IE 492 PROJECT

FINAL REPORT

Online Frequency Assignment Problem

Supervisor
Tınaz Ekim Aşıcı

Ali Özgür Çetinok 2012402054
Berkin Tan Arıcı 2013402096
Selin Bayramoğlu 2013402105

Abstract

Online frequency assignment problems (FAP) arise in wireless communication domain in a variety of forms. In this project, we consider traveling radiophone users as communication agents in a terrestrial region. If two agents want to make a call and the distance between them is below a threshold they have to be assigned different frequencies, otherwise interference occurs. Given a set of available frequencies, we need to assign a frequency to each call as soon as it happens in such a way that any interference is avoided. If no frequency can be assigned to a call, we assume that the call is dropped. The objective is to minimize the number of dropped calls. The dynamic call network can be represented by a graph where vertices represent the locations of calls and edges between vertices are present whenever the related calls are close enough to each other that can cause interference. Then, the objective is to color this graph with a given number of colors in such a way that no two adjacent vertices get the same color. The problem has an online nature as vertices (and consequently edges) appear and disappear in a given time horizon. In this paper, we provide an experimental study on online FAPs. We use online algorithms from the literature, develop online algorithms and formulate IP models that provide optimal offline solutions using the full instance information. We compare the results and analyze the conditions in which some of the algorithms perform significantly better than the others.

Özet

Çevrimiçi frekans atama problemleri ile kablosuz iletişim alanında farklı biçimlerde karşılaşılır. Bu projede, belirli bir bölgede iletişim, hareket eden telsiz kullanıcılar arasında gerçekleşmektedir. Eğer iki kullanıcı bu bölgenin dışındaki kişileri aramak istiyor ve aralarındaki mesafe önceden belirlenmiş bir eşik değerinin altındaysa, konuşmalara farklı frekansların atanması gerekir. Aksi durumda enterferans oluşacaktır. Belirli bir frekans kümesi varlığında yapılması gereken bir arama yapıldığı anda kullanılabilir frekanslardan birini ona atamaktır. Eğer bir aramaya hiçbir frekans atanamıyorsa, o arama düşürülür. Bu durumda amaç, toplam düşürülen arama sayısını en aza indirmektir. Zamanla değişen aramalar ağı bir çizge ile gösterilebilir. Köşeler aramaların konumlarını belirtirken, bağlar aynı frekans atandıklarında enterferansın oluşacağı (komşu) aramaları birbirine bağlar. Bu durumda amaç, çizgedeki komşu aramaları farklı renklere boyamaya çalışarak, toplam boyanmamış köşe sayısını en aza indirmektir. Ele alınan problemde köşelerin ve bağların zamanla çizgeye eklenip çıkarılması, problemin çevrimiçi oluşunu gösterir. Bu projede çevrimiçi frekans atama problemleri üzerine deneysel bir çalışma sunuyoruz. Önceki çalışmalarda kullanılan çevrimiçi algoritmaları ve geliştirdiğimiz yeni algoritmaları oluşturduğumuz senaryolarda denedikten sonra elde edilen sonuçları, geliştirdiğimiz tam sayılı programlama modellerinin verdiği optimal çevrimdışı sonuçlarla karşılaştırıyoruz. Arama ağlarının yapıları değiştikçe, bazı algoritmaların diğerlerine göre niçin daha iyi sonuçlar verdiğini araştırıyoruz.

Table of Contents

1. Introduction

In today’s world, demand for communication and connection increases tremendously. Satellites, TV broadcasts, cell phone networks and many other technologies use radio waves as their transmission media. A sound or an image can be transferred via waves and connection between two devices can be established in the same manner. However, in a huge network where a lot of devices exist and radio waves are used rather often, the utilization of radio wave frequencies might get tricky. The problem basically arises from the notion of interference, which stands for the situation where radio waves clash with each other and lower the quality of the connection or the content being transmitted.

Frequency assignment has to be viewed from an industrial engineering point in 21st century where the population using cellular data and demand for communication increase. When a user in a region makes or receives a call, a frequency is assigned to the call from the available radio spectrum allocated to the device or service. This frequency remains the same during the call. While an active user is on the phone, other calls may arise in the same region very close to the current one, and they have to be assigned frequencies in the radio spectrum that are different from the ones that are already in use. The assignment of the same frequency to simultaneous calls within very close distance causes interference, which reduces the quality of calls significantly and may result in dropped calls. The limited bandwidth of the radio spectrum requires the reuse of frequencies in a way that minimizes the number of dropped or bad quality calls used and costs associated with them.

In this project, we consider a frequency assignment problem (FAP) in contexts such as military areas and rescue teams. Individuals are situated in a region, each having a communication device. These devices act as base stations, that is to say they don't need a base station to operate and they connect to each other directly. Radiophone is a device that works with this principle, however frequency assignment is nonexistent in these radio phone networks. The transmitter users broadcast the message through a frequency which they adjust manually. Other receivers in the region that adjust their frequencies to the same frequency as the transmitters' can listen to that particular broadcast. Another technology is full duplex device-to-device technology where two devices connect to each other in both directions with a single frequency. Once a person wants to call another person, a central authority allocates a frequency to that pair to be used during the call. So, this technology is more related to the problem studied in this project. However, the constraints or the feasibility of this technology is ignored. This is a fairly new technology being developed currently.

In the real world calls occur in pairs. There is a caller and a receiver. In the first version of the problem examined, only one side of the call might be considered. The paired nature of the call is ignored to simplify the problem. This situation can be observed in real life, too. For example, when a disaster happens, people who are in the region tries to call other people for help that are not in the region. Here the people that are assumed to be out of that region are probably in irrelevant regions, so their interference is not important. Only people who are in the disaster region and the frequencies assigned to them can be considered.

This problem can be modeled as a graph coloring problem. Graph coloring is a problem in graph theory where it is aimed to color the vertices of a graph in a way that no two adjacent vertices share the same color. A proper coloring of the graph can be described as coloring the connected vertices differently. The coloring task is equivalent to assigning a frequency to a call, where assigning the same frequency to any two adjacent calls is not desired. Coloring of an incoming call should be provided at the arrival time of that call. We classify the start and end of calls as events, because they change the state of the network. Whenever a call is initiated, a unit disk appears in the region representing the communication agent. In the graph representation of the region, this agent, represented as a vertex, is connected by an edge to other agents it intersects and it is colored differently from those agents. Similarly, whenever a call ends, the vertex corresponding to the unit disk in the region is deleted from the graph along with its associated edges and its frequency becomes available to possible calls that can occur in short time in close distance.

In the second version of the problem, a more realistic case is considered in which calls are communicating with each other and the implemented algorithms are compared. The validity of the unit disk approach will be lost and additional operations are performed on the network to obtain a graph representation. A similar procedure is followed as the outgoing calls version of the problem.

The online version of this problem is the main focus of this project. An online problem is one where not all the input is known a priori, and decisions must be made based on past events without complete information about the future. The probability of a call arriving at a point in time, its location, start time and duration are the elements that make this problem online. Therefore, whenever a call arrives, a frequency has to be assigned to it right away, if there is any available, without the knowledge of future calls.

The online algorithms developed in this project are applied on different instances and their performances are observed. The offline problem where the input is completely known a priori is solved using these generated instances by integer programming (IP) and the solutions of the online algorithms in terms of the number of dropped calls can be compared against the optimal offline solution.

After implementing these solutions, numerous instances are generated and the results for different heuristics and exact solution are investigated. Pooling the results, statistical analysis is done where the performance measure is the number of dropped calls out of a given number of total calls. The pooling is done in different ways. The instances are primarily grouped by the number of total calls, however there are different groupings such as sparse/dense instances. Conclusions derived from the results indicate that Greedy (First

Available Fit) performed quite well and it is preferable in most cases, bearing in mind its computational convenience. However, the Ring heuristic developed performs better than Greedy (First Available Fit) in sparse networks.

In this report, the approaches encountered in the literature, extended explanation of the steps taken in IP formulation, assumptions, heuristic implementations, proposal of new heuristics and statistical results are presented.

2. Problem Definition, Requirements and Limitations

2.1. Problem Definition

The emerging communication industry created a modern issue to the world: frequency assignment problems. Since there are limited frequencies available to use today and each call needs a frequency to be successful, the optimization point of view in allocation of frequencies to the calls became essential.

A frequency assignment problem is an online problem that needs to be solved by developed heuristics exactly when a call is generated. As discussed in the introduction part, there are two versions of the problem considered in this project, outgoing calls and paired calls. The former follows a graph theoretical approach. The setting can be transformed into a unit disk graph G(V,E) where each call v, $v \in V$, is shown as the center of a unit disk and two simultaneous calls that are within a distance of less than 2 units, where the corresponding unit disks intersect, are connected by an edge. This unit disk approach in communication networks studied in [1] in detail. When a snapshot is taken, the graph shows us the calls going on at the time (active calls) as the current vertices of the graph, and the restriction imposing the assignment of different frequencies to close-enough calls will define the edges of the graph.

The latter, paired calls case, is slightly different. Since there are two different individuals to consider frequency assignment, the unit disk graph assumptions do not hold. In a 1-frequency paired call instance consisting of 2 calls, consequently 4 individuals, if any 2 of 4 individuals are close enough, then all of the 4 individuals are considered to interfere with each other. The enclosed area is transferred into a whole new graph, which paired calls and their intersections combined. By using this graph as the new data set, paired calls solution follows the same procedure as the outgoing calls case. However, the heuristics implemented in outgoing calls that use location information may not be applicable to the paired calls case, since the geographical information is lost during the new graph creation process.

As given in the definition of the problem, the frequencies are assigned in an online manner. That is to say, the frequency is committed to a customer (and this frequency stays the same until the end of that call) without knowing how this particular assignment would affect the future state of the network. The inevitable call losses and preventable inefficiencies can be distinguished and identified by examining some cases. For this purpose, a visual environment is created by using an agent-based simulation tool, NetLogo, with which problematic cases can be identified more easily.

Using this simulation environment different instances are created with different parameters. The most important characteristic of the instances created is the density of the graph. Very sparse networks are not desired where it is trivial to color the graph with given colors. The instances examined have different interference situations and are dense enough. The density of the graph is controlled by a couple of parameters while generating the instances. These parameters are the call arrival probability, call duration, call location and the area of the enclosed region. By trying different parameter combinations, variously populated instances are generated and calls are colored with one of the most basic heuristics, color with the first available color.

One of the situations identified was the so-called P4 structures. The existence of P4 structures makes a graph imperfect as stated in [2]. In online problems, the 2-color colorability property of a P4 structure can be lost easily. Figure 1 below explains such a case.

In this project, occurrence of problematic cases such as P4 is the main focus. While simple heuristics are employed such as First Available Fit, Random and Least Frequently Used, different heuristics are also developed to address the P4 cases stated above. The results of the heuristics on the same instances are compared with each other. Even though the focus was not purely on P4 problems, P4 or similar structures can change the number of calls dropped severely and it is possible that the new heuristics would improve the result by handling such cases.

The heuristics developed are called Greedy Location and Ring heuristics. These heuristics are discussed in detail in the following pages of the paper. One thing to bear in mind is that these algorithms are computationally more expensive compared to Random or LFU, thus can be out of the abilities of the hardware used in the device. Furthermore, the existence of such networks, where the frequency is allocated to a call by an authority is possibly hypothetical. According to the literature review, the communication devices that can connect to each other directly exist [3], however technical issues of this technology regarding frequency assignments and operating principles are beyond our comprehension of signal and communication electronics.

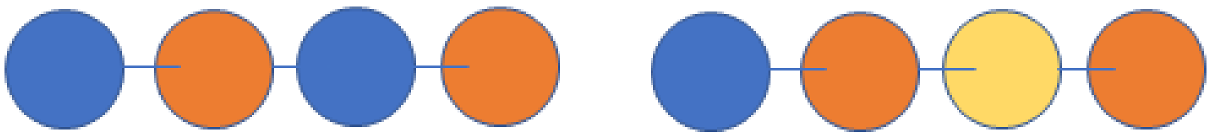


Figure 1: Left: 4 calls with 2 colors, Right: 4 calls with 3 colors

2.2. Requirements and Limitations

The online frequency assignment problem studied in this paper is a complex problem which includes some restrictions. To simplify the problem and make it possible to solve, requirements and limitations are identified and considered.

Millions of individuals are making calls every day which leads to the accumulation of an immense volume of calls in the databases of communication providers. Therefore, gathering data from these providers and observing those data both online and offline would increase the computational complexity. In order to avoid the complexity, a synthetic data is generated by simulation softwares. The optimal solution is complex and NP-Hard by Integer Programming Solution. Keeping the data small would help comparing online algorithms with the best possible solution. Also, since D2D Duplex Technology is considered and it is not widely used, creating synthetic data in simulation makes sense.

2.3. A Systemic View of the Handled Design Problem

In Figure 2 below, the main role is to propose the communication service provider a more efficient way of assigning frequencies, enabling them to provide the same service quality with less number of frequencies, consequently with lower costs. The performance indicator is the number of dropped calls. Here, the aim is to increase the service provided to the customers with the same number of frequencies. Eventually, when the heuristics help preventing the non-inevitable dropped calls, the company could decrease the number of frequencies it owns or ask for more money for the increased quality of the communication with the same number of frequencies it has. In both cases, the company would make more profit and it is a desirable action to take in terms of the gainings.

