Faculty of Electronics and Information Technology - WUT

EARIN

Project 7: Safety Cameras - Final report

Filippos Malandrakis
Patrick Mila

General approach

This project presented as with an additional difficulty on top of the optimization method, which was determining whether a position is covered by a camera or not. Of course, for an empty room this was a task of trivial difficulty, as the only factor to be considered was the radius of the cameras. On the other hand, a randomly designed room with walls obstructing our view at every possible point was more of a challenge. Our approach to tackle this problem was the following:

```
for every point(candidate) in the radius of the camera(pos):
  l1 -> the line connecting candidate and pos
  for every point(testing point) between candidate and pos:
    l2-> the line connecting testing point and pos
    if abs(angle(l1, l2)) < 30°:
      return 0
return 1
```

We tested many different values for the least acceptable angle between l1 and l2, and 30 degrees seemed to work optimally in most cases. Approaches based solely on the equations of these two lines weren't so successful nor consistent. More details on our method can be found in *validityTest* function of *cameraInitialization*.
The eventually used optimization method is a genetic algorithm with multiple interconnected populations, to combat and slow early convergence.

Code outline

Generally, we won't dwell too much on the code at this part, as we have provided detailed comments for each function and step on it.

- *cameraInitialization.py* plays a major part in the initialization sector of our implementation, as it determines which positions can be covered for every possible camera placement(*tryCameras* function), and stores the results in specifically created structures which can be forwarded to the main part of our implementation. Also, *calculateMaxMinCoverage* function is used by the problem

generator to determine the minCoverage values that can be set by the user, for random grids.

- *problemGenerator.py* is, as its name states a simple problem generator which produces an input grid. User-specified parameters are the dimensions of the grid, the radius of the cameras used, the minimum coverage and the design of the grid(empty/random).
- *cameras.py* is the heart of our implementation. After the camera initialization is over, it proceeds to breed our initial genepool(s) (*createGene* function). Each created gene(solution) comprises a list of positions(coordinates) of all the cameras and its fitness score(calculated by the *fitnessFunction*). The fitness score depends mostly on the number of cameras a gene uses(aka the length of its list) and also on the sum of how many times each position is covered by a camera(acting as a tiebreaker, aka coverage score). Obviously, the lower the score the better. The function *checkCamera* is of cardinal importance to creating the best-quality (starting) genes possible. When every population is set, evolution begins. For each round(epoch), for every population 2 chosen parents crossover(*crossover* function) to eventually create 2 children, which are fixed or mutated(*fixOrMutate)* function, although in this project these two notions seem to be really close. This process may occur more than one time in one epoch for each population. Subsequently, the population kills the worst parent and replaces it with the best created child. Every few epochs, migrations between the populations happen in a circular manner. After each epoch, we check if our peak fitness score improved. If the score doesn't improve over a specified number of rounds, our implementation terminates.
- *config.py* specifies all the user-set parameters of our main implementation, which will be analysed on the following section.

