A Comparison of Predictive Dialer Algorithms

Rob Johnston

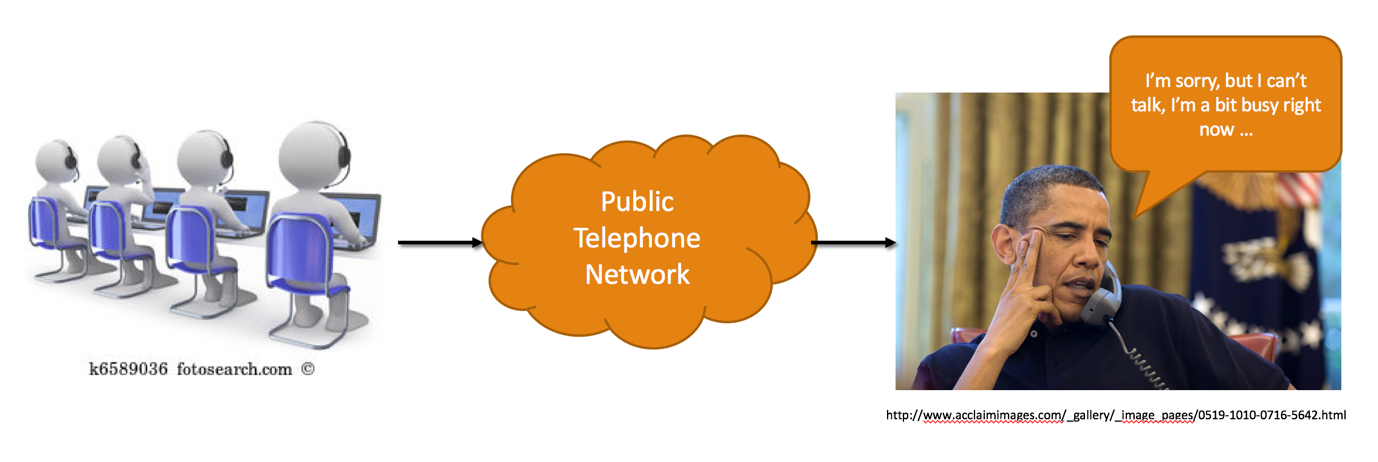
CSCI E-81  
rob@monach.com

# Abstract

Outbound call centres rely on predictive dialer algorithms to optimise the amount of time call centre agents talk to people. A number of algorithms have been proposed and we analyse four algorithms and compare their results. We take a different approach to a number of existing papers by attempting to mimic real call centre conditions. In order to achieve this, we utilise historical call data from an existing call centre.

# Introduction

The use of call centres is pervasive in today’s society. They typically perform two functions: inbound call centres take calls initiated by customers whereas outbound call centres are typically used for telemarketing or market research where the call centre attempts to contact potential customers [1] (Filho, da Cruz, Seara, Steinmann, 2007). This project is concerned with only outbound call centres.



A call centre is staffed by call centre agents. They typically use a client computer to log onto a ‘campaign’ and the call centre software will begin to initiate calls. When someone answers the call is then connected to an agent.

## Predictive Dialers

A predictive dialer is a computer application that generates telephone calls automatically. An algorithm will take into account a number of parameters such as agent availability and percentage of calls that will be answered in order to predict the number of calls that should be made. The result of the algorithms calculations is parameter known as the dial level and it represents the number of calls the predictive dialer should make per second. The algorithm will re-evaluate the dial level on a regular basis and increase or decrease it as necessary.

We can look at the predictive dialer algorithm as solving an optimisation problem and there are three constraints that the algorithm must take into account:

1. The abandonment rate. In a number of countries there is legislation that demands the number of abandoned calls is less than 5% of all answered calls.
2. The agent utilisation. This is defined as the amount of time that the agent is talking to a person, as opposed to listening to ringing tone and other non-productive time. Ideally agent utilisation would be close to 100%.
3. Trunk availability. This is the number of telephone lines that are available to the call centre to make a call. As more calls are made so are more telephone lines consumed. A number of years ago trunks were physical telephone lines rented from the telephone company. However, these days call centres tend to use voice-over-IP trunks and, depending on the size of the internet connection, trunks can almost be considered unlimited.

## Comparison of Algorithms

The purpose of this project is to compare a number of different predictive dialing algorithms. We evaluate each algorithm for ease of use and understanding, for robustness and accuracy. We have selected four algorithms to evaluate.

We begin by evaluating progressive dialing, which can be thought of a special degenerative case of predictive dialing. In progressive dialing a call is only generated whenever an agent is free. It’s not entirely predictive as there is no prediction involved – the dialer will only react to agent availability.

Next we evaluate a constant dial level algorithm. This algorithm has one parameter, the constant dial level (number calls to generate per second) and will dial that number regardless of what else is happening. This, and the progressive dialing algorithm, will be used as a baseline to compare the next two algorithms.

A number of papers in the academic literature provide analytic models to solve this problem. In our third comparison, we investigate one such model [2] (Fourati and Tabbane, 2010) and analyse the ability of a statistical model to predict a dynamic, non-deterministic system.

Finally, we implement a genetic algorithm as described in Amaral and Vital [3]. There are a number of challenges in implementing a genetic algorithm in a real-time environment and we work through some of those challenges and offer some enhancements to the original algorithm.

