OULUN AMMATTIKORKEAKOULU

Samuli Saukkonen

USING MACHINE LEARNING TO FIND OPTIMAL BEAM GROUPS

USING MACHINE LEARNING TO FIND OPTIMAL BEAM GROUPS

Samuli Saukkonen
Bachelor's thesis
Autumn 2023
Degree Programme in Information Technology
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information Technology, Option of Device and Product Design

Author(s): Samuli Saukkonen
Title of thesis: Using machine learning to find optimal beam groups
Supervisor(s): Kari Jyrkkä
Autumn 2023:
Number of pages: 38

The object of the thesis was to use machine learning to find optimal radio beams that work together without interfering each other. Stochastic hill climbing and genetic algorithm were used to find optimal beam groups from beams measurement data. Beams signal to noise ratio in the beam groups were used for verifying if the results were good.

A machine learning program was created that worked as a microservice. It was integrated to a cloud-based web service where users could start the microservice and view the available results.

Especially the genetic algorithm worked well. It was able to find working beam groups from the given data. However, a problem arose as the machine learning program used only measurement data's azimuth results. As the beams can be directed both on azimuth and elevation plane machine learning could not handle this. Further update for this would be needed to get results to work with real use cases.

This thesis verified that machine learning can be used to find optimal beams groups. It also highlighted how time and money consuming creating machine learning projects can be.

PREFACE

I would like to thank my line manager Tuomo Partanen for giving me a chance to carry out my thesis for Nokia. Thanks go also to my team members Joni Barsk, Teemu Kontio, Juha Jeskanen and Sakari Naumanen for helping me. I would also like to thank my thesis supervisor Kari Jyrkkä.

CONTENTS

1 INTRODUCTION

Nokia is a large multinational company with many different subsidiaries. One of the biggest subsidiaries is Mobile Networks. One of the key products that Mobile Networks creates are first class radio products. These products are used by telecommunication providers to give the best possible mobile network to the end customers i.e., general public using their mobile phones.

Modern radios use beamforming technology. This means that the antenna of the radio can direct multiple radio beams to different directions. This allows multiple mobile phones to connect to the radio from different directions at the same time. The more different beams the radio can produce at specific frequency, the more mobile phones it can serve at the same time. However, if the radio produces too many beams and the beams are interfering with each other, the signal quality from the radio to the mobile phone will become too low. If this happens, the connection between the radio and the mobile phone becomes slow and prone to errors or worse the mobile phones cannot connect to the radio.

In the thesis the idea was to use machine learning techniques to solve this problem. The objective was to select optimal beams with machine learning while maintaining good signal quality for those beams. In the future this could also be used for antenna and radio designers. They would generate hundreds or thousands of simulated beams and the machine learning program would pick from those beams the ones which work correctly for them. It would remove the need for doing these things manually.

Part of the thesis was also to select best algorithms to use. A Python based program would be created that would implement those algorithms. This program would then be running as a micro-service that could be accessed from a Nokia internal cloud-based web service for test engineers.

2 BEAMFORMING, ALGORITHMS AND PYTHON

2.1 Beamforming

In beamforming radios use signal processing and multiple antennas to send or receive data at the same time in the same frequency. Beamforming is done by changing phases of signals from the antennas. This allows the beams to be directed to different directions. These changes increase performance when compared to non-beamforming radios. As the beams are also narrower, they can reach further. This also allows to add more beams in the same general area. Each beam can have a different user so the same radio can serve more users at the same time in the same frequency. Non-beamforming radio would have only one radio beam and it could serve only one user at the same time in the same frequency. In FIGURE 1 the difference of non-beamforming and beamforming radio antenna can be seen. (1., 2.)

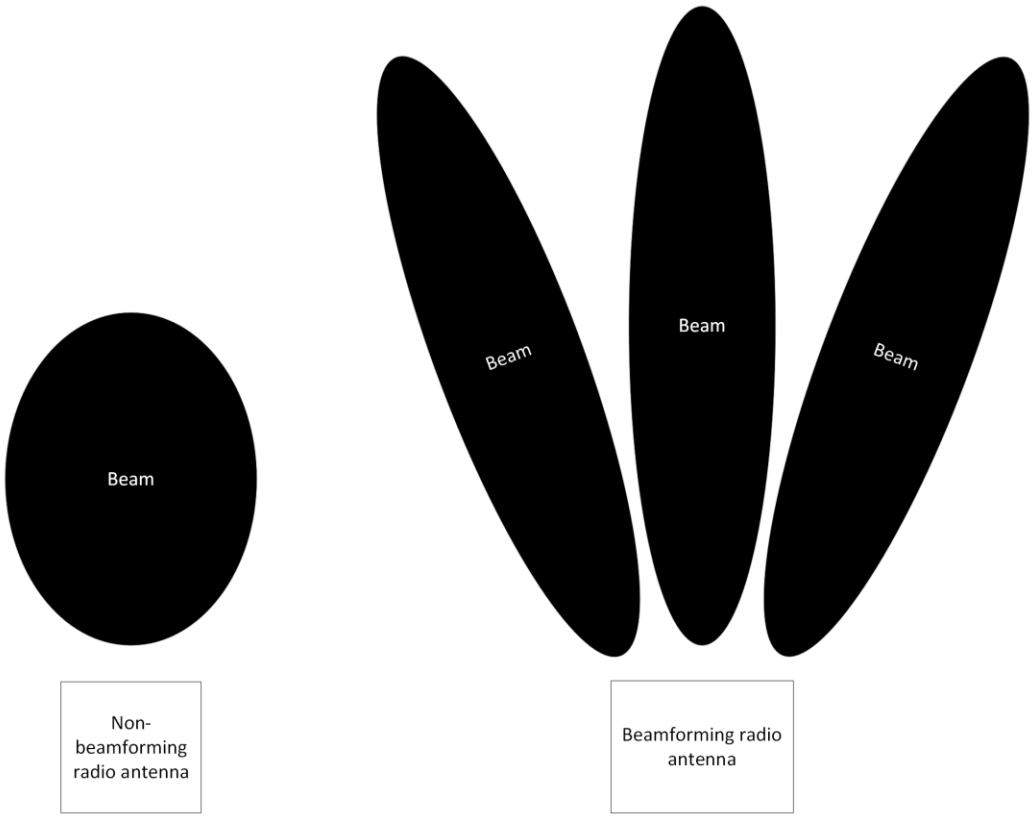

FIGURE 1. Simplified image of non-beamforming and beamforming radio antenna. On the left side there is a non-beamforming radio antenna, and it has only one short beam. On the right there is beamforming radio antenna with three narrow beams going to different directions.

Beamforming allows radios to serve as many users as there are beams. In an ideal case there would be an infinite number of beams. To get a working beam, it needs good enough signal to noise ratio. Beams also have side lobes. These side lobes are smaller than the main beam, but they can interfere with other beams. For a beam to have good enough signal to noise ratio it needs to be far enough from the other main beams. It also needs to be far enough from the other beams side lobes which can interfere with the beam. In FIGURE 2 there is an example of a beam with side lobes. (3.)

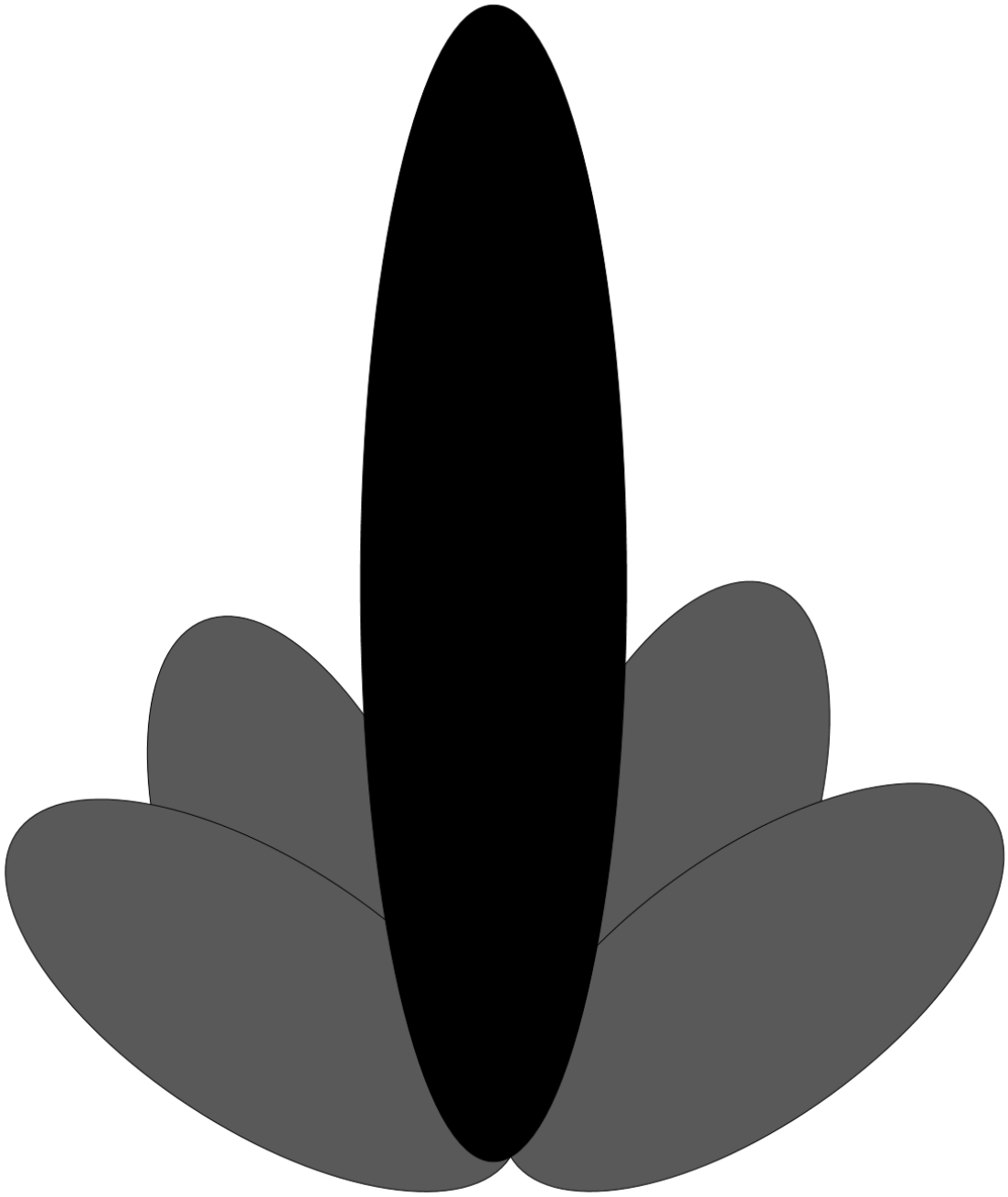

FIGURE 2. This is a simplified image of a beam (black) and its side lobes (grey). These side lobes might interfere with other beams.

