Developing a Strategy for Optimal Call Center Utilization

A Major Qualifying Project submitted to the

Faculty of Worcester Polytechnic Institute

In partial fulfillment of the requirements for the

Degree in Bachelor of Science in Actuarial Mathematics

By

[illegible]

Mihnea Andrei

[illegible]

James Elmore

Report Submitted to:

Professor Jon Abraham, Advisor

Dean Arthur Heinricher, Advisor

SatMap Inc., Ittai Kan

ABSTRACT

SATMAP Inc., a global call management software company, aims to investigate a call center structure where a variety of customer inquiries can be handled by an arrangement of agents who individually are trained on various subsets of the possible inquiry types. This problem boils down to a complicated application of queuing theory and optimizations research. For this project, we devised several objective functions for call-agent matching techniques in order to maximize revenue under this framework. In order to expand their current Linear Programming approach of agent assignment, we researched Quadratic Programming and Lagrange Multipliers as possible solutions. After several mathematical and implementation issues, we returned to Linear Programming from a different perspective.

Table of Contents

Executive Summary

In this project, we are working with SATMAP, a company that develops customer service strategies for call centers. Currently, most call centers have a simple first come, first served strategy. However, SATMAP employs a wide variety of techniques to make sure that the match between each agent and each call received is as good as possible. We consider that each incoming call is concerned with a problem that can be solved using only one service provided by the call center. Meanwhile, an agent could be available to help customers with a wide variety of problems across several possible services. In this setup, the first natural question that comes to mind is how much time should each agent spend on each service in order to maximize the boost in revenue?

This project comes as a continuation of the work that James Elmore did for the company, during his summer internship, on creating a simulation out of SATMAP's strategy. However, as indicated before, the MQP is mainly focused on optimizing the utilization of agents across skills. In order to do so, our team simulates the data that a calling center might have when employing SATMAP's strategy. In turn, this data can be used to define various objective functions for optimization. Depending on the objective functions, our team investigated different optimization methods:

- Linear Programming
- Quadratic Programming
- Lagrange Multipliers

In our project, we tried to create objective functions that had a strong connection to the reality. However, in our attempt we have encountered various issues.

As we will show, the Linear Programming Method recommends assigning each agent's time entirely on the skill on which they perform the best. However, allowing utilization of each agent on a variety of skills increases our choice of agents for each call and allows for a potentially better agent to call match.

Our team also developed a work around the issue that the Linear Programming arises. When optimizing, instead of looking at all the calls that are coming in, it would be a better idea to only consider the n_b calls that best match n_b agents. This way, our optimization program will run more often and it will also be able to adapt faster to new data.

The Quadratic Programming is very difficult to implement since our matrix is indefinite. There are algorithms that can solve a convex quadratic optimization problem in polynomial time, but there are very few methods of solving an indefinite quadratic optimization problem and all of those have NP-complexity. Moreover, the Lagrange Multipliers Method gives computational errors, due to the fact that some of our data is very close to 0. When dividing by those data points, the result becomes inaccurate.

Our research indicates that a solution to the Quadratic Programming objective function exists, but further analysis for implementation is required. Furthermore, matching multiple agents to multiple calls is an effective method for revenue and choice boost.

Introduction to concepts

We consider that each call that arrives at the calling center is concerned with a problem that can be solved using only one skill. For example, in a telephone company calling service center, calls from customers with concerns about the termination of a plan or about connection problems would represent two different types of problems that can be solved using two different skills. Hence, we say that a call can only have one skill. Meanwhile, an agent could help customers with a wide variety of problems. Therefore, an agent can have multiple skills (and we say that they are logged into multiple skills). Hence, the first natural question that comes to mind is, given the strategy that SATMAP employs, how much time should each agent spend on each skill in order to maximize the boost in revenue?

But in order to answer this question, we realize that the amount of time that an agent spends on a skill depends on how good a particular agent is on the skills that he is logged into and on the value per unit of time that the skill has. Let us denote by A_{ij} the time that agent i spends on skill j. We are trying to find the optimum A_{ij}'s that would maximize the revenue boost. As mentioned before, in order to do so, we will have to introduce more variables that measure not only the marginal revenue boost that an agent brings to the company, but also the ranking of that agent on each skill. Let $\$_j = \frac{E[Revenue]}{avg\ handling\ time}$, obtaining thus, from a financial point of view, a measure of how valuable the skill is to the company. The more valuable the skill is, the more time agents should spend on that skill. Hence, the two measures are directly proportional. Also, for the ranking, let z_{ij} be the z score of agent i on skill j.

