Optimal Bid Pacing for Online Ad Impression Vickrey Auction Markets

---

A Major Qualifying Project Report:

Submitted to the Faculty of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Bachelor of Science

by

Youwei Hu
Jeremy Macaluso

Advisors:

Marcel Blais

William Martin

Stephan Sturm

Sponsor:
Chitika, Inc.

This report represents the work of WPI undergraduate students submitted to the faculty as evidence of completion of a degree requirement. WPI routinely publishes these reports on its website without editorial or peer review. For more information about the projects program at WPI, please see http://www.wpi.edu/academics/ugradstudies/project-learning.html

Abstract

The purpose of this MQP is to develop a model to predict costs and construct a bidding strategy for Cidewalk, an online advertisement platform for mobile marketing developed by Chitika, Inc.. We create algorithms to minimize the total cost of a set of Vickrey auctions for ad impressions. We develop an equation to calculate the estimated total cost of winning a set of auctions and from that we are able to obtain an optimal bidding price for the next auction.

Acknowledgements

First, we would like to thank Professor Marcel Blais, Professor William Martin, and Professor Stephan Sturm for advising our project and providing us with guidance and encouragement each week.

We would also like to thank our project sponsor, Chitika, Inc, for all the support throughout the Project. In particular, Venkat Kolluri, Joseph Regan, Stijn Peeters, and Zachary Hueras were all instrumental for this project.

Authorship

Section 6, General Distributions was authored by Jeremy Macaluso with editorial help by Youwei Hu. Section 7, Implementation, and sections related to Computer Science were authored by Youwei Hu with editorial help by Jeremy Macaluso. All other sections were authored and edited by both Jeremy Macaluso and Youwei Hu.

Contents

1 Introduction

Cidewalk is a Mobile application made by Chitika, Inc. for small business owners to quickly and easily create a localized ad campaign. When a customer purchases a $1 service, they will get 1000 advertisement (ad) impressions in one city in one day. Cidewalk uses an ad market to purchase ad space to fulfill the request. On this market, space for an ad can be purchased, one at a time, by winning a Vickrey Auction.

Let's say Mike owns a pizza restaurant in Worcester, and he has ingredients to make far more pizzas than what the customers have been actually ordering. He decides it is more profitable to make a special offer to customers and sell them all rather than just throwing the perishable ingredients away. He decides to use Cidewalk to advertise the great deal locally. He enters the content of the advertisement he wants to make, and he chooses the service that 1000 ad impressions will be delivered at the cost of $1. Soon Bill, who is playing a game on his iPhone at his home in Worcester, sees an advertisement saying the pizza place half a mile away has a discount of 50% today. He thinks that this is a good deal, clicks on the advertisement, and calls the pizza place for the delivery of a pizza. Right before Bill actually sees the advertisement, there was something more going on. The game application Bill was using had space for an ad. This game uses an ad market to find the most valuable ad it can show. When Bill opens the application, the ad market sends a message to many companies like Chitika that this space is available for purchase. Any number of companies can bid on this space, and as a Vickrey auction, whoever bids the highest wins and pays the amount that the second highest bidder bids. For example, if the highest value offered by competitors of Chitika was $0.0012 and Chitika offered $0.0017, according to the rules of a Vickrey Auction, Chitika won this bid and only has to pay $0.0012. This is how Cidewalk works, but Bill is only one impression delivered out of 1000.

In this paper we present the background necessary for an understanding of online advertisement impressions and Vickrey Auctions. We develop models by assuming other competitors bid randomly uniformly, randomly binomially, and randomly multinomially. We also discuss a faster algorithm to estimate the total paid price for a set of auctions to win and from that to obtain the optimal bidding price for the next auction. We discuss comparisons of complexity and errors of different methods we develop, and at last we discuss ideas and issues regarding implementation of the method.

2 Background

This section introduces relevant background material related to advertising and Chitika.

2.1 Ad Impressions

An impression (in the context of online advertising) is a measure of the number of times an ad is seen. Clicking or not is not taken into account. Each time an ad displays it is counted as one impression. 'Impression' is a measurement of responses from a Web server to a page request from the user browser, which is filtered from robotic activity and error codes, and is recorded at a point as close as possible to opportunity to see the page by the user. Counting impressions is the method by which most Web advertising is accounted and paid for, and the cost is quoted in CPI (cost per impression). (Contrast CPC, which is the cost per click and not impression-based).

For example, if you go to the internet to look for a recipe for frittatas, and go to a web page with the recipe and see several advertisements, each of those advertisements gets one impression. It does not matter if you do not click on the advertisement, or even if it is too far down the page for you to see it, it counts as one impression.