2.2 Algorithms

2.2.1 Solution and fitness function

In this thesis, a solution or solutions are the result of the algorithms. The solution is a group of beams. For searching a beam group with four beams, its solution would consist of four beams.

One important aspect of any algorithm is how to evaluate if the solution is good or in other words what is its fitness. Fitness function is used to evaluate the solution. Fitness function can be simple or complicated. As this thesis is dealing with radio beams, a signal to noise ratio is a good way to find the fitness of a solution. If all the beams have a good signal to noise ratio, the solution is good. If all the beams have bad signal to noise ratio, the solution is bad. Fitness function can be used to get fitness for solutions and then the best one can be selected as the best solution. (4., 6.)

2.2.2 Stochastic hill climbing

Stochastic hill climbing is a local search algorithm and a variation of hill climbing algorithm. Hill climbing algorithms are used to find solutions for global optimization problems. They start by creating a random solution to the given problem and go step by step closer to the optimal solution. In hill climbing algorithm selecting the best neighbouring solution would be the next step. Selecting the best neighbouring solution would be done using the fitness function. Stochastic hill climbing varies in that its next step is a random solution which is better than the previous solution. Again, verifying if the solution is better would be done using the fitness function. (4., 5.)

Hill climbing algorithms can have a problem of getting stuck in local maximum. In hill climbing, the algorithm will try to find the solution from the neighbourhood of the current solution. If it cannot find a better solution in the neighbourhood, it will assume that this is the best solution. This might happen even though there is a better global solution available. However, there are ways to prevent this by randomly restarting the hill climbing. In FIGURE 3 there is an example of this problem. (4., 5.)

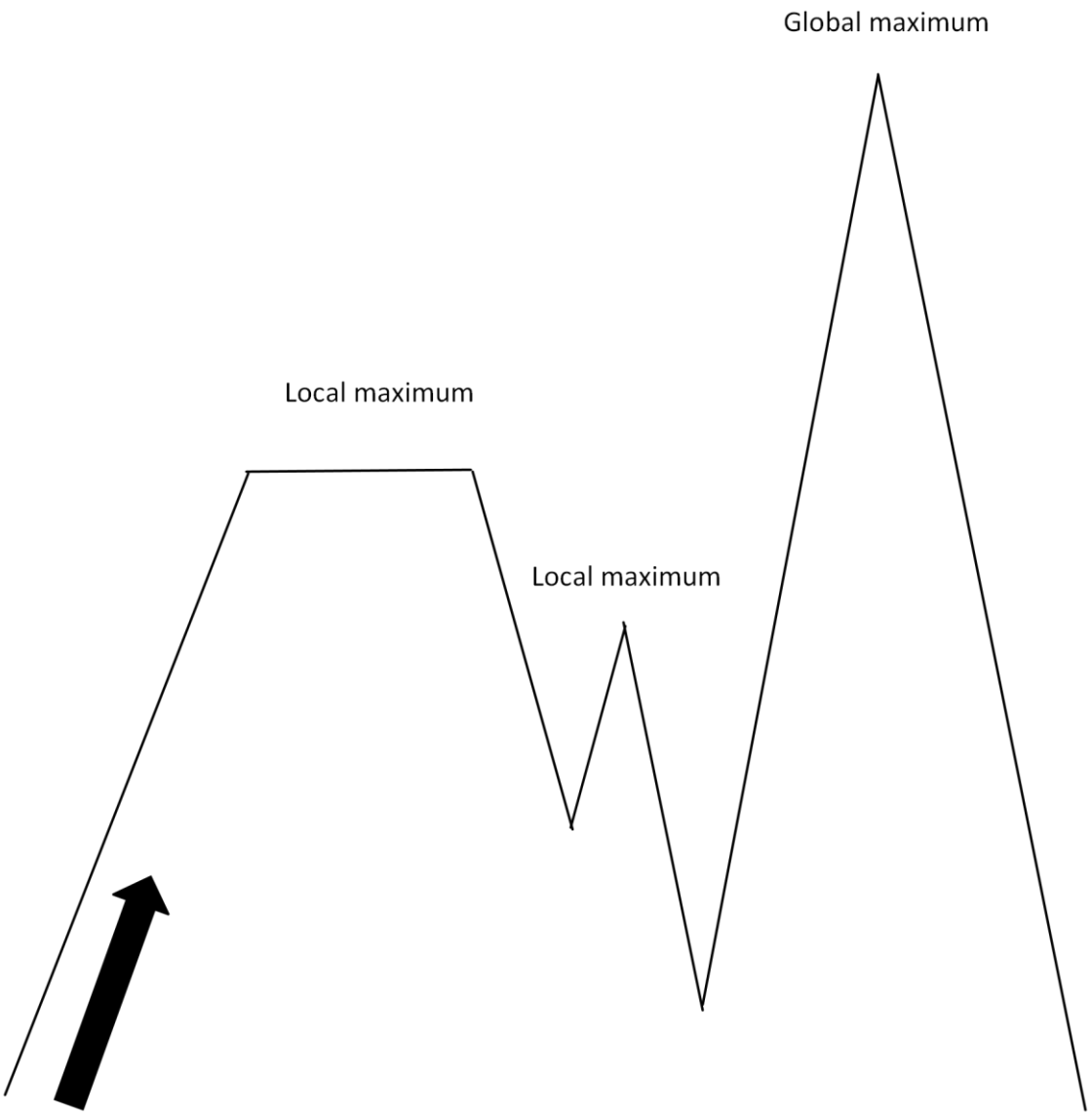

FIGURE 3. In this example finding the solution starts from the bottom left corner. The algorithm gets stuck in the first local maximum and cannot find the global maximum.

The algorithm which was used in this thesis was a modified stochastic hill climbing. In normal stochastic hill climbing the next solution would always be a random solution. This would make finding the solution slow when dealing with millions of different solutions. Modified stochastic hill climbing used the same randomness for finding the solutions. However, it did save the good parts of the found solution that were found using the fitness function. If parts of the solution were over the wanted limit, those would be saved and used as the base for finding the next solution. This added speed. However, it also brought the problem of getting stuck in local maximums like in the normal hill climb algorithm. (4., 5.)

2.2.3 Genetic algorithm

Genetic algorithm implements the simplified version of the theory of evolution to be used as an algorithm. It can be used to find solutions to different optimization problems similarly as hill climbing algorithms. However, they work completely differently. As hill climbing algorithms tries to find a solution more randomly step by step, genetic algorithm uses evolutionary approach. (4., 6.)

When implementing genetic algorithm, the first problem is trying to figure out what would be the genetic representation of the solution. All the beams that would be available would be a beam pool or a gene pool. A genetic solution consists of genes and in this case one gene would be one beam. Thus, representation of a genetic solution would be a group of beams and the beams would be selected from the beam pool. The number of genes or beams in the solution would be the number of beams wanted to be in the result. (4., 6.)

Next a starting population would be created. Population is a set of solutions, and one solution is a group of beams. Population has solutions that can be evaluated for finding the best solution or it can be used to create new populations. The size of the population can be anything from tens to thousands of solutions. The bigger the population, the more time and the more computing power is needed for finding the solutions. Solutions in the starting population would be randomly generated from the genes in the gene pool. The whole population would be evaluated using the fitness function, and the best solution would be selected as the current best solution. (4., 6.)

After finding the best solution from the starting population, a new population would be needed. As the algorithm uses evolution it would produce children from parents. One parent is a one solution in the population. Two randomly selected parents would produce two children. At the first point children would be direct copies of their parents. But as the algorithm is based on evolution, it would also crossover the two parents' genes or in this case beams. This means that a random crossover point would be selected in the genes, and the parents' genes would mix up at that point. After that the children would not be direct copies of the parents but rather a mix of both parents' genes. Additionally, a crossover rate would be used. This would be a percentage of how often the crossover would happen. This would guarantee that at least some of the solutions in the old population would continue to the new population. New children would be created until the new population size would match the old populations size. (4., 6.)

One additional evolutionary approach in the genetic algorithm is mutation. Mutation would happen for the new children and mutation would mutate the individual genes in the children. In this case a gene would be a beam. If a gene would have a mutation, a new random gene would be selected from the gene pool to replace it. Mutation rate would dictate how often it happens. Mutation rate would be a percentage similarly as the crossover rate was. (4., 6.)

After all this, the children would be the new population. It would be evaluated similarly as the starting population. If a better solution would be found in the evaluation, it would be selected as the current solution. This would finish the run of the first generation. The next generation would start by creating children from the last population. This loop would continue for a wanted number of generations. After finishing all the wanted generations, the result would be the best solution that the genetic algorithm found. (4., 6.)

2.3 Python

Python was selected to be the programming language of the program. It had multiple reasons. A cloud-based web services backend that would use this program was already done using Python. Python programs can be run as a microservice. Microservices are independent programs that can run on their own. This was important because the program would be a microservice that would be connected to the web services through backend. Python is also often used in machine learning projects and there is a lot of good documentation available on those. Python is also a popular language, and it has good support available. (4., 7., 8.)

2.3.1 Pytest and test-driven development

Pytest is a test framework for Python. It allows to create automatically run unit tests. Tests can be used to verify that the code works correctly. It is also relatively simple to use, and it allows a fast workflow in coding. (9.)

