Performance of Optimizing People's location

Given a zoning plan, optimizing people's location increases the overall average score based on the people's preferences, which indicates that most people are satisfied with the zoning plan. One algorithm for this optimization is to swap a person A and a person B if the swapping increases the resulting score. By repeating this swapping, it will eventually converge to the optimal. The proof of this convergence is quite obvious. Since we allow only swapping the locations, the combination of people's location is finite. For every swapping, the score increases. However, since we are handling a large number of people, the running time is quite long. For instance, for around110,000 people it takes more than 5 minutes to converge.

Performance improvement

	Avg. running time for 10,000 swapping
Single thread	> 1 minutes
16 threads	9 seconds
CUDA	0.065 seconds

First, I tried multi-threading to speed up, which achieved around **9 seconds** for one iteration with 16 threads. Next, I used CUDA and the average computation time for one iteration becomes **0.065 seconds**.

Hierarchical optimization

While it is possible to increase the performance by using multi-threading or CUDA, I also found that the variance in the scores by the people's different spatial distribution for a given zoning plan is smaller than the one by the different zoning plans. Thus, it is better and efficient to employ the hierarchical approach where we first optimize the zoning plans and then optimize the people's locations.

Performance of People Allocation

Even if we employ the hierarchical approach, we still allocate people at least one time in order to compute the score of the given zoning plan. Although it does not take long compared to the optimization of people's location, it still takes some time. For instance, for around 110,000 people and some stores and other kinds of facilities, it takes 0.031seconds to 0.078 seconds. Since we run a large number of MCMC steps, the response time can be an hour or more. One solution would be to use parallelization by using multi-threading or CUDA. Instead of running a large number of MCMC steps in a single chain, we can do chain-level parallelization. For instance, we run 1,000 MCMC chains, each of which runs the MCMC steps for 100 times. Each chain returns the best plan so that we get 1,000 best plans. Finally, we choose the best 3 plans out of them.