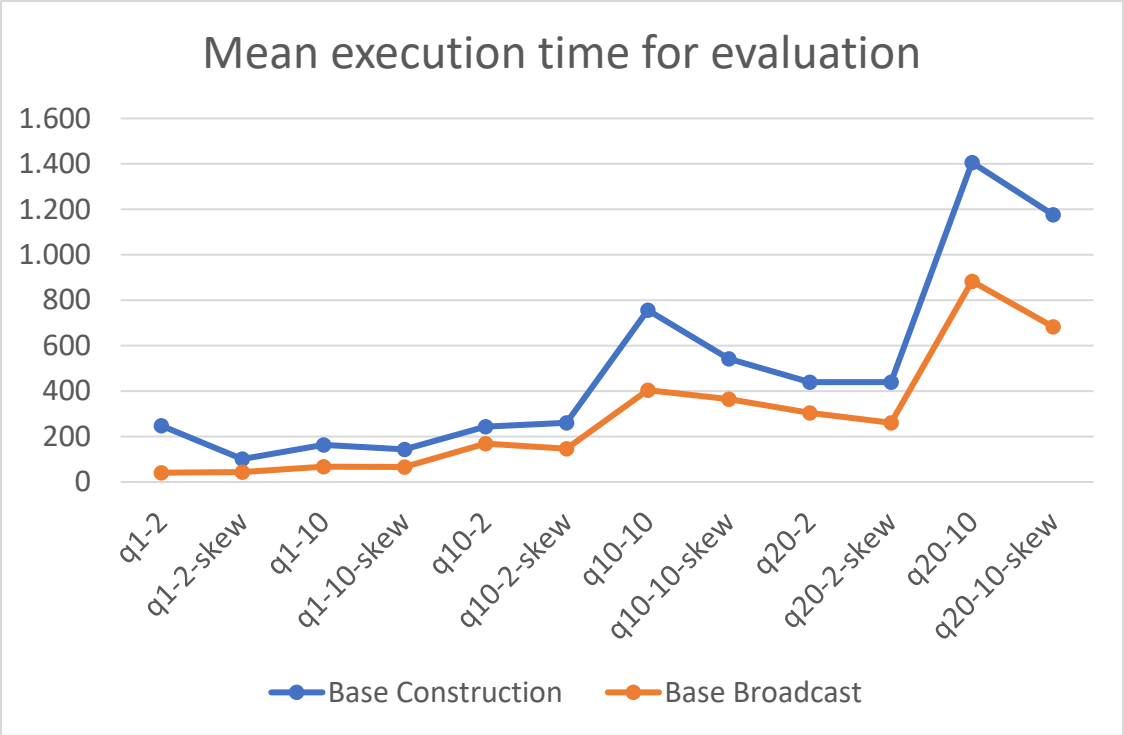
Table showing mean and standard deviation of execution times (without initialization) of the different constructions ran for 10 iterations on the cluster

As expected, from the experiments with different sized datasets, we can see that the Exact NN takes far more time to execute than any of the LSH implementations, and as the set gets bigger the number of operations and time to execute Exact NN grows much faster in comparison to the slower growth for LSH.

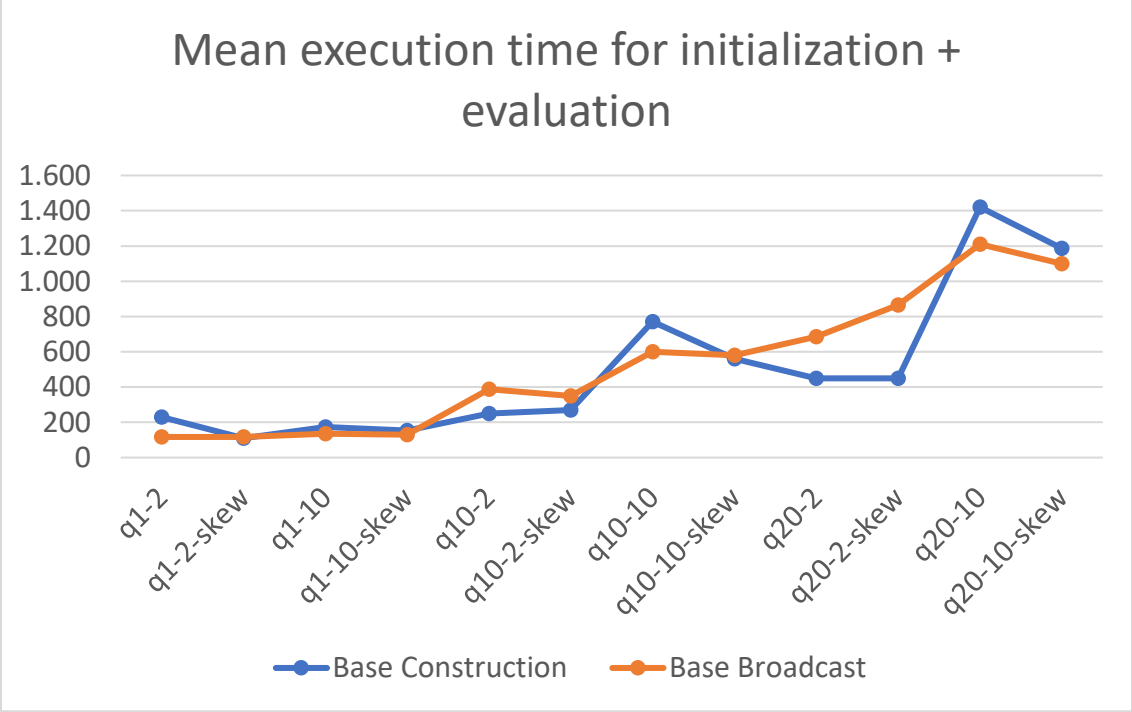
Comparing the different LSH implementations, as illustrated in the table above, we can see that for a small dataset we get closer execution times for both Base and Broadcast (while Exact NN is hundreds of times slower even for the corpus-1), however as the dataset grows, the execution time for Broadcast is visibly smaller. This is mostly because all the MinHash computations are performed locally by the workers, thanks to the broadcasted maps. This avoids extra shuffling and data transfers between partitions. We can also see that the standard deviation of the time execution of the Broadcast implementation is smaller for all queries, making it the most stable. The execution times for Base Construction Balanced are far higher than the other two implementations as load balancing introduces overhead (one extra shuffle/aggregation and possibly slow implementation of the get_Partition method)). The balanced implementation is about 2-3 times slower for the small dataset but gets up to hundreds of times slower than the Base and Broadcast versions for bigger data sets.