One thing that unit tests can be used for is test-driven development or TDD. Test-driven development is a different way of coding. In it tests are written first and the code that passes the tests is written afterwards. Test-driven development results to excellent code coverage because every part

of the code that has been written is tested. This then gives some sort of guarantee that the code does what it is supposed to do. Test-driven development also forces to write better and simpler code because complicated code is hard to test. Good test coverage generated by test-driven development helps also when refactoring code or adding new features. Old tests guarantee that the new features have not broken anything in the code. Lastly tests generated by test-driven development also serve as documentation for the code. They tell what is happening in the code which then helps to understand it better. This can be helpful especially when working with legacy code. (10., 11.)

Test-driven development was used in all the software developing phases of this thesis. In the development process this allowed easier changes to the code. Changes could be easily verified by the unit tests. In the end the microservice that was created had 98% test coverage. This means that most of the code was covered by unit tests and future development work can rely on those tests.

2.3.2 Multiprocessing

Running machine learning algorithms can need a lot of processing power. They often also need to run a lot of different calculations which can be slow. It would be beneficial to run those calculations in parallel using all the CPU cores that the computer has to offer. Python has a Global Interpreter Lock which allows only one thread to run at a time. What this means is that threading in Python cannot run multiple requests at the same time. When multiple requests are wanted to run at the same time multiprocessing is needed. Multiprocessing can start multiple different Python instances and run those in parallel. This can help to run the program much faster than using only single Python instance. (12., 13.)

Implementing multiprocessing in Python is a relatively simple process because there are libraries available for this. Libraries can be imported and used in the code. All you need to do is use the libraries with parameters and the libraries do all the work for you. Parameters depend on what libraries are used. For example, they can be the function that is wanted to run in parallel and how many multiprocessing instances are started. These libraries also allow waiting until all the returns are available and then the program can continue running the next part of the code. (12., 13.)

Both multiprocessing and threading were used in the thesis. Threading was used to get the wanted data from the databases and multiprocessing was used in the machine learning process. Multiprocessing allowed expediting the machine learning and getting the results faster.

3 CREATING THE PROGRAM

3.1.1 Background information

For an initial version of the machine learning program, a set of simulated beams were selected. Simulated beams were selected instead of real measured beams. This was done because the beams would be ideal, and this would reduce difficulties working with poor data. The selected set contained 40 different beams and requirement was to get a beam group with N number of beams to work together with good enough signal to noise ratio. In FIGURE 4 there is a chart displaying an example of a beam group.

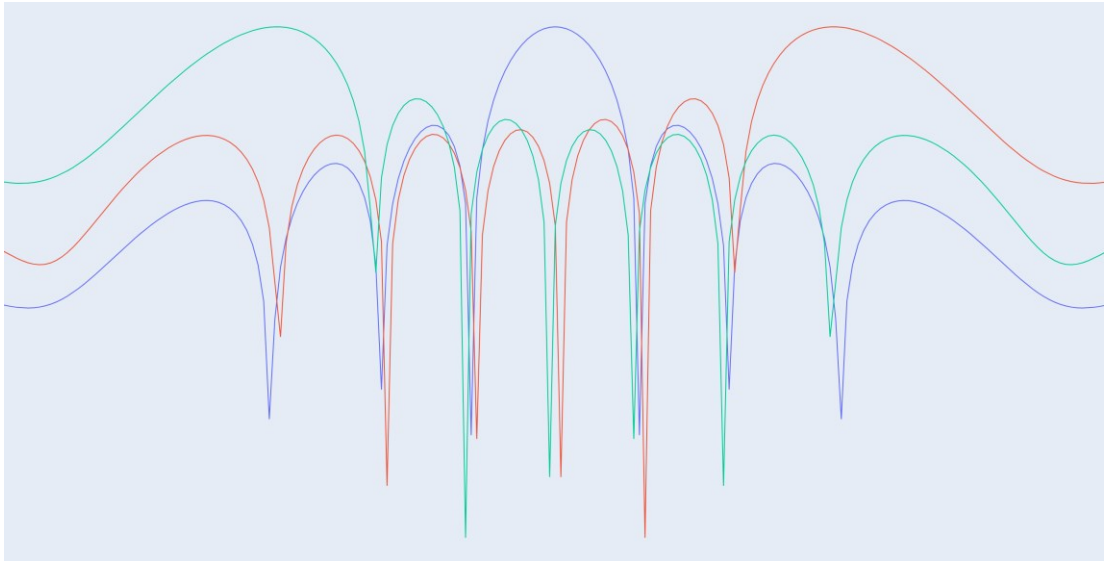

FIGURE 4. Example of a beam group with three beams.

A decibel value X was chosen as the signal to noise ratio limit. The beam group would need to be in 120-degree field of view. With three radios with 120-degree field of view facing different directions would result full 360-degrees of coverage area. The problem with a set of 40 beams and selecting N number of beams from there is that there would be millions of different combinations available. This can be calculated by using formula $C_k^n = \frac{n!}{k!(n-k)!}$ where n is number of the available beams and k is number of the wanted beams per beam group. Because there are so many different combinations available, machine learning would be helpful in beam selection.

The program using machine learning algorithms would be running in a Nokia internal cloud-based web page for test engineers. It was decided that the program would be running as its own microservice. Microservices are independent programs that run on their own. They are connected to other programs for example with message brokers or REST APIs. As the microservices are independent, they can be easily updated and maintained. Microservices can also be scaled independently if needed. The machine learning program was done from the beginning in a way that it could be easily implemented as a microservice. This meant that the program could work independently. The program was also made so that it could be started from the outside. Returning the results to the outside from the program was also taken into consideration. (8.)

3.1.2 Creating comparison tools and stochastic hill climbing algorithm

Beams simulated measurements were converted to Pandas DataFrames. In the beginning, the first step was just to have the ability to combine some number of beams and calculate for each of those what is the maximum peak signal to noise ratio of those beams. In FIGURE 5 an example of a beam group with a poor signal to noise ratio can be seen.

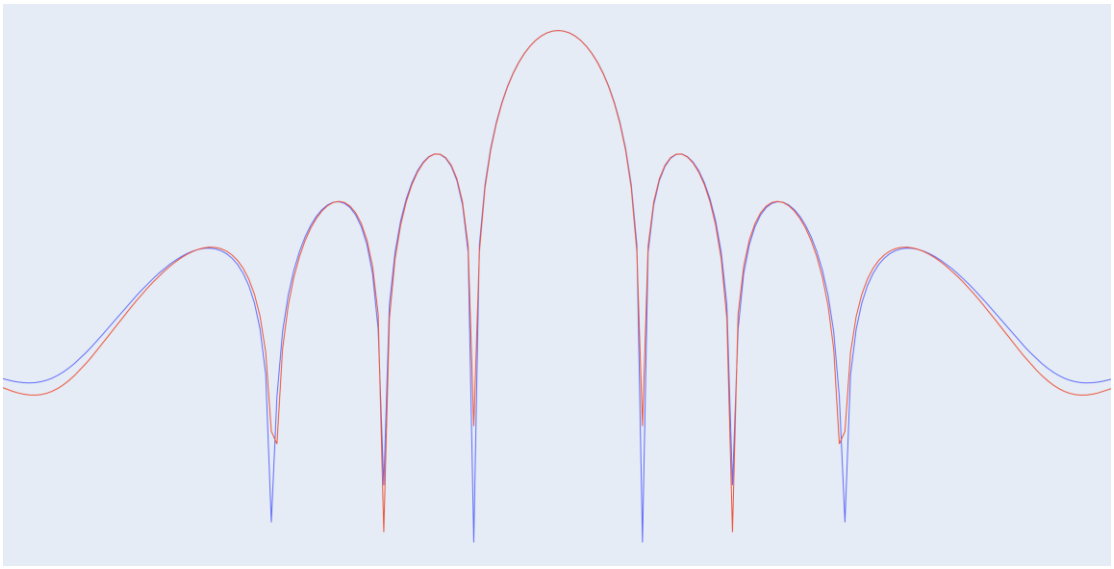

FIGURE 5. In this chart there are two beams with poor signal to noise ratio. Red line and blue line are almost completely over each other.

A beam group that has more beams over wanted decibel limit would be selected as a winner in the comparison tool. If beam groups would have the same number of beams over the decibel limit, their mean signal to noise ratio would be compared. The one where the mean would be higher would be

selected as the winner. This comparison tool would be used in the algorithms to select the better beam group. In FIGURE 6 can be seen how comparison tool works.