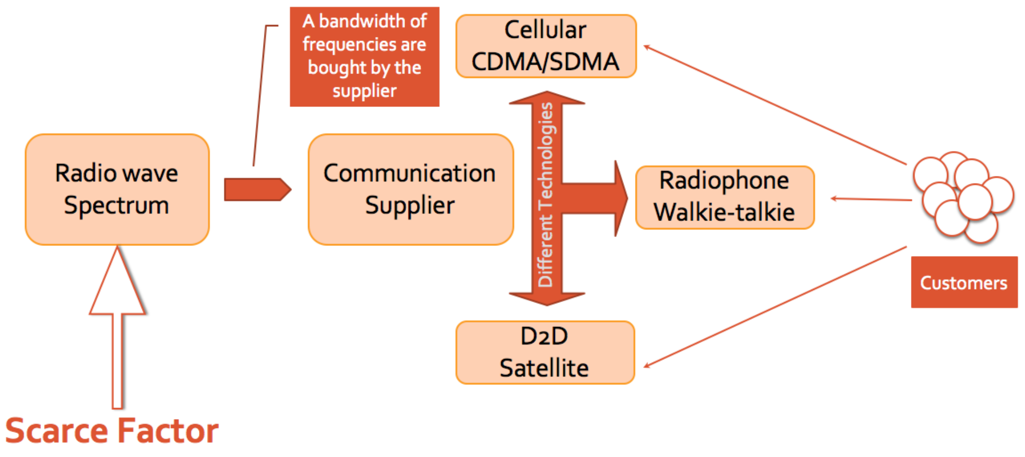


Figure 2: Context Diagram

2.4. Performance Criteria and Potential Improvements

The performance criteria of Online Frequency Assignment project is the number of dropped calls achieved by different heuristics. To measure these performances, the offline solution and solutions generated by other algorithms are compared to each other. With this comparison, best working algorithms for different situations are determined. Deviations from the best possible solution are calculated to measure the overall performance of algorithms.

There is always a space for improvement in this problem. The total number of frequencies can be increased in different situations. Network densities can be rearranged and new algorithms may be developed in order the improve the best algorithm and get close to the lower bound on the number of dropped calls, which was obtained by the integer programming solutions.

3. Analysis for Solution/Design Methodology

3.1. Literature Overview

There are numerous studies on online frequency assignment using graph coloring and network models. However, these papers study the problem in different ways. These ways vary in approaches and assumptions. Most of the papers approach the problem in a theoretic way where they aim to find lower and upper bounds to online algorithms. Yet, there are few studies that approach the problem empirically. In this paper, it is preferred to conduct an empirical analysis on the problem. Here, instead of finding theoretical competitive ratios of algorithms to make conclusions on their performances, they are tested on synthetically created data which imitate real life scenarios.

Different instances are created and the results are pooled to make statistical evaluations.

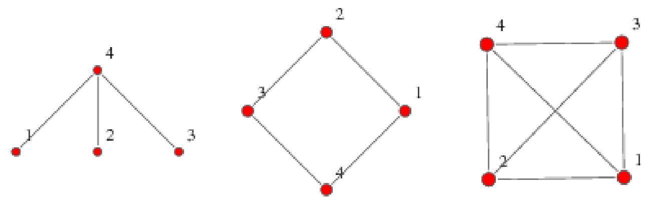
In terms of assumptions made to model the networks, the researches vary sizably. Some papers focus on hexagonal base station areas where they consider calls in these areas [4]. Others take circular or rectangular areas into account. In this paper, the area considered is a rectangular shape. This area can be considered as a coverage area of a base station where calls go out to a region out of this coverage area. Similarly, it can be considered a terrestrial region where device-to-device connections are possible without an infrastructure. This is an emerging technology as explained in [3] and [5].

The way interference is defined varies between different studies. Interference is an effect relying on a distribution. Interference is explained in detail in [6]. However, many studies assume that interference is certain and constant in an area. In this project, the interference is assumed to be constant in a surrounding circular area of a communicating agent. The diameter of this area is a parameter. Here, it is given a constant value which yields enough interference so that the problem is worth studying.

3.2. Alternative Solution Approaches

As discussed in the previous sections, frequency assignment problems need to be solved by algorithm development and implementation. Algorithm development is a non-stopping process, therefore analyzing and comparing the algorithms are significant steps in the solution methodology. All the algorithms generated and implemented are alternative solution approaches to the problem.

Firstly, the common algorithms found in the literature overview were implemented into the problem. NetLogo and C++ platforms are used to implement these algorithms and obtain solutions. NetLogo is an agent based simulation tool which helps to generate synthetic data as well as visualizing the process. The implementation in NetLogo enabled the instances to be solved during the data generation. The calls are generated and at the same time the algorithm decided whether an incoming call can be accommodated or not. Implementing algorithms in two distinct platforms was beneficial for checking the validity of the C++ codes and improve visuality during the data generation process.



$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

Figure 3: Neighborhood Matrix

A graph can be represented in various forms as data structures such as edge list, adjacency list or adjacency matrix. In this project adjacency matrix approach is preferred and referred to as "neighborhood matrix". How this representation works can be understood from Figure 3.

The data generated consists of 4 dimensions: x-coordinate, y-coordinate, start time and the end time of a particular call. After generation of the full data, the neighborhood matrix of that instance is generated by processing the full data. The neighborhood matrix represents the call network throughout the simulation period. The dimensions of the matrix is *# of Calls* ***x*** *# of Calls*. If, say, call *i* is present in the region and in close proximity to call *j*, and furthermore if these two calls are active at the same time, even for a second, the *ij*th entry of the neighborhood matrix is 1. This entry indicates that *i* and *j* may interfere with each other, consequently they can't be assigned the same frequency.

The neighborhood matrix explained incorporates location and time information. Also, the indexes provide information on the arrival order of the calls. These properties of the matrix helps algorithms to solve the coloring problem in an on-line manner, even if the whole instance is fed into the algorithm at once. This is ensured by forcing the algorithms to color the calls starting from the lower indexes to higher indexes, while only considering the neighboring relations between current calls and previous calls.

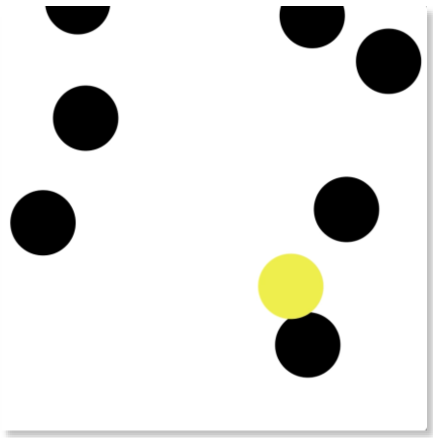


Figure 4: Greedy Algorithm

The Greedy algorithm was the first algorithm implemented. It has a basic structure. Suppose there are four frequencies in the spectrum and each one is represented by a color. The Greedy algorithm is provided with a priority sequence of colors. The first element of the sequence is what the algorithm tries to assign to a new incoming call. If it is not proper to use this color, algorithm continues with the next color in the sequence. In Figure 4, there is a snapshot of the Greedy algorithm’s simulation. Black has a higher priority than other colors, therefore black is assigned whenever it is possible. When black is not available in the spectrum, the algorithm assigns the second high-priority color, which is yellow in this snapshot.

"Least Frequently Used" is a widely used phrase in computer science and industrial engineering literature. The adapted version of LFU Algorithm, which is called simple LFU uses a basic logic. When a call arrives, the system assigns the least frequently used color until that moment to the call. This algorithm tries to balance the number of times different colors are used to accommodate calls.

The Random algorithm is the last one of the common algorithms taken from the literature. Sometimes, making decisions randomly yields good results. What this algorithm does is assigning each available color with an equal probability to an arriving call.

Other than the common algorithms taken from the literature, two additional distinct algorithms are developed. The first one is the Greedy Location algorithm. It has the same structure as Greedy, but location information of the calls are utilized. This algorithm partitions the region we consider into circular sub-regions. In these sub-regions, different color sequences are used by Greedy.

The other algorithm developed is the Ring algorithm. In this algorithm, a similar logic to the Greedy Location heuristic is employed. The region defined this time is not a circular region but a region that resembles a ring around a center. In this defined region, the incoming calls are forced to be called with a different color than the center of the region.

Algorithm development is an essential procedure in the process of frequency assignment, however making comparisons and checking the efficiency of these algorithms is an important process in deciding on the best performing algorithm. While evaluating the algorithms' efficiencies, there must be a reference point to evaluate the results. This reference point must represent the most efficient way of allocating the frequencies to a given instance where we know everything a priori. In this project, two integer programming models have been developed that result in solutions that allocate frequencies optimally but with slight differences that reflect the mechanisms behind the models.

The first IP model yields the absolute best way of allocating frequencies. Given the whole instance, the model minimizes the total number of dropped calls. However, in online algorithms, the common procedure is assigning a new call a frequency among the available ones, according to a preset rule. The IP solution can output a solution in which a call that arrives at a certain time point is dropped to accommodate multiple calls that arrive later. This situation is illustrated in Figure 5. Suppose there is only one frequency and three calls arrive in the order given in the figure, then any online algorithm would keep the first call, and drop the other two calls, as in the picture on the left. However, the IP model would, knowing the whole instance, drop the first one to save the other two, as in the picture on the right.

Although such solutions provide a baseline to compare the performances of online algorithms, the IP model can be modified to reflect the nature of online algorithms so that a more reasonable evaluation of the performances of online algorithms can be made. This concern is resolved in the second IP model (improved IP model) where only additional variables and constraints are added to the previous model. Here, for each call, if there is at least one frequency available, then the model is forced to assign an available frequency to the call. This might not be achieved in the first model.

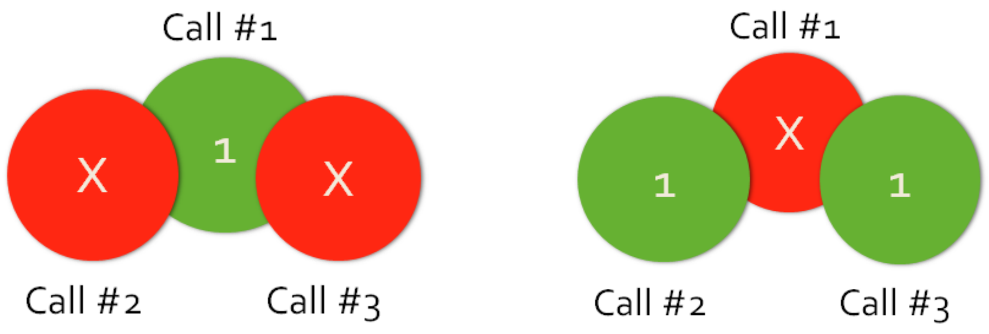


Figure 5: Representation of Frequency Allocation

3.3. Assumptions

In order to avoid complexity and obtain solutions from alternative solution approaches, several assumptions have made.

First, outgoing calls from a certain region is considered. By this assumption, problem is simplified to the Unit Disk Graph Approach and algorithms could be easily implemented and solved through the simulation softwares. After solving this complexity, the paired call case this is similar to real life is adopted and the algorithms are compared in the latter case too.

Frequencies lie in a spectrum which has certain boundaries. However, in this spectrum, there are infinite number of frequencies, since there are infinite numbers. The problem arises because of the interference between close frequency values. When to close frequency values are chosen for intersecting calls, these calls' can mix with each other and call qualities may drop. Therefore, choosing distant frequency values was the second assumption of this project, in order to avoid these situations.

World is a globe that has lots of geographical shapes and disorders. With the result of these disorders, the frequency ranges are not exactly circular, they depend on the height and shapes in that geographical region. To eliminate this issue, Unit Disk Graph approach is implemented. Similarly, interference is an effect relying on a probability distribution. Two close calls may not interfere with each other even if they use the same frequency. The chance of observing interference is inversely proportional to the distance around the communication agent. The following graph depicts the distribution related to this situation. In this project, it is assumed that the interference probability is constant and certain in interference region.

Interference is a widely known concept in frequency assignment. Normally, when two calls intersect and have similar frequencies, they do not drop, interference occurs and call quality drops. In order to simplify the problem, interference is not allowed in this project. There cannot be intersecting calls with the same frequency, so the later arriving one must be dropped if there is no other frequency available for it.

3.4. IE Tools to be Integrated to Implement to the Proposed Methodology

In this project, several different industrial engineering tools have been used. Agent based simulation is used in implementing online algorithms and generating instances. Since time points are discrete, duration times are created using the Poisson distribution in the outgoing calls case, and geometric distribution in the paired calls case. The arrival of a call at a certain point in time is regarded as a Bernoulli random variable with parameter p with which the density of the corresponding graph is adjusted. The algorithms are also implemented using C++. Operations research tools are used to formulate the integer programming models. The statistical tool R is used to create instances in the paired calls case and in both cases, it creates neighborhood matrix which gives adjacency relations between calls. In evaluating the performances of online algorithms, statistical tools are employed.

3.5. Brief Overview of the Selected Approaches

Instances are generated in NetLogo and solved by the online algorithms exactly when they are generated. These instances are also solved by C++ to validate the results in NetLogo. After having the full data at the end of simulation, the data obtained is used in integer programming by R and Gurobi softwares. Later, the results of each algorithm and offline solution are compared in order to find out which algorithms act better in different conditions and their variations from the best possible solutions. This procedure is summarized in Figure 6 below.