Tweakable parameters

Now, we are going to break down every parameter inhabiting *config.py*

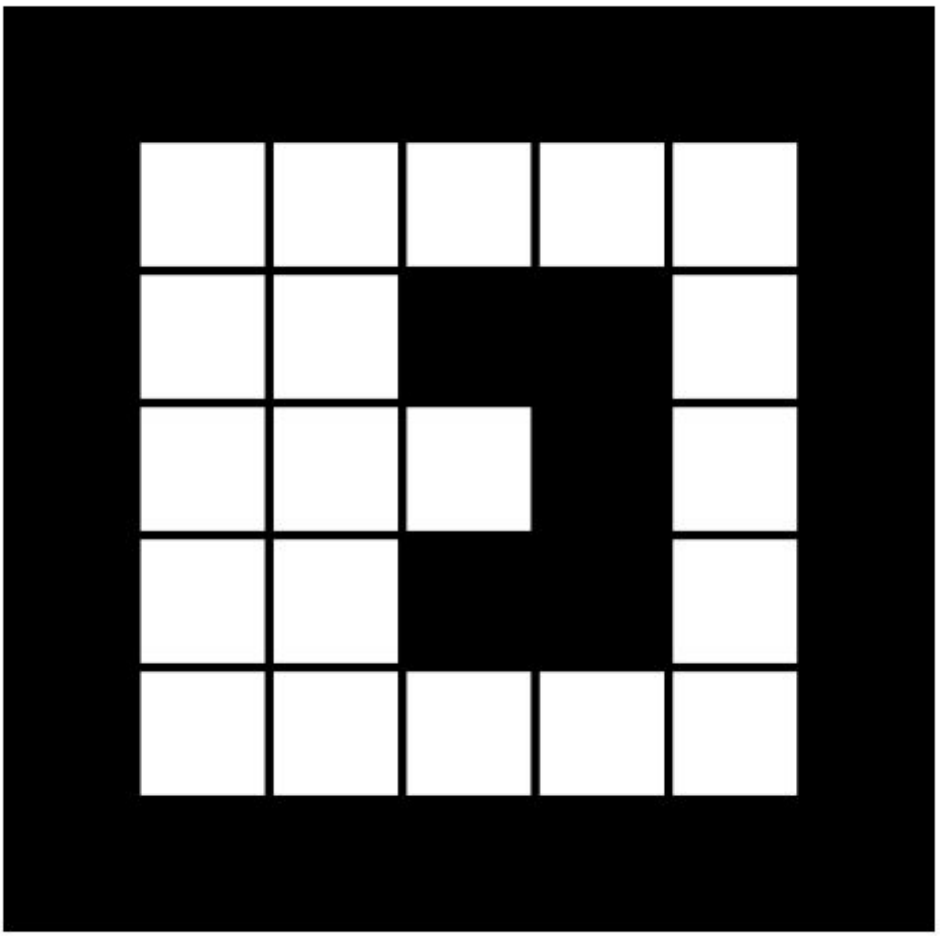
.

- Number of populations: This size is inversely proportional to the size of each population, meaning that the more populations we have, the smaller they will be. Although more populations allow us to mix up things more, we have to be careful not to exaggerate here, bearing in mind the dimensions of our grid.
- Patience: As stated before, this value determines how many rounds without improvement are we going to accept. Our tests usually showed that if we don't have any improvement in 200-300 rounds, we probably won't have any improvement at all, of course with exceptions. Ideally, we would like to set an extremely high value to this parameter to take every chance, but that would translate into a huge amount of wasted time. So to be more realistic and take our chances in a time-mindful way, we set this value to 500 rounds.
- Fitness function's parameters(a, b): These two parameters are used to distinguish that the length of a solution(a) is much more important than its coverage score(b) to the overall fitness score of the solution. So, we set a = 1 and b = 0.000001, to make that difference easily distinguishable.
- Sample size: This size determines how many genes we are going to sample for the parent role. From this sample, we will choose the two fittest parents. Obviously, if the sample is too big, the process will be equivalent to choosing directly the two fittest parents of the genepool. So we want to keep a small and controllable sample.
- Step: This value determines the genes / children analogy. For every one hundred genes, we should have 1 to 3 children max, so as not to disturb the evolution process too much.
- Migration time: This value determines how frequently(in terms of rounds) we will migrate genes between the populations. It is the glue between the populations, and obviously must be used with prudence. If it is too low, the populations probably won't be able to establish any kind of progress. If it is too high, it might not be able to deter early convergence, and it makes the whole 'many populations' notion useless and damaging(each population will be on its own).