# Method

An intrinsic element in developing a predictive dialer algorithm is a simulator that can simulate call centre behaviour. There are two methods of simulation [1] (Filho, da Cruz, Seara, Steinmann, 2007):

1. The use of distribution curves and probabilities in order to produce ‘synthetic’ call behaviour,
2. The use of real call centre data in order to ‘replay’ real call behaviour.

We believe the second method is superior and have chosen that as our basis of simulation. Unfortunately, there is no published code for a simulator that we could reuse. Therefore, as part of this project a simulator was written and will be published for others to use in future work.

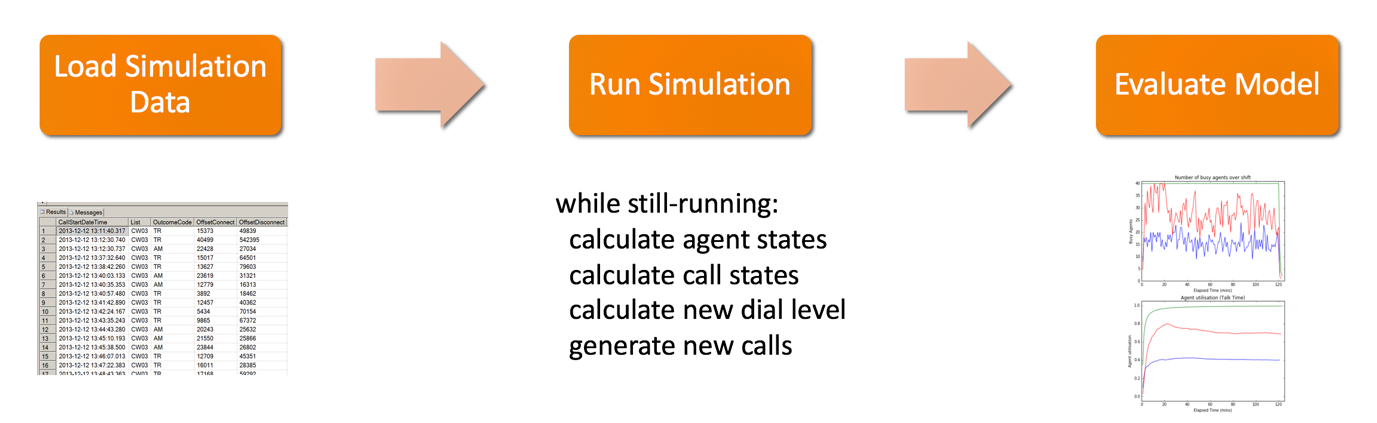


Figure 1 The process of running an experiment

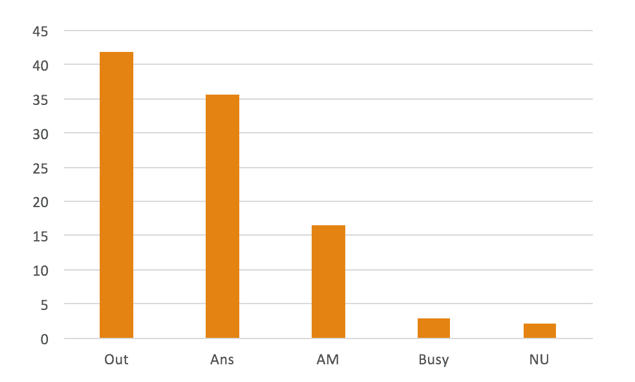
The process for running a simulation is shown, above:

1. Data that has been extracted from the call centre is loaded into the simulator,
2. An algorithm is chosen and is run through the simulator,
3. The results are then analysed.

### Simulation Data

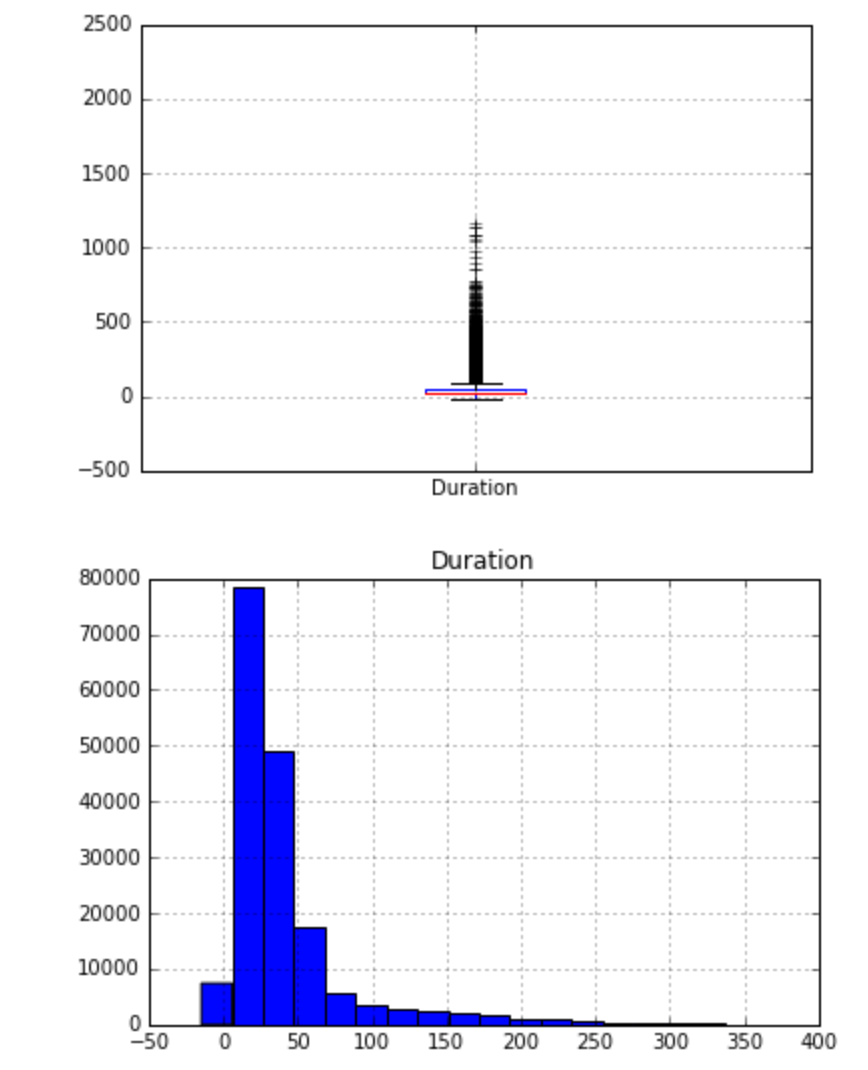
The data for the simulator is taken from the historical logs of a working call centre. We have access to historical records going back over the last ten years. These records consist of mostly telemarketing and market research campaigns. For this project we selected data for a single telemarketing campaign that spanned three months.