Let us first start with introducing the optimization problem.

Optimization

The better an agent performs on a particular skill, the more time they should spend on it. Hence, we realize that the objective function for the optimization should have terms that represent the three measures presented above ($z_{ij}, A_{ij}\ and\ \$_j$).

Also, SATMAP imposes 2 constraints on the A_{ij}'s:

a) Preserve each agent's total utilization across the skills that he is logged into

$$\sum_{j=1}^{n_s} A_{ij} = v_i, \qquad for\ all\ \ i = \overline{1, n_a}\,, \qquad \text{where } n_a \text{ is the total number of agents}$$

b) Preserve the total agent utilization on each skill

$$\sum_{i=1}^{n_a} A_{ij} = w_j, \qquad for\ all\ \ j = \overline{1, n_s}\,, \qquad \text{where } \mathrm{n_s} \text{ is the total number of skills}$$

We are given $0 < v_i < 1$ and $0 < w_j < 1$.

In the following sections we will present the different approaches that we attempted in order to solve this problem, the reasoning behind using them and the according background information.

Linear Programming

Introduction

A linear program has a linear objective function and linear constraints which can include both equalities and inequalities. We call a feasible point one that satisfies the constraints. The feasible set is a polytope. The standard form of a linear program is:

$$\min c^T x\,, \qquad subject\ to\ Ax = b, x \geq 0$$

In the rest of the chapter, we will assume that the matrix A has full row rank. This is equivalent with having a pivot position in every row, assumption which is reasonable for our optimization since our matrix of constraints is the result of an intricate strategy.

Definition (Basic Feasible Point): [1]*Let x be a feasible point with at most m nonzero components and let B(x) be the set of indices i for which* $x_i \neq 0$. *Then define a new matrix B that has as columns the* i^{th} *column of A, for all* $i \in B(x)$. *If the obtained matrix B is invertible, then we say that x is a basic feasible point.*

However, why are feasible points important in linear programming? The definition doesn't give us a concrete understanding of the concept, however, the following theorem will.

Theorem: All basic feasible points are vertices of the feasible polytope $\{ x \mid Ax=b, x \geq 0\}$.

Proof: Let x be a basic feasible point and, without loss of generality, assume that $B(x) = \{1,2,\ldots,m\}$. Hence, from the definition presented above, we know that $x_{m+1} = x_{m+2} = \cdots = x_n = 0$ (*) and we also have that the matrix $B = [\,A_1 \;\; A_2 \;\; A_3 \ldots A_m]$ is invertible.

Let us assume, by contradiction, that x is on a line between 2 other basic feasible points, y and z. Hence, we can write $x = \alpha y + (1-\alpha)z$, for $0 < \alpha < 1$. By looking at this equation componentwise and by using (*), we obtain that $y_i = z_i = 0$, for all $i = \overline{m+1, n}$. Let $x_B = (x_1, x_2, \ldots, x_m)$ and similarly define y_B and z_B.

Moreover, since x, y and z are all feasible points, they have to satisfy the constraints:

$$Ax = Ay = Az = b \;\Rightarrow\; \sum_{i=1}^{m} A_i x_i = \sum_{i=1}^{m} A_i y_i = \sum_{i=1}^{m} A_i z_i = b \;\Rightarrow Bx_B = By_B = Bz_B = b$$

However, we know that B is invertible and, by multiplying by its inverse to the left of the last equation, we obtain that $x_B = y_B = z_B \Rightarrow x = y = z$, a contradiction with the assumption that x was on the line between y and z.

[1] Jorge Nocedal and Stephen J. Wright

Now that we have a better understanding of what a basic feasible point is, we can give the main result in Linear Programming:

Theorem (Fundamental Theorem of Linear Programming): If the system of constraints has a solution, then at least one such solution is a basic optimal point.

Proof: Let x be a solution with the minimal number of nonzero entries (p), and let $x_i \neq 0$, for all $i = \overline{1, m}$.