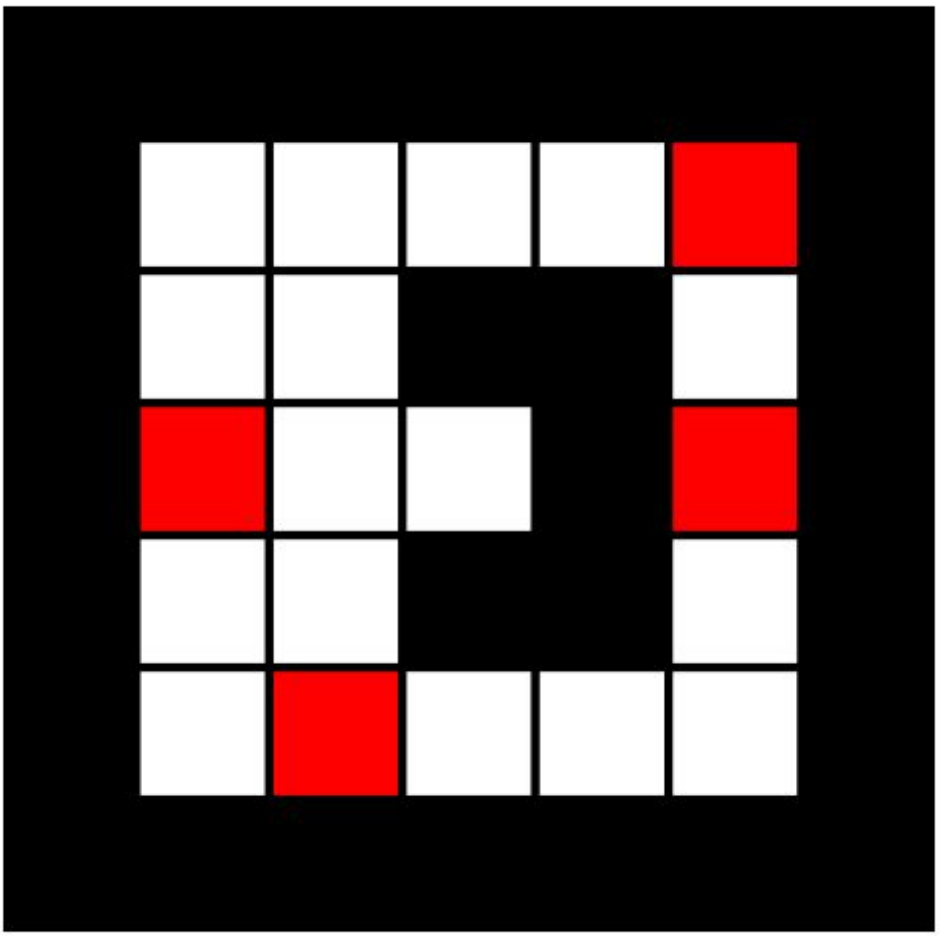
- Mutation ratio: This ratio determines how many children should be mutated instead of fixed. Generally, this ratio should be really low, like 1 or 2%. However, as said before, in our project mutating and fixing does not seem to make much difference, so this value is mostly typical and set to 1.

Simple example

For this section, we choose a little yet distinctive 7X7 grid, with walls inside the room as well. Camera radius is set to 2, and minimum coverage to 1. For this problem, anything more than one population of 80 genes would be an overkill. We also set our sample size to 5 genes. The input grid is shown below(walls with black, floor with white):



The optimal solution suggested by our program is the following(cameras with red):



Various parameter sets trials

All parameter sets were tested with a 50X50 empty room, camera radius 8 and minimum coverage 7. For each parameter set, we ran 20 simulations, keeping track of the average number of rounds and fitness score, as well as best and worst fitness score achieved. Our base parameter set is: populations = 5, sampleSize = 10, migrationTime = 5, step = 40. Of course, each choice of this set will be defended and proved by our results. Each parameter set tested changes one value of the base parameter set.

While reviewing the results of our trials, our verdict will be based upon two equally important and critical metrics. First, keeping as low as

possible the average number of rounds. Second, getting consistently good fitness scores, as close to the best score we have seen as possible.