Mean execution time for evaluation
The Balanced implementation explodes for large datasets
1.000.000
900.000
800.000
700.000
600.000
500.000
400.000
300.000
200.000
100.000
0
q1-2
q1-2-skew
q1-10
q1-10-skew
q10-2
q10-2-skew
q10-10
q10-10-skew
q20-2
q20-2-skew
q20-10
q20-10-skew
Base Construction
Base Balanced
Base Broadcast


Mean execution time for evaluation
logarithmic scale
1.000.000
100.000
10.000
1.000
100
10
1
q1-2
q1-2-skew
q1-10
q1-10-skew
q10-2
q10-2-skew
q10-10
q10-10-skew
q20-2
q20-2-skew
q20-10
q20-10-skew
Base Construction
Base Balanced
Base Broadcast

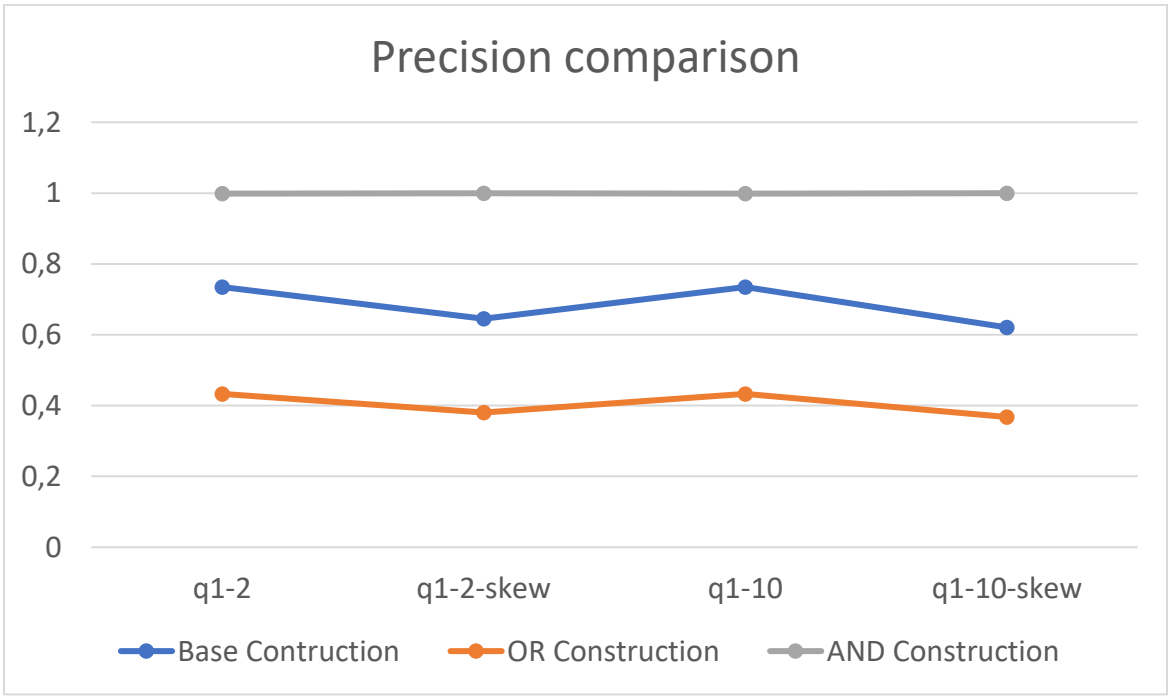


I also calculated the initialization time for each operator and as expected, it is negligible for the Base and Balanced implementation, but more pronounced in the Broadcast implementation due to the broadcast that is done in this initialization phase. This makes the Base LSH operator faster for some larger datasets, as seen below, when the execution time is taken as the time to initialize the LSH instance + one call to eval for the given query.