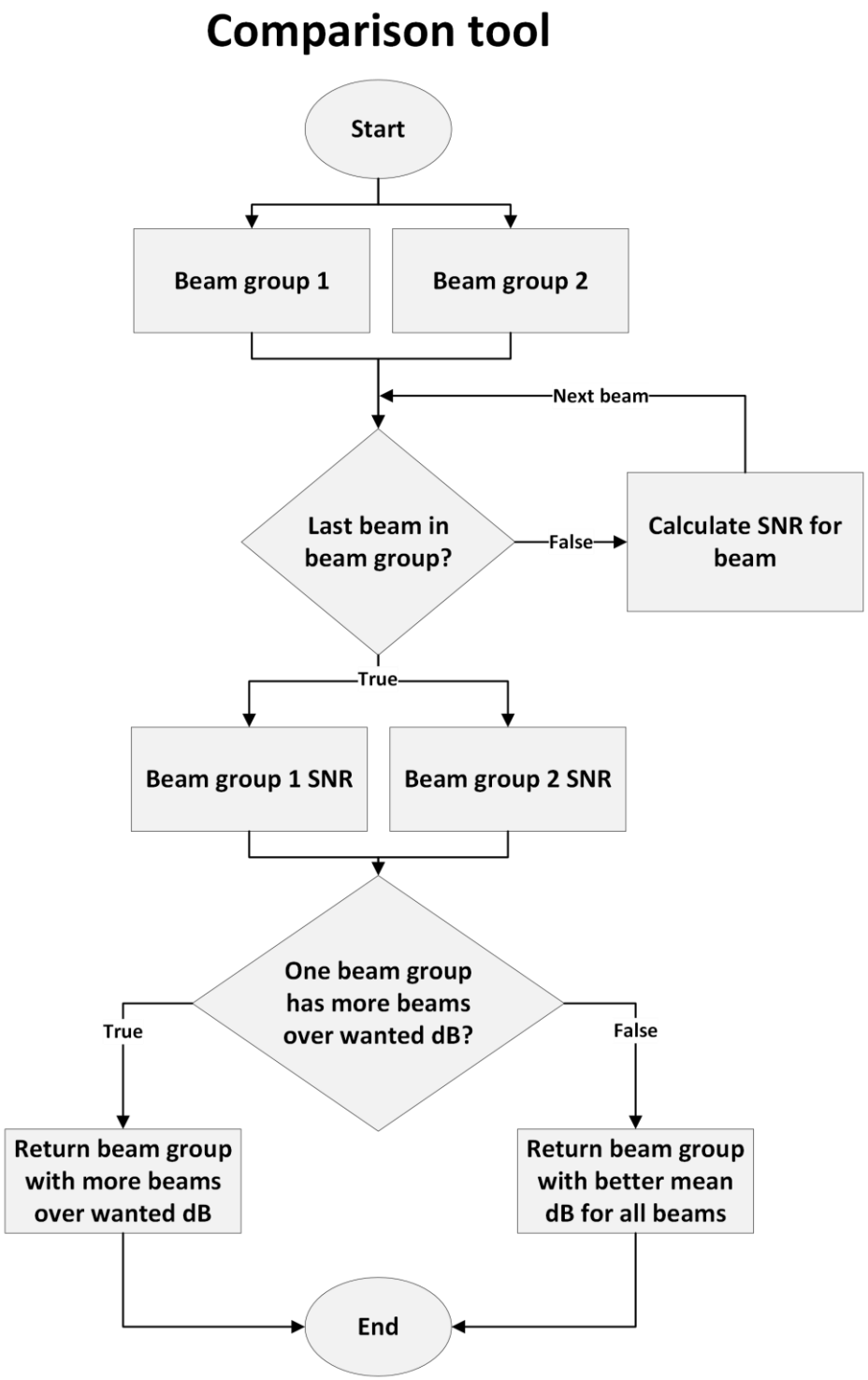

FIGURE 6. Comparison tool compares signal to noise ratios of two beam groups and returns the better beam group.

Stochastic hill climbing algorithm was the first algorithm that was tried in the program. In this implementation of it, the program would create two beam groups from randomly selected beams and compare them together. The program would select the winner by using comparison tool. Additionally, if the winner would have beams over the wanted decibel limit, the program would save those beams. The program would then use those saved beams when creating the next beam group for comparison. Rest of the beams in the new beam group would be random beams.

For example, the wanted number of beams in the beam group would be four. The winner beam group would have one beam over the wanted decibel limit. That one good beam would be a step closer to the solution in the stochastic hill climbing algorithm. For the next beam group for comparison, the program would use that one good beam as a starting point and select randomly additional three beams. The comparison tool would do the comparison between the last winner and the new beam group and return the winner of this comparison. This would be one iteration of stochastic hill climbing algorithm. The program would always use the beams of the latest winners that are over the wanted decibel limit as a starting point for the next beam group. Doing this step-by-step process for multiple of iterations would get more and more good beams in a beam group and closer to the optimal solution. In FIGURE 7 there is a diagram that explains the stochastic hill climbing algorithm.

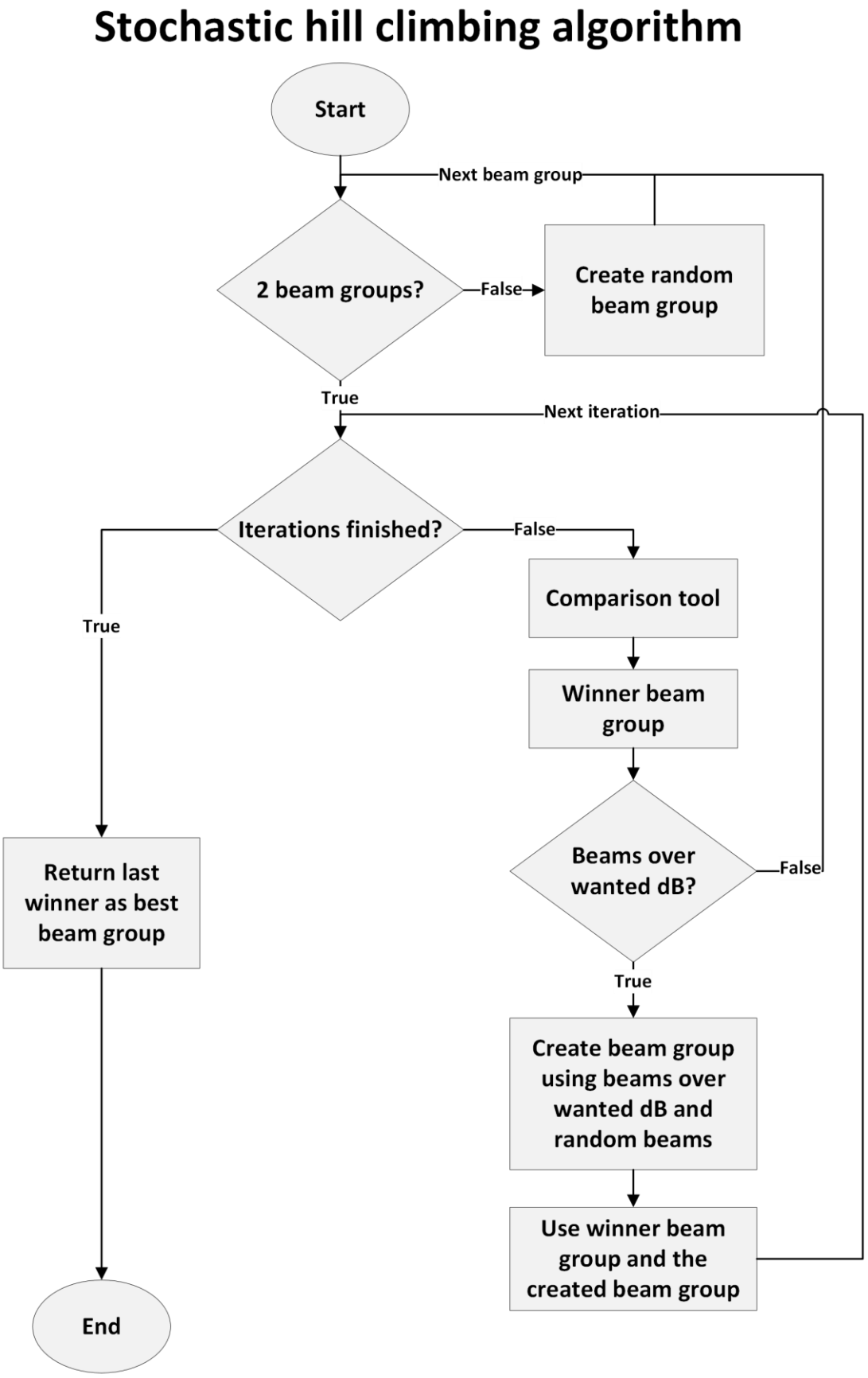

FIGURE 7. Stochastic hill climbing algorithm creates and compares beam groups in a loop. In the end it returns the best beam group it found.

Testing stochastic hill climbing algorithm with simulated data seemed to work if the wanted beams per beam group were lower than the initial value of N number of beams. However, with wanting more beams, getting good beam groups seemed to be harder and harder. One reason for this might be that the algorithm finds a few good beams that work together but then it gets stuck there. As it keeps using those beams that at one point were good, they might not be good beams for the overall result. This is the problem that was mentioned in the chapter 2.2.2.

3.1.3 Creating genetic algorithm

The implementation of the genetic algorithm of the program would use the basic components from the comparison tool generated for stochastic the stochastic hill climbing algorithm. Genetic algorithm also needed to do similar comparisons to get the best possible beam group.

Creating a population is a starting point in genetic algorithm. The population is a set of beam groups. The beam groups in the first population would be created from random beams the same way as a single beam group was created in the stochastic hill climbing algorithm. The size of the population was set to be the number of wanted beams to the second power. For example, for four beams that would be 16. For beam groups inside this population, the best beam group would be selected as the winner with the comparison tool. This winner would then be compared to the winners of the generations of the future population. In FIGURE 8 there is a diagram of genetic algorithm.