We analyse the data to understand the nature of the data and any patterns that may be evident.



In looking at a histogram of the call outcomes we see that only around 36% of calls are answered. The other 64% of calls either ring out, are answer machines, busies or the number is unobtainable.

Figure 2 Call Outcome Distribution



An analysis of the duration of answered calls reveals that the data is skewed and there is a very long tail. The boxplot is distorted due to the existence of a small number of outliers. Care has to be taken when evaluating these outliers as in this example a call of around 40 minutes (2480 seconds) is unusual but can occur.   
  
We plot a histogram of only the calls under 400s in order to get a better understanding of the underlying distribution. The majority of calls are under a minute in length and these are considered the ‘short’ calls. The distribution of long vs short calls is important in predicting when an agent will become available.

Figure 3 Length of answered calls

### Simulator

The associated code included with this project consists of the following:

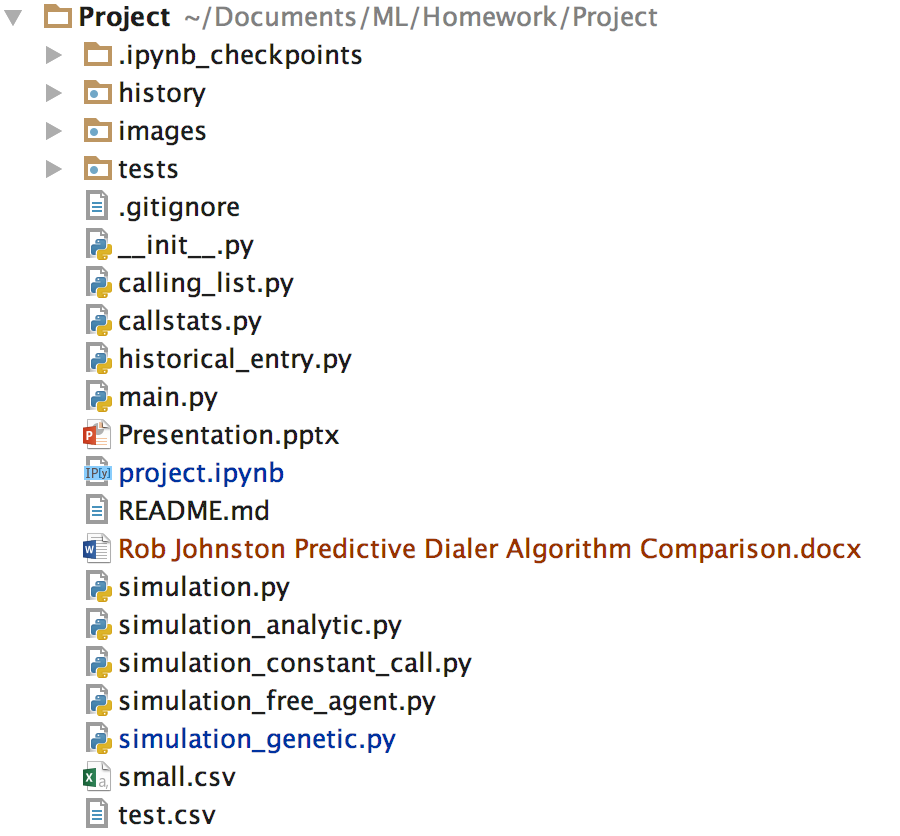


Figure 4 Code Directory

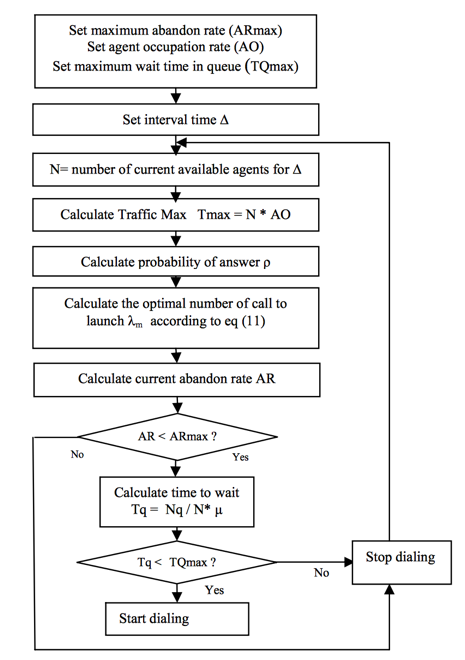
1. calling\_list.py is a data store for the historical call records,
2. callstats.py represents a single call record and is responsible for determining the outcome of the call,
3. historical\_entry.py allows us to store results in a pickle for later analysis,
4. main.py, this is where we can choose what algorithm to run,
5. simulation.py is the base class simulator,
6. simulation\_xxxx.py implement all the different algorithms.

## Baseline Algorithms - Progressive Dialing and Constant Dial Level

The progressive dialing algorithm consists of nothing more than reporting that we need a call whenever an agent becomes free. Equally simple is the constant call algorithm: we report the same value for the dial level at all times. This value for the dial level is passed into the algorithm at the start and there is no opportunity for it to change.