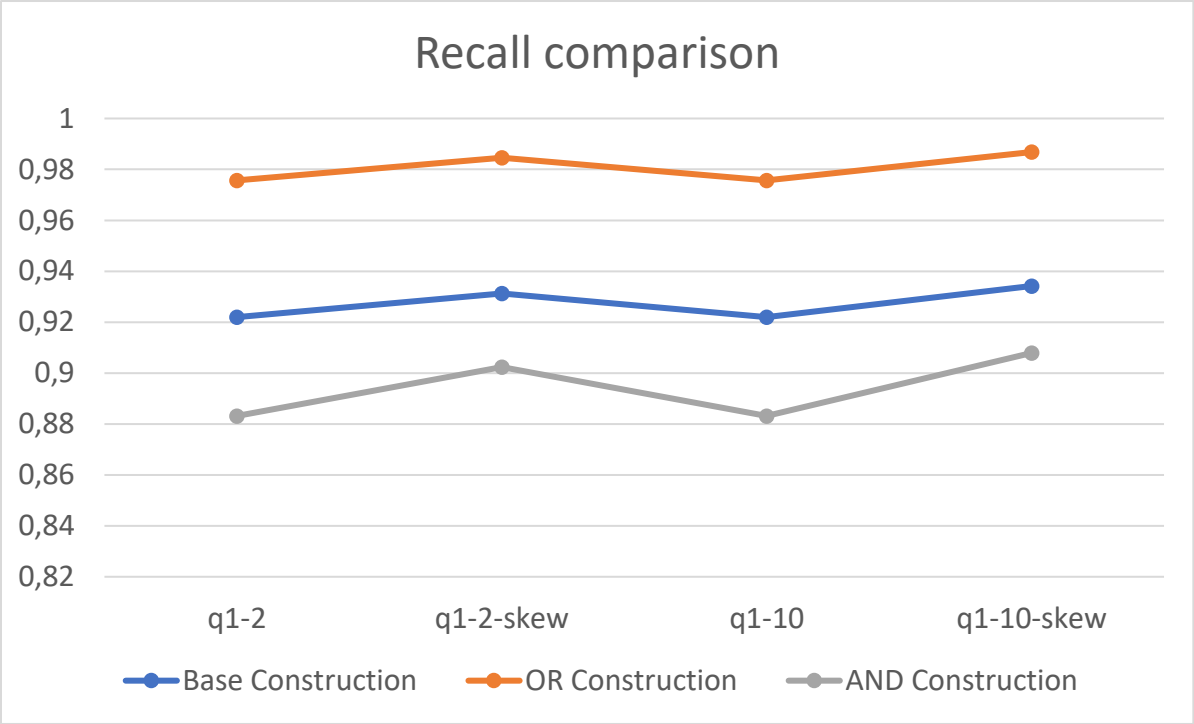


Accuracy comparison

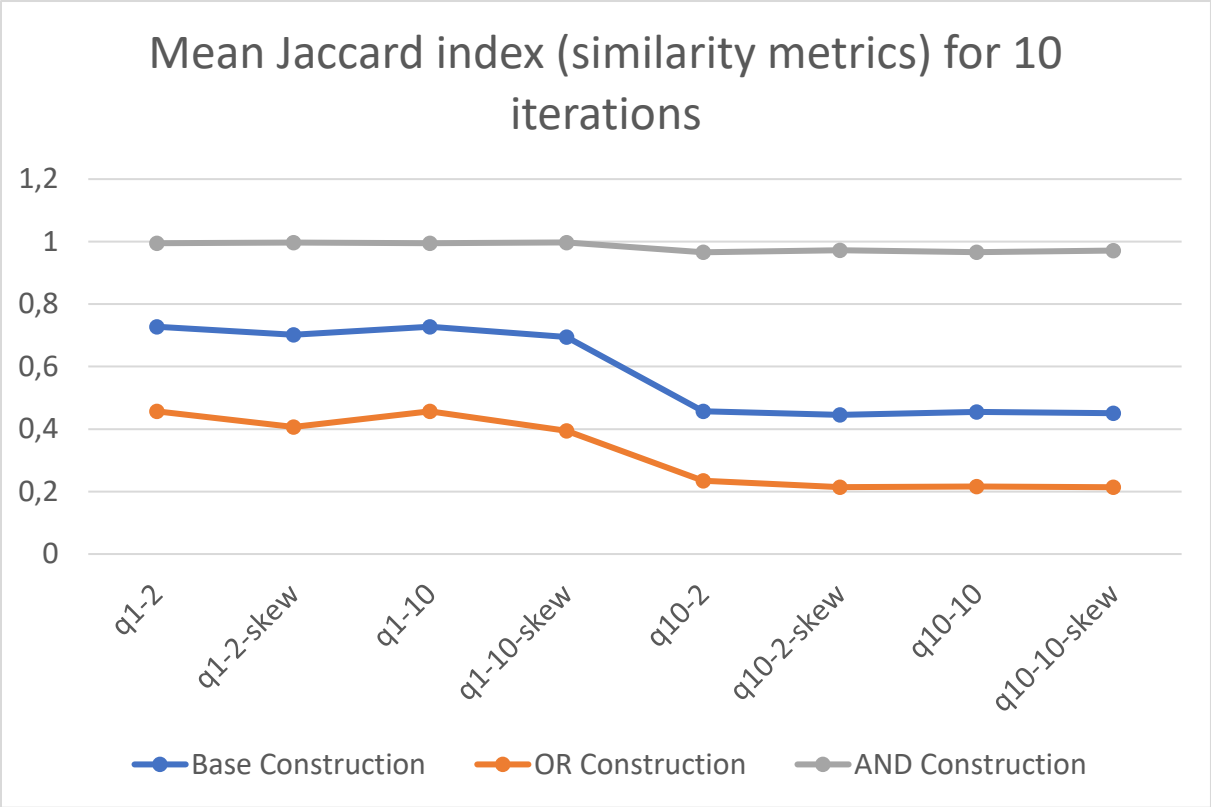
To evaluate the goodness of these operators I calculated (averaged for 10 different runs) the precision and recall of their outputs compared to the correct result – the Exact NN output and the average Jaccard similarity (1 minus the nearest neighbor distance) for each of their outputs. All the LSH operators give the same results on these metrics, as they are basically calculating the same thing only in different ways. The precision obtained for the LSH implementations seemed quite poor (<0.8) with a good recall (>0.92). Knowing that the precision is the true positives divided by the sum of true positives and false positives, decreasing the number of false positives should increase the precision. As expected, an AND construction (of 4 Broadcast instances) has an almost perfect precision, as it decreases the false positives.



However, this construction is prone to false negatives, thus it yields a smaller recall (which is defined as the true positives over the sum of the true positives and the false negatives). As seen in figure below using an OR construction with 4 Broadcast instances (which reduces the number of false negatives) increases the recall but decreases the precision as it is more sensitive to false positives. Ideally a combination of these two operators should yield a good precision and recall, however that was beyond the requirements for this project :)