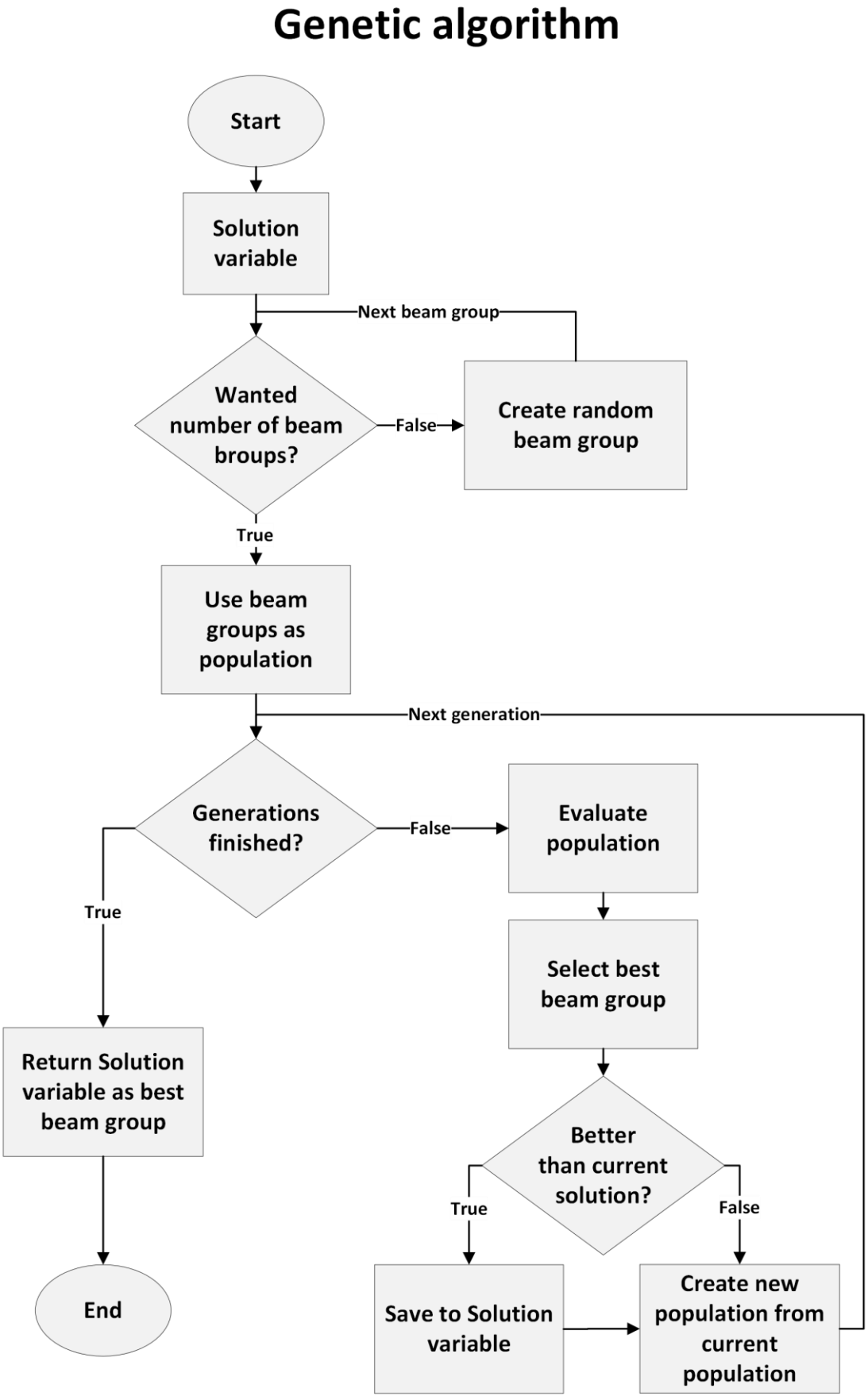

FIGURE 8. Simplified version of how the genetic algorithm returns the best beam group.

The next step would be to create the new population from the current population. A random beam group from the population would be selected as parent candidate. It would be compared to the wanted number of other randomly selected beam groups from the population. The winner of this comparison would be returned as a parent. This would allow to select only the best beam group to be a parent. This selection would be done until there would be the same number of parents as there were beam groups in population. In FIGURE 9 there is a diagram displaying how the parents are created.

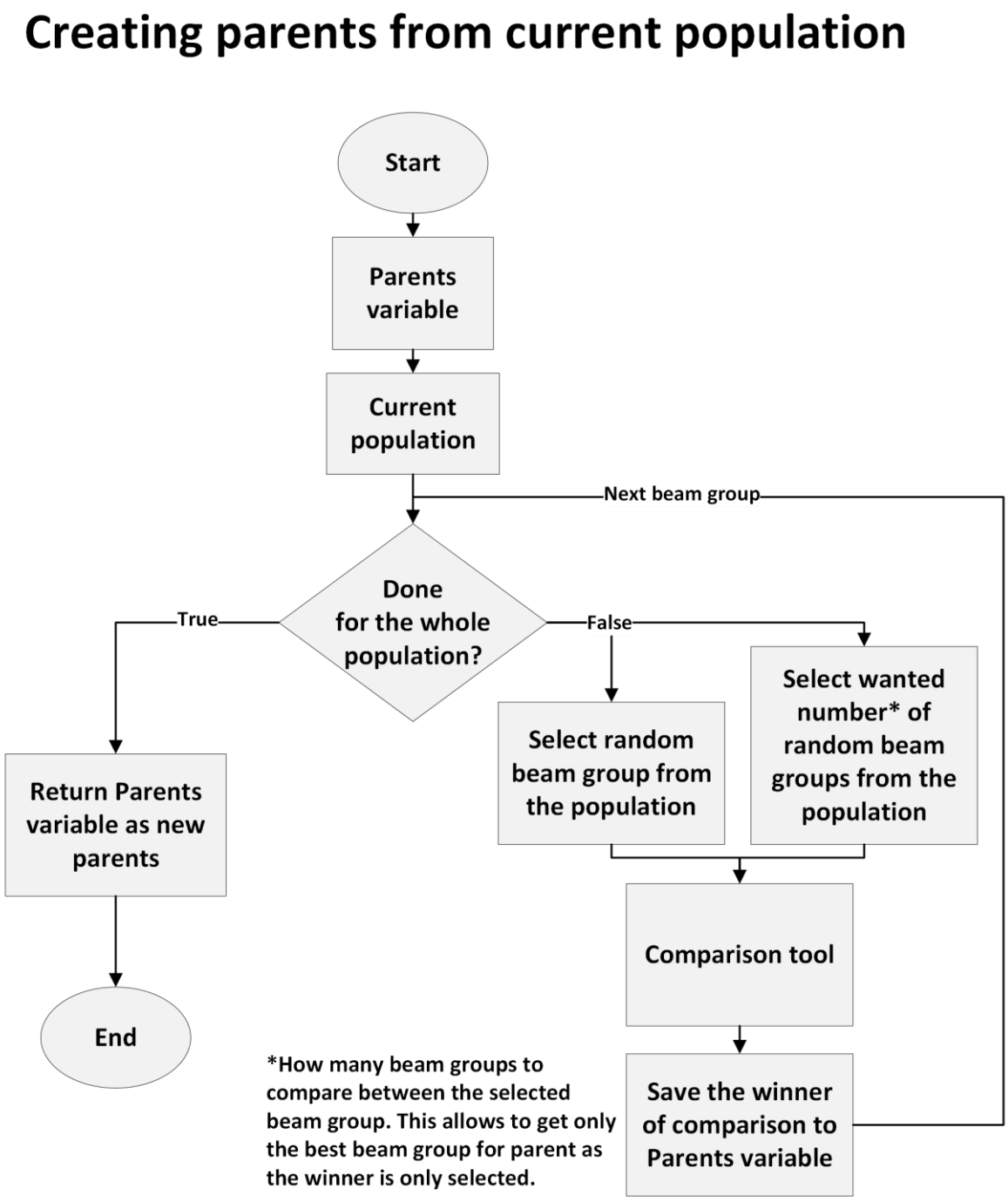

FIGURE 9. Diagram of how parents are created from the current population.

After having the parents selected, children would be created of them. Normally children would be a direct copy of parent. However, by adding crossover rate as a percentage crossover would be done between two parents. For example, having a crossover rate of 90 percent, 90 percent of the parents would be crossed over to get new children and 10 percent would be direct copies of their parents. In the crossover the program would select a random point in the beam group to do the crossover. For example, if the program would try to get four good beams and it also randomly picked the crossover point to be after the first beam. First children would have first beam from the first parent and the last three beams from the second parent. Second children would have first beam from the second parent and the last three beams from the first parent. This would be done until there would be the same number of children as there were parents in the beginning. These children would now be the new population. See FIGURE 10 for how crossover and creating children works.

As this algorithm somewhat simulates evolution, mutation would also affect some of the children. Mutation rate works in a same way as the crossover rate. However, mutation rate affects children's individual beams. For example, for mutation rate of 10 percent, 10 percent of the children's beams would change to a random beam. This randomness also added a problem: beam group cannot have the same beam twice. This same problem also affects crossing over parents because parents might have the same beams in them, or parents could even be identical. To be able to deal with these situations additional checks were added to be sure that a beam group would never have the same beam twice. If the same beam would appear twice in a beam group, one of them would be changed to a random beam. This also adds some additional random beam changes which could be beneficial. In FIGURE 10 there is a diagram showing how mutation and creating children works.