## Analytic Algorithm

The algorithm for the analytic model described in Fourati and Tabbane is shown below in the left column. It relies on calculating the probability of a call being answered together with the weighted average length of the call (shown on the right).



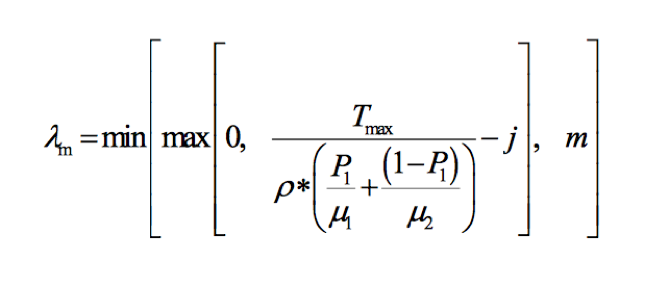


Figure Equation 11

Note that:

*Tmax =* number of avail agents (approx.)

*m* = number of available trunks,

*j* = number of available agents,

*p* = probability of a call being answered,

= weighted ave. length of a call

Figure Analytic Algorithm

## Genetic Algorithm

The genetic algorithm described in Amaral and Vital [3] works by considering three time periods of length T:

1. T is the current time period. The dialer is currently dialing live calls based on the current dial level,
2. During period T the dialer is also calculating what would have been an optimal dial level for the previous period, that is T-1,
3. Once the algorithm has calculated an ideal historical dial level it is applied to the next period of calling (T+1)

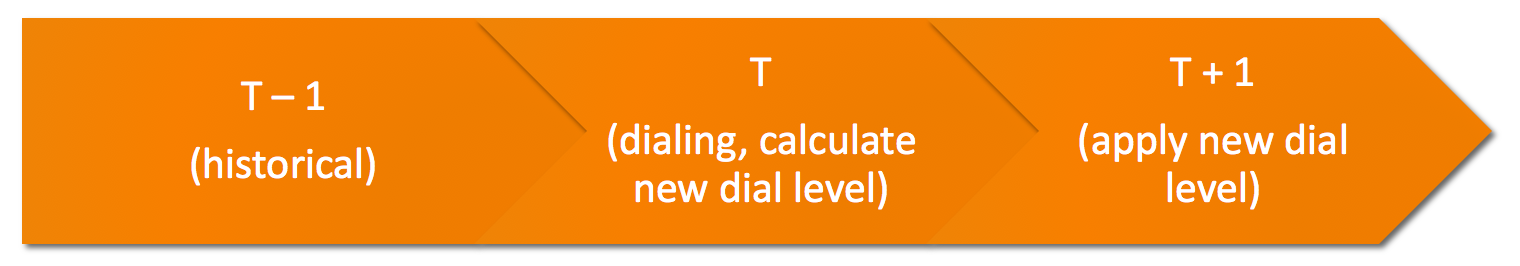
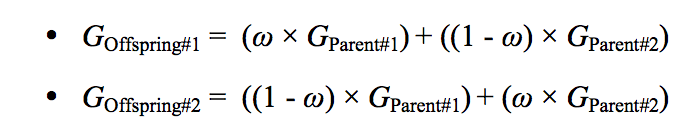


Figure Genetic Algorithm Time Periods

In our implementation we initially run the algorithm every 15 minutes with a window of 30 minutes.

Our fitness function maximises the agent utilisation. However, if the abandonment rate goes above the maximum abandonment rate then we return a negative value for the fitness function. This is to discourage bad behaviour amongst our population.

To cross two chromosomes a strategy of weighted means is used:



We choose the best 50% of the population to become the parents and then crossover random parents to ensure our population pool is replenished with children.

Finally, the mutation strategy is, depending on mutation probability, to stress the offspring by adding a random value +/- 50% of the original value.