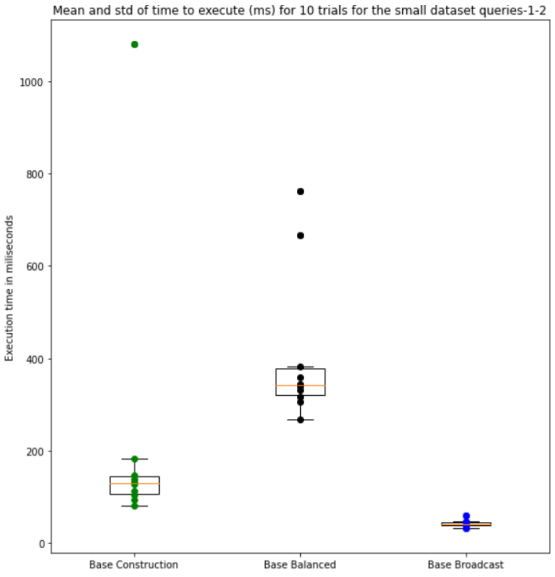


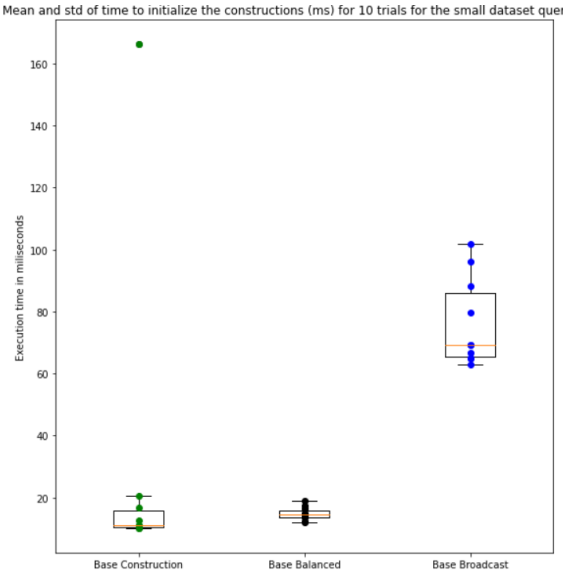
I also calculated the average Jaccard similarity for the LSH operators and the two constructions for 10 runs. As we can see the best results are obtained with the AND construction, as it has the highest precision, it gives the highest similarity, thus smallest distance. And similarly, the OR construction (low precision) yields the lowest similarity.

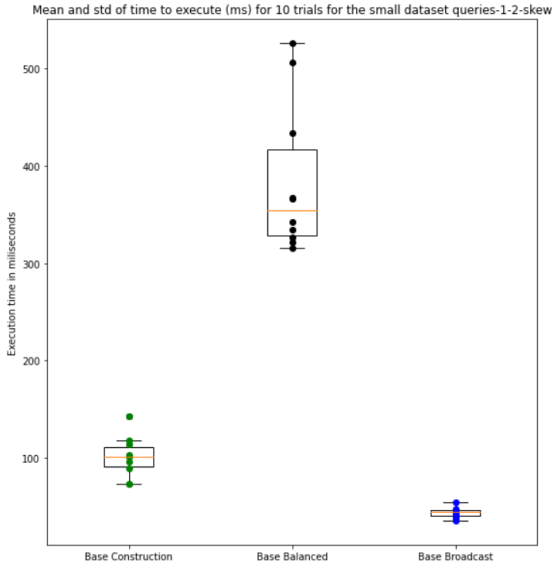


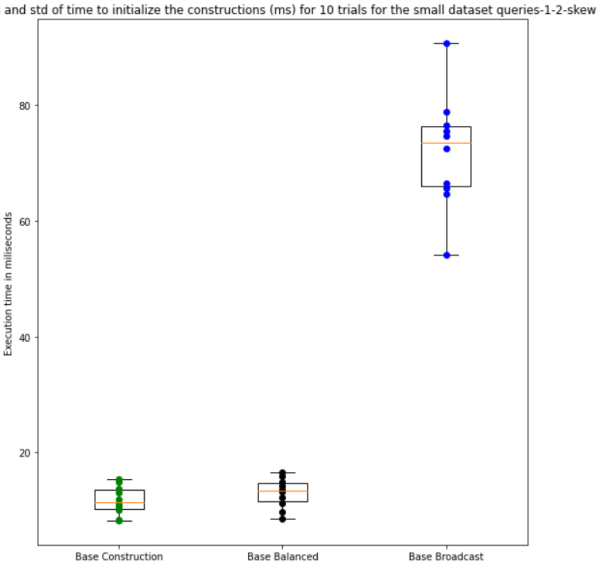
In conclusion, considering these results, we can see that the Broadcast implementation is the most performant when evaluating these large query sets. The Base construction gives similar or better results for some of the datasets, if after the initialization we only evaluate **one** query (due to the Broadcast initialization overhead, which would be negligible if we evaluate more queries). Thus, using a combination of an AND construction and OR construction of Base Construction Broadcast instances is expected to yield the optimal precision, recall and Jaccard similarity results.

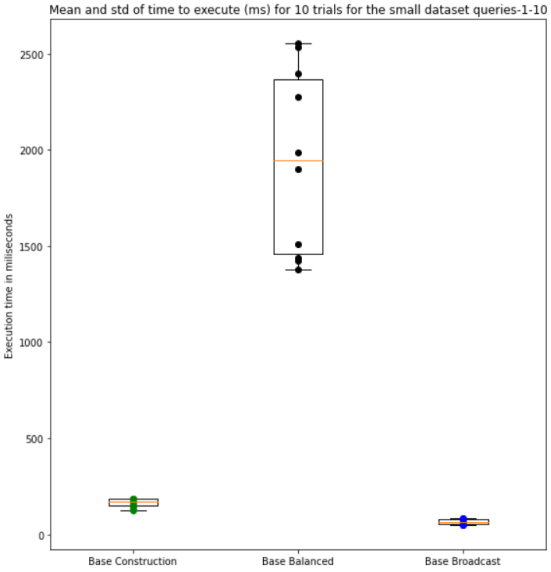
Below are some plots of mean and std of execution times (left for time to execute an “eval” call, right for instance initialization time) for the 10 runs for each query.