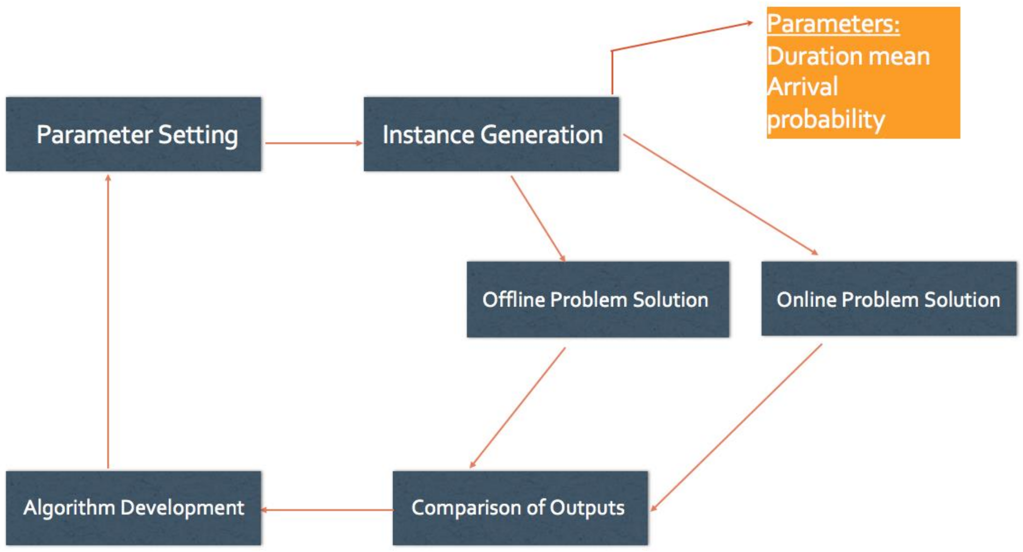


Figure 6: Overview of the Design Process

4. Development of Alternative Solutions

As explained above, the instances are synthetically generated with different parameter settings. Instances are examined and compared according to objective function values and parameters. Instances where we observe dense networks results in unacceptable ratios of dropped calls, which is worth studying. Firstly, Greedy, LFU and Random algorithms are implemented and compared with each other. On the other hand, the improved IP models are employed to solve the same instances to find an exact solution.

By comparing the results of IP and other heuristics, how far the results of the algorithms are from the optimal case is measured. This optimality gap is considerably big (around 60-80%) and we wanted to come up with different algorithms to close this gap.

The problems or cases that might lead to an inefficient way of allocation are identified by visually examining the instances. Two algorithms are proposed to eliminate the occurrence of such cases. These two algorithms are Greedy Location and Ring algorithms. These algorithms’ development processes and pseudo codes will be discussed in this section. The Figure 5 above illustrates the design and development process of algorithms.

4.1. Greedy Algorithm

Greedy Algorithm was the first algorithm implemented. It is implemented in C++ (Appendix A) and NetLogo (Appendix B). NetLogo solves the problem dynamically as the

```
for all calls i do
    for calls j in 1..i do
        if neighbor[i,j]=1 then
            if colorOf(j)=colorONE then colorONECounter++; (repeated for all colors)
        end if
    end for
    for c in {a,b,c,d}
        if colorONE=0 colorOf(i)=a;
        else if colorTWO=0 colorOf(i)=b;
        else if colorTHREE=0 colorOf(i)=c;
        else if colorFOUR=0 colorOf(i)=d;
        else drop the call;
set all color counters to 0;
end for
```

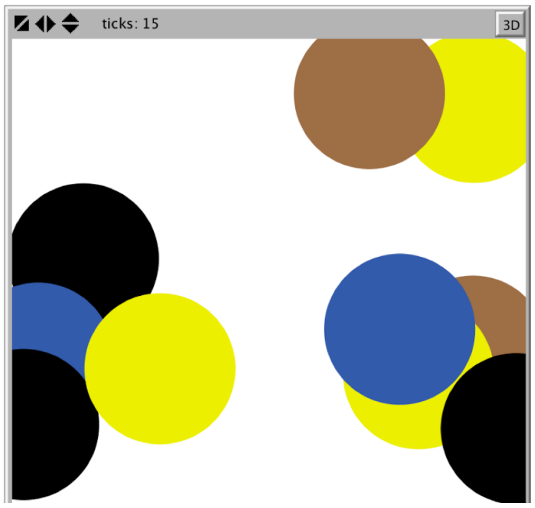
Figure 7: Pseudo Code of Greedy Algorithm

data is generated. The pseudo code is given in Figure 7 below:

The C++ code is provided with the neighborhood matrix and a given number of colors. As a procedure, what the algorithm does is, starting from them lowest indexed calls, it checks whether there are other previous calls in its neighborhood colored with the first color in the color sequence. If there is not, it colors the call with that color and moves on to the next indexed call, otherwise it move to the next color in the sequence. If no color is available in that region, the call is dropped and moved to the next call.

4.2. Simple Least Frequently Used Algorithm

Simple LFU Algorithm is another algorithm that was encountered in the literature. It is used in many other problems in industrial and computer engineering. The procedure is pretty simple. The algorithm keeps track of number of each frequency used throughout the horizon. When a call arrives, it assigns the least used frequency on that time from the available frequencies. It is implemented in C++ (Appendix C) and NetLogo.



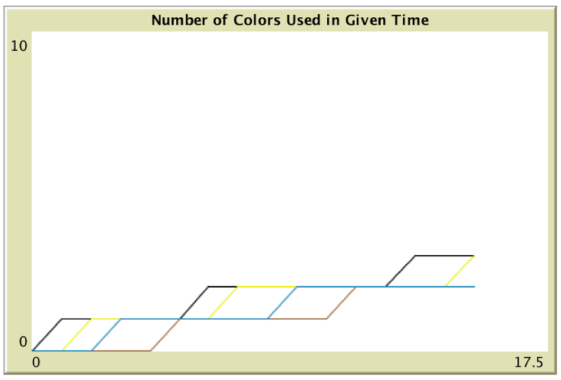


Figure 8: Simple LFU Algorithm

In Figure 8 above, the Simple LFU algorithm is depicted in NetLogo Software. The graph illustrates the number of times frequencies are used. It can be seen from the graph that the frequencies try to catch up with each other. The pseudo code is provided in Figure 9 below.

The procedure of the algorithm is, it starts with the lowest indexed call. It checks what color is the one that is used least up until that moment. If that color is available to that call at that location, it assigns that color. If that color is not available, the next least frequently used color check for availability. If no color is available for that call, it is dropped. This algorithm tries to use every color as much as possible, creating an evenly colored network. The pseudo code of the Simple LFU algorithm is given in Figure 9 below:

```
for all calls i do
    for calls j in 1..i do
        if neighbor[i,j]=1 then
            if colorOf(j)=colorONE then colorONECounter++; (repeated for all colors)
        end if
    end for
    if any colorCount() is not 0
        for all color c in {a,b,c,d}
            if overallCount(a)<bigM & colorCount(a)=0 then
                colorOf(i)=a & LFUvalue=overallCount(a) &overallCount(a)++;
            if overallCount(b)<LFUvalue & colorCount(b)=0 then
                colorOf(i)=b & LFUvalue=overallCount(b)& overallCount(b)++;
            if overallCount(c)<LFUvalue & colorCount(c)=0 then
                colorOf(i)=c & LFUvalue=overallCount(c)& overallCount(c)++;
            if overallCount(d)<LFUvalue & colorCount(d)=0 then
                colorOf(i)=d & LFUvalue=overallCount(d)& overallCount(d)++;
    else drop the call;
set all color counters to 0;
end for
```

Figure 9: Pseudo Code of LFU Algorithm

4.3. Random Algorithm

Random algorithm is another heuristic taken from the literature. As the name suggest, it does a random assignment. The C++ implementation can be found in Appendix D. The pseudo code is given in Figure 10 below:

```
for all calls i do
    for calls j in 1..i do
        if neighbor[i,j]=1 then
            if colorOf(j)=colorONE then colorONECounter++; (repeated for all colors)
        end if
    end for
    if any colorCount() is not 0
        while(colorOf(i)=NULL)
            pick a color randomly;
            if colorCount(pickedColor)=0 colorOf(i)=pickedColor;
        end while
    else drop the call;
set all color counters to 0;
end for
```

Figure 10: Pseudo Code of Random Algorithm

The procedure is straightforward. If even a single color is available to an incoming call, algorithm selects a color randomly and checks if that color is available to that call. If yes, the call is color, otherwise algorithm picks a color randomly again. This process continues until an available color is picked. The algorithm continues with the next indexed call.

4.4. Greedy Location Algorithm

GL tries to address the problem with a simple approach. The way this algorithm solves the defined problem can be explained as follows.

When a call is received at a certain location, if that location is not owned by another call (in other words, that location is not in the surrounding circle of another call), algorithm defines a surrounding circular region to this call and it becomes the center/owner of that region. Otherwise, if this incoming call is situated in an already defined circular region, the call is associated with the center of that region. Each call that is a region center/owner represent a sequence of frequencies. This sequence is used by the First Fit algorithm employed in that region. Naturally, the center of the call is colored with the first color of that sequence. An incoming call is given the same frequency of the centered call if it is possible. This algorithm basically aims to tackle cases as illustrated below in Figure 11. Note that GL algorithm is used in Case 2, however in Case 1, there is a different algorithm.

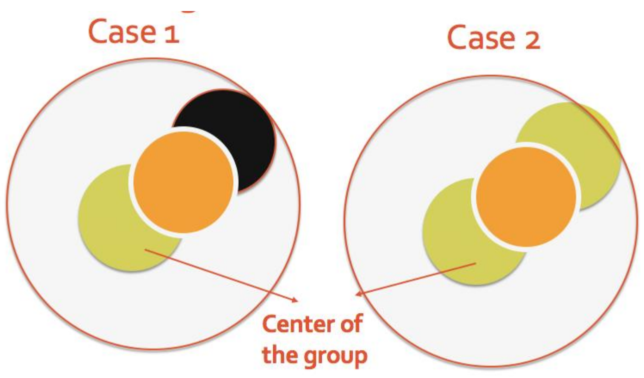
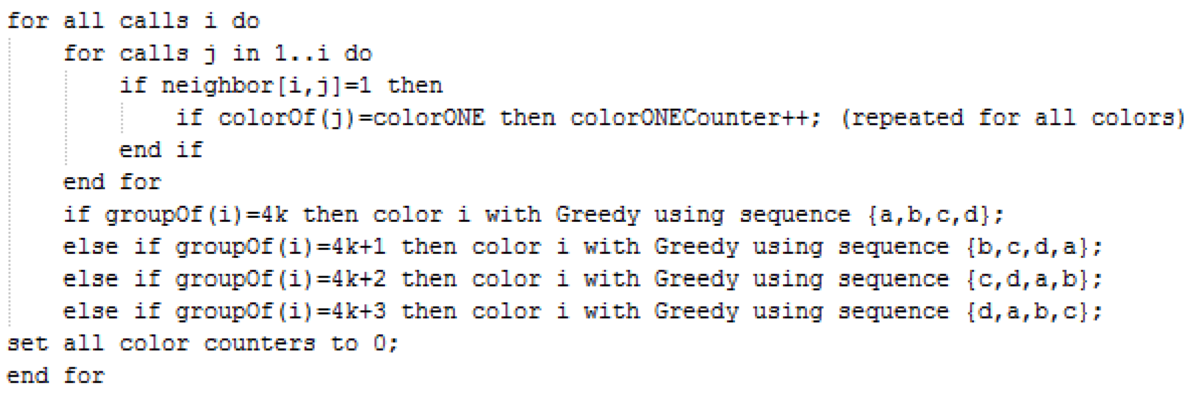


Figure 11: Visualization of GL Algorithm

In both illustrations of Figure 10, the centers of these regions are the ones that arrived first. In case 1, the second call is colored black for whatever the reason (the outside of the circle is ignored). The third call arrives and is colored with orange. Here three colors are used to color three calls. However, if the second call was forced to be colored with the same color as the center of the region (as can be seen in the case 2), three calls would be colored by two colors.

By forcing the incoming call to be colored the same color as the center of the region, the usage of that color is maximized and the region becomes more suitable to use other colors in the emergency cases. The C++ code can be found in Appendix E. The pseudo code is provided below in Figure 12. The neighborhood matrix and group matrix is evaluated in R and provided to the algorithm.



```
for all calls i do
    for calls j in 1..i do
        if neighbor[i,j]=1 then
            if colorOf(j)=colorONE then colorONECounter++; (repeated for all colors)
        end if
    end for
    if groupOf(i)=4k then color i with Greedy using sequence {a,b,c,d};
    else if groupOf(i)=4k+1 then color i with Greedy using sequence {b,c,d,a};
    else if groupOf(i)=4k+2 then color i with Greedy using sequence {c,d,a,b};
    else if groupOf(i)=4k+3 then color i with Greedy using sequence {d,a,b,c};
set all color counters to 0;
end for
```

Figure 12: Pseudo Code of GL Algorithm

The group matrix is generated in this way: starting from the first call, the R code assigns a group ID to that call and the calls that are in the group of that call. Once the first call and its group is constructed, it moves to the next lowest indexed call that has not been group yet. That call and the calls that are in its group receives another ID. This procedure continues in this manner. The group IDs are basically the index numbers of the groups.

4.5. Ring Algorithm

This rules aims to tackle situations illustrated in Figure 13 below. On the left hand side, there is an illustration of a ring. In the Ring algorithm, the color of the center has the least priority to be used to color the calls in shaded area. To the right, a situation where this rule can work by using two colors instead of three in coloring four calls.