2.2 Vickrey Auctions

A Vickrey auction is an auction where each participant silently bids a certain amount. The winner is the highest bidder, and they pay an amount equal to the second-highest bid. This is a commonly used auction style for ad impression markets for several reasons. First, it is a blind auction, so bidders only need the information about the impression and do not need information about other bidders. This makes the process much faster than relying on auctions with several rounds. Second, because the winner pays the bid of the second highest bidder, increasing or decreasing their bid does not change the amount they end up paying. It only changes whether they win or not. This makes it more difficult to game the system.[2]

⟨ Redacted by sponsor ⟩

2.3 Cidewalk

Cidewalk is a Mobile application for small business owners to quickly and easily create a localized ad campaign. It gives the customer the ability to easily create an ad that for one dollar will acquire one thousand impressions in one city in one day. After a customer submits their request, Cidewalk uses an ad market to purchase ad space to fulfill the request. On this market, space for an ad can be purchased, one at a time, by winning a Vickrey Auction. For each customer, Cidewalk needs to win one thousand of those auctions, from the ones which are accptable in the correct city, in one day.

If the grocery store from the previous example were using Cidewalk, they would have paid Chitika a dollar earlier that day. Over the course of the day, Chitika bids on many ad spaces and would win some and lose some. By the end of the day, Chitika will win at least a thousand auctions. The ad will receive a thousand impressions, many of which will lead to further interest from the users. Chitika's profit from the purchase will depend on the total amount paid for the ad spaces.

3 Mathematical & Computer Science Background

This section provides a short introduction to several important mathematical concepts which will be used in the paper.

3.1 The Central Limit Theorem

The Central Limit Theorem (CLT) states that the distribution of the scaled and normalized sum of n independent and identically distributed random variables converges to the standard normal distribution as n goes to infinity. As the sum of n independent Bernoulli random variables is binomially distributed, this can in particular be used to approximate a binomial distribution with large enough n by a normal distribution.

Let X be a binomial random variable based on n trials with success probability p. Then X has approximately a normal distribution with $\mu = np$ and $\sigma = \sqrt{npq}$. In particular, for $x \in \{0, 1, ..n\}$,

$$\begin{aligned}\mathbb{P}(X \leq x) = B(x; n, p) &\approx (\text{area under the normal curve to the left of } x + .5) \\ &= \int_{z=-\infty}^{x+.5} \frac{1}{\sqrt{npq2\pi}} e^{\frac{(z-np)^2}{2npq}} dz\end{aligned} \tag{1}$$

.

In practice, the approximation is adequate provided that both $np \geq 10$ and $nq \geq 10$. [6, section 3.3.5]

This result is useful for us because it is much faster to calculate an approximation of a normal distribution than to calculate an approximation of a binomial distribution.

3.2 Order Statistics

Denote X a real-valued variable for a set of trials, and let $x = (x_1, x_2, ..., x_n)$, the observed values of a sample of size n corresponding to this variable. The order statistic of rank k is the k^{th} smallest value in the data set, and is usually denoted $x_{(k)}$.

The distribution for the k^{th} order statistic can be calculated using the following formula [3],

$$\begin{aligned}\mathbb{P}(X_{(r)} \leq x) &= \mathbb{P}(\text{there are at most } n-k \text{ observations greater than } x \text{ out of } n \text{ trials}) \\ &= \sum_{j=0}^{n-k} \binom{n}{j} \mathbb{P}(X > x)^j \mathbb{P}(X \leq x)^{n-j}\end{aligned} \tag{2}$$

Order statistics are useful as a way to find an optimal bid.

3.3 Big-Oh notation

Big-Oh notation , also known as one of Landau's symbol, is a efficient way to describe how fast a function grows or declines.

We say that $f(n) = \mathcal{O}(g(n))$ if there exist N and $c > 0$ such that, for $n > N$

$$f(n) \leq c \cdot g(n). \quad [9] \tag{3}$$

3.4 Data Structures

Tree A tree is a hierarchical data structure made up of nodes and branches without any cycles. A tree consists root, nodes, edge, leaf.

Root A root node is the top node of a tree. A root node does not have parent node.

Node A node is a data structure which contains a value. Child nodes are one or more nodes that follow a node which is called parent node. All nodes have a parent except root node.

Edge An edge is the connection between nodes.