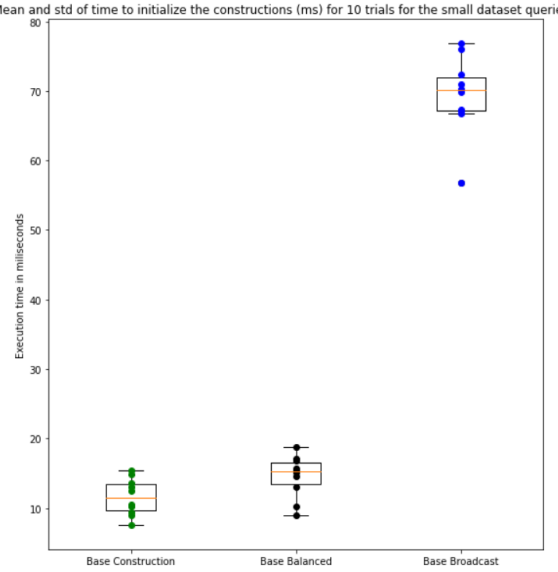

Mean and std of time to execute (ms) for 10 trials for the small dataset queries-1-2
Execution time in milliseconds
0
200
400
600
800
1000
Base Construction
Base Balanced
Base Broadcast

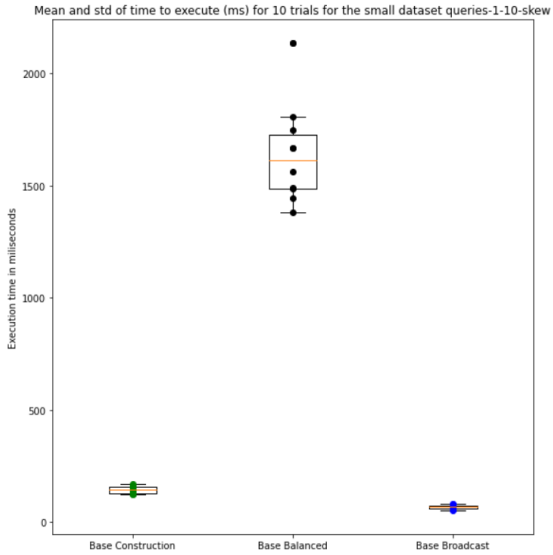

Mean and std of time to initialize the constructions (ms) for 10 trials for the small dataset queries-1-2
Execution time in milliseconds
20
40
60
80
100
120
140
160
Base Construction
Base Balanced
Base Broadcast

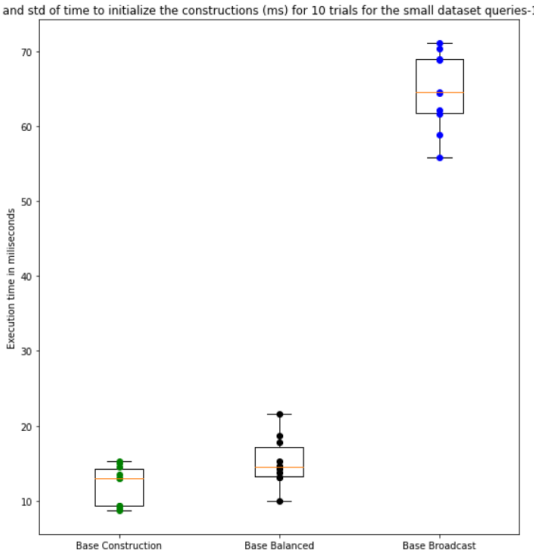

Mean and std of time to execute (ms) for 10 trials for the small dataset queries-1-2-skew
Execution time in milliseconds
100
200
300
400
500
Base Construction
Base Balanced
Base Broadcast

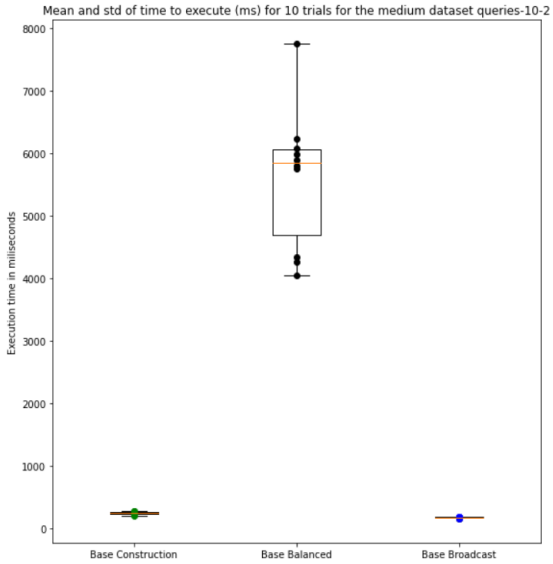

Mean and std of time to initialize the constructions (ms) for 10 trials for the small dataset queries-1-2-skew
Execution time in milliseconds
20
40
60
80
Base Construction
Base Balanced
Base Broadcast

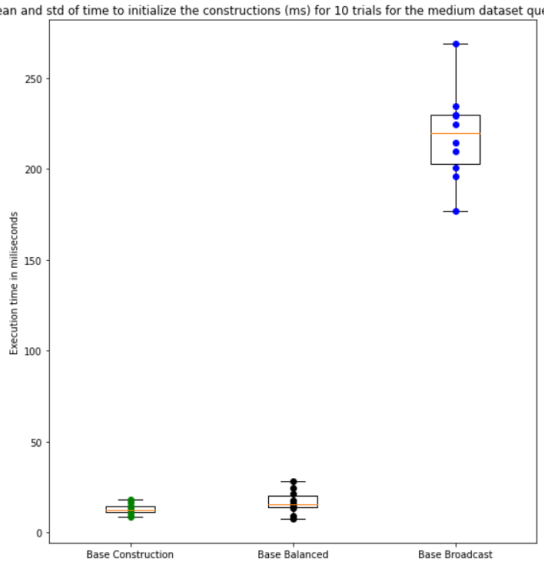

Mean and std of time to execute (ms) for 10 trials for the small dataset queries-1-10
Execution time in milliseconds
0
500
1000
1500
2000
2500
Base Construction
Base Balanced
Base Broadcast

