Experimentelle Algorithmik

Übung 1

- Niklas Neesemann *(Matrikelnr. 609789)*
- Leonhard Thyssen *(Matrikelnr. 573374)*

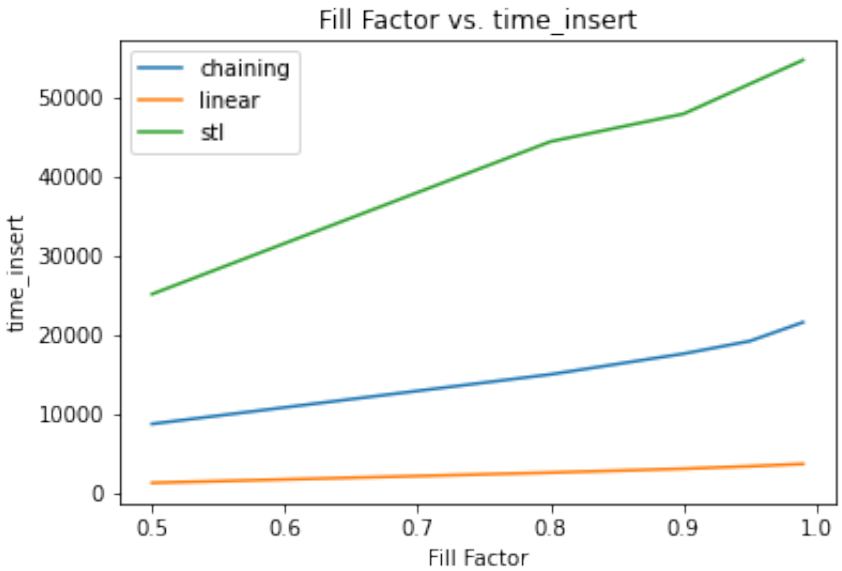
Github Repo: here

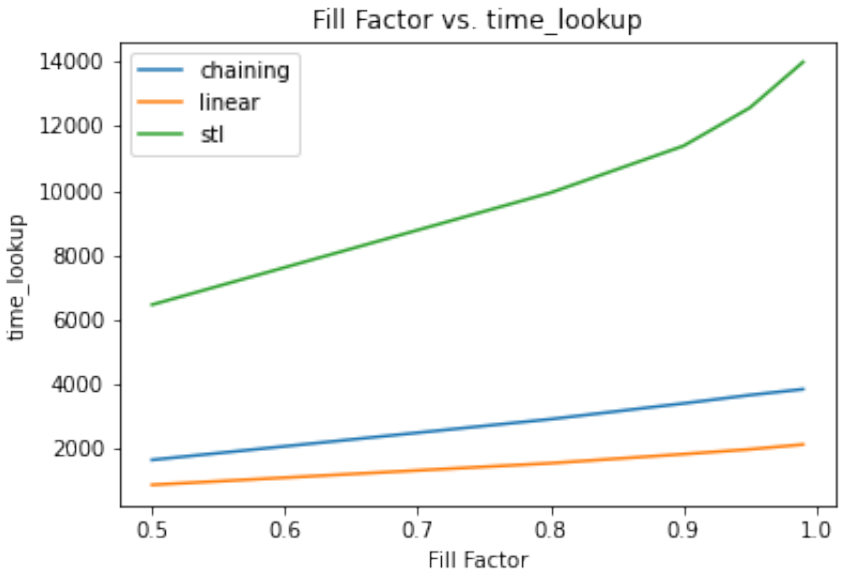
1.)

1.a)

See the simex archive: data.tar.gz

1.b)





2.)