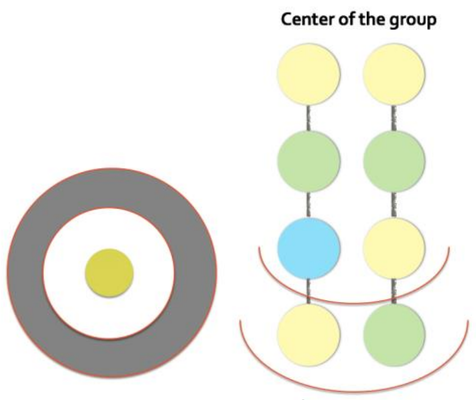


Figure 13: Visualization of Ring Algorithm

The procedure is as follows: When a call arrives, if it is not falling into the ring of another call, it becomes the center of that group and it is colored with an available color with Greedy. Otherwise, if the call is in the ring of another call, that call is associated with the center of that group. This call is colored with Greedy using a sequence where the last color in the sequence is the color of the center of that group. The C++ code is provided in Appendix F with the neighborhood and group matrices. The pseudo code can be seen below in Figure 14:

```
for all calls i do
    for calls j in 1..i do
        if neighbor[i,j]=1 then
            if colorOf(j)=colorONE then colorONECounter++; (repeated for all colors)
        end if
    end for
    if groupOf(i)>0 then color i with Greedy using sequence {a,b,c,d};
    else if groupOf(i)<0 then do
        find the color of the call k with groupOf(k)=-groupOf(i)
        if colorOf(k)=a then color i with Greedy using sequence {b,c,d,a};
        else if colorOf(k)=b then color i with Greedy using sequence {c,d,a,b};
        else if colorOf(k)=c then color i with Greedy using sequence {d,a,b,c};
        else if colorOf(k)=d then color i with Greedy using sequence {a,b,c,d};
    end else if
set all color counters to 0;
end for
```

Figure 14: Pseudo Code of Ring Algorithm

The group matrix here is slightly different than the one in GL algorithm. Here, the center of the group gets indexed with the order of that group and the rest of the calls that are in the ring part of that group gets the negative of ID of that group. This was necessary to identify the center and the other members of that group.

4.6. The Initial Offline Solution

After obtaining the results of the online algorithms applied on the instances, it is important to observe what the optimal assignments of frequencies would be if the whole instances were known beforehand. Therefore, two integer programming (IP) models are developed to obtain offline solutions to the online problem. It must be noted that without obtaining an offline solution, it is still possible to compare the performances of different online algorithms applied on the same instances. However, knowing the offline solutions makes possible the evaluation of how good the online solutions are in terms of their deviations from the optimal objective values. For example, if some algorithms yield results that are in general too far from the optimal values, then those algorithms should be improved.

There are two parameters and a neighborhood matrix that are input into the models. In both of the models, N is the number of calls and K is the number of frequencies. Let i be call index ($i = 1,2,...,N$), and k be frequency ($k = 1,2,...,K$). It is important to recall the assumption that no two calls can arrive at the same point in time. Therefore, the way the instances are output and read ensures that the call indices actually give the order in which calls arrive, e.g. the call with index i = 1 arrives first. The neighborhood matrix is $A[i,j]$. It is a binary $N \times N$ upper triangular matrix which stores the adjacency relationships between calls and an entry $A[i,j]$ is equal to 1 if and only if call i arrives before call j ($i < j$), they are close enough (within a distance of 5 units), and are present in the network together for at least one unit of time.

In the first model, there are two sets of decision variables and they are all binary. The first one is $e[i]$ which is defined for all calls. $e[i]$ gets the value of 1 if call i is dropped in the solution, and 0 if it is not dropped and assigned a frequency. The second set of variables is $c[i,k]$ which indicates whether or not call i is assigned frequency k.

Given the parameters, neighborhood matrix and decision variables, the first IP formulation is defined as follows:

Minimize
$$\sum_{i=1}^{N} e[i]$$
subject to
$$\sum_{k=1}^{K} c[i,k] = 1 - e[i] \quad \forall i, \tag{1}$$
$$A[i,j] * (c[i,k] + c[j,k]) \leq 1 \quad \forall k, \forall i, for\ j = i+1, i+2, \ldots, N, \tag{2}$$
$$e[i] \in \{0,1\} \quad \forall i, \tag{3}$$
$$c[i,k] \in \{0,1\} \quad \forall i,\ \ \forall k. \tag{4}$$

In the above model, the objective is to minimize the number of dropped calls. In constraint (1), the state of the call is determined by checking if any frequency is assigned to it. The left hand side of the equation can either get 0 or 1. If it is 0, then it has not been assigned any frequency in the solution, which forces it to be dropped. However, if it is assigned, this makes the sum 1 and forces the call to be kept in the system. Constraint (2) makes sure that if calls i and j, i < j, are neighbors of each other, they cannot be assigned any frequency at the same time. Either only one of them can be assigned that frequency or both are not assigned. Note that this constraint poses no restrictions on the assignments of non-neighbor calls and also neighbor calls i and j, where i>j, due to the upper triangular structure of the neighborhood matrix and saves computational work. Constraints (3) and (4) indicate the binary structure of the decision variables. The CMPL implementation of this model can be found in Appendix G.

With the offline solutions for instances, we can determine the percentage deviations of solutions of the online algorithms from the optimal results. Although these results are useful, there might be situations in which dropping an earlier call when there are available frequencies would save later coming calls and this formulation would choose to do that if it improves the objective value. Such results might not be suitable to compare with the results of the online algorithms because in online algorithms, a frequency is chosen according to a rule among the available ones and if there is at least one available frequency when a call arrives, then one cannot simply decide to drop the call. In order to incorporate the online structure into the IP model, an additional set of variables and constraints are included in the second IP model (namely improved IP).

4.7. The Improved Offline Solution

The newly introduced variable is $y[i,k]$ which is a binary variable that indicates if frequency k is ever assigned to the previous calls in the neighborhood of call i.

Given the previous IP model and the additional set of variables, here is the Improved IP formulation:

Minimize
$$\sum_{i=1}^{N} e[i]$$

subject to

$$\sum_{k=1}^{K} c[i,k] = 1 - e[i] \quad \forall i, \tag{5}$$

$$A[i,j] * (c[i,k] + c[j,k]) \leq 1 \quad \forall k, \forall i, for\ j = i+1, i+2, \ldots, N, \tag{6}$$

$$\sum_{j=1}^{i-1} A\,[j,i] * c[j,k] \geq y[i,k] \quad \forall i, \forall k, \tag{7}$$

$$\sum_{j=1}^{i-1} A\,[j,i] * c[j,k] \leq M * y[i,k] \quad \forall i, \forall k, \tag{8}$$

$$\sum_{k=1}^{K} y\,[i,k] \geq K * e[i] \quad \forall i, \tag{9}$$

$$\sum_{k=1}^{K} y\,[i,k] \leq K - 1 + K * e[i] \quad \forall i, \tag{10}$$

$$e[i] \in \{0,1\} \quad \forall i, \tag{11}$$
$$c[i,k] \in \{0,1\} \quad \forall i, \forall k, \tag{12}$$
$$y[i,k] \in \{0,1\} \quad \forall i, \forall k. \tag{13}$$

The objective function and constraints (5), (6), (11) and (12) were already in the initial formulation. Constraints (7) and (8), and constraints (9) and (10) work in pairs. The former two make sure that if frequency *k* is ever assigned to a call *j*, *j* = 1,2,...,*i*-1, which is also a neighbor of call *i*, then this forces *y*[*i*,*k*] to be 1. If it has never been used before, then this means that frequency *k* is available to call *i*, which makes *y*[*i*,*k*] 0. The latter two constraints link *y*[*i*,*k*]'s to *e*[*i*]. For call *i*, if the sum of *y*[*i*,*k*]'s over *k*'s is *K*, then no frequency is available and this forces *e*[*i*] to be 0. However, if the sum is less than K, meaning that at least one frequency is available for use, then the model is forced to keep call *i* in the network and *e*[*i*] is forced to be 1. Finally, constraint (13) indicates the binary structure of variables *y*[*i*,*k*]. The CMPL implementation of this model can be found in Appendix H.

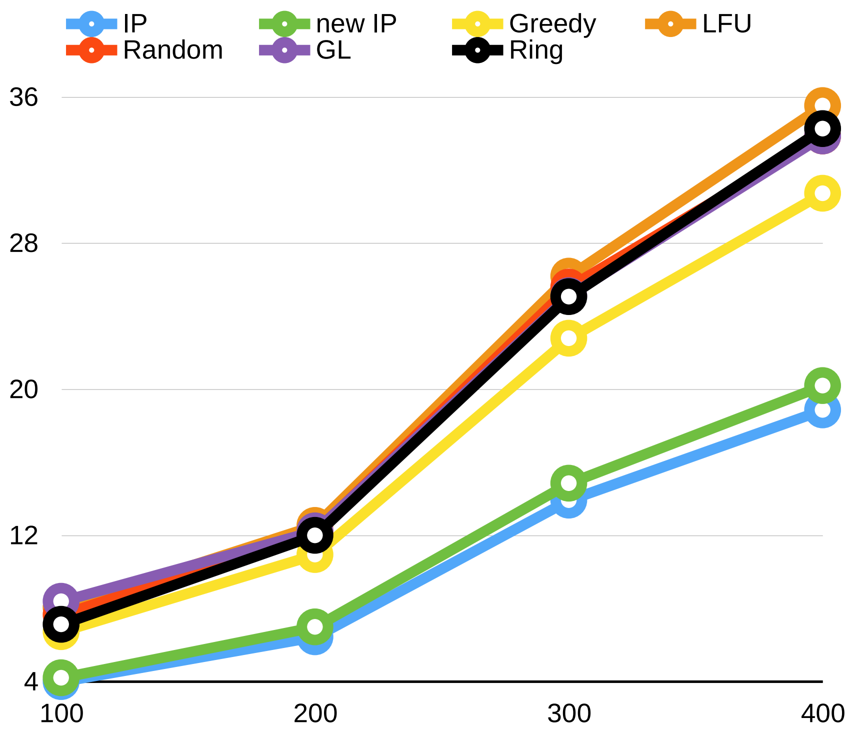
The offline results of the improved IP formulation has become more comparable to the results of the online algorithms. It is important to note that the improved formulation is computationally more expensive. In the first model, there were *N* *e*[*i*] and *N x K* *c*[*i*,*k*] variables, making a total of *N x (K+1)* variables. However in the improved model, we have *N x K* additional variables due to the introduction of *y*[*i*,*k*], which makes the total number of variables in this model *N x (2K+1)*. In addition, the number of constraints also increase in the improved model. In the first model, there were *N* constraints coming from Constraint (1), and *N x* (*N* - 1)/2 constraints coming from Constraint (2). In the second model, there are *N x K* constraints from Constraint (7), *N x K* constraints from Constraint (8), *N* constraints from Constraint (9) and *N* constraints from Constraint (10), in addition to those in the first model. Therefore, given an instance, it takes much more time to get an offline solution from the improved model. Also, it has been observed that not only the number of variables and constraints, but also the number of non-redundant constraints matter in terms of computational efficiency. For example, given the number of calls, an increase in the probability of a call arriving makes the graph denser and adds more adjacency relations to the system, making more constraints non-redundant and it takes more time than the situation where there is a lower probability. In cases where the running times of the IP models exceed two minutes, the performances of online solutions have been compared without checking their deviations from the optimal offline solutions.

5. Comparison of Alternatives and Recommendations

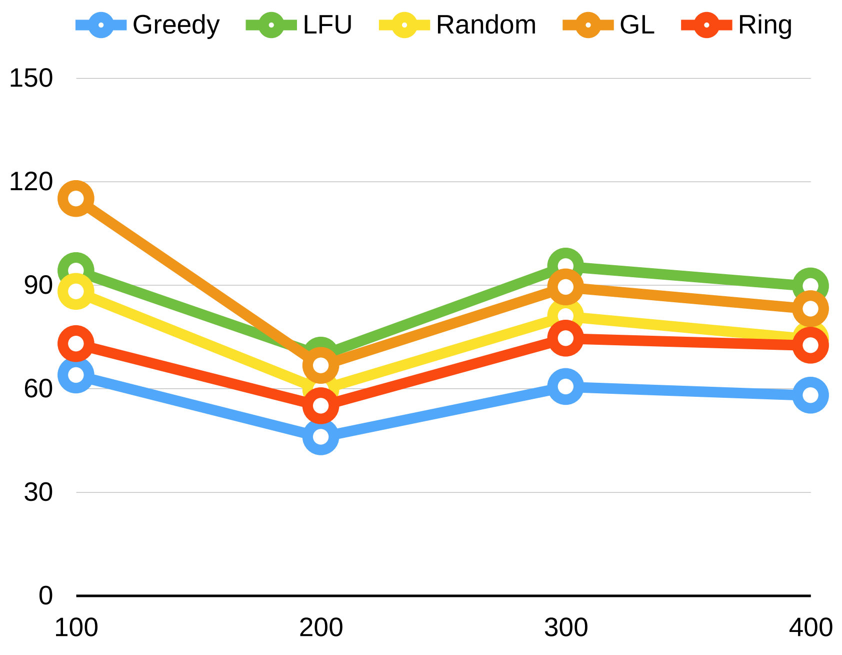
5.1. Numerical Studies and Evaluation Procedure

The results of these developed algorithms are recorded with the same instances where other algorithms and the IP models were employed. These results contain instances with various settings and characteristics. Instances are created in a way that vary in density because pooling the results of different settings would give a robust statistical inference. However, the data is also pooled by different features also. Whole instances are divided into two where in one group we put sparse networks and in the other dense networks.

In overall results the greedy algorithm works the best as we already expected. They are given in Graphs 5.1 and 5.2 below.