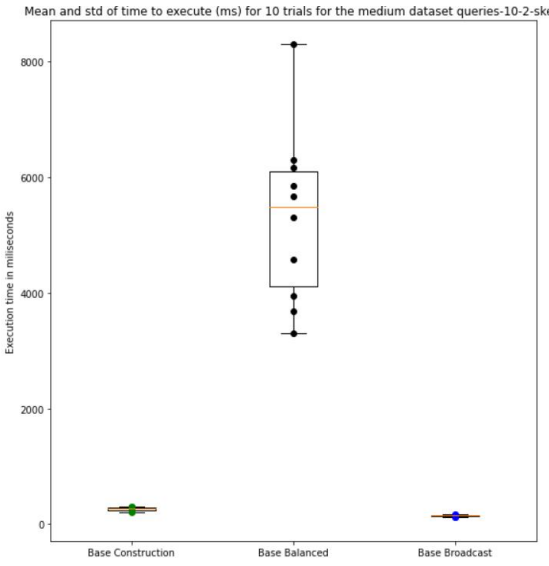

Mean and std of time to initialize the constructions (ms) for 10 trials for the small dataset queries-1-10
Execution time in milliseconds
10
20
30
40
50
60
70
80
Base Construction
Base Balanced
Base Broadcast

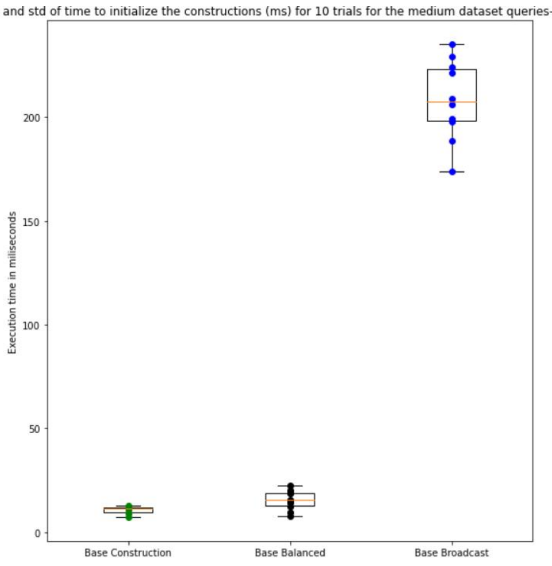

Mean and std of time to execute (ms) for 10 trials for the small dataset queries-1-10-skew
Execution time in miliseconds
0
500
1000
1500
2000
Base Construction
Base Balanced
Base Broadcast

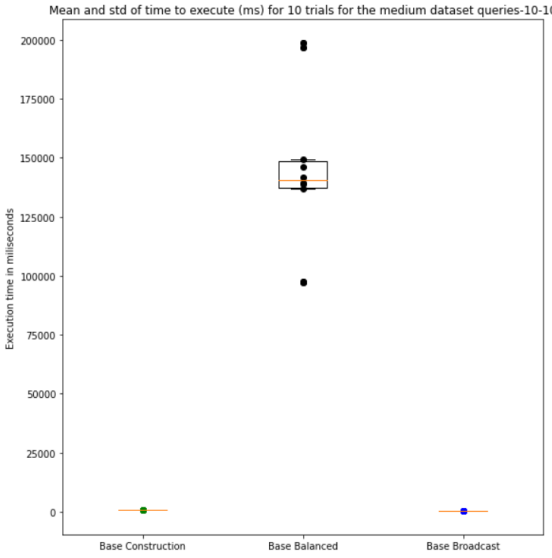

Mean and std of time to initialize the constructions (ms) for 10 trials for the small dataset queries-1-10-skew
Execution time in miliseconds
10
20
30
40
50
60
70
Base Construction
Base Balanced
Base Broadcast

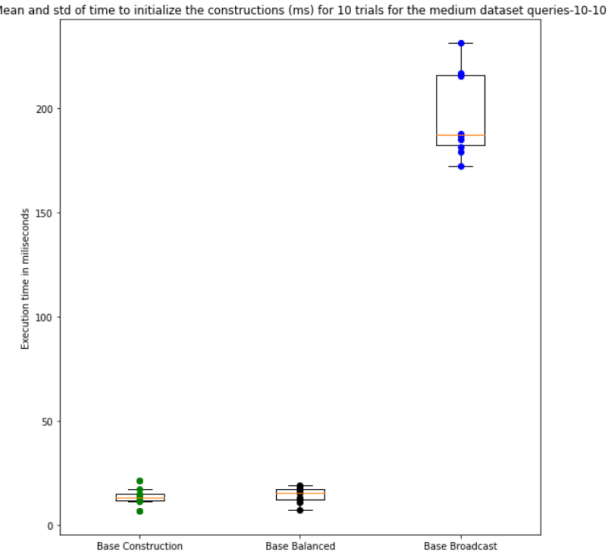

Mean and std of time to execute (ms) for 10 trials for the medium dataset queries-10-2
Execution time in miliseconds
0
1000
2000
3000
4000
5000
6000
7000
8000
Base Construction
Base Balanced
Base Broadcast

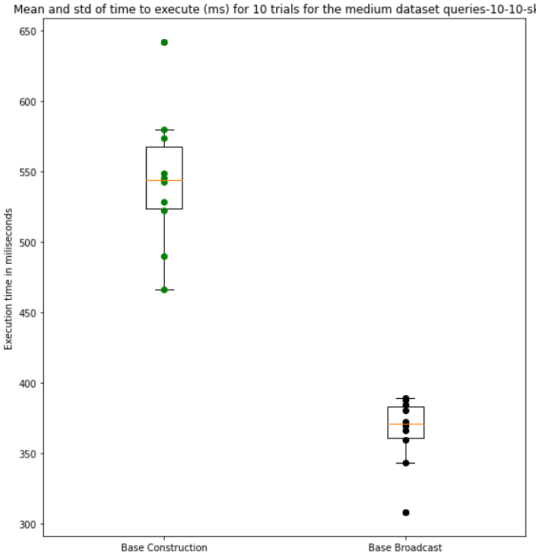

Mean and std of time to initialize the constructions (ms) for 10 trials for the medium dataset queries-10-2
Execution time in miliseconds
0
50
100
150
200
250
Base Construction
Base Balanced
Base Broadcast