Rating examples:

- If a parameter set has some really good and some really bad results, its value is deemed lower than a parameter set getting consistently ok results.
- If a parameter set scores really well but runs for a long amount of rounds, its value is deemed lower than a parameter set getting ok fitness scores and running for a much shorter amount of rounds.

Below we present a table containing all of our trials. We should note that all fit scores are converted to integers, as the coverage score part doesn't matter to us, only to our implementation(as a guide).

Parameter set	Average #rounds	Average fit score	Best fit score	Worst fit score
Base set	1669.8	86.05	83	90
sampleSize = 2	2649.1	86.3	81	91
sampleSize = 5	2044.95	85.7	83	88
sampleSize = 20	1394.7	86.35	82	90
populations = 1	1330.05	88.15	85	92
populations = 2	1415.45	87.5	84	91
migrationTime = 1	1477.55	86.65	82	91
migrationTime = 10	1695.65	86.25	82	90
migrationTime = 20	1646.1	86.8	81	92
step = 80	1541.6	87.9	84	92

Regarding the sample size, it can be observed that the more genes we sample for the parent role, the faster we arrive at our final solution, with

some negative impact on our fitness score. Eventually, we concluded using sampleSize = 10, which seemed the most balanced choice. It is interesting that without sampling(sampleSize = 2), we get almost the same fit score we would get with extensive sampling(sampleSize = 20) but almost on double the amount of rounds.
Moving on to the populations, it's obvious that while more populations increase little by little the average rounds, they also improve drastically our fit score. As a result, we concluded on 5 populations, gaining over 2 points of fit score compared to an implementation without multiple populations(populations = 1).
As far as migrationTime is concerned, our previous analysis is totally confirmed. Too frequent(migrationTime = 1) or too sparse(migrationTime = 20) migration leads to worse results compared to a balanced migration policy. Migrating every 5 rounds proves to be the optimal strategy, after a very tight race with the 10 rounds option. Interesting facts for this part are that too frequent migration drops the average number of rounds and sparse migration delivers the most inconsistent results of all the parameter sets(11 points difference between best and worst score achieved).
Concluding with the step size, producing 1.25 children per 100 genes (step = 80) drops the average rounds by a little margin, but it also implodes our fit score. This was totally expected as less children leads to earlier convergence and smaller diversity in every population. We chose producing 2.5 children per 100 genes(step = 40).

Various sizes trials

All trials were held with the base parameter set specified on the previous section, with the same method as before. Every room was empty. Although we would have liked to try even bigger sizes, holding a sufficient amount of simulations would have exceeded the deadlines of this project.

Grid size	Average #rounds	Average fit score	Best fit score	Worst fit score
7X7	530.85	4.05	4	5
20X20	686.55	13.9	13	16
50X50	1669.8	86.05	83	90
100X100	2327.25	373.9	360	384

As we can see, the dimensions' rise results in more rounds and of course biggest fit scores. This is expected, as the number of possible solutions to be explored and their length soars for bigger grids. Also, as the dimensions grow, it is notable that the average fit score centers between our best and our worst score.

Notable tested mechanisms

On our path to arrive to the final implementation, we experimented with many ideas, the most interesting of which are listed below:

- After producing two children:
  - Keeping best child and best parent*(made it to the final implementation)*
  - Keeping both children -> In many cases really good parent genes were lost
  - Keeping best child and worst parent -> Good to deter early convergence, but damaging in the long run
  - Keeping worst child and worst parent -> Very good to mix things up in the beginning, but keeping us from making actual progress
  - Keeping best child and worst parent if genepool is 'starting to converge', otherwise best parent. 'Starting to converge' is defined with the help of the following ratio.

$$\frac{worst\ gene's\ fitness}{best\ gene's\ fitness}$$

If this ratio is bigger than a user-set constant, then the genepool is starting to converge. -> Although really promising, finding a consistent value for the constant proved to be an unbearable task.

- In addition to the circular/ring topology, we also attempted to migrate between populations in a complete net topology, with no restrictions. However, the results were less consistent than desired, so we dropped this idea.