Graph 5.1: Comparison of Overall Results



Graph 5.2: Comparison of Overall Results (in Percentages)

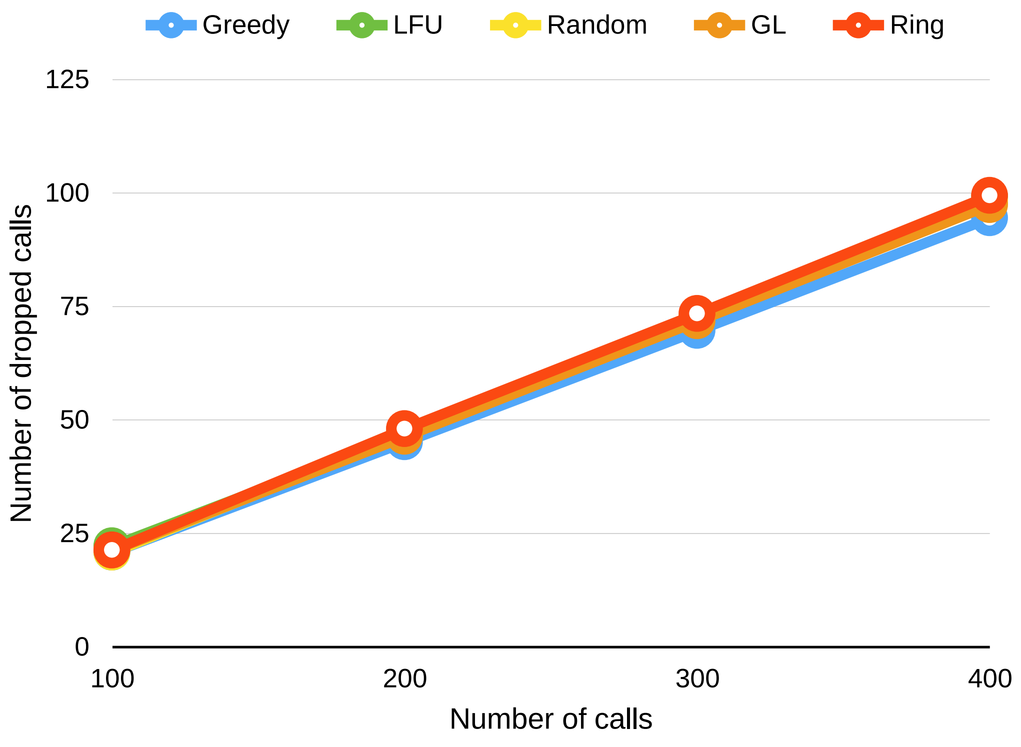
Graph 5.1 represents the number of calls dropped out of the total number of calls. Graph 5.2 represents the percentage deviation from the best possible solution, which is the improved IP solution. Table 5.1 represents the percentage deviation values from improved IP.

Number of Calls	**Greedy**	**LFU**	**Random**	**GL**	**Ring**
100	64	94	88	115	73
200	46	70	60	67	55
300	61	95	81	89	75
400	58	90	74	83	73

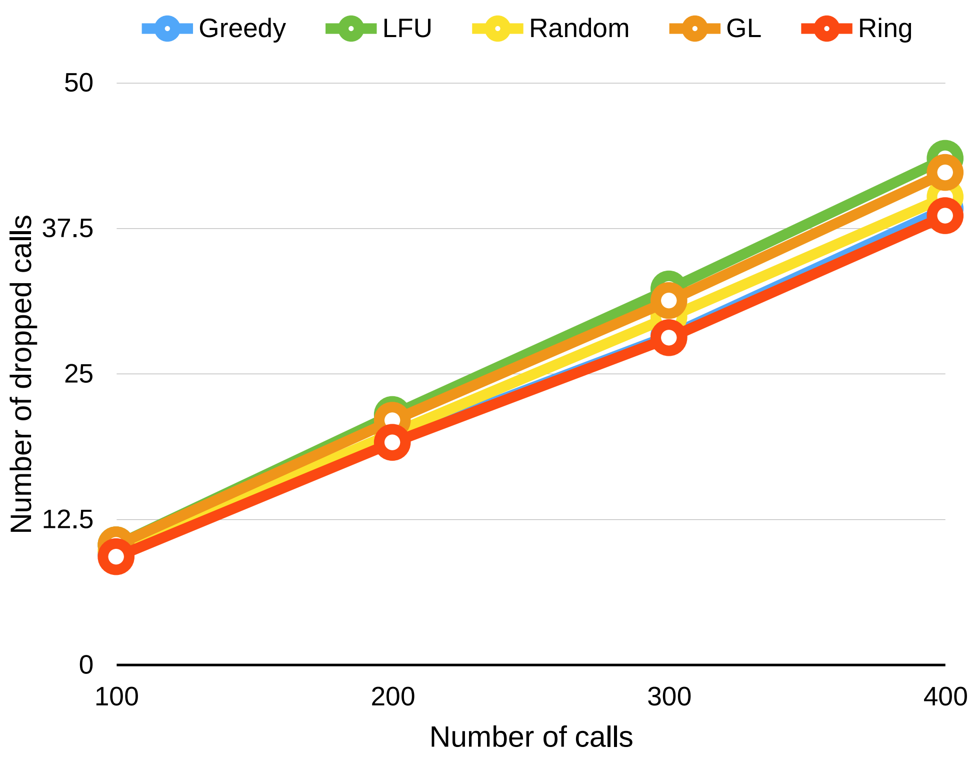
Table 5.1: Percentage Deviations from IP Solution

When the overall results are observed, the Greedy algorithm is 57% worse than the best possible solution on average. Simple LFU is 87%, random is 76%, GL is 89% and Ring is 69% worse than the improved IP solution.

On the other hand, when the data is pooled according to the instance density, an interesting result is observed. In sparse networks, the Ring algorithm works better than the Greedy algorithm, which is the best algorithm in dense groups. These results can be seen in Graphs 5.3 and 5.4 below.



Graph 5.3: Results in Dense Networks

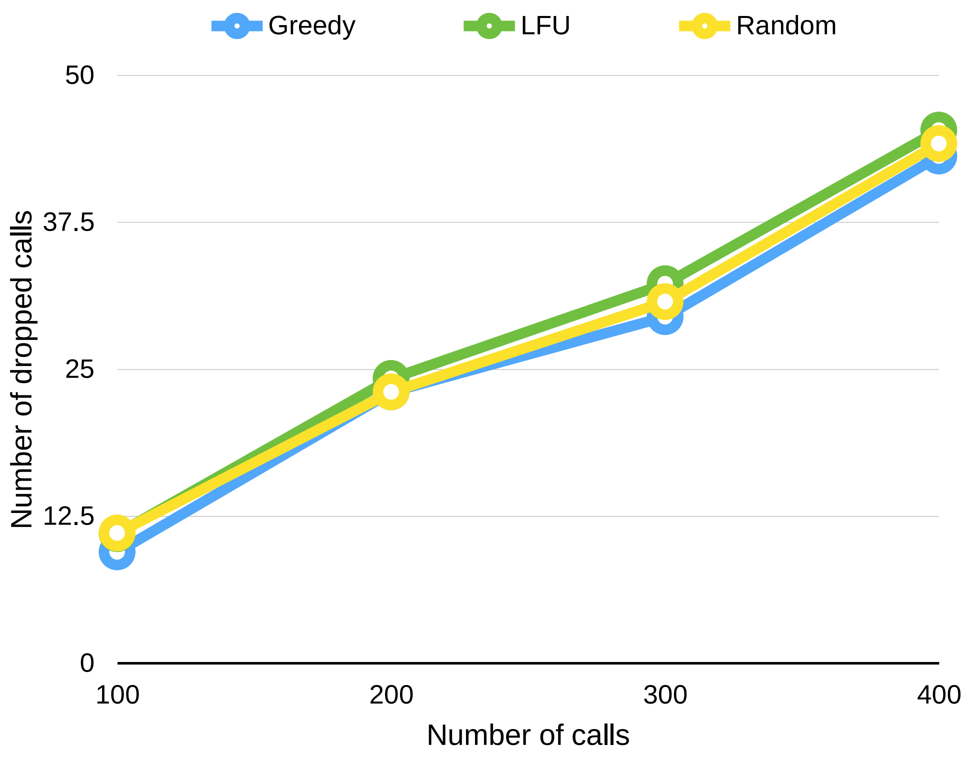


Graph 5.4: Results in Sparse Networks

Although the Ring algorithm works best in sparse networks, it gives the worst performance in dense networks. The reason of it is investigated for this situation. The reason is that, in sparse networks, different rings and groups do not intersect with each other so the way the algorithm colors the calls that arrive into the ring has a meaning. However, in dense networks, a lot of different groups intersect with each other so the meaning of the way the algorithm colors the calls in the rings conflicts and loses its meaning.

After obtaining these results, the paired calls case is investigated with three possible algorithms to implement: Greedy, LFU and Random. The ranking of these algorithms did not change in the paired calls case, too and the results for this case is shown in Graph 5.5.

The results for all instances in both outgoing calls and paired calls settings can be examined in Appendices I and J.



Graph 5.5: Results in Paired Calls Case

5.2. Justification of the Solution

5.2.1. Limitations

The limitations were mainly time related. For example, in the paired calls case, when the number of calls was greater than 100 and the call arrival probability was 0.9, the improved IP solutions could not be generated within two minutes, a running time threshold that had been preset. Therefore, the results of the online algorithms could not be compared with optimal offline solutions. Also, for each combination of parameters, five instances were generated using different seeds. In order to evaluate the performances and robustness of the algorithms in a more reliable way, it is suggested that more instances should be generated.

5.2.2. Sensitivity

It is desired that online algorithms are robust to changes in circumstances. When a parameter is slightly changed, it is expected that in general the algorithm that used to perform best again performs best. This is due to the fact that parameters are uncertain and may be approximations to real life situations, and small deviations from real life parameters should not change the performances of online algorithms drastically if they are to be applied in real life cases.

For all algorithms, it is observed that changing the number of calls all other things being equal does not affect the performances of the online algorithms. So it can be concluded that the algorithms proposed in this project are robust to changes in the number of calls. However, changing the call arrival probability from 0.5 to 0.9 changed the success ranking (Greedy performed best when it was 0.5, but Ring performed best when it was 0.9), so the online algorithms are sensitive to changes in the call arrival probability.

5.3. Further Assessment of the Recommended Solution

The way the problem is studied in this paper follows scientific steps. The problem is online and highly stochastic, so theoretic approaches might not give good solutions to the real life problem. The assessments of the algorithms based on instance generation that are created with different parameters captures the real life variability of the networks. The algorithms suggested based on this evaluation should give decent results when applied to the real life.

The results of the newly developed Ring algorithm on sparse networks performs a cut above. It is surprising that such a complex algorithm performs decent. However, the variability in the performance of this algorithm is the downside of its complexity.

6. Suggestions for a Successful Implementation

The heuristics presented in this paper are developed under certain assumptions. The results in real life may not be perfectly correlated. A communication provider has to consider these simplifications if they want to realize this improvement in their network.

A good combination of solution can be using Ring algorithm during times where the network is sparse. Sparsity of the network can be measured in different ways as suggested in the literature as can be seen in the *Measuring Sparsity* chapter of [7]. The provider can set a threshold of sparsity and once that threshold is reached, the algorithm can switch to First Fit, which performs better in dense networks. By choosing this way while assigning frequencies, the total dropped call number will be minimized in both sparse and dense networks. For instance, when an earthquake occurs in the meantime, networks will be dense and algorithm can switch to First Fit.

A successful implementation of this project depends on the current technological advancements, too. First of all, the algorithms has to be supported by the hardware of the provider. For example Ring algorithm can be computationally expensive. Also, the network has to be using a communication technology which detects the exact locations of the calls and shares the other arriving calls location to those devices. This communication technology must support device to device Duplex technology as well. Frequencies must assigned from the main device which all devices in the network has an access. These are all advanced technological improvements and relate to high investments. Cost benefit analysis must be conducted in order to check that if the investment is worth it.

In this project, the simulations were containing total calls between 100 and 400. These low call numbers result in low dropped call numbers and little differences between different algorithms. However, when real life case is investigated, there are millions of calls and the difference in algorithms would grow to ten thousands. Ten thousand customers' satisfaction is significant for a communication provider.

After implementation of this project, the algorithm development stage must always continue until reaching the best possible solution. Developing algorithms have little costs, but when a perfect algorithm is implemented the costs of communication provider may drop sharply.

7. Conclusions and Discussion

In this report, design and implementation of the frequency allocation problem was discussed. The main aim was minimizing the dropped calls in a communication network by using frequency allocation algorithms. To reach this aim, different online algorithms are developed and compared with each other and the offline solution.

The main industrial engineering tools used in this project are optimization, simulation and statistics. Optimization methods were used in the integer programming solution of the model. Simulation is used while solving the online problems and instances were generated with respect to several distributions, which are a sign of statistical tools used.

As discussed in this report, frequency assignment is an emerging problem in the current developing world and techniques will get better as time passes and researches increase. It was really significant to address this problem and come up with several algorithms to improve the current structure. After this project, the algorithm development stage must continue to achieve better results, which are closer to the best possible solutions. A further step may be applying these algorithms to the real data obtained from a telecommunication company in order to verify the algorithm and check that if D2D Duplex technology would be available to use in certain locations, which can minimize the total cost. In addition, the simplifications and assumptions that made in this project to increase solvability may be changed with real life conditions too.

Finally, implementing this project may seem too far as D2D technology is very rarely used right now, communication industry is exponentially growing for the last 10 years. Being prepared for such a technological advancement will be a significant advantage for a communication provider, in order to minimize the current costs and focus on the algorithm development stage for frequency allocation. This project is one of the first steps of the preparation period for new technological advancements and generated algorithms stated in this paper will be strong candidates for the frequency assignment problem.