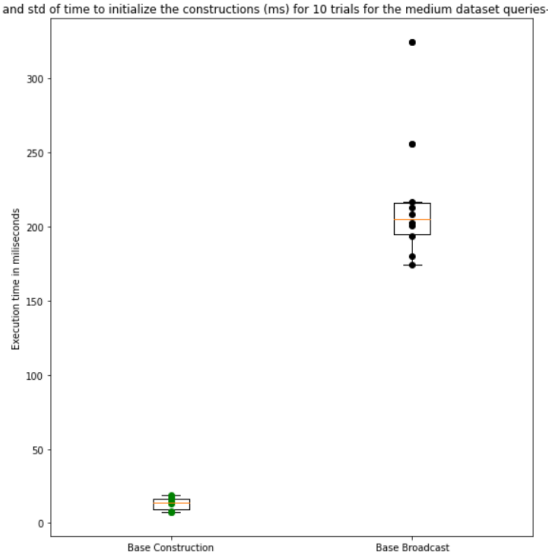

Mean and std of time to execute (ms) for 10 trials for the medium dataset queries-10-2-skew
Execution time in miliseconds
0
2000
4000
6000
8000
Base Construction
Base Balanced
Base Broadcast

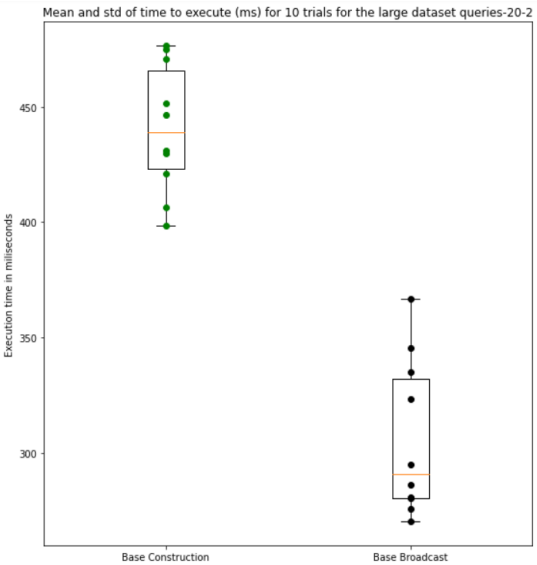

Mean and std of time to initialize the constructions (ms) for 10 trials for the medium dataset queries-10-2-skew
Execution time in miliseconds
0
50
100
150
200
Base Construction
Base Balanced
Base Broadcast

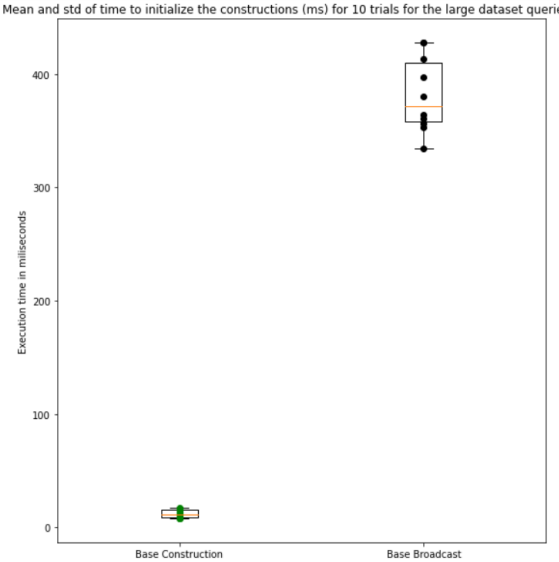

Mean and std of time to execute (ms) for 10 trials for the medium dataset queries-10-10
Execution time in milliseconds
200000
175000
150000
125000
100000
75000
50000
25000
0
Base Construction
Base Balanced
Base Broadcast

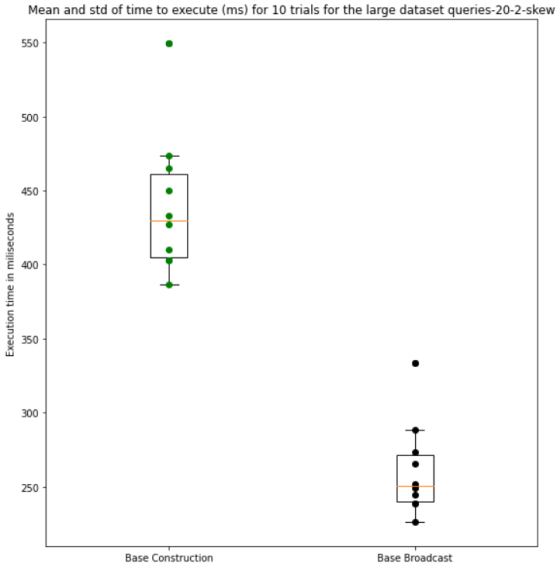

Mean and std of time to initialize the constructions (ms) for 10 trials for the medium dataset queries-10-10
Execution time in milliseconds
200
150
100
50
0
Base Construction
Base Balanced
Base Broadcast

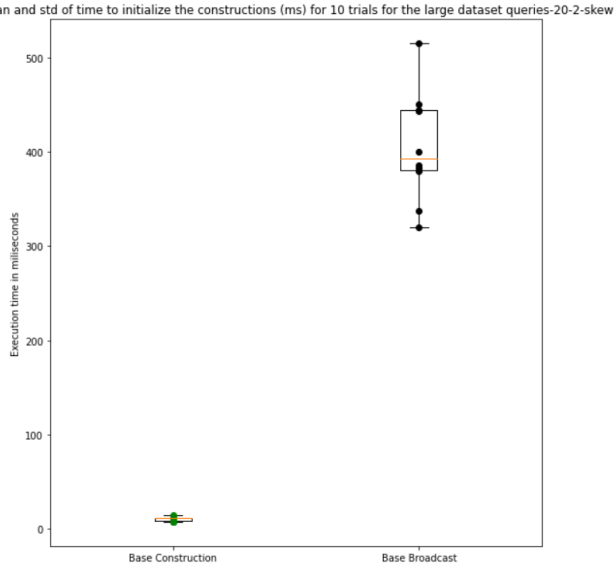

Mean and std of time to execute (ms) for 10 trials for the medium dataset queries-10-10-skew
Execution time in milliseconds
650
600
550
500
450
400
350
300
Base Construction
Base Broadcast