From the constraints, we know that $b = \sum_{i=1}^{p} A_i x_i$. Let us assume now that $\{A_1, A_2, \dots, A_p\}$ is a set of linearly dependent vectors. Hence, we can write one of the vectors as a linear combination of the others:

$$A_p = \sum_{i=1}^{p-1} c_i A_i\,, for\ some\ real\ c_i \ \Leftrightarrow 0 = \left(\sum_{i=1}^{p-1} d_i A_i\right) - A_p$$

Let $x(\varepsilon) = x + \varepsilon(d_1, d_2, d_3, \dots, d_{p-1}, -1, 0, \dots, 0)^T = x + \varepsilon z$ With this notation we now have:

$$Ax(\varepsilon) = Ax + \varepsilon A(d_1, d_2, d_3, \dots, d_{p-1}, -1, 0, \dots, 0)^T = b + \varepsilon\left(\sum_{i=1}^{p-1} d_i A_i - A_p\right) = b, for\ any\ \varepsilon$$

Now since x is optimal, we know that $c^T(x + \varepsilon z) \geq c^T x \ \Leftrightarrow\ c^T \varepsilon z \geq 0, for\ any\ \varepsilon$. If we now choose a positive ε and a negative one, we arrive at the conclusion that $c^T z = 0$. Therefore, we obtain that $c^T x(\varepsilon) = c^T x$.

The last step of the proof is to find an $\bar{\varepsilon}$ that would contradict our initial assumption. We can find an $\bar{\varepsilon}$ such that $x_i(\bar{\varepsilon}) = 0$. In this way, we obtained another solution $x(\bar{\varepsilon})$ that has $p - 1$ nonzero entries, which, as desired, contradicts the assumption that x has the minimal number of nonzero entries.

Therefore, $\{A_1, A_2, \dots, A_p\}$ is a set of linearly independent vectors (*). Let us assume that $p > m$. This would imply that the set of vectors is linearly dependent, since there are more vectors than the dimension of the space. Therefore, we conclude that $p \leq m$.

If $p = m$, we can choose $B(x) = \{1,2, \dots, m\}$ and, from the definition, x is a basic feasible point.

If $p < m$, we recall that A has full row rank. Therefore, we know that A has a pivot position in every row, which, by using (*), implies that A has a pivot position in columns $\{1, 2, \ldots, p\}$. We conclude that for $B(x) = \{1,2, \dots, p\}$, x is a basic feasible point.

Using the 2 theorems presented above, we realize that if A has full row rank, then at least one optimal solution will be at the vertices of the polytope that represents our feasible region.

Application of concepts

The first and simplest idea for an objective function is $f(\vec{A}) = \sum_{i,j} \$_j A_{ij} z_{ij}$. Let us remember that i represents the agents, while j represents the skills. If we consider to have n_a agents in total and if we consider to have n_s skills in total, then $i = \overline{1, n_a}$ and $j = \overline{1, n_s}$. However, when implemented, the linear program assigns either 1 or 0 to the A_{ij}'s. As we have seen in the introductory section, at least one optimal point is at the vertex of our polytope. Hence, this assignment is not surprising. However, this is not an acceptable result since we would like to have as many agents in reserve on every skill as possible. But why do we need this? Let us say that we only have 2 agents working on skill 5, than, if all of the sudden many calls that require assistance related to skill 5 arrive, the 2 agents will not be able to answer as many calls as 10 agents would have been able to. This is in contradiction to one of the business objectives that SATMAP has: serving as many calls as possible and this is also why the condition $0 < A_{ij} < 1$ is important.

In a later section of the paper we will see how we can go around this issue, just by looking at the initial problem from a new perspective. Before that though, in the next section, we will see our attempts of using quadratic programming for finding optimized A_{ij}, with $0 < A_{ij} < 1$.

Quadratic Programming

Introduction

Definition (General Quadratic Program): Find $\min_x \frac{1}{2} x^T G x + x^T d$, *subject to* $a_i^T x = b_i$ *for all* $i \in E$ *and* $a_i^T x \geq b_i$ *for all* $i \in I$, *where G is symmetric.*

If G is a positive semidefinite matrix, the QP is called to be convex. Convex quadratic programs can be solved. In R there are functions that can solve any such QP. However, the problem of solving QP's for which the matrix G is not positive semidefinite is more difficult since those functions can have multiple stationary points and local minima.

As we will see in the next section, the biggest problem that we encountered using this method was trying to create an objective function that has strong roots in the real world, while trying to obtain a matrix G that is positive semi-definite. Even more, the entries in our matrix G will depend on z_{ij} and $\$_j$, which, in turn, change with time, accordingly to SATMAP's strategy.