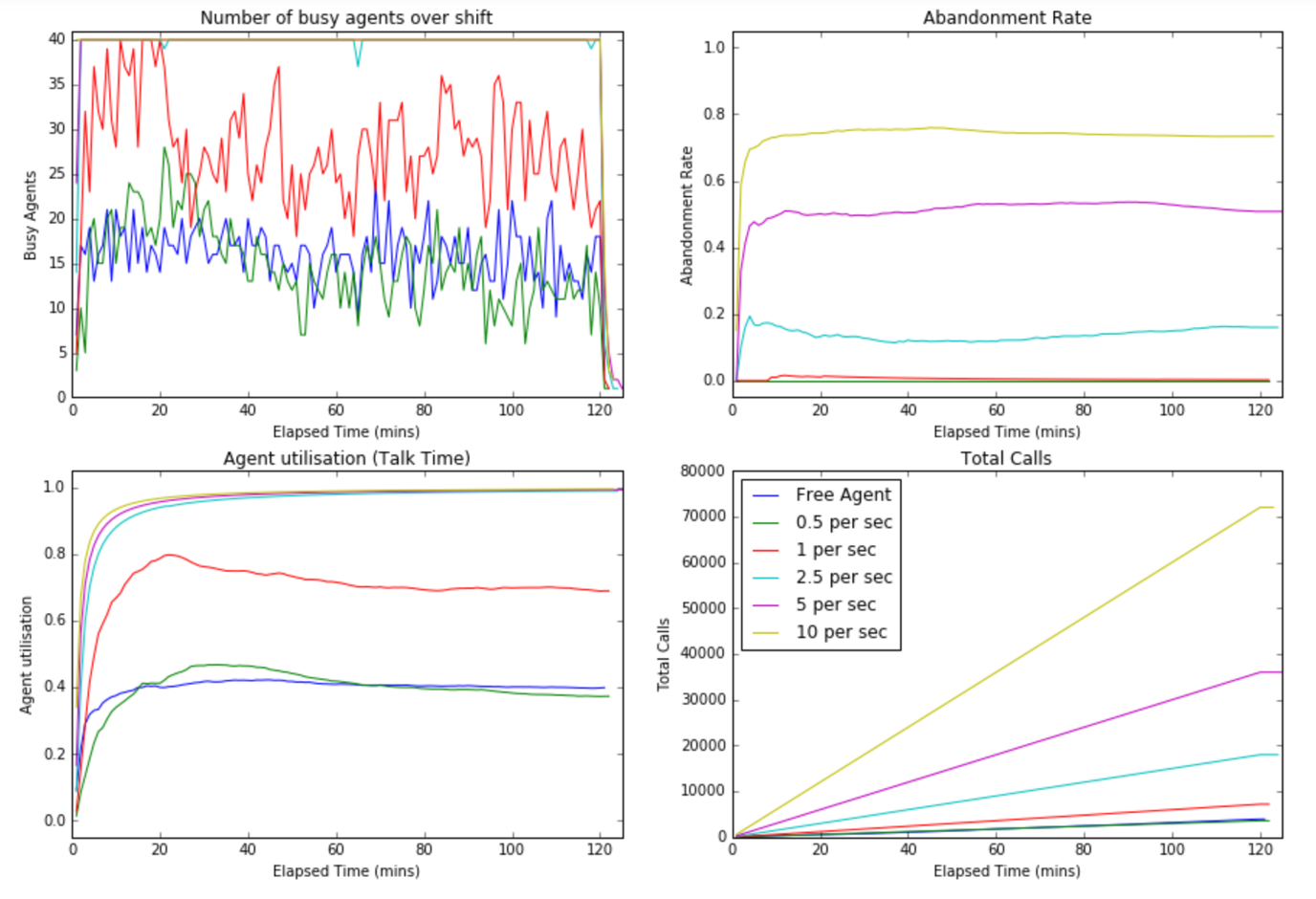
Note: the original study ran the simulation for five hours. However, this is unrealistic in a real call centre as shift’s generally only run for two-three hours before agents go for a break. We have, therefore, kept to a two-hour shift period.

