Experiment 1

Introduction:

The kmeans clustering algorithm is working with a randomization that is triggered based on the number of misses. Misses are when the current solution is not better than the best solution found so far(see figure 1). The number of misses after which a random solution is chosen was set arbitrarily in the beginning because sometimes in a difficult district (3 at example) no greedy or random connection could be made for a long time so this is trying to get the algorithm out of a dead end. Becuase initially these parameters were set to an arbitrary value it is tested whether different settings have an effect on how fast a good solution is found and the best solution found.

Method:

First the randomness parameters where set to 20 and 10, so if in 5% of the total runtime (see figure 1: if miss < numIt/20) nothing could be found a random competitor from the history is chosen and when after 10% of iterations still nothing happens the district is reconnected randomly. To test which parameters work best a range of these with a 2:1 ratio was tried out.

To get a reliable measure the program was executed 10 times with 300 iterations for each setting. The average of the 10 runs then was plotted to find a difference in how fast a good solution was found. To test if some setting can find better solutions than the others a t-test was used to test for differences in mean values of all 10 runs per setting. Because of the low N of 10 and the therewith low power an α-level of 0.1 was chosen. Plots for the best districts found by each parameter can be found in the experiment folder.

Results:

Regarding the efficiency, thus how fast a setting finds a reasonable good optimum, it seems that there is almost now difference at all between different settings. In figure 3 can be seen that the differences are almost negligible. However it seems that 20/40 and 80/160 seem the most promising because they produce a good solution (rated by visual inspection with the ellbow-method) is reached (marginally) faster and also yield a better overall-result in the end. On could argue that these changes will cancel out with a high number of iterations. When testing the best solution achieved after 300 iterations none of the differences between group and total means were significant .

However when testing the different parameters of random against no randomization the 20/40 ($p < .1$) and the 80/160 ($p < .1$) parameters showed a significant improvement in comparison to when the random method was not used.

Conclusion:

Even when the evidence is small, there is some evidence for the neccesity of using a random method within kmeans. The best performance (fastest descend in price and best final price) could be achieved when after *numIt/160* misses a random contestant is chosen for the next iteration and after *numIt/80* misses the district is randomized by making random connections.

```
random = [5,10]
contestants.sort(key = lambda x: x.costs)
if district.costs <= contestants[0].costs:
    if plot == "winner":
        visualize(district, True, count)
# If the solutionis not better than the best so far its a miss
else:
    miss += 1
    # If there are more misses in a row than 10% of the number of
    # iterations the list is shuffled so that later a random contestant
    # is chosen for the next iteration
    if miss > numIt /random[1]:
        shuffle(contestants)
        # If this still does not yield better results in 20% of the
        # number of iterations the district is randomly reconncted
        if miss > numIt / random[0]:
            miss = 0
            district.disconnect()
            district.connectRandom()


# Here the district for the next iteration is chosen
district = deepcopy(contestants[0])
```

Figure 1: Randomizing the beginstate for next iteration if no good solutions are found

```
random = [[5,10],[10,20],[20,40],[40,80],[80,160],[160,320],[320,640]]
```

Figure 2: All tried settings for random

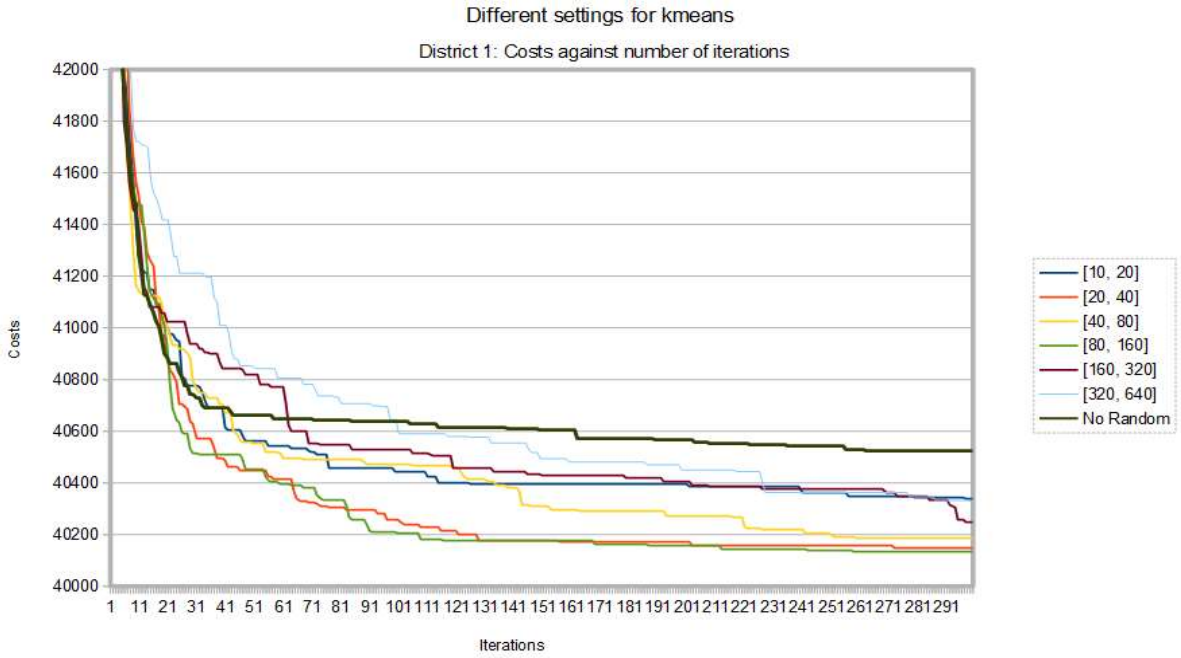

Figure 3: Different random settings for kmeans

	P-value	
settings	Against all	against no random
[5, 10]	0,30	0,10
[10, 20]	0,28	0,25
[20, 40]	0,18	**0,09**
[40, 80]	0,27	0,11
[80, 160]	0,14	**0,08**
[160, 320]	0,43	0,14
[320, 640]	0,28	0,23
No Random	0,15	

Table 1: results ttests