Application of concepts

After the attempt with the linear program, we decided to turn our attention to creating a quadratic objective function. The first one that comes to mind is $f(\vec{A}) = \sum_{i,j} \$_j A_{ij}^2 z_{ij}$. However, it doesn't take too long to realize that such a function would also have the maximum for A_{ij} 0 or 1. If $z_{ij} < 0 \Rightarrow A_{ij} = 0$ and if $z_{ij} > 0 \Longrightarrow A_{ij} = 1$.

Therefore, we would need a penalty function that would keep the optimum solution of our objective function in the interval $(0,1)$. But what should this penalty be?

In the conversations we had with SATMAP, we realized the importance of freedom of choice for the agents. The more choices we have for an agent, the better the potential match between him and the call and, therefore, more revenue boost is added. Hence, a possible penalty function could be the opportunity cost of an agent i been on any other skill than the current one (i been the current skill).

$$f(A_{ij}) = \sum_{i,j} \$_j z_{ij} A_{ij} - \sum_{i,j,k} \$_j \$_k (A_{ij} - A_{ik})^2 \left(\frac{z_{ij} + z_{ik}}{2}\right)^2$$

When we write this function in the form of a quadratic program, we obtain that G has as entries:

$$on\ the\ main\ diagonal{:}\ -(2n_a - 4)\$_j^2 z_{ij} - 2\$_j^2 \sum_{k=1}^{n_s} z_{ik}\,, for\ all\ i \neq j$$

$$not\ on\ the\ main\ diagonal{:}\ 2\$_j \$_k (z_{ij} + z_{ik})$$

Unfortunately, the resulting matrix is not positive semidefinite and, therefore, as suggested at the beginning of this section the optimization problem becomes a lot more complex.

Lagrange Multipliers Method

Introduction

Assume that we are trying to find the minimum or maximum of a function $f(x_1, x_2, \dots, x_n)$ subject to $g_i(x_1, x_2, \dots, x_n) = c_i\ ,\ \ for\ i = \overline{1, m}$. The Lagrange Multipliers method consists of creating another function (usually denoted by $\mathcal{L}$) and of introducing some new variables, called Lagrangian

Multipliers (usually denoted by λ) such that $\frac{\partial \mathcal{L}}{\partial \lambda_i} = g_i(x_1, x_2, \dots, x_n) - c_i$. This is because we want the extrema of $\mathcal{L}$ to satisfy the equality constraints. Hence, our new function should look like:

$$\mathcal{L}(\vec{x}, \vec{\lambda}) = f(x_1, x_2, \dots, x_n) - \sum_{i=1}^{n} \lambda_i (g_i(x_1, x_2, \dots, x_n) - c_i)$$

As we have seen, the extrema of this new function satisfies the constraints and, moreover, the Lagrange Multipliers Method claims that $\mathcal{L}$ and f have the same extrema. But how are we going to apply the Lagrange Multipliers method to our problem? And more importantly, what function should we choose? As mentioned before, this function should emphasize the choice that an agent has when it comes to the skills that he can be used on. The more skills he can be logged on, the more choice that agent has.

Application of concepts

If we are looking to maximize the choice that an agent has when it comes to the skills that he is logged on, why not try to maximize the variance of the time that agents spend on a certain skill? However, optimizing the variance itself is difficult since this function is not even polynomial. Therefore, we decided to come up with a linearized variance function. Each variance type term is also weighted by the dollar value of the skill and by the performance of the agent on the particular skill:

$$f(\vec{A}) = \sum_{i=1}^{n_a} \sum_{j=1}^{n_s} z_{ij} \$_j^2 (A_{ij} - \mu_j)^2 , \qquad where\ \mu_j = \frac{\sum_{i=1}^{n_a} A_{ij}}{n_a} \ \ \forall\, j = \overline{1, n_s}$$

Before delving more into the calculations associated with the Lagrange Multipliers Method, let us observe that $\sum_{i=1}^{n_a} A_{ij}$ is exactly the constraint to our optimization problem and let $\sum_{i=1}^{n_a} A_{ij} = v_{n_a+j}$, $\forall\, j = \overline{1, n_s}$. We also remember that we are trying to preserve each agent's time spent across the skills that he is logged into: $\sum_{j=1}^{n_s} A_{ij} = v_i,\ \forall\ i = \overline{1, n_a}$.