2.a)

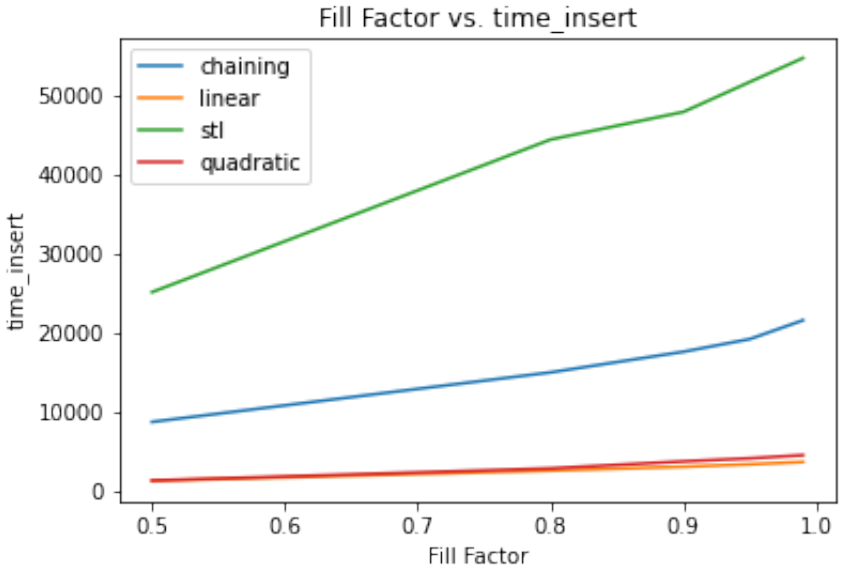
See the Quadratic Probing Implementation here (starts at line 161)

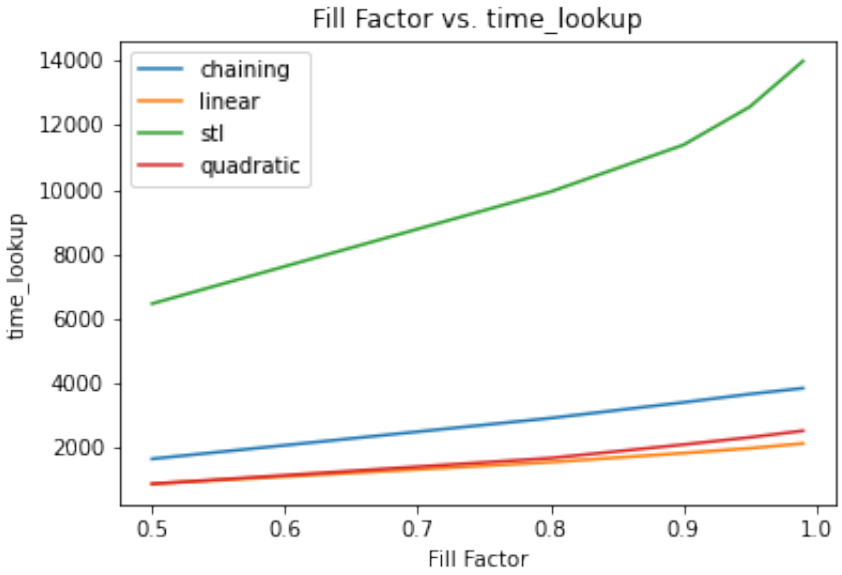
2.b)

See the raw `perf stat` cache misses results here

The number of cache misses grows in the following order (independent of the fill-factor [0.25, 0.50, 0.75, 0.90]): Chaining, linear, quadratic. This lies in the nature of the algorithms and how the cache works. The cache can grab a certain amount of bytes into its memory and when we access a cell/node it can grab the bytes around it, which makes another access much faster. Chaining stores elements with the same hash in a (linked) list, which might cause the elements to lie next to each other in the memory. So the next element that will be traversed could already be in the cache memory. Linear and especially quadratic probing, in contrast, hashes to (a completely) different index, hence to a different location in memory, which is why the next accessed cell is not in the cache.