# Summary Results

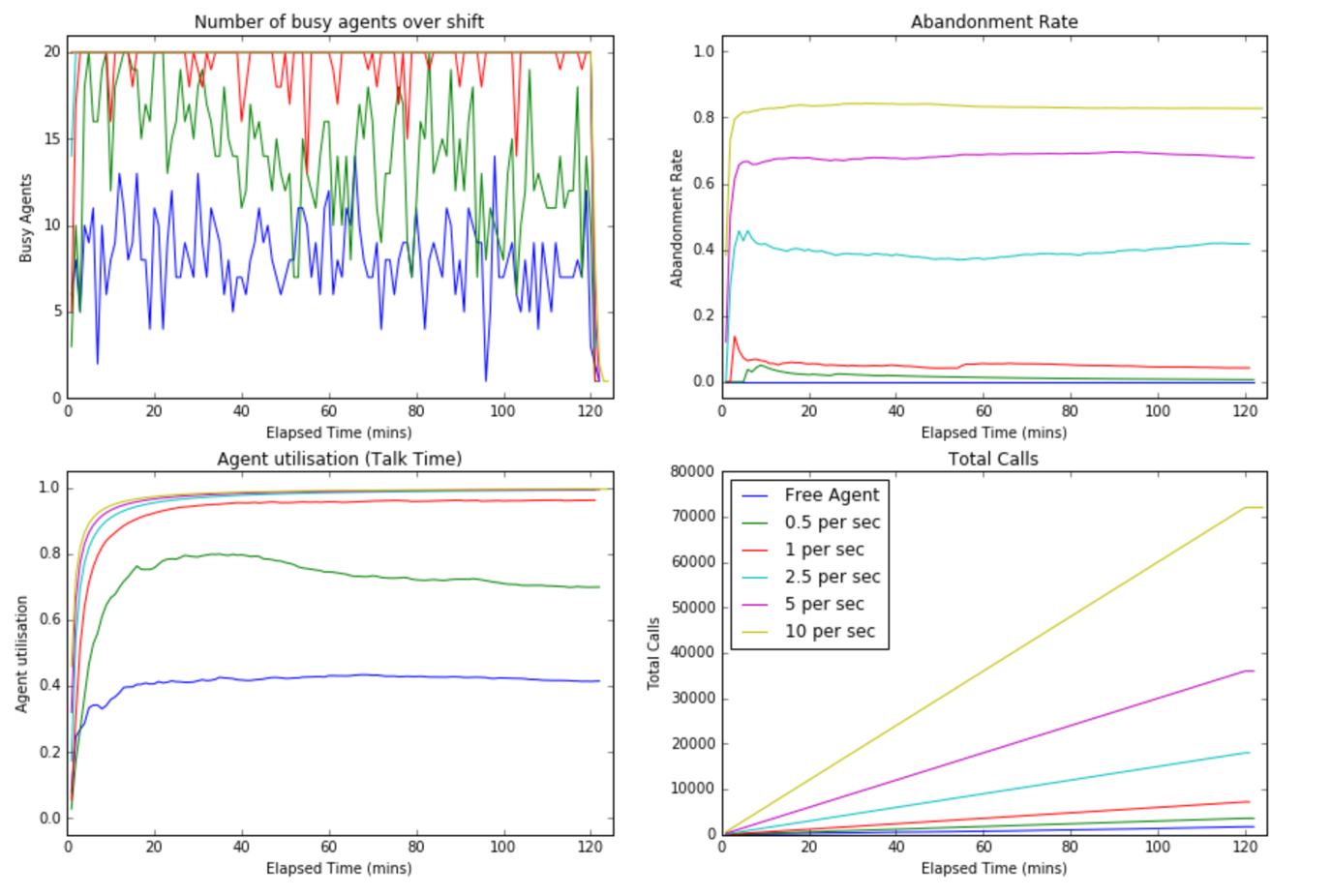
## Baseline Algorithms - Progressive Dialing and Constant Dial Level

For our baseline we ran a progressive dialing algorithm (Free Agent) and a constant dial level algorithm. We repeated the later with dial levels set to 0.5, 1, 2.5, 5 and 10 seconds.

In the first run we used 40 agents and ran for a shift of two hours. The progressive and 0.5 calls/sec did poorly on agent utilisation at only around 40% utilisation. However, neither abandoned any calls. The 1 call/sec dial level worked well in this scenario with utilisation rate of around 75% and a minimal abandonment rate (around 1%). The remaining attempts (2.5, 5, 10 calls/sec) all reached close to 100% utilisation but failed to keep the abandonment rate below 5% (18%, 52% and 73% respectively).



Using 40 agents the constant dial level of 1 call/sec produced respectable results. Therefore, we ran the same suite of tests but used 20 agents instead. When analysing these results, we see the agent utilisation is greatly increased to around 95% but the abandonment rate goes past 10% in the early part of the shift and levels out to around 5% by the end of the campaign. We also note that the results for the progressive (Free Agent) algorithm are almost identical to the 40 agent test. The number of agents does not appear to affect the utilisation or abandonment rate for that algorithm.



## Analytic Algorithm

The analytic algorithm did not produce reliable results. This could be due to a number of factors:

1. It is based on a statistical model and was intended to be run using synthetic data and not real life data,
2. If the abandonment rate is breached it stops dialing until the abandonment rate goes below the threshold. However, the only way to do this is to keep dialing conservatively to increase the number of calls, and the algorithm does not do this,
3. It maximises the number of trunks to a value of double the number of agents. The results of 90% agent utilisation are interesting as analysis of ten years of historical data indicates that a rate of between 2.5 and 3 trunks are needed per agent. It’s possible that the analytic algorithm works due to the limiting ceiling of the max trunk level,
4. The weighted average of calls is used but the units are not specified. Possibly we have misinterpreted the intention and a programming error is causing issues.