With this notation, our function becomes:

$$f(\vec{A}) = \sum_{i=1}^{n_a}\sum_{j=1}^{n_s} z_{ij}\$_j^2 \left(A_{ij} - \frac{v_{n_a+j}}{n_a}\right)^2$$

Moreover, since A_{ij} represents the proportion of time that agent i spends on skill j, we realize that $0 \leq A_{ij} \leq 1$. Even more, SATMAP would like the agents to spend time on each skill that they are logged into ($0 < A_{ij}$). For now, let us try to embed the weaker condition, $0 \leq A_{ij}$, into our function. The first idea that comes to mind is to consider $A_{ij} = X_{ij}^2$. With this change of variable, our function becomes:

$$f(\vec{A}) = \sum_{i=1}^{n_a}\sum_{j=1}^{n_s} z_{ij}\$_j^2 \left(X_{ij}^2 - \frac{v_{n_a+j}}{n_a}\right)^2$$

Note also that the condition $0 \leq A_{ij}$ and the fact that $v_i < 1, \forall i = \overline{1, n_a + n_s}$ implies that $A_{ij} < 1$.

Under those conditions, our Lagrange function will be:

$$\mathcal{L}(\vec{A}, \vec{\lambda}) = \sum_{i,j} z_{ij}\$_j^2 \left(X_{ij}^2 - \frac{v_{n_a+j}}{n_a}\right)^2 + \sum_{i=1}^{n_a}\lambda_i\left(v_i - \sum_{j=1}^{n_s} X_{ij}^2\right) + \sum_{j=1}^{n_s}\lambda_{n_a+i}\left(v_{n_a+i} - \sum_{i=1}^{n_a} X_{ij}^2\right)$$

$$\frac{\partial \mathcal{L}}{\partial X_{ij}} = 4z_{ij}\$_j^2 X_{ij}\left(X_{ij}^2 - \frac{v_{n_a+j}}{n_a}\right) - 2\lambda_i X_{ij} - 2\lambda_{n_a+i}X_{ij} = 0\ , \forall i = \overline{1, n_a}\ , \forall j = \overline{1, n_s}$$

Hence, we either obtain that $X_{ij} = 0$ or $4z_{ij}\$_j^2\left(X_{ij}^2 - \frac{v_{n_a+j}}{n_a}\right) - 2\lambda_i - 2\lambda_{n_a+i} = 0$. Obviously, $X_{ij} = 0$ will not satisfy our equality constraints. Therefore, from the second equation we obtain:

$$A_{ij} = X_{ij}^2 = \frac{v_{n_a+j}}{n_a} + \frac{\lambda_i + \lambda_{n_a+j}}{2z_{ij}\$_j^2}, \forall i = \overline{1, n_a}\ , \forall j = \overline{1, n_s}$$

Now that we have found A_{ij}, we can go back and plug into the equality constraints. Fix an $= \overline{1, n_a}$:

$$v_i = \sum_{j=1}^{n_s} A_{ij} = \sum_j \left(\frac{v_{n_a+j}}{n_a} + \frac{\lambda_i + \lambda_{n_a+j}}{2z_{ij}\$_j^2} \right) = \left(\sum_j \frac{1}{2z_{ij}\$_j^2} \right) \lambda_i + \left(\sum_j \frac{1}{2z_{ij}\$_j^2} \lambda_{n_a+j} \right) + \frac{1}{n_a} \left(\sum_j v_{n_a+j} \right)$$

$$\Leftrightarrow \left(\sum_j \frac{1}{2z_{ij}\$_j^2} \right) \lambda_i + \left(\sum_j \frac{1}{2z_{ij}\$_j^2} \lambda_{n_a+j} \right) = v_i - \frac{1}{n_a} \left(\sum_j v_{n_a+j} \right) \; (*)$$

Similarly, if we fix $j = \overline{1, n_s}$ and we use the equality constraints, we obtain:

$$v_{n_a+j} = \sum_{i=1}^{n_a} \left(\frac{v_{n_a+j}}{n_a} + \frac{\lambda_i + \lambda_{n_a+j}}{2z_{ij}\$_j^2} \right) = \frac{v_{n_a+j}}{n_a} n_a + \sum_i \left(\frac{1}{2z_{ij}\$_j^2} \lambda_i \right) + \left(\sum_i \frac{1}{2z_{ij}\$_j^2} \right) \lambda_{n_a+j} \Leftrightarrow$$

$$\Leftrightarrow \sum_i \left(\frac{1}{2z_{ij}\$_j^2} \lambda_i \right) + \left(\sum_i \frac{1}{2z_{ij}\$_j^2} \right) \lambda_{n_a+j} = 0 \; (**)$$