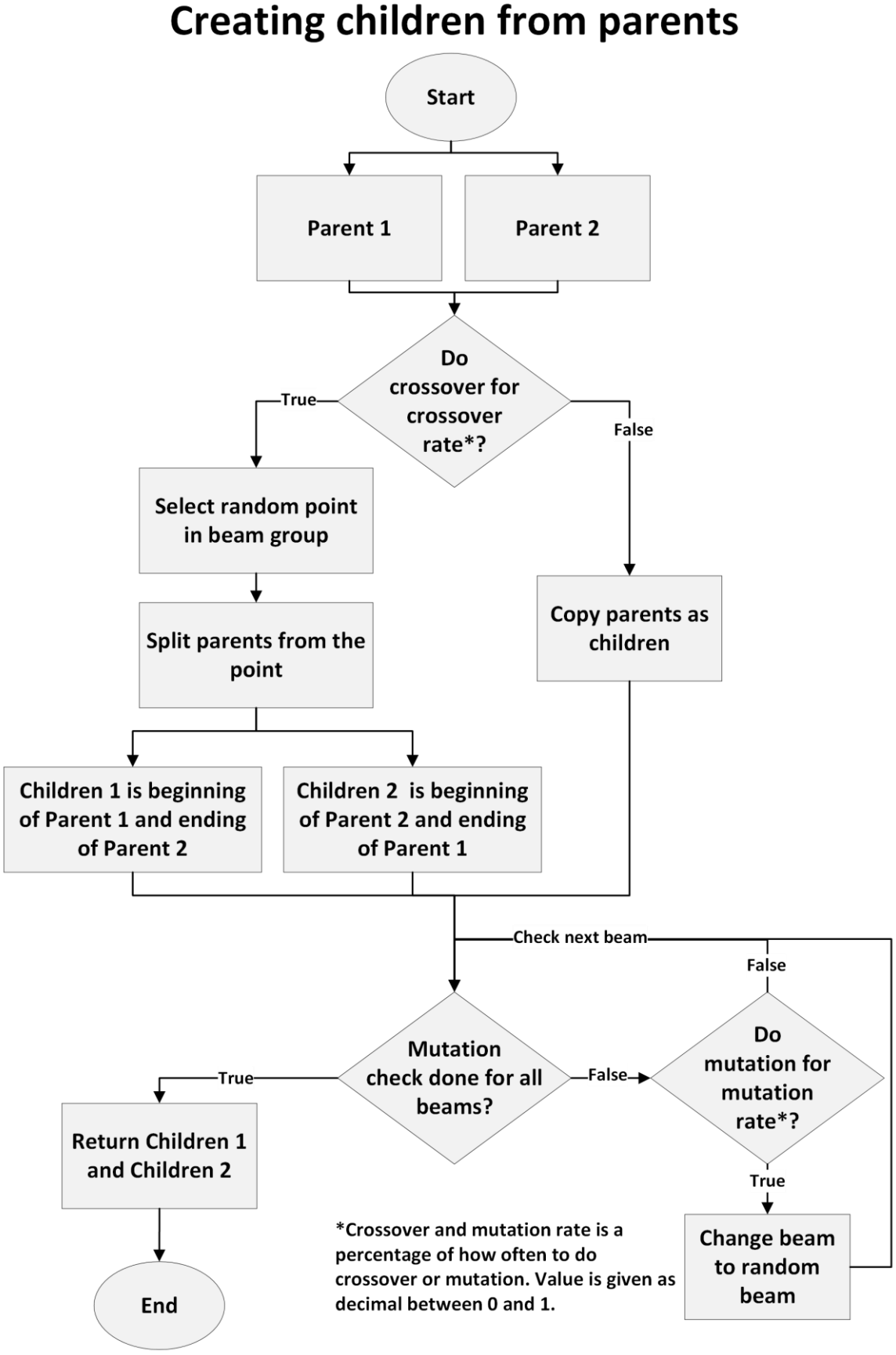

FIGURE 10. Diagram showing how new children are created from two parents. This would be done for all the parents. In the end, there would be a new population that is the size of the parents.

With the new population ready, it would be evaluated to select the best children from there. This would be done with the comparison tool. If the winner from the new population would be better than the last solution, it would be selected as the current solution. Generating new populations and evaluating them would continue for as many generations as would be wanted. In the end the best solution would be returned. See FIGURE 8 for a reminder how the genetic algorithm works.

3.1.4 Findings while testing

There were a lot of different parameters that could be changed when running these algorithms. Stochastic hill climbing was simpler, and it had mainly one parameter, how many iterations to run. The genetic algorithm however was more complicated. The population size could be changed, how many generations to run, what would the crossover rate or mutation rate be and how many parents would be compared in the parent selection. This added some complexity to find best parameters to find the good beam groups in an optimal time.

Different parameters were tried out. The problem was that there were so many different parameters that could be changed so it was difficult to find optimal parameters. It was also found out that after changing some parameters, the program did not return any good results. Findings were that optimal population range would be the number of wanted beams to the power of two. Crossover rate for 90 percent and mutation rate for 10 percent were selected. In the parent selection comparison 6 parents were a good compromise. With these settings running the program would take around 40 minutes.

Testing was done trying to find N number of beams with X dB signal to noise ratio from the given set of 40 beams. It soon became evident that with these beams it would be impossible to find N number of beams with X dB signal to noise ratio. This was later confirmed by experts that indeed with those beams it would be impossible. This was a good first confirmation that the program was indeed working at least somewhat correctly because it could not find impossible solutions.

The next step was changing the number of beams to N - 2 beams with X dB signal to noise ratio. This also seemed impossible. The program could find N - 3 beams with over X dB signal to noise ratio but the signal to noise ratio of the last beam would always be only a little bit under the wanted

X dB. The program was even successful finding different beam groups that all had N - 3 beams with over X dB signal to noise ratio and one beam that was lower. This then confirmed that even finding good N - 2 beams from this set of beams would be impossible.

It was also apparent that the genetic algorithm seemed to be working from generation to generation finding better and better solutions. Sometimes it would find worse solutions for a while as it would be stuck in local maximum found in the population. However, because of the continuously happening crossover and mutation it would get out of the local maximum in goal to find the global maximum. Running program for 100 generations it always seemed to find the good N - 3 beams and one lower quality beam in the end which verified that the program is working as planned at least with this set of beams.

Stochastic hill climbing algorithm was added to run in the program after the genetic algorithm. However, it would only run if the genetic algorithm did not find any solutions with all the beams in the beam group being over the wanted decibel limit. The reason for this was that if the genetic algorithm had found one or multiple good solutions, there would not be any need to run the stochastic hill climbing algorithm. But if the solution would be missing one or two good beams stochastic hill climbing algorithm could be used to try to find the missing good beams.

3.1.5 Multiprocessing

The genetic algorithm seemed to be running relatively slowly so it raised a question: could any optimizing be done? Threading was the first option that was studied. However soon after examining how threading works in Python a problem arose. In Python threading can only use a single CPU core at the same time. As all the calculations are done in CPU, threading cannot really make these calculations run any faster. Threading would only be useful in Python for something where CPU processing would not be needed all the time, for example downloading multiple files from a website at the same time. (14.)

Multiprocessing was studied as an alternative. In multiprocessing multiple Python instances are started. The only limiting factor for those is how many CPU cores the system has as every new Python instance would need its own core. Multiprocessing seemed to be a solution. Multiprocessing

was first added for creating parents from the population. In the selection process of the parents, the whole population size was compared as many times as there were comparisons wanted in the parent selection. For example, for a population of 100 where six parents would be compared, it would have 600 comparisons. Multiprocessing was also added to the evaluating of the population. Evaluation also used a lot of comparison tool as the whole population was compared to each other to find out the best beam group. As both these cases used a lot of comparison tool, it seemed that it was the limiting factor on how fast the genetic algorithm can run. (12., 13.)

Tests were done to find out how much faster the program could be with multiprocessing. One limiting factor for the tests was that the remote computer where this development work was done had only two CPU cores available. For that reason, measurements for only one and two cores were available. Measurements for one and two cores from ten generations of genetic algorithm was taken and their mean value was calculated. The measurement data was then extrapolated from 2 cores up to 6 cores and 100 generations. Of course, this could not give accurate results but at least it gave a trend of what would happen if more cores were added. The findings were that adding multiple cores could significantly reduce the running time of the program. In FIGURE 11 there is a graph from the measurement results.

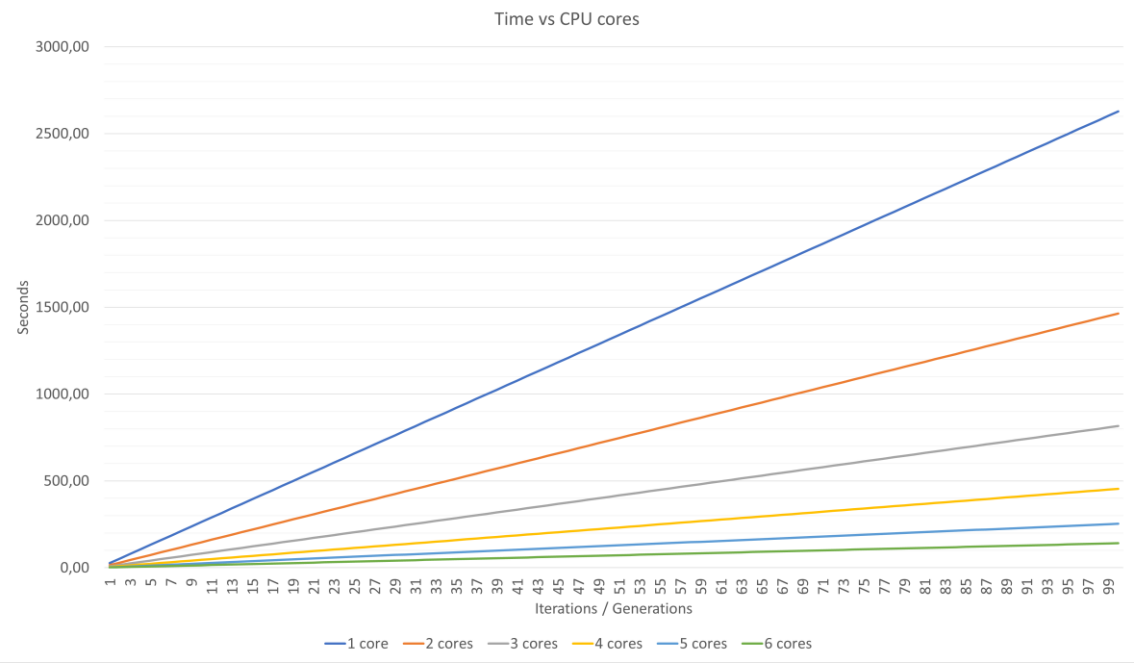

FIGURE 11. In this graph one core and two cores are measured, cores from 3 to 6 are extrapolated from the measurements. In these findings can be seen that using just one more core almost halves the time used. If this trend would continue with adding cores, the running time could be significantly reduced with multiple cores available.

3.1.6 Program as microservice

Building blocks for the microservice were already available. The machine learning program was already made from the beginning to be able to work independently as a microservice. Another important part was the other Python based microservices already available in the cloud-based web service. Connection interfaces could be copied and modified versions of those could be used in the new microservice.

Microservice had two different ways of connecting to the Python based backend. First it would listen to the new messages from RabbitMQ service. RabbitMQ is an open-source message broker service that can deliver messages from one service to another. Microservice would listen to these RabbitMQ messages and based on those messages start the task of finding the beam groups. RabbitMQ also has queueing ability. This was helpful as it could be used to limit the messages for the microservice. Microservice would finish the task of the first message and then pick another from the RabbitMQ's queue. (15.)

The second way for connecting to the backend was using HTTP request to the backends REST API. REST APIs are set of rules and design principle how applications can communicate with each other. HTTP requests could be used to get information from the database through backends REST API. Measurement data for the beams would also come through the backend but from a different measurements database. After finding the best beam groups, the results would be saved to the database using HTTP requests to the backends REST API. See FIGURE 12 for the connections of the microservice. (17.)