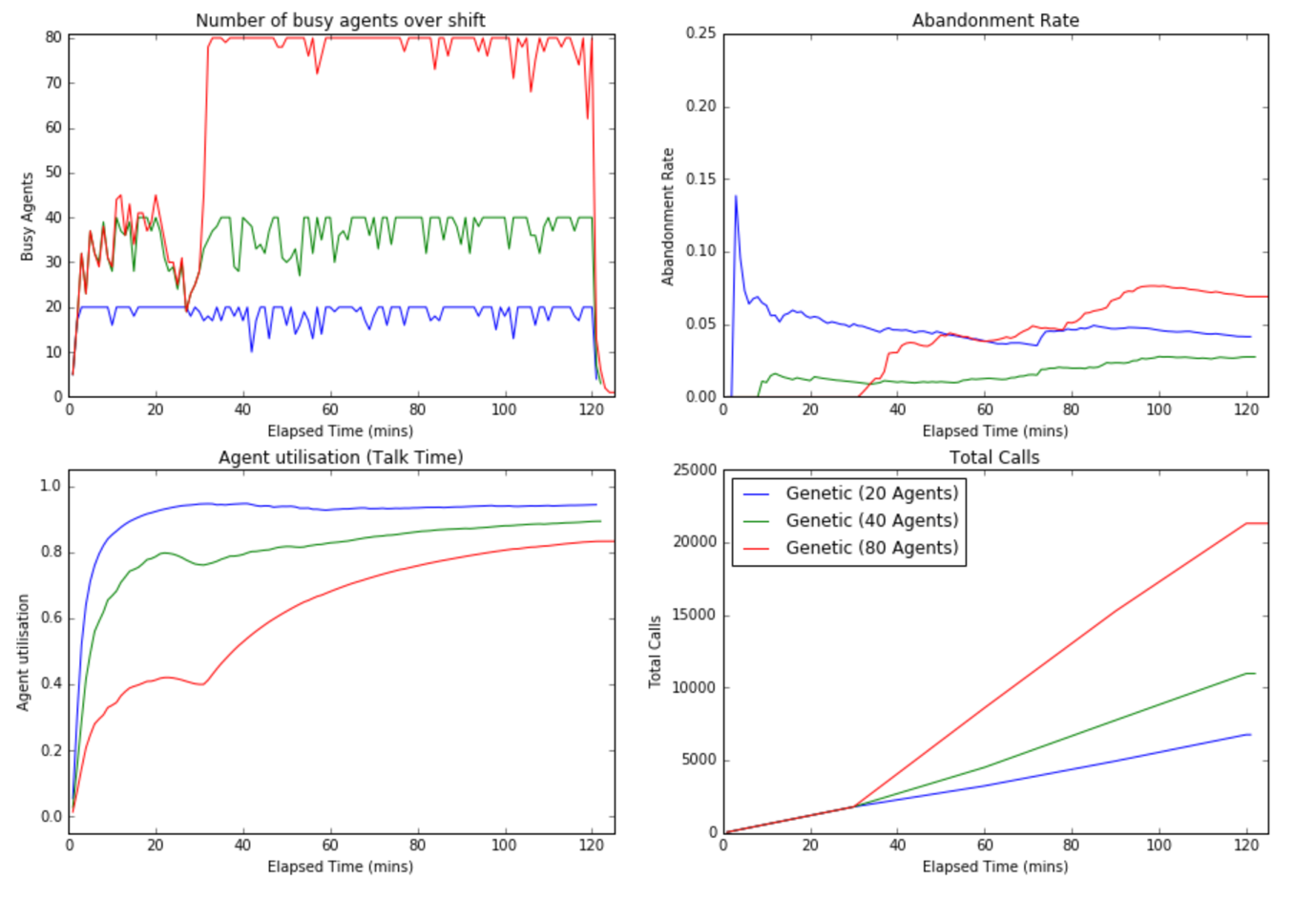
We did not pursue this algorithm as it appears to be brittle to changing conditions and our results from the genetic algorithms proved to be more rewarding.

## Genetic Algorithm

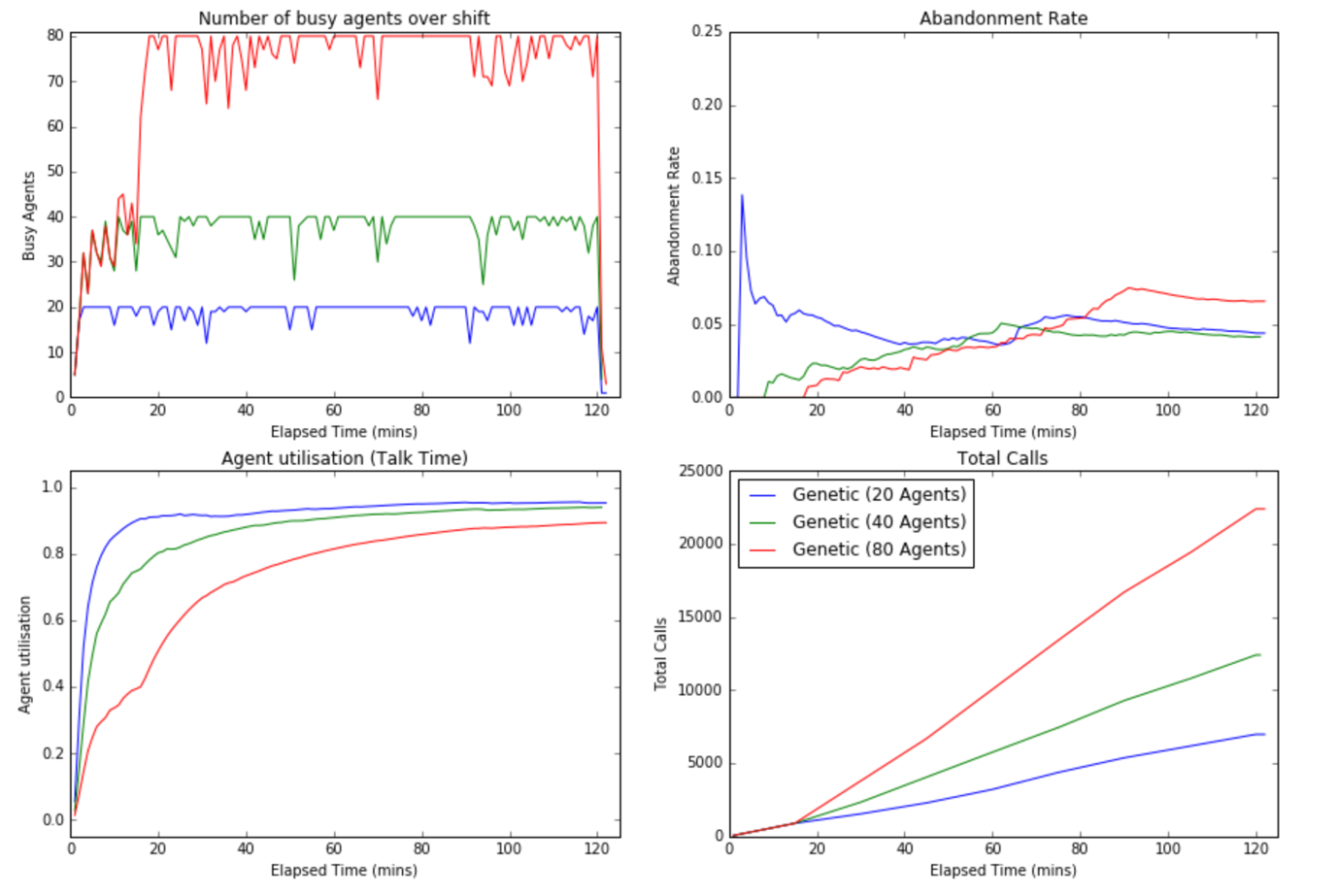
The genetic algorithm takes no parameter as is found in the constant call algorithm: instead, the intention is to let a genetic algorithm figure out the dial level for itself. To observe how it handles different conditions we run the algorithm three times: once with 20 agents and then with 40 and then 80 agents.