Hence, by using equations (*) and (**), we obtain a linear system of equalities, where $\vec{\lambda}$ is the unknown. All the coefficients of this system are known and, even more, we have $n_a + n_s$ equations in $n_a + n_s$ variables. The coefficient matrix for this system will be:

$$Q = \begin{pmatrix} q_{11} & 0 & \cdots & q_{1n_a+1} & q_{1n_a+2} & \cdots & q_{1n_a+n_s} \\ 0 & q_{22} & & q_{2n_a+1} & q_{2n_a+2} & \cdots & q_{2n_a+n_s} \\ \vdots & & \ddots & & \vdots & & \\ 0 & 0 & & q_{n_an_a} & q_{n_an_a+1} & \cdots & q_{n_an_a+n_s} \\ q_{n_a+11} & q_{n_a+12} & & q_{n_a+1n_a+1} & 0 & \dots & 0 \\ q_{n_a+21} & q_{n_a+22} & \cdots & 0 & q_{n_a+2n_a+2} & \cdots & 0 \\ \vdots & \vdots & & \vdots & \vdots & & \vdots \\ q_{n_a+n_s1} & q_{n_a+n_s2} & & 0 & 0 & \dots & q_{n_a+n_sn_a+n_s} \end{pmatrix}$$

Also,

$$b = \left(v_1 - \frac{1}{n_a} \left(\sum_j v_{n_a+j} \right), \dots, v_{n_a} - \frac{1}{n_a} \left(\sum_j v_{n_a+j} \right), 0, 0, \dots, 0 \right)^T$$

Now, if we look at the augmented matrix for our system, we realize that $q_{ii}, \forall i = \overline{1, n_a}$ are the pivot positions. Hence, we will have at least n_a fixed variables. However, it is still possible for the system to have many solutions or even no solutions if Q is not invertible.

As we implemented this idea into R, the biggest problem was exactly the fact that Q was almost singular. Another problem is the fact that programs work with approximations of irrational numbers. Since some of our entries in the matrix Q are themselves sums of reciprocals, the computational error accumulates at each operation. Moreover, if agent i is rated as been almost average on skill j, then z_{ij} will be very close to 0 and, consequently the respective entry in the matrix Q will become extremely big.

Batches of agents-same problem, new perspective

Linear Programming for Batches of Agents

Introduction

As mentioned before in the Quadratic Programming approach, the more choice we have for the agents, the more revenue boost the company will have. But how do we capture this "choice"? We might be able to come up with an objective function that penalizes the lack of choice, but there is an easier way. We could answer calls only when there are, for example, *nb* agents available (or only after a predefined amount of time). In this scenario, the matching could be done among the best *nb* agents and the best *nb* calls. By having a bigger pool of agents to select from, we guarantee to have a better matching than when we would only have one agent available. As soon as we have a batch of *nb* agents available, in order to find the corresponding A_{ij}'s, we set up a new linear programming function.

Application of concepts

For the purposes of this approach, we decided to generate an objective function that would encourage good matches and put emphasis on high performing agents. In general, we classified matches made into four categories depending on the quality of the match (good and bad) and on the performance of the agent (high and low). The following table depicts which of the resulting 4 different types of matches should have the biggest impact on our objective function (with 1-higest and 4-lowest). A good match between an agent and a call is considered to be one in which their respective percentiles are the closest among all the possible pairings. Also, the measure used for computing agent's performance was their z-scores.

	High Performance	Low Performance
Good Match	1	3
Bad Match	2	4

Table 1: Crude Match Type Breakdown

Hence, we would need a function that is increasing with quality of match, and increasing in agent performance. However, the term that quantifies the agent's performance should be predominant. The most natural increasing function in the quality of the match is: $1 - |a_{ij} - c_{ij}|$, where a_{ij} and c_{ij} represent the agent and call percentiles, respectively. Moving on, another very natural increasing function in agent's performance would be the exponential. However, since we wanted to have a higher effect on our linear function from our agent's performance, we decided to use $\frac{1-e^{-a_{ij}}}{1-e^{-1}}$. This function is increasing in agent percentile and it was scaled such that when $a_{ij} = 1$, the function is 1.