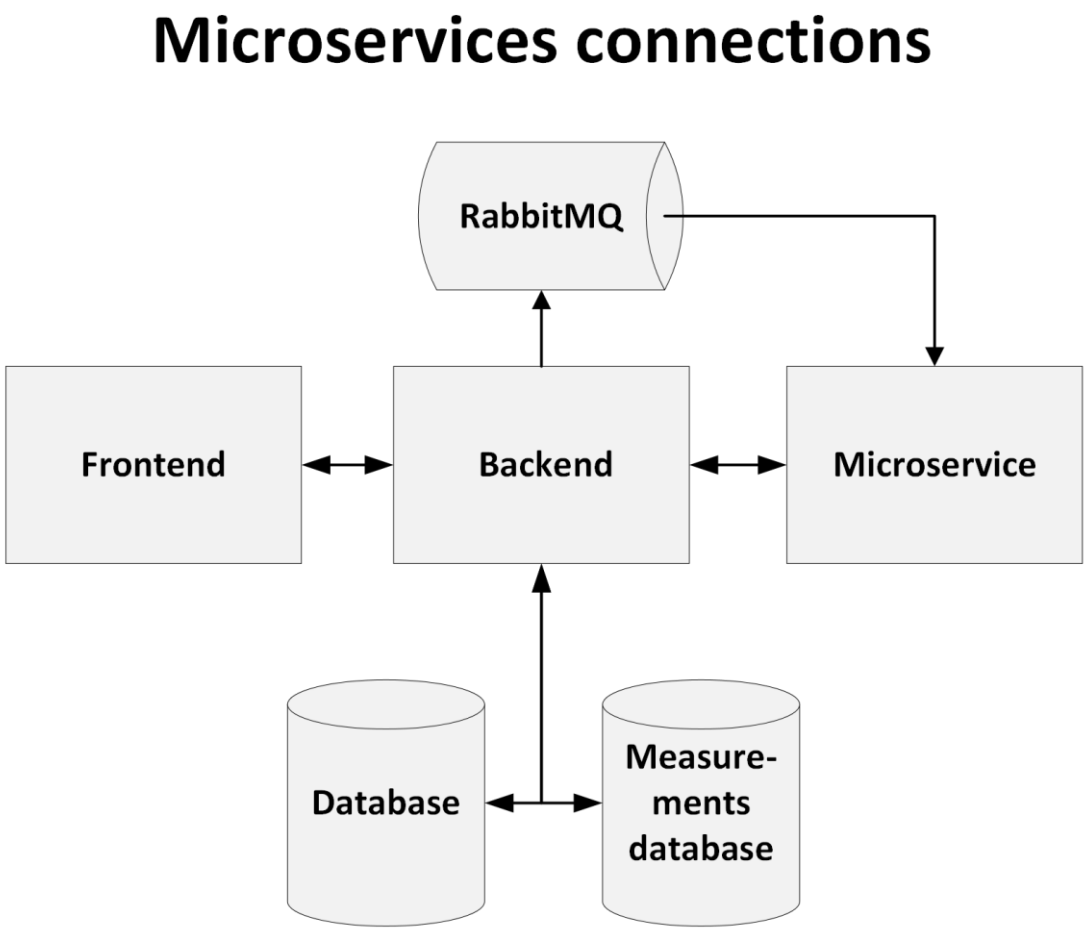

FIGURE 12. Frontend would call backend to start the microservice. Backend would send message through RabbitMQ to the microservice. Microservice would get the needed data from the database and the measurements database through backend. Microservice would save results to the database through backend. Frontend could then view those results from the database through backend.

User interface was on the cloud-based web service. This web service was already available, and it had been made using Angular framework. A new web page was added there. The user could fill in a form in that page and star the microservice by submitting it. The users could also view the results in that page in a table and see graphs of the beams found by the machine learning microservice.

3.1.7 Testing the microservice

When the connections between the backend and the microservice were established, testing the microservice and machine learning could start. Using the real measurement data lead to some difficulties. Data was transformed into Pandas DataFrame. As the measurement data format was completely different from the simulation data format, the same data transformation could not be

used. Data structure for the measurement data was more complex therefore more complicated Pandas transformations were needed.

Another problem was found when running the machine learning with the real measurement data. The results did not seem to match the expectations. The problem was that the machine learning could not find as many beams for the beam groups as was assumed to be found. The reason for this was that the beams could be directed also in the elevation level, not only in azimuth level. If radio is watched from the top, azimuth is left and right and elevation is up and down. This caused a problem because the program was done to use only the azimuth measurements and not the elevation measurements. In FIGURE 13 there are two beams from azimuth plane and in FIGURE 14 there are those same beams from elevation plane.

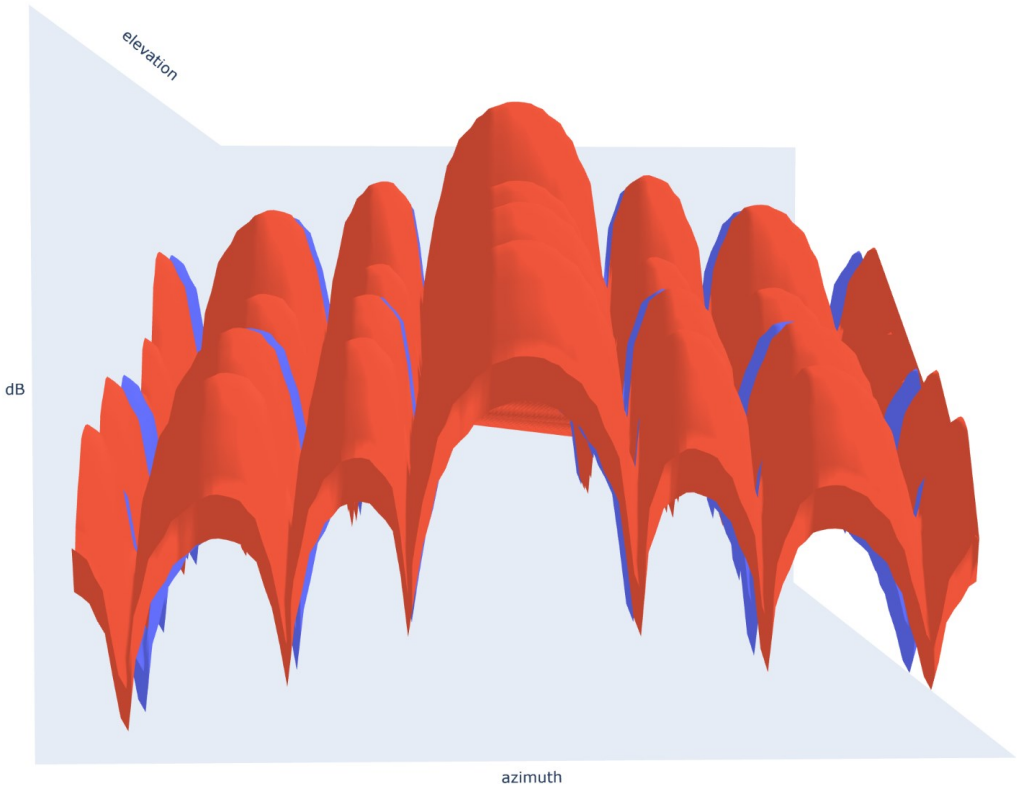

FIGURE 13. Two beams from the azimuth plane. Red beam blocks the blue beam if the machine learning is done only from the azimuth plane.

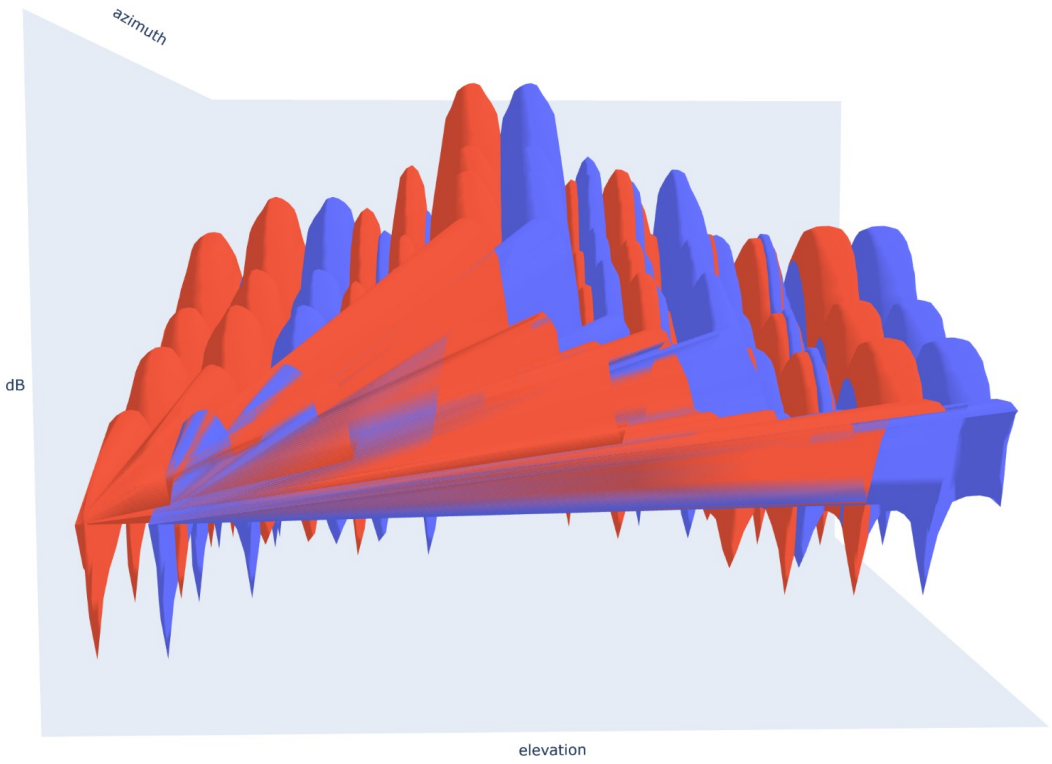

FIGURE 14. Two beams from the elevation plane. Both beams fit if the machine learning is done only from the elevation plane.