We observe two noticeable features in the agent utilisation:

1. in the first thirty minutes the behaviour for 40 and 80 agents is almost identical, and equally poorer than the rest of the shift. This is due to the window, T, that we have chosen. The algorithm looks at the past 30 minutes’ behaviour when calculating the new dial level. This has the effect of delaying the start of the genetic algorithm until 30 minutes into the shift in order to give it enough data to run the first time. The default behaviour at the beginning of the campaign is to run at 1 call/sec and this produces very poor results.
2. When the genetic algorithm does run it does a remarkable job of finding an appropriate dial level – the agent utilisation jumps to close to 100%. Equally impressive is that it can do this over different numbers of agents.



In an attempt to combat the 30 minute ‘warm-up’ period we modified T to be 15 minutes and analysed the results. We still have a problem in the first 15 minutes but we’ve managed to increase the agent utilisation to over 80% now.



One interesting observation is that the results for 80 agents shows the abandonment rate rising to above 5% and takes a while to head back down. This appears to be a result of a combination of our fitness function and crossover and mutation strategies. We reward a high utilisation in the fitness function which encourages high dial levels. Higher dial levels tend to come with higher abandonment rates. As we crossover our best parents there is a risk that we encourage ‘bad behaviour’ in the children. Once we have a population of delinquent dial levels we rely on the mutation strategy to help mutate a dial level to below the 5% threshold. However, we can equally mutate to a higher dial level. This all combines to generating a population that is reluctant to get below the 5% threshold.

# Final Results

The progressive dialing algorithm was the consistent across all algorithms: it’s abandonment rate was consistently the best at 0% and its agent utilisation was consistently the worst at 40%.

The constant dial level algorithm proved to be better in certain scenarios. Unfortunately, it’s simplistic approach proved to be brittle and it didn’t handle changing conditions well.

The genetic algorithm proved to be able to compensate for changing conditions. It does, however, require domain knowledge to be able to optimise the fitness, crossover and mutation strategies otherwise bad behaviour is rewarded.

# Summary & Future Work

The use of real call centre data seems to put additional ‘strain’ on call centre algorithms. However, as this is how they will ultimately be run, it is essential that algorithm design takes this into consideration. The use of genetic algorithms produced respectable results which we believe can be improved upon with further study.

The current work can benefit from the following:

1. As discussed, modifying the fitness function and crossover strategies of the genetic algorithm to ensure the abandonment rate is respected.
2. As well as using real call data, use real agent activity. This will simulate agents logging on and off at different times and will test how flexible the algorithms are to additional change.
3. Making the simulator open source will give others interested in researching predictive dialer algorithms a common baseline to benchmark their work.

The genetic algorithm has a restriction that the number of generations has to be small due to the real-time constraints of running it in a live call centre. However, future work may look at a two phase approach:

1. A number of different algorithms are used in an ensemble (including Genetic algorithms) to analyse a vast, diverse set of call data to produce optimal dial levels in many different scenarios,
2. A second phase uses the generated ‘truth’ values in a supervised learning algorithm, such as neural networks, to produce a function that can respond immediately to changing conditions within in a call centre.

# References

1. Filho P. J., da Cruz, G. F., Seara, R., Steinmann, G., 2007, *Proceedings of the 2007 Winter Simulation Conference, 2007*
2. Fourati, S., and Tabbane, S., 2010, Optimization of a Predictive Dialing Algorithm, *2010 Sixth Advanced International Conference on Telecommunications*
3. Amaral, P. M. T., Vital, M. M., 2014, Predictive Dialer Intensity Optimization Using Genetic Algorithms, *International Journal of Machine Learning and Computing, Vol. 4, No. 3, June 2014*
4. Mitchell, M., 1995, Genetic Algorithms: An Overview, *Complexity,* vol. 1, no 1, pp 31-39, 1995