8. References

[1] Clark, Brent N., Charles J. Colbourn, and David S. Johnson. "Unit disk graphs." Discrete Mathematics 86.1-3 (1990): 165-66. Web.

[2] Hougardy, Stefan. "Perfect graphs with unique P4-structure." Discrete Mathematics 165-166 (1997): 421. Web.

[3] Tehrani, Mohsen Nader, Murat Uysal, and Halim Yanikomeroglu. "Device-to-device communication in 5G cellular networks: challenges, solutions, and future directions." IEEE Communications Magazine 52.5 (2014): 86. Web.

[4] Chan, Joseph Wun-Tat, Francis Y. L. Chin, Deshi Ye, and Yong Zhang. "Online frequency allocation in cellular networks." Proceedings of the nineteenth annual ACM symposium on Parallel algorithms and architectures - SPAA 07 (2007): 1-2. Web.

[5] Fodor, Gabor, Stefano Sorrentino, and Shabnam Sultana. "Network Assisted Device-to-Device Communications: Use Cases, Design Approaches, and Performance Aspects." Smart Device to Smart Device Communication (2014): 135. Web.

[6] Heath, Robert W., and Marios Kountouris. "Modeling heterogeneous network interference." 2012 Information Theory and Applications Workshop (2012): 1-2. Web.

[7] Nešetřil, Jaroslav, and Patrice Ossona De Mendez. Sparsity Graphs, Structures, and Algorithms. Berlin: Springer Berlin, 2014. Print.

9. Appendices

9.1. Appendix A. C++ code for the Greedy Algorithm

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <fstream>
#include <string>
#include <random>
#include <math.h>

using namespace std;

int main() {
	int i;
	int j;
	int N;
	char frequency[5] = {0,45,105,35,-1};
	int sayi;
	int count1 = 0;
	int count2 = 0;
	int count3 = 0;
	int count4 = 0;
	fstream file1("neighborC.txt");
	fstream file2("neighborC.txt");
	int colcount = 0;

	while (!file1.eof())
	{
		file1 >> sayi;
		colcount++;
	}
	N = int(sqrt(colcount));

	int* callColor = new int[N];

	int** neigh = new int*[N];
	for (i = 0; i < N; i++)
	{
		neigh[i] = new int[N];
	}

	while (!file2.eof())
	{
		for (i = 0; i < N; i++) {
			for (j = 0; j < N; j++) {
				file2 >> neigh[i][j];
			}
		}
	}
	cout << "Algorithm: Greedy" << "\n" << "\n";
	cout << "Number of calls: " << N << "\n" << "\n";
	callColor[0] = 0;

	for (i = 1; i < N; i++)
	{
		for (j = 0; j < i; j++)
		{
			if (neigh[j][i] == 1)
			{
```

```
                        if (callColor[j] == frequency[0])
                                count1++;
                        if (callColor[j] == frequency[1])
                                count2++;
                        if (callColor[j] == frequency[2])
                                count3++;
                        if (callColor[j] == frequency[3])
                                count4++;
                }
        }

        if (count1 == 0)
                callColor[i] = frequency[0];
        else if (count2 == 0)
                callColor[i] = frequency[1];
        else if (count3 == 0)
                callColor[i] = frequency[2];
        else if (count4 == 0)
                callColor[i] = frequency[3];
        else callColor[i] = frequency[4];

        count1 = 0;
        count2 = 0;
        count3 = 0;
        count4 = 0;
    }

    int droppedCounter = 0;

    cout << "ID" << "\t" << "Color" << "\n";

    for (i = 0; i < N; i++)
    {
        cout << i << "\t" << callColor[i] << "\n";
        if (callColor[i] == -1)
                droppedCounter++;
    }
    cout << "\n";
    cout << "Number of dropped calls: " << droppedCounter << "\n";

    getchar();
    return 0;

}
```

9.2. Appendix B. NetLogo code for the Greedy Algorithm

```
globals[time droppedcalls]

breed[calls call]

calls-own[starttime finishtime duration]

to setup


  clear-all
  random-seed 20
  ask patches[set pcolor white]
  reset-ticks
  set time 0

end

to Go


if count turtles = totalNumberOfCalls
[
  ask turtles with [shape = "circle"]
  [set size 0]
  stop]

    set time ticks

    create-calls ifelse-value (ProbabilityOfCall >= random-float 1) [1] [0][
    set starttime time
    set duration random-poisson durationmean
    set finishtime starttime + duration
    setxy random-xcor random-ycor
    set shape "circle"
    set size 5
    set color black
    ]

ask calls[
  if finishtime = ticks
  [set size 0]
]
  ask calls with [color = black and starttime = ticks]
  [
  if count turtles with[color = black and finishtime > ticks] in-radius 5 >= 2
 [
     if count turtles with[color = brown and finishtime > ticks] in-radius 5 = 0
      [ set color brown ]
     if count turtles with[color = blue and finishtime > ticks] in-radius 5 = 0
      [ set color blue ]
     if count turtles with[color = yellow and finishtime > ticks] in-radius 5 = 0
      [ set color yellow ]
     ]
  ]
ask calls with [color = black and starttime = ticks]
  [
  if count turtles with[color = black and finishtime > ticks] in-radius 5 >= 2
 [
   set shape "x"
   set color red
```

```
    set size 3
    set droppedcalls droppedcalls + 1]
   ]

ask calls with [color = yellow and starttime = ticks]
   [
   if count turtles with[color = yellow and finishtime > ticks] in-radius 5 >= 2
  [set shape "x"
    set color red
    set size 3
    set droppedcalls droppedcalls + 1]
   ]

   ask calls with [color = brown and starttime = ticks]
   [
   if count turtles with[color = brown and finishtime > ticks] in-radius 5 >= 2
  [set shape "x"
    set color red
    set size 3
    set droppedcalls droppedcalls + 1]
   ]

   ask calls with [color = blue and starttime = ticks]
   [
   if count turtles with[color = blue and finishtime > ticks] in-radius 5 >= 2
  [set shape "x"
    set color red
    set size 3
    set droppedcalls droppedcalls + 1]
   ]

tick

     export-world "492.csv"


end
```

9.3. Appendix C. C++ code for the Simple LFU Algorithm

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <fstream>
#include <string>
#include <random>
#include <math.h>

using namespace std;

int main() {
       int i;
       int j;
       int N;
       char frequency[5] = { 0,45,105,35,-1 };
       int sayi;
       int count_close[4] = { 0,0,0,0 };
       int count_all[4] = { 0,0,0,0 };
       fstream file1("neighborC.txt");
       fstream file2("neighborC.txt");
       int colcount = 0;
       int minFreqIndex,minCall;

       while (!file1.eof())
       {
              file1 >> sayi;
              colcount++;
       }
       N = int(sqrt(colcount));

       int* callColor = new int[N];

       int** neigh = new int*[N];
       for (i = 0; i < N; i++)
       {
              neigh[i] = new int[N];
       }

       while (!file2.eof())
       {
              for (i = 0; i < N; i++) {
                     for (j = 0; j < N; j++) {
                            file2 >> neigh[i][j];
                     }
              }
       }
       cout << "Algorithm: Simple LFU" << "\n";
       cout << "Number of calls: " << N << "\n";
       callColor[0] = 0;

       for (i = 1; i < N; i++)
       {
              if (callColor[i-1] == frequency[0])
                     count_all[0]++;
              if (callColor[i-1] == frequency[1])
                     count_all[1]++;
              if (callColor[i-1] == frequency[2])
                     count_all[2]++;
              if (callColor[i-1] == frequency[3])
                     count_all[3]++;

              for (j = 0; j < i; j++)
              {
                     if (neigh[j][i] == 1)
                     {
                            if (callColor[j] == frequency[0])
                                   count_close[0]++;
                            if (callColor[j] == frequency[1])
                                   count_close[1]++;
                            if (callColor[j] == frequency[2])
                                   count_close[2]++;
                            if (callColor[j] == frequency[3])
                                   count_close[3]++;
                     }
              }
```

```
                minCall = 1000;

                for (int m = 0; m < 4; m++)
                {
                        if (count_close[m] == 0)
                        {
                                if (count_all[m] < minCall)
                                {
                                        minCall = count_all[m];
                                        minFreqIndex = m;
                                }
                        }
                }

                if (minCall == 1000)
                        callColor[i] = frequency[4];
                else
                        callColor[i] = frequency[minFreqIndex];

                for (int m = 0; m < 4; m++)
                        count_close[m] = 0;

        }

        int droppedCounter = 0;

        cout << "ID" << "\t" << "Color" << "\n";

        for (i = 0; i < N; i++)
        {
                cout << i << "\t" << callColor[i] << "\n";
                if (callColor[i] == -1)
                        droppedCounter++;
        }
        cout << "\n";
        cout << "Number of dropped calls: " << droppedCounter << "\n";

        getchar();
        return 0;

}
```

9.4 Appendix D. C++ code for the Random Algorithm

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <fstream>
#include <string>
#include <random>
#include <math.h>
#include<ctime>
#include<cstdlib>

//Random Algorithm

using namespace std;

int main() {
        srand(1);
        int i;
int j;
int N;
char frequency[5] = { 0,45,105,35,-1 };
int sayi;
int count1 = 0;
int count2 = 0;
int count3 = 0;
int count4 = 0;
fstream file1("neighborC.txt");
fstream file2("neighborC.txt");
int colcount = 0;

while (!file1.eof())
{
	file1 >> sayi;
	colcount++;
}
N = int(sqrt(colcount));

int* callColor = new int[N];

int** neigh = new int*[N];
for (i = 0; i < N; i++)
{
	neigh[i] = new int[N];
}

while (!file2.eof())
{
	for (i = 0; i < N; i++) {
		for (j = 0; j < N; j++) {
			file2 >> neigh[i][j];
		}
	}
}
cout << "Algorithm: Random" << "\n" << "\n";
cout << "Number of calls: " << N << "\n" << "\n";


double firstrand = rand() / double(RAND_MAX);

if (firstrand<0.25 && firstrand >= 0)
	callColor[0] = frequency[0];
if (firstrand<0.50 && firstrand >= 0.25)
	callColor[0] = frequency[1];
if (firstrand<0.75 && firstrand >= 0.50)
	callColor[0] = frequency[2];
if (firstrand<1 && firstrand >= 0.75)
	callColor[0] = frequency[3];

for (i = 1; i < N; i++)
{
	for (j = 0; j < i; j++)
	{
		if (neigh[j][i] == 1)
		{
			if (callColor[j] == frequency[0])
				count1++;
			if (callColor[j] == frequency[1])
				count2++;
			if (callColor[j] == frequency[2])
				count3++;
```

```
                if (callColor[j] == frequency[3])
                    count4++;
            }
        }

        if (count1 == 0 || count2 == 0 || count3 == 0 || count4 == 0) {


            if (i != 0)
                callColor[i] = -5;
            while (callColor[i] == -5) {
                double randK = rand() / double(RAND_MAX);
                if (randK<0.25 && randK >= 0.0 && count1 == 0)
                    callColor[i] = frequency[0];
                else if (randK >= 0.25 && randK<0.5 && count2 == 0)
                    callColor[i] = frequency[1];
                else if (randK >= 0.5 && randK<0.75 && count3 == 0)
                    callColor[i] = frequency[2];
                else if (randK >= 0.75 && randK <= 1 && count4 == 0)
                    callColor[i] = frequency[3];
            }
        }
        else callColor[i] = frequency[4];

        count1 = 0;
        count2 = 0;
        count3 = 0;
        count4 = 0;
    }

    int droppedCounter = 0;

    cout << "ID" << "\t" << "Color" << "\n";

    for (i = 0; i < N; i++)
    {
        cout << i << "\t" << callColor[i] << "\n";
        if (callColor[i] == -1)
            droppedCounter++;
    }
    cout << "\n";
    cout << "Number of dropped calls: " << droppedCounter << "\n";

    getchar();
    return 0;