The following picture depicts $f(x) = x$ and $g(x) = \frac{1-e^{-x}}{1-e^{-1}}$. We notice that $g(x) \geq f(x)$, which is exactly what we were looking for:

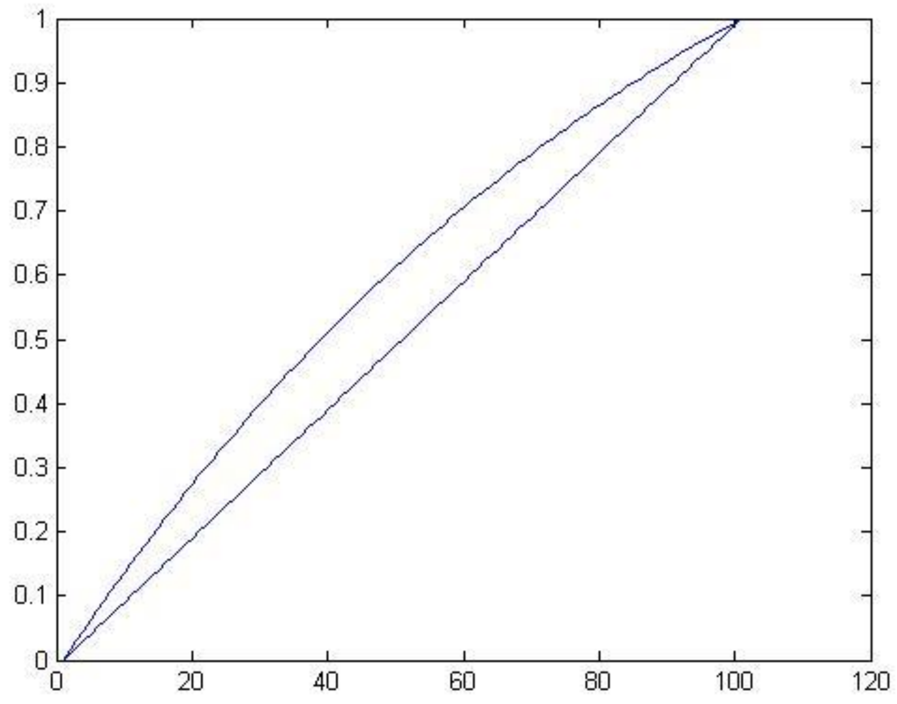

Figure 2: Comparison of Exponential Effect Versus Linear

Hence, by multiplying the function that measures the quality of the match with the one that measures an agent's performance and by adding the result over all the agents and skills available in the particular batch, we obtain the following objective linear function:

$$f(x) = \sum_{i=1}^{nb} \sum_{j=1}^{ns} (1 - |a_{ij} - c_{ij}|) \frac{1 - e^{-a_{ij}}}{1 - e^{-1}} A_{ij} \$_{ij}$$

Unlike the other methods that we investigated, an optimal value for this linear program is easily calculable and will be easily implementable for simulation. A sample code for testing the flexibility of this linear program can be found in the appendix.

Conclusions & Recommendations

Our original intentions for this project were quite ambitious. The first discussions around the scope and goals for the project centered around three major components: developing a utilization strategy, testing that strategy, and finally (assuming time allowed) developing a method of validating a given strategy. After non-disclosure agreement delays, mathematical issues, and finally computational issues, we were only capable of completing the first of these objectives.

Of the three approaches for agent utilization that we investigated, the quadratic programming approach and the batch match linear programming approach showed the most promise. While the quadratic programming objective function was highly indefinite, several mathematical properties imply existence of a solution as well as the absence of strict local minimums or maximums that would hamper convergence to a true optimal point. Because of this we believe that given enough time, an indefinite quadratic programming solver could be developed with sufficient results for simulation. The batch match approach has a fairly simplistic form and an optimal batch of matches will always be computable. While we were able to find optimal solutions for the batch match LP for some parameter investigations, implementation setbacks have prevented a full blown simulation run. The Lagrange multipliers approach showed promise but due to the use of z-scores and other small parameters, singularity and near zero issues arose. This approach showed the most difficulty and issues in implementation. Because of this, we recommend that it is not further investigated.

Next steps for the continuation of research on this project most likely should be on the implementation side. After the development of several quadratic programming objective functions, a positive-definite objective seems very difficult to derive given the varying signs and scales of the parameters available as building blocks. Because of this, a rudimentary indefinite quadratic programming solver would be of interest. An easy and natural next step would be the implementation of the batch match LP. It shows some promise as an opportunity to increase choice, even in the case where only 2-4 matches are available. In addition, its ease of implementation makes it an attractive and quick deliverable.