Leaf Leaves are the nodes without child node. [8][10]

3.5 Hash Table

A hash table is data structure to implement dictionaries efficiently. Hash table allows us to directly access an ordinary array and examine it in $\mathcal{O}(1)$ time. Therefore hash tables are a lot efficient than any other table look up structures or search trees. [8]

3.6 Numerical Methods

3.6.1 Trapezoidal Rule

Assume that the interval $[a, b]$ is divided into N intervals and denote the nodes as $[x_k, x_{k+1}]$ with step size $h = \frac{(b-a)}{N}$. Use equally spaced nodes $x_k = a + kh$, for k $= 0, 1, ..., N$. The composite trapezoidal rule for N intervals can be expressed as

$$T(f, h) = \frac{h}{2} \sum_{k=1}^{N} [f(x_{k-1}) + f(x_k)]. \tag{4}$$

This is an approximation to the integral of f over $[a, b]$, and we write

$$\int_a^b f(x)dx \approx T(f, h). \quad [11] \tag{5}$$

3.6.2 Composite Simpson Rule

Assume that $[a, b]$ is divided into $2N$ intervals and denote the nodes as $[x_k, x_{k+1}]$ with step size $h = \frac{(b-a)}{2N}$. Use equally spaced nodes $x_k = a + kh$ for $k = 0, 1, ..., 2M$. The composite Simpson rule for $2N$ intervals can be expressed as:

$$S(f, h) = \frac{h}{3} \sum_{k=1}^{N} (f(x_{2k-2}) + 4f(x_{2k-1}) + f(x_{2k})). \tag{6}$$

This is an approximation to the integral of f over $[a, b]$, and we write

$$\int_a^b f(x)dx \approx S(f, h). \quad [11] \tag{7}$$

3.6.3 Romberg Integration

We first obtain approximations by applying trapezoidal rule repeatedly with $n = 1, 2, 4, 8, 16, ...,$. And we denote the result as $R_{1,1}$, $R_{2,1}$, $R_{3,1}$, etc. Then we obtain approximations of $R_{2,2}$, $R_{3,2}$, $R_{4,2}$, etc. using extrapolation by,

$$R_{k,2} = R_{k,1} + \frac{1}{3}(R_{k,1} - R_{k-1,1}), \text{for k = 2,3,....} \tag{8}$$

Then obtain approximations $R_{3,3}$, $R_{4,3}$, $R_{5,3}$, etc., by

$$R_{k,3} = R_{k,2} + \frac{1}{15}(R_{k,2} - R_{k-1,2}), \text{for k = 3,4,....} \tag{9}$$

In general, after we obtain the appropriate $R_{k,j1}$ approximation, we obtain the approximations from

$$R_{k,j} = R_{k,j-1} + \frac{1}{4^{j-1} - 1}(R_{k,j-1} - R_{k-1,j-1}), \text{for k = j, j+1,....} \quad [12] \tag{10}$$

3.6.4 Gaussian Quadrature

Gaussian quadrature chooses the points to evaluate instead of equal spaced intervals. Gaussian Quadrature method is based on Newton-Cotes formula that for the n nodes $x_1, x_2, ..., x_n$ on the interval $[a, b]$ and coefficients $w_1, w_2, ..., w_n$ is

$$\int_{-1}^{1} f(x)dx \approx \sum_{i=1}^{n} w_i f(x_i). x \in [-1, 1] \tag{11}$$

Gaussian quadrature chooses x_i and w_i to minimize error. This choice of x_i is made by finding the roots of an nth-order Legendre polynomial $P_n(x)$. [12][23]

3.7 Programming Languages

3.7.1 Erlang

Erlang is a functional language and has concurrency at the programming language level instead of in the operating system. It is easy to conduct parallel programming. Also Erlang does not encounter problems like shared memory corruption because there is no shared memory. Finally, Erlang can be run on not only a single processor, but also multi-core processor and network processors. [13]

3.7.2 MATLAB

There are millions of engineers and scientists using MATLAB, which is a high-level language and interactive environment. Scientists and engineers use MATLAB to perform computational tasks in high level mathematics. In particular for numerical computation, MATLAB makes it possible to analyze data, create models, and develop algorithms using a range of methods. "Core math functions use processor-optimized libraries to provide fast execution of vector and matrix calculations."[18]

4 Pacing and Minimizing Costs

Since Chitika already has a business model which includes a Vickrey auction bidding price for Cidewalk, the best way to increase profit is to minimize costs. Through the course of the day, there are many auctions, and Cidewalk needs to win a certain number of those auctions. Because Cidewalk has a promise of impressions, rather than a price per click, the value of the space cannot be determined by the click rate. In some of those auctions, the competitors' bids will be higher, and in some the competitors' bids will be lower. If we win the auctions where the other bidders are bidding higher, it will cost more than if we win auctions where they bid lower.

We examine some data that has been collected. Table 4.1 depicts auction data from a 6 minute period durring which Chitika bid completely random amounts.