}
```

9.5. Appendix E. C++ code for the Greedy Location Algorithm

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <fstream>
#include <string>
#include <random>
#include <math.h>

using namespace std;

int main() {
    int i;
int j;
int N;
char frequency[5] = { 0,45,105,35,-1 };
int sayi;
int count1 = 0;
int count2 = 0;
int count3 = 0;
int count4 = 0;
fstream file1("neighborC.txt");
fstream file2("neighborC.txt");
fstream file3("groups.txt");
int colcount = 0;

while (!file1.eof()){
	file1 >> sayi;
	colcount++;
}

N = int(sqrt(colcount));

int* callColor = new int[N];

int** neigh = new int*[N];
for (i = 0; i < N; i++)
{
	neigh[i] = new int[N];
}

while (!file2.eof())
{
	for (i = 0; i < N; i++) {
		for (j = 0; j < N; j++) {
			file2 >> neigh[i][j];
		}
	}
}

int* group = new int[N];

while (!file3.eof())
{
	for (i = 0; i < N; i++) {
			file3 >> group[i];
	}
}


cout << "Algorithm: Greedy Location" << "\n" << "\n";
cout << "Number of calls: " << N << "\n" << "\n";
callColor[0] = 0;

for (i = 1; i < N; i++)
{
	for (j = 0; j < i; j++)
	{
		if (neigh[j][i] == 1)
		{
			if (callColor[j] == frequency[0])
				count1++;
			if (callColor[j] == frequency[1])
				count2++;
			if (callColor[j] == frequency[2])
```

```
                count3++;
            if (callColor[j] == frequency[3])
                count4++;
        }
    }

    if (group[i] % 4 == 1) {
        if (count1 == 0)
            callColor[i] = frequency[0];
        else if (count2 == 0)
            callColor[i] = frequency[1];
        else if (count3 == 0)
            callColor[i] = frequency[2];
        else if (count4 == 0)
            callColor[i] = frequency[3];
        else callColor[i] = frequency[4];
    }else if (group[i] % 4 == 2) {
        if (count2 == 0)
            callColor[i] = frequency[1];
        else if (count3 == 0)
            callColor[i] = frequency[2];
        else if (count4 == 0)
            callColor[i] = frequency[3];
        else if (count1 == 0)
            callColor[i] = frequency[0];
        else callColor[i] = frequency[4];
    }else if (group[i] % 4 == 3) {
        if (count3 == 0)
            callColor[i] = frequency[2];
        else if (count4 == 0)
            callColor[i] = frequency[3];
        else if (count1 == 0)
            callColor[i] = frequency[0];
        else if (count2 == 0)
            callColor[i] = frequency[1];
        else callColor[i] = frequency[4];
    }else if (group[i] % 4 == 0) {
        if (count4 == 0)
            callColor[i] = frequency[3];
        else if (count1 == 0)
            callColor[i] = frequency[0];
        else if (count2 == 0)
            callColor[i] = frequency[1];
        else if (count3 == 0)
            callColor[i] = frequency[2];
        else callColor[i] = frequency[4];
    }


    count1 = 0;
    count2 = 0;
    count3 = 0;
    count4 = 0;
}

int droppedCounter = 0;

cout << "ID" << "\t" << "Color" << "\n";

for (i = 0; i < N; i++)
{
    cout << i << "\t" << callColor[i] << "\n";
```

```
			if (callColor[i] == -1)
				droppedCounter++;
		}
		cout << "\n";
		cout << "Number of dropped calls: " << droppedCounter << "\n";

		getchar();
		return 0;

}
```

9.6. Appendix F. C++ code for the Ring Algorithm

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <fstream>
#include <string>
#include <random>
#include <math.h>

using namespace std;

int main() {
	int i;
	int j;
	int N;
	char frequency[5] = { 0,45,105,35,-1 };
	int sayi;
	int count1 = 0;
	int count2 = 0;
	int count3 = 0;
	int count4 = 0;
	fstream file1("neighborC.txt");
	fstream file2("neighborC.txt");
	fstream file3("groups.txt");
	int colcount = 0;

	while (!file1.eof()) {
		file1 >> sayi;
		colcount++;
	}

	N = int(sqrt(colcount));

	int* callColor = new int[N];

	int** neigh = new int*[N];
	for (i = 0; i < N; i++)
	{
		neigh[i] = new int[N];
	}

	while (!file2.eof())
	{
		for (i = 0; i < N; i++) {
			for (j = 0; j < N; j++) {
				file2 >> neigh[i][j];
			}
		}
	}
```

```
int* group = new int[N];

while (!file3.eof())
{
        for (i = 0; i < N; i++) {
                file3 >> group[i];
        }
}


cout << "Algorithm: Ring" << "\n" << "\n";
cout << "Number of calls: " << N << "\n" << "\n";
callColor[0] = 0;

for (i = 1; i < N; i++)
{
        for (j = 0; j < i; j++)
        {
                if (neigh[j][i] == 1)
                {
                        if (callColor[j] == frequency[0])
                                count1++;
                        if (callColor[j] == frequency[1])
                                count2++;
                        if (callColor[j] == frequency[2])
                                count3++;
                        if (callColor[j] == frequency[3])
                                count4++;
                }
        }



        if (group[i] > 0) {
                if (count1 == 0)
                        callColor[i] = frequency[0];
                else if (count2 == 0)
                        callColor[i] = frequency[1];
                else if (count3 == 0)
                        callColor[i] = frequency[2];
                else if (count4 == 0)
                        callColor[i] = frequency[3];
                else callColor[i] = frequency[4];
        }
        else if (group[i] < 0) {
                int centerColor;
                for (int k = 0; k < i; k++) {
                        if (group[k] == -group[i]) {
                                centerColor = callColor[k];
                        }

                }

                if (centerColor == frequency[0]) {

                        if (count2 == 0)
                                callColor[i] = frequency[1];
                        else if (count3 == 0)
                                callColor[i] = frequency[2];
                        else if (count4 == 0)
                                callColor[i] = frequency[3];
                        else if (count1 == 0)
```

```
                                callColor[i] = frequency[0];
                        else callColor[i] = frequency[4];

                }
                else if (centerColor == frequency[1]) {
                        if (count3 == 0)
                                callColor[i] = frequency[2];
                        else if (count4 == 0)
                                callColor[i] = frequency[3];
                        else if (count1 == 0)
                                callColor[i] = frequency[0];
                        else if (count2 == 0)
                                callColor[i] = frequency[1];
                        else callColor[i] = frequency[4];
                }
                else if (centerColor == frequency[2]) {
                        if (count4 == 0)
                                callColor[i] = frequency[3];
                        else if (count1 == 0)
                                callColor[i] = frequency[0];
                        else if (count2 == 0)
                                callColor[i] = frequency[1];
                        else if (count3 == 0)
                                callColor[i] = frequency[2];
                        else callColor[i] = frequency[4];
                }
                else if (centerColor == frequency[3]) {
                        if (count1 == 0)
                                callColor[i] = frequency[0];
                        else if (count2 == 0)
                                callColor[i] = frequency[1];
                        else if (count3 == 0)
                                callColor[i] = frequency[2];
                        else if (count4 == 0)
                                callColor[i] = frequency[3];
                        else callColor[i] = frequency[4];
                }
                else
                {
                        if (count4 == 0)
                                callColor[i] = frequency[3];
                        else if (count1 == 0)
                                callColor[i] = frequency[0];
                        else if (count2 == 0)
                                callColor[i] = frequency[1];
                        else if (count3 == 0)
                                callColor[i] = frequency[2];
                        else callColor[i] = frequency[4];


                }
        }


        count1 = 0;
        count2 = 0;
        count3 = 0;
        count4 = 0;
}

int droppedCounter = 0;
```

```
        cout << "ID" << "\t" << "Color" << "\n";

        for (i = 0; i < N; i++)
        {
                cout << i << "\t" << callColor[i] << "\n";
                if (callColor[i] == -1)

                        droppedCounter++;
        }
        cout << "\n";
        cout << "Number of dropped calls: " << droppedCounter << "\n";

        getchar();
        return 0;

}
```

9.7. Appendix G. CMPL code for the Initial IP Model

```
%arg -solver gurobi

parameters:

Arcset[,] := readcsv("neighbor.txt");
N_calls := count(Arcset[1,]);
K := 4; //number of colors
callset := 1..N_calls;
colorset := 1..K;

variables:

e[1..N_calls]: binary;
c[1..N_calls, 1..K]: binary;

objectives:

cost: sum{m in callset: e[m]} ->min;

constraints:

constraint1 {m in callset: sum {k in colorset: c[m,k]} = 1 - e[m]; }
constraint2 {m in callset, n in m+1..N_calls, k in colorset: Arcset[m,n]*c[m,k] +
Arcset[m,n]*c[n,k] <= 1; }
```

9.8. Appendix H. CMPL code for the Improved IP Model

```
%arg -solver gurobi

parameters:

Arcset[,] := readcsv("neighbor.txt");
N_calls := count(Arcset[1,]);
```

```
K := 4; //number of colors
callset := 1..N_calls;
colorset := 1..K;
M:=1000000;

variables:

e[1..N_calls]: binary;
c[1..N_calls, 1..K]: binary;
y[1..N_calls, 1..K]: binary;

objectives:

cost: sum{m in callset: e[m]} ->min;

constraints:

constraint1 {m in callset: sum {k in colorset: c[m,k]} = 1 - e[m]; }
constraint2 {m in callset, n in m+1..N_calls, k in colorset: Arcset[m,n]*c[m,k] +
Arcset[m,n]*c[n,k] <= 1; }
constraint3 {m in callset: y[m,1]+y[m,2]+y[m,3]+y[m,4] >= 4*e[m]; }
constraint4 {m in callset: y[m,1]+y[m,2]+y[m,3]+y[m,4]-3 <= 4*e[m]; }
constraint5 {m in 1..N_calls, k in colorset: sum{a in 1..m-1: Arcset[a,m]*c[a,k]}>=y[m,k];}
constraint6 {m in 1..N_calls, k in colorset: sum{a in 1..m-1:
Arcset[a,m]*c[a,k]}<=M*y[m,k];}
```

9.9. Appendix I. Instances and Results for Outgoing Calls

Run ID	Seed	Call Probability	Duration Mean	Number of Calls	Area	IP	new IP	Greedy	LFU	Random (seed=1)	GL1	GL2
1	1	0.5	25	100	20x20	2	2	7	6	6	10	6
2	2	0.5	25	100	20x20	0	0	2	2	2	5	2
3	5	0.5	25	100	20x20	3	3	4	6	6	6	5
4	10	0.5	25	100	20x20	3	3	3	3	4	5	4
5	20	0.5	25	100	20x20	4	4	5	7	4	5	4
						2.40	2.40	4.20	4.80	4.40	6.20	4.20
							0.00	75.00	100.00	83.33	158.33	75.00
1	1	0.7	25	100	20x20	1	1	3	4	6	5	4
2	2	0.7	25	100	20x20	7	8	9	12	11	11	10
3	5	0.7	25	100	20x20	2	2	4	5	4	5	5
4	10	0.7	25	100	20x20	1	1	4	5	7	6	5
5	20	0.7	25	100	20x20	4	4	6	5	6	7	5
						3.00	3.20	5.20	6.20	6.80	6.80	5.80
							6.67	62.50	93.75	112.50	112.50	81.25
1	1	0.9	25	100	20x20	6	6	8	12	12	12	11
2	2	0.9	25	100	20x20	4	4	8	12	9	10	9
3	5	0.9	25	100	20x20	8	8	11	14	10	12	12
4	10	0.9	25	100	20x20	**8**	**9**	16	15	14	14	14
5	20	0.9	25	100	20x20	7	8	11	13	14	13	11
						6.60	7.00	10.80	13.20	11.80	12.20	11.40
							6.06	54.29	88.57	68.57	74.29	62.86
1	1	0.5	25	200	20x20	5	5	12	13	10	14	9
2	2	0.5	25	200	20x20	0	0	5	6	4	7	4
3	5	0.5	25	200	20x20	7	7	9	12	11	11	10
4	10	0.5	25	200	20x20	5	5	6	7	8	8	7
5	20	0.5	25	200	20x20	6	6	7	14	8	10	7
						4.60	4.60	7.80	10.40	8.20	10.00	7.40
							0.00	69.57	126.09	78.26	117.39	60.87
1	1	0.7	25	200	20x20	4	4	8	14	12	14	14
2	2	0.7	25	200	20x20	12	13	17	20	21	19	18
3	5	0.7	25	200	20x20	6	7	11	11	10	11	15
4	10	0.7	25	200	20x20	4	5	13	12	15	13	11
5	20	0.7	25	200	20x20	6	6	9	10	11	11	10
						6.40	7.00	11.60	13.40	13.80	13.60	13.60
							9.37	65.71	91.43	97.14	94.29	94.29
1	1	0.9	25	200	20x20	15	16	21	26	27	25	27
2	2	0.9	25	200	20x20	16	18	27	26	29	28	30
3	5	0.9	25	200	20x20	16	16	24	28	26	25	25
4	10	0.9	25	200	20x20	12	14	26	25	25	24	27
5	20	0.9	25	200	20x20	15	18	24	27	27	25	26
						14.80	16.40	24.40	26.40	26.80	25.40	27.00
							10.81	48.78	60.98	63.41	54.88	64.63
1	1	0.5	25	300	20x20	6	6	14	17	12	18	13
2	2	0.5	25	300	20x20	0	0	6	9	8	12	5
3	5	0.5	25	300	20x20	11	11	14	18	15	16	15
4	10	0.5	25	300	20x20	5	5	6	10	10	11	8
5	20	0.5	25	300	20x20	8	8	12	20	13	15	13
						6.00	6.00	10.40	14.80	11.60	14.40	10.80
							0.00	73.33	146.67	93.33	140.00	80.00
1	1	0.7	25	300	20x20	10	10	18	24	26	23	24
2	2	0.7	25	300	20x20	17	18	25	27	29	27	25
3	5	0.7	25	300	20x20	10	11	18	18	16	18	20
4	10	0.7	25	300	20x20	8	8	20	19	21	19	20
5	20	0.7	25	300	20x20	12	12	16	19	20	19	20
						11.40	11.80	19.40	21.40	22.40	21.20	21.80
							3.51	64.41	81.36	89.83	79.66	84.75
1	1	0.9	25	300	20x20	28	30	38	44	47	44	45
2	2	0.9	25	300	20x20	24	27	39	41	43	40	44
3	5	0.9	25	300	20x20	27	28	38	44	41	41	41
4	10	0.9	25	300	20x20	17	20	37	39	38	34	39
5	20	0.9	25	300	20x20	26	29	41	44	45	40	44
						24.40	26.80	38.60	42.40	42.80	39.80	42.60
							9.84	44.03	58.21	59.70	48.51	58.96
1	1	0.5	25	400	20x20	7	7	17	21	16	22	16
2	2	0.5	25	400	20x20	2	2	9	12	10	16	9
3	5	0.5	25	400	20x20	12	12	15	19	17	17	16
4	10	0.5	25	400	20x20	7	7	8	13	12	13	10
5	20	0.5	25	400	20x20	10	11	18	24	17	19	17
						7.60	7.80	13.40	17.80	14.40	17.40	13.60
							2.63	71.79	128.21	84.62	123.08	74.36
1	1	0.7	25	400	20x20	15	16	26	35	36	32	33
2	2	0.7	25	400	20x20	23	24	35	40	41	37	34
3	5	0.7	25	400	20x20	15	16	27	26	26	28	33
4	10	0.7	25	400	20x20	14	14	28	27	30	28	31
5	20	0.7	25	400	20x20	16	18	22	28	27	28	27
						16.60	17.60	27.60	31.20	32.00	30.60	31.60
							6.02	56.82	77.27	81.82	73.86	79.55
1	1	0.9	25	400	20x20	34	36	46	54	56	53	58
2	2	0.9	25	400	20x20	30	33	50	53	56	52	56
3	5	0.9	25	400	20x20	35	38	52	61	55	56	58
4	10	0.9	25	400	20x20	28	31	52	57	52	52	56
5	20	0.9	25	400	20x20	35	38	56	63	57	55	60
						32.40	35.20	51.20	57.60	55.20	53.60	57.60
							8.64	45.45	63.64	56.82	52.27	63.64

Run ID	Seed	Call Probability	Duration Mean	Number of Calls	Area	IP	new IP	Greedy	LFU	Random (seed=1)	GL1	GL2
1	1	0.5	50	100	20x20	14	14	16	20	18	17	17
2	2	0.5	50	100	20x20	15	15	21	21	21	17	19
3	5	0.5	50	100	20x20	7	7	14	13	15	13	13
4	10	0.5	50	100	20x20	6	6	10	10	9	9	8
5	20	0.5	50	100	20x20	8	8	13	15	13	16	15
						10.00	10.00	14.80	15.80	15.20	14.40	14.40
							0	48	58	52	44	44
1	1	0.7	50	100	20x20	13	14	20	20	18	21	19
2	2	0.7	50	100	20x20	16	17	26	25	24	24	23
3	5	0.7	50	100	20x20	12	14	22	23	20	23	20
4	10	0.7	50	100	20x20	19	19	26	28	28	28	26
5	20	0.7	50	100	20x20	13	14	23	22	23	24	24
						14.6	15.6	23.4	23.6	22.6	24	22.4
							6.849315068	50	51.28205128	44.87179487	53.84615385	43.58974
1	1	0.9	50	100	20x20	16	16	27	31	29	26	28
2	2	0.9	50	100	20x20	23	24	32	32	30	33	34
3	5	0.9	50	100	20x20	19	22	32	29	29	29	30
4	10	0.9	50	100	20x20	25	29	33	34	34	34	35
5	20	0.9	50	100	20x20	22	22	30	30	27	29	29
						21	22.6	30.8	31.2	29.8	30.2	31.2
							7.619047619	36.28318584	38.05309735	31.85840708	33.62831858	38.0531
1	1	0.5	50	200	20x20	25	25	35	38	37	37	37
2	2	0.5	50	200	20x20	18	20	28	31	30	27	28
3	5	0.5	50	200	20x20	24	24	36	34	37	37	32
4	10	0.5	50	200	20x20	17	18	28	28	27	27	25
5	20	0.5	50	200	20x20	17	17	27	32	27	32	32
						20.20	20.80	30.80	32.60	31.60	32.00	30.80
							2.97029703	48.07692308	56.73076923	51.92307692	53.84615385	48.07692
1	1	0.7	50	200	20x20	28		43	48	40	48	43
2	2	0.7	50	200	20x20	38		54	55	53	54	53
3	5	0.7	50	200	20x20	34		54	55	50	52	52
4	10	0.7	50	200	20x20	38		58	57	59	61	54
5	20	0.7	50	200	20x20	31		47	50	47	48	50
						33.8	-	51.2	53	49.8	52.6	50.4
							-	-	-	-	-	-
1	1	0.9	50	200	20x20	41		62	68	62	62	68
2	2	0.9	50	200	20x20	55		72	71	74	74	73
3	5	0.9	50	200	20x20	44		64	61	64	66	66
4	10	0.9	50	200	20x20	47		63	68	66	68	71
5	20	0.9	50	200	20x20	49		68	68	69	66	67
						-	-	65.8	67.2	67	67.2	69
							-	-	-	-	-	-
1	1	0.5	50	300	20x20	35	35	52	57	56	54	52
2	2	0.5	50	300	20x20	27	29	40	44	45	41	42
3	5	0.5	50	300	20x20	35	35	55	55	54	54	48
4	10	0.5	50	300	20x20	27	27	43	45	44	42	40
5	20	0.5	50	300	20x20	26	26	41	48	42	50	45
						30	30.4	46.2	49.8	48.2	48.2	45.4
							1.333333333	51.97368421	63.81578947	58.55263158	58.55263158	49.34211
1	1	0.7	50	300	20x20	49		67	77	71	76	68
2	2	0.7	50	300	20x20	54		81	82	76	77	79
3	5	0.7	50	300	20x20	52		76	80	75	76	75
4	10	0.7	50	300	20x20	59		84	84	87	86	82
5	20	0.7	50	300	20x20	54		77	81	76	79	79
						53.6	-	77	80.8	77	78.8	76.6
							-	-	-	-	-	-
1	1	0.9	50	300	20x20			96	103	99	95	106
2	2	0.9	50	300	20x20			107	111	113	114	112
3	5	0.9	50	300	20x20			100	98	99	103	101
4	10	0.9	50	300	20x20			98	102	101	105	104
5	20	0.9	50	300	20x20			103	98	101	102	98
						-	-	100.8	102.4	102.6	103.8	104.2
							-	-	-	-	-	-
1	1	0.5	50	400	20x20	47	48	70	75	73	76	72
2	2	0.5	50	400	20x20	38	40	57	64	65	58	59
3	5	0.5	50	400	20x20	50	51	74	75	71	72	65
4	10	0.5	50	400	20x20	41	42	65	65	64	62	61
5	20	0.5	50	400	20x20	39	39	59	67	57	68	61
						43.00	44.00	65.00	69.20	66.00	67.20	63.60
							2.33	47.73	57.27	50.00	52.73	44.55
1	1	0.7	50	400	20x20			101	113	104	112	102
2	2	0.7	50	400	20x20			107	112	102	104	105
3	5	0.7	50	400	20x20			108	113	105	108	106
4	10	0.7	50	400	20x20			108	111	114	111	107
5	20	0.7	50	400	20x20			102	108	100	101	101
						-	-	105.2	111.4	105	107.2	104.2
							-	-	-	-	-	-
1	1	0.9	50	400	20x20			135	140	139	133	147
2	2	0.9	50	400	20x20			140	146	148	146	145
3	5	0.9	50	400	20x20			141	138	141	145	141
4	10	0.9	50	400	20x20			133	140	137	140	137
5	20	0.9	50	400	20x20			140	140	141	142	136
						-	-	137.8	140.8	141.2	141.2	141.2
							-	-	-	-	-	-

9.10. Appendix J. Instances and Results for Paired Calls

Run ID	Seed	Call Prob	Call End Prob	# of Calls	Area	new IP	Greedy	LFU	Random (seed=1)	Run ID	Seed	Call Prob	Call End Prob	# of Calls	Area	newIP	Greedy	LFU	Random (seed=1)
1	1	0.9	0.1 (+10)	100	20x20	13	14	17	18	1	1	0.9	0.1 (+10)	200	20x20		36	37	35
2	2	0.9	0.1 (+10)	100	20x20	5	9	12	13	2	2	0.9	0.1 (+10)	200	20x20		26	26	29
3	3	0.9	0.1 (+10)	100	20x20	10	13	17	17	3	3	0.9	0.1 (+10)	200	20x20		36	39	32
4	4	0.9	0.1 (+10)	100	20x20	12	14	14	15	4	4	0.9	0.1 (+10)	200	20x20		31	36	30
5	5	0.9	0.1 (+10)	100	20x20	10	15	17	15	5	5	0.9	0.1 (+10)	200	20x20		36	38	35
						10	13	15.4	15.6							-	33	35.2	32.2
							30	54	56								-	-	-

Run ID	Seed	Call Prob	Call End Prob	# of Calls	Area	newIP	Greedy	LFU	Random (seed=1)	Run ID	Seed	Call Prob	Call End Prob	# of Calls	Area	newIP	Greedy	LFU	Random (seed=1)
1	1	0.9	0.1 (+10)	300	20x20		40	42	42	1	1	0.9	0.1 (+10)	400	20x20		69	67	66
2	2	0.9	0.1 (+10)	300	20x20		43	42	40	2	2	0.9	0.1 (+10)	400	20x20		67	64	71
3	3	0.9	0.1 (+10)	300	20x20		46	51	46	3	3	0.9	0.1 (+10)	400	20x20		62	64	59
4	4	0.9	0.1 (+10)	300	20x20		49	47	47	4	4	0.9	0.1 (+10)	400	20x20		54	58	62
5	5	0.9	0.1 (+10)	300	20x20		40	47	46	5	5	0.9	0.1 (+10)	400	20x20		61	63	60
						-	43.6	45.8	44.2							-	62.6	63.2	63.6
							-	-	-								-	-	-

Run ID	Seed	Call Prob	Call End Prob	# of Calls	Area	new IP	Greedy	LFU	Random (seed=1)	Run ID	Seed	Call Prob	Call End Prob	# of Calls	Area	newIP	Greedy	LFU	Random (seed=1)
1	1	0.7	0.1 (+10)	100	20x20	8	14	14	13	1	1	0.7	0.1 (+10)	200	20x20		26	25	28
2	2	0.7	0.1 (+10)	100	20x20	4	8	9	13	2	2	0.7	0.1 (+10)	200	20x20		22	22	21
3	3	0.7	0.1 (+10)	100	20x20	8	11	13	12	3	3	0.7	0.1 (+10)	200	20x20		27	23	26
4	4	0.7	0.1 (+10)	100	20x20	6	11	12	10	4	4	0.7	0.1 (+10)	200	20x20		23	25	23
5	5	0.7	0.1 (+10)	100	20x20	8	11	11	13	5	5	0.7	0.1 (+10)	200	20x20		25	30	29
						6.80	11.00	11.80	12.20							-	24.6	25	25.4
							61.76	73.53	79.41								-	-	-

Run ID	Seed	Call Prob	Call End Prob	# of Calls	Area	newIP	Greedy	LFU	Random (seed=1)	Run ID	Seed	Call Prob	Call End Prob	# of Calls	Area	newIP	Greedy	LFU	Random (seed=1)
1	1	0.7	0.1 (+10)	300	20x20		30	31	28	1	1	0.7	0.1 (+10)	400	20x20		57	50	51
2	2	0.7	0.1 (+10)	300	20x20		30	31	34	2	2	0.7	0.1 (+10)	400	20x20		50	48	48
3	3	0.7	0.1 (+10)	300	20x20		36	37	34	3	3	0.7	0.1 (+10)	400	20x20		43	53	51
4	4	0.7	0.1 (+10)	300	20x20		36	39	38	4	4	0.7	0.1 (+10)	400	20x20		39	42	44
5	5	0.7	0.1 (+10)	300	20x20		25	34	32	5	5	0.7	0.1 (+10)	400	20x20		39	44	47
						-	31.4	34.4	33.2							-	45.6	47.4	48.2
							-	-	-								-	-	-

Run ID	Seed	Call Prob	Call End Prob	# of Calls	Area	new IP	Greedy	LFU	Random (seed=1)	Run ID	Seed	Call Prob	Call End Prob	# of Calls	Area	newIP	Greedy	LFU	Random (seed=1)
1	1	0.7	0.3 (+10)	100	20x20	2	4	7	5	1	1	0.7	0.3 (+10)	200	20x20	8	14	15	14
2	2	0.7	0.3 (+10)	100	20x20	1	3	3	4	2	2	0.7	0.3 (+10)	200	20x20	2	10	10	9
3	3	0.7	0.3 (+10)	100	20x20	0	4	6	4	3	3	0.7	0.3 (+10)	200	20x20	6	10	12	10
4	4	0.7	0.3 (+10)	100	20x20	2	6	5	5	4	4	0.7	0.3 (+10)	200	20x20	4	13	10	11
5	5	0.7	0.3 (+10)	100	20x20	3	5	8	9	5	5	0.7	0.3 (+10)	200	20x20	4	11	15	14
						1.6	4.4	5.8	5.4							4.80	11.60	12.40	11.60
							175	262.5	237.5								141.67	158.33	141.67

Run ID	Seed	Call Prob	Call End Prob	# of Calls	Area	newIP	Greedy	LFU	Random (seed=1)	Run ID	Seed	Call Prob	Call End Prob	# of Calls	Area	newIP	Greedy	LFU	Random (seed=1)
1	1	0.7	0.3 (+10)	300	20x20	7	12	19	15	1	1	0.7	0.3 (+10)	400	20x20	11	23	30	26
2	2	0.7	0.3 (+10)	300	20x20	8	13	20	18	2	2	0.7	0.3 (+10)	400	20x20	7	20	24	16
3	3	0.7	0.3 (+10)	300	20x20	6	14	15	17	3	3	0.7	0.3 (+10)	400	20x20	11	26	25	25
4	4	0.7	0.3 (+10)	300	20x20	4	14	16	15	4	4	0.7	0.3 (+10)	400	20x20	8	17	23	20
5	5	0.7	0.3 (+10)	300	20x20	2	14	13	9	5	5	0.7	0.3 (+10)	400	20x20	8	20	25	17
						5.40	13.40	16.60	14.80							9.00	21.20	25.40	20.80
							148.15	207.41	174.07								135.56	182.22	131.11