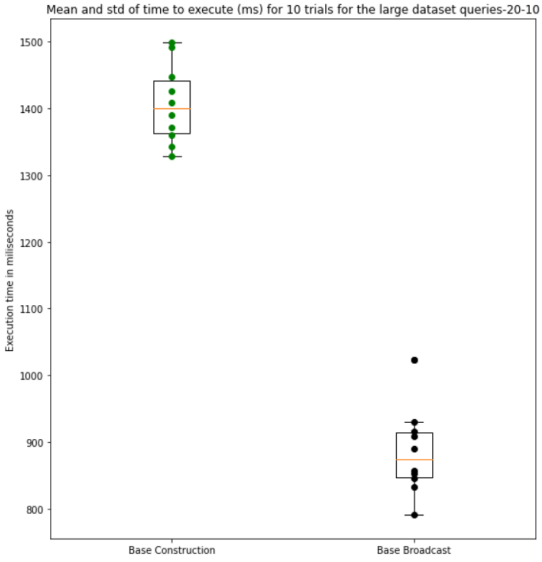

Mean and std of time to initialize the constructions (ms) for 10 trials for the medium dataset queries-10-10-skew
Execution time in milliseconds
300
250
200
150
100
50
0
Base Construction
Base Broadcast

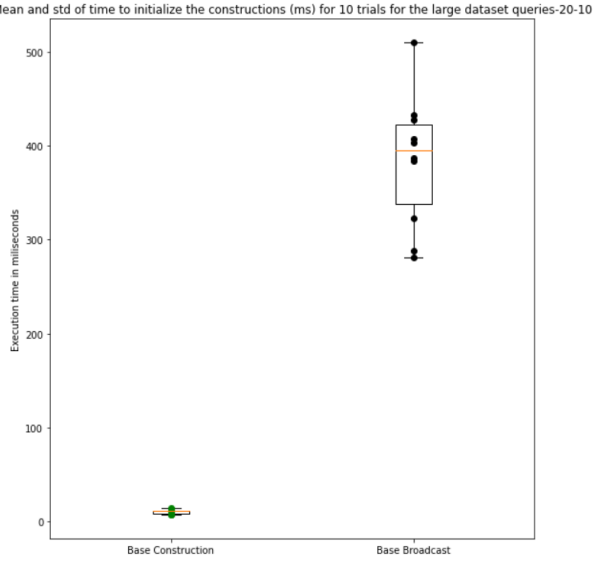

Mean and std of time to execute (ms) for 10 trials for the large dataset queries-20-2
Execution time in milliseconds
450
400
350
300
Base Construction
Base Broadcast

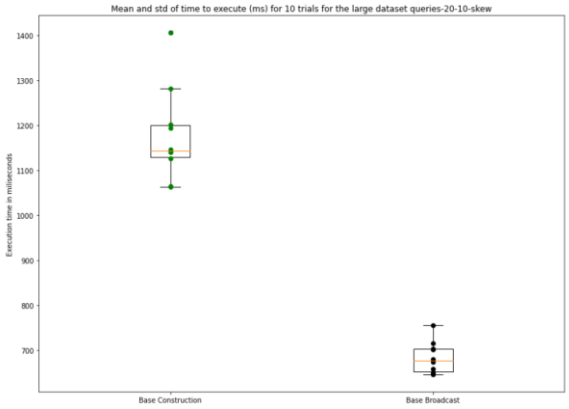

Mean and std of time to initialize the constructions (ms) for 10 trials for the large dataset queries-20-2
Execution time in milliseconds
400
300
200
100
0
Base Construction
Base Broadcast

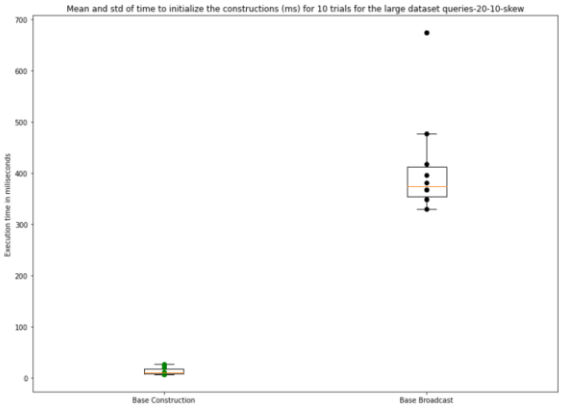

Mean and std of time to execute (ms) for 10 trials for the large dataset queries-20-2-skew
Execution time in milliseconds
550
500
450
400
350
300
250
Base Construction
Base Broadcast


Mean and std of time to initialize the constructions (ms) for 10 trials for the large dataset queries-20-2-skew
Execution time in milliseconds
500
400
300
200
100
0
Base Construction
Base Broadcast


Mean and std of time to execute (ms) for 10 trials for the large dataset queries-20-10
Execution time in milliseconds
1500
1400
1300
1200
1100
1000
900
800
Base Construction
Base Broadcast


Mean and std of time to initialize the constructions (ms) for 10 trials for the large dataset queries-20-10
Execution time in milliseconds
500
400
300
200
100
0
Base Construction
Base Broadcast


Mean and std of time to execute (ms) for 10 trials for the large dataset queries-20-10-skew
Execution time in milliseconds
1400
1300
1200
1100
1000
900
800
700
Base Construction
Base Broadcast


Mean and std of time to initialize the constructions (ms) for 10 trials for the large dataset queries-20-10-skew
Execution time in milliseconds
700
600
500
400
300
200
100
0
Base Construction
Base Broadcast