Fortunately, this problem could be fixed by creating a 3D antenna pattern from the azimuth and elevation measurements. Information for how this can be done was received from Nokia Principal System Architect Tomi Haapala. Data points from the 3D antenna pattern could be used similarly as using only the azimuth measurement points. This would add more points to the calculations so the time to run the machine learning would increase. To avoid adding too many data points and to keep the running times reasonable, only a few degrees of elevation measurements would be used. This sliced 3D antenna pattern would provide all the information for the calculations to be accurate, while cutting out all the unnecessary data. In FIGURE 15 there is 3D antenna pattern from two beams. (17.)

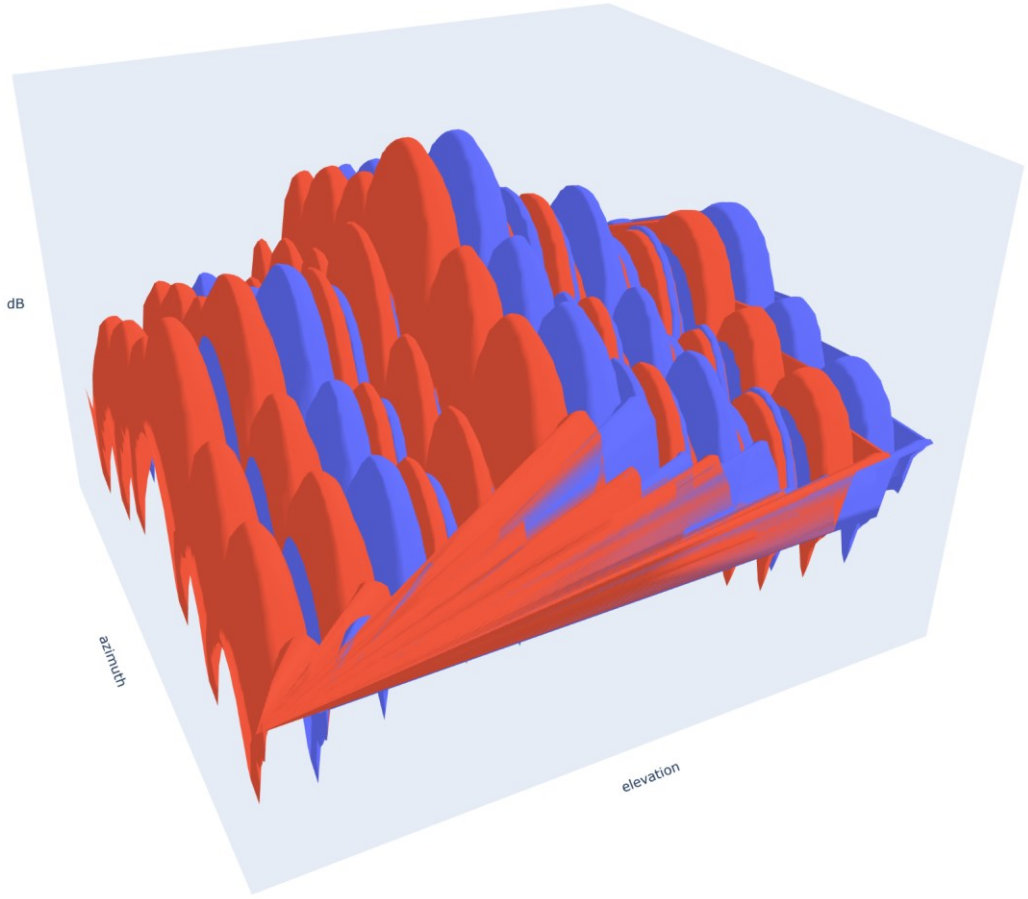

FIGURE 15. If the machine learning is done from 3D antenna pattern these beams would work together. This is the reason for the need to be able to get the 3D antenna pattern from the azimuth and elevation measurements.

4 CONCLUSION

The thesis was done in two parts. The first part was doing the machine learning with the algorithms and everything that was needed for it. The second part was full-stack web development by doing the microservice and integrating it into the cloud-based web service.

In the first part of the thesis the first task was to find correct algorithms to be used and see if they would work in this use case. Working algorithms were found successfully and a working program for those was made. Although algorithms worked and found good beam groups, the progress was rather slow. To speed it up the program could be changed to use GPU's instead of CPU's. This would also need hardware architecture changes. Another option could be moving the microservice to a server with the ability to use more cores. Then the multiprocessing could utilize all those cores and return results faster.

In the end stages of the thesis the azimuth elevation problem rose. Although the fix for this was found, time constraints for finishing the thesis did not allow to implement the fix before finishing the thesis. If this would have been known earlier, it could have been done in time. This problem could have been avoided completely if there would have been more knowledge about beamforming in the beginning of the thesis.

In the second part of the thesis the microservice using the machine learning running on Python was created. It was able to start the machine learning and return results. User interface was the cloud-based web services frontend that used Angular framework. After updates to the frontend users were able to start the microservice and see the results from the machine learning. Updates were also needed to be done on the backend where Flask framework was running on Python. Results were saved to a MongoDB database. Microservice used backends REST API and RabbitMQ for the connections needed. Everything worked together and could be used by users.

In the future the web service could also show 3D antenna patterns of the beams. This could allow better visualization for the users to see how the beams work together. Additional parameters for the machine learning could also be added. In this version some of the choices were hard coded to the microservice so not all the parameters were available for the users.

In the end this was just the first version of this microservice and its implementation. After user feedback it could be updated to better suite their needs. However, this still proved that making machine learning proof of concept and implementing it to that cloud-based web service is possible.

5 DISCUSSION

This thesis became a lot bigger than was initially thought. At first the idea was just to get some sort of machine learning proof of concept to work. After that went quite well, implementing it into the cloud-based web service was added to the thesis. This then added more complexity, but it also allowed to create a machine learning project from the early stages of studying it to the end implementation. To me this was important as it allowed me to show my skills in terms of machine learning as well as the full-stack development perspective.

For me this was an educational project. I had studied machine learning before, but I had not made any real machine learning projects. When studying machine learning the basics are often already available, and implementing everything to work correctly is only needed. This project was completely different. First, I needed to study algorithms and how they could work. Then I had to create proof of concepts of those to see if they would work. After they were verified to work there was the need to implement all to work in a microservice. Then I had to get the real measurement data for the machine learning algorithms and transform the data to correct format. Finally, I had to integrate the microservice to work with the could-based web service.

The thesis also added more to my interest towards machine learning. It was wonderful to see that the algorithms worked, and they found better and better results after each iteration. Even though this was how it was supposed to work, it was still astonishing for me to see it working. The thesis also improved my understanding of what is needed to get a machine learning project to work and what kind of difficulties might be faced when working on it.

Nowadays machine learning and artificial intelligence are buzzwords used in the media. However, a fact often forgotten is how much work is needed to create even an relatively simple machine learning project, let alone a full project that can be used by a regular user. This thesis took around 15 weeks. On average maybe 30 hours a week were used for creating the machine learning program, creating the microservice and integrating it to the cloud-based web service. From that can be calculated that the time spend on working on this thesis was around 450 hours. Is this amount of time acceptable for companies to create machine learning projects? Can they accept that somebody uses that amount of time and money to a machine learning project that might not even work?

These are some questions companies need to think about when they want to run their own machine learning and artificial intelligence programs.

REFERENCES

1. Ehab Ali, Mahamod Ismail, Rosdiadee Nordin, Nor Fadzilah Abdulah 2017. Beamforming techniques for massive MIMO systems in 5G: overview, classification, and trends for future research. Search date 10.11.2023. Beamforming techniques for massive MIMO systems in 5G: overview, classification, and trends for future research | SpringerLink
2. Rappaport Theodore S., Sun Shu, Mayzus Rimma, Zhao Hang, Azar Yaniv, Wang Kevin, Wong George N., Schulz Jocelyn K., Samimi Mathew, Gutierrez Felix 2013. Millimeter Wave Mobile Communications for 5G Cellular: It Will Work! Search date 27.10.2023. Millimeter Wave Mobile Communications for 5G Cellular: It Will Work! | IEEE Journals & Magazine | IEEE Xplore
3. Oxford Reference. Side lobe. Search date 12.11.2023. Side lobe - Oxford Reference
4. Stuart Russell, Peter Norvig 2022. Artificial Intelligence: A Modern Approach. Fourth edition. Pearson.
5. Jason Brownlee. Iterated Local Search From Scratch in Python. Search date 19.11.2023. Iterated Local Search From Scratch in Python - MachineLearningMastery.com
6. Jason Brownlee. Simple Genetic Algorithm From Scratch in Python Search date. 19.11.2023. Simple Genetic Algorithm From Scratch in Python - MachineLearningMastery.com
7. Python. Welcome to Python.org. Search date 19.11.2023. Welcome to Python.org
8. IBM. What are microservices? Search date 27.10.2023. What are Microservices? | IBM
9. Pytest. pytest: helps you write better programs. Search date 19.11.2023. https://docs.pytest.org/
10. Testdriven.io. What is Test-Driven Development? Search date 19.11.2023. What is Test-Driven Development? | TestDriven.io
11. Agile Alliance. TDD. Search date 19.11.2023. What is Test Driven Development (TDD)? | Agile Alliance
12. Python. Multiprocessing — Process-based parallelism. Search date 10.11.2023. multiprocessing — Process-based parallelism — Python 3.12.0 documentation
13. Jason Brownlee. Multiprocessing in Python. Search date 19.11.2023 Multiprocessing in Python - MachineLearningMastery.com
14. Python. Threading — Thread-based parallelism. Search date 10.11.2023. threading — Thread-based parallelism — Python 3.12.0 documentation

15. RabbitMQ. RabbitMQ: easy to use, flexible messaging and streaming. Search date 24.11.2023. RabbitMQ: easy to use, flexible messaging and streaming — RabbitMQ
16. IBM. What is a REST API? Search date 11.12.2023. What is a REST API? | IBM
17. Haapala, Tomi 2023. Principal System Architect, MN RF A&S Radio Sys Arch SG. Nokia. Teams meeting 21.11.2023.