2.c)





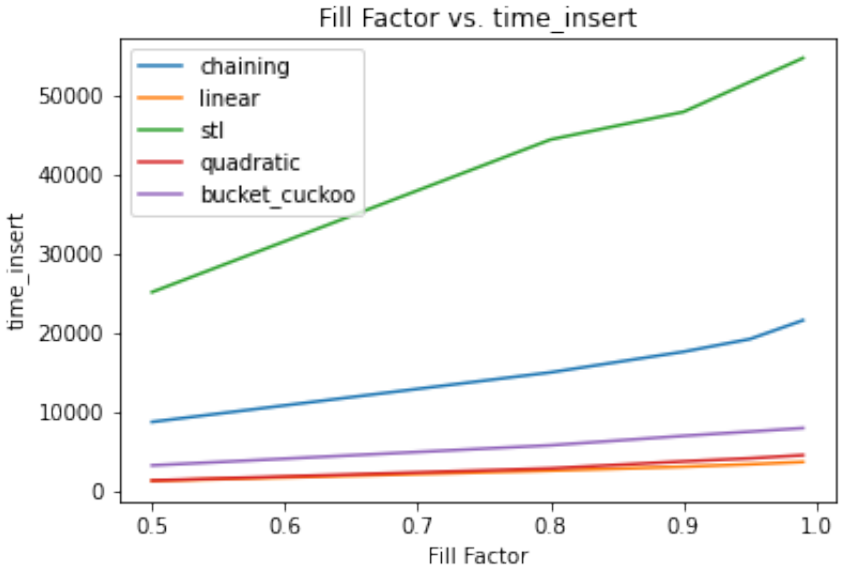
Compared to linear probing and chaining it performs poorly.

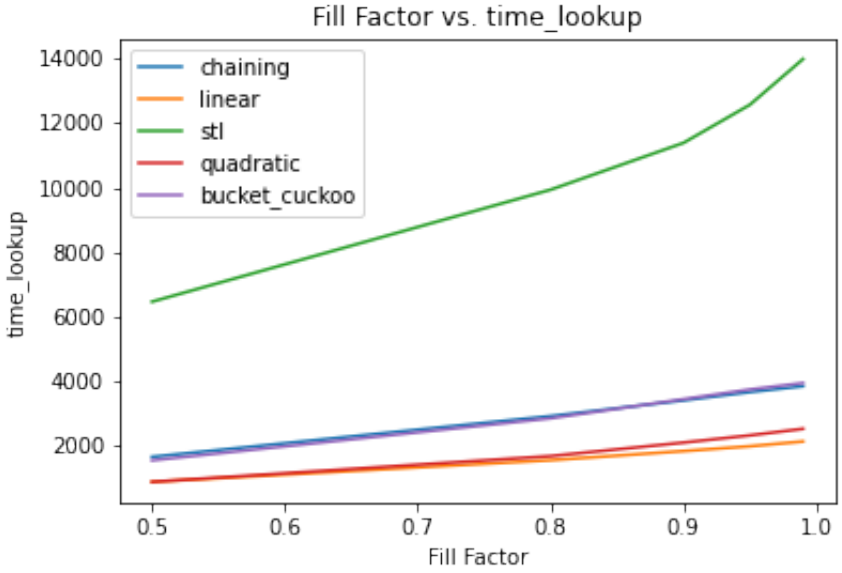
3.)

3.a)

We have chosen a threshold of log2 (M) for the maximal length of the eviction chain. This link leads to the file with the implementation (starts at line 225).

3.b)


Fill Factor vs. time_insert
chaining
linear
stl
quadratic
bucket_cuckoo
time_insert
0
10000
20000
30000
40000
50000
0.5
0.6
0.7
0.8
0.9
1.0
Fill Factor


Fill Factor vs. time_lookup
chaining
linear
stl
quadratic
bucket_cuckoo
time_lookup
2000
4000
6000
8000
10000
12000
14000
0.5
0.6
0.7
0.8
0.9
1.0
Fill Factor