Appendix:

Batch-Match Linear Programming Code Sample:

```
#Inputs:
#queue - nskillsXm matrix where m is the max number of calls in the skill queues taken over skills
#groupseq - Information matching the callgroup of each of the calls in the call queue
#agentson - matrix indicating which agents are currently available to handle calls
#agentinfo - agent percentile information
#callinfo - call percentile information
#nskills - number of skills in this simulation, 3 is the default
batchmatch <- function(queue, groupseq, agentson, agentinfo, callinfo, nskills = 3)
{
 nagents = dim(agentson)[1]
 agentsonsub = agentson
 considered = rep(0,nagents)

 navail = 0
 for(b in 1:nagents)
 {
  navail = navail + max(agentson[b,])
  considered[b] = max(agentson[b,])
 }

 agentsonsub = agentson[as.logical(considered),]
 ncalls = sum(queue > 0)

 #Condition Matrix Generation:
 {

 #Boolean matrix defining all *possible* matches. Used for constraint conditions in LP.
 potenmatch = matrix(0, navail, ncalls)

 for(s in 1:nskills)
 {
  callsconc = which(queue[s,] > 0)
  agentsconc = which(agentsonsub[,s] == 1)

  #print(callsconc)
  #print(agentsconc)
  if(max(callsconc) > 0 && max(agentsconc) > 0)
  {
   potenmatch[agentsconc, callsconc]
  }
 }

 #Definition of LP constraints.
```

```
#Dimensions: A[navail + ncalls, navail*ncalls], b[navail+ncalls]
#Note: The first dimension of A defines the index of the condition.
#     The second dimension indicates a unique match between a caller and an agent.
#     The index for a match between agent i, and call j (in condition k of A):
#                         A[k, navail*j + i]

#If navail != ncalls, our conditions will vary from a symmetric case.
balance = navail - ncalls
rowadjust = matrix(0, navail, navail)
coladjust = matrix(0, ncalls, ncalls)
A = NULL
b = rep(1, navail + ncalls)

if(sum(potenmatch) == 0)
{
  cat("Error: No Matches Available")

  cat("Queue:")
  print(queue)

  cat("Agent Log Status:")
  print(agentson)
}

#When the number of agents and calls are equal the conditions are straightforward.
#The "rowadjust"ed conditions say that an agent can only match to one call
#The "coladjust"ed conditions say that a call can only match to one agent
for(p in 1:navail)
{
  rowadjust = 0*rowadjust
  rowadjust[p,p] = 1

  A = rbind(A, c(rowadjust%*%potenmatch))
}

for(p in 1:ncalls)
{
  coladjust = 0*coladjust
  coladjust[p,p] = 1

  if(balance <= 0) #agents < calls
  {
    A = rbind(A, c(potenmatch%*%coladjust))
  }else #agents > calls
  {
    A = rbind(c(potenmatch%*%coladjust), A)
  }
}
```

```
  if(balance == 0)
  {
    meq = rep("=",2*navail)
  }else
  {
    neq = min(navail,ncalls)
    meq = c(rep("=", neq), rep("<=", navail + ncalls - neq))
  }

  }

  cvec = matrix(0, navail, ncalls)

  for(s in 1:nskills)
  {
    calls = queue[s, which(queue[s,]>0)]
    agents = which(agentsonsub[,s] == 1)

    for(c in calls)
    {
      for(a in agents)
      {
        cvec[a,c] = abs(callinfo[groupseq[c],2] - agentinfo[a,s])*ifelse(abs(agentinfo[a,s]) >= 1, -
agentinfo[a,s], 1)
      }
    }
  }

  cvec = c(cvec)
  result = solveLP(cvec, b, A, const.dir = meq, lpSolve = TRUE, verbose = 1)[[3]]
  batchmatch = matrix(result, navail)
  result = matrix(0,min(ncalls,nagents),2)
  count = 1

  for(h in 1:navail)
  {
    if(max(batchmatch[h,]) == 1)
    {
      result[count,] = c(h,which(batchmatch[h,] == 1))
      count = count + 1
    }
  }

  result
}
```

Bibliography

Wright, Jorge Nocedal and Stephen J. *Numerical Optimization*. New York: Springer, 1999.