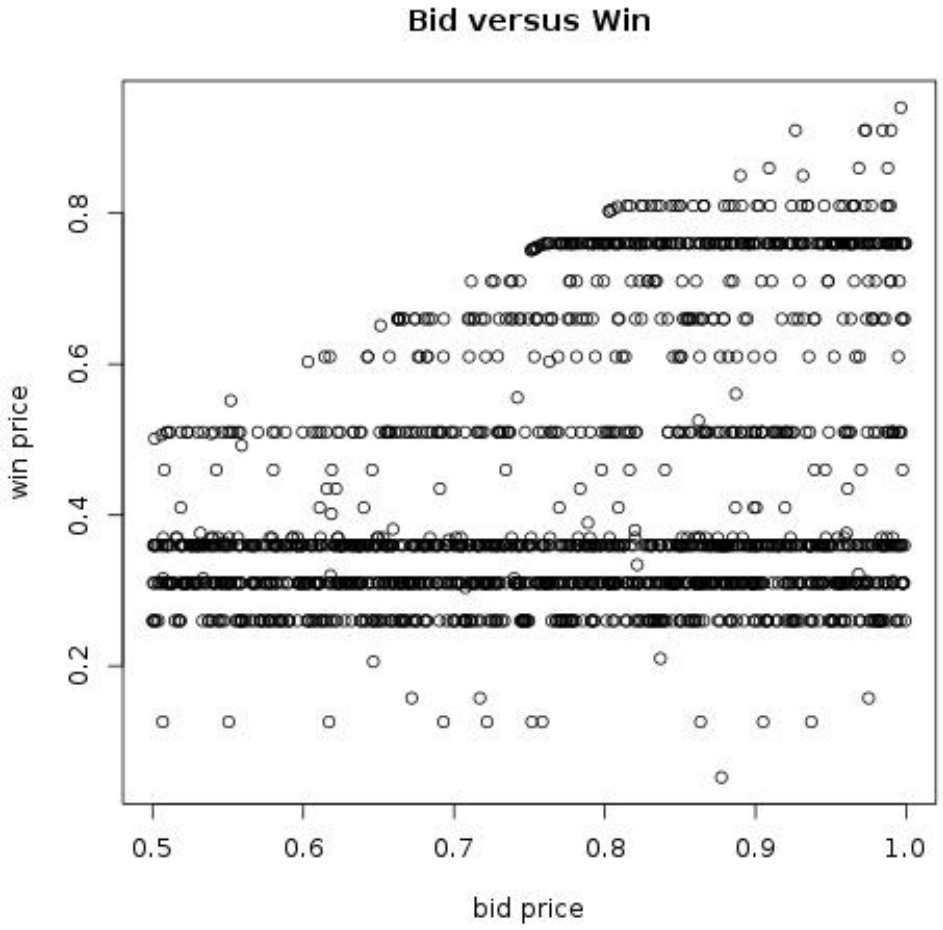

Table 4.1

The data only exists for auctions where Chitika won, and for each of those the amount which Chitika paid has been recorded. This amount is the second highest bid because the auction is a Vickrey auction. There are many data points clustered in horizontal lines in the data. This is because many companies decide whether or not to bid a fixed price, instead of deciding how much to bid. Thus such an auction participant bids either their fixed price or not at all.[5]

For the problem of minimizing costs, Chitika wants to win auctions for acceptable ad spaces over the entire day in cases where the amount they have to pay is smallest. From previous bids they can see how often they win at any given price, and how much it costs when they win. This cannot say very much about individual bidders, but it can be used to estimate the distribution of the highest other bidder as a whole. Fortunately only a distribution for the highest other bidder is needed to find an optimal bid because of the Vickrey style auction.

Let n acceptable ad requests arrive sequentially by an index i with a total requirement of r ads to show. Denote our bidding price by x_i and let c_i denote the second highest bid (our winning price). For a set of requests an optimization problem is

$$\min \sum_{i=1}^{n} \{c_i : c_i < x_i\} \tag{12}$$

$$s.t. \sum_{i=1}^{n} \{1 : c_i < x_i\} > r \tag{13}$$

.

We assume that all of the highest other bids, the c_i are identically independently distributed according to a fixed distribution. We use the notation, $\{c_i : c_i < x_i\}$ to return c_i when $c_i < x_i$, and 0 otherwise. Similarly, $\{1 : c_i < x_i\}$ is 1 when $c_i < x_i$, and is 0 otherwise. This is known as Iverson Notation.

5 Modeling for Specific Distributions of Next Highest Bidders

We begin by trying to find a general solution by simplifying the problem. We look at the solutions to several specific distributions of second highest bidders to see if those solutions lead to a solution for a general distribution.

5.1 The Uniform Distribution

We first considered a uniform distribution of second highest bidders. This simplifies several equations in predictable ways.

Assume the second highest bid is randomly uniformly distributed between 0 and 10. Let n be the number of ads we can bid on. Let r be the number of ads we have to win. Let x be the amount we are bidding. For the case $n = 2$ and $r = 1$ we have 2 ads to bid on, and we have to win 1 of them. We now have to find out how much we should bid. Because we only have to win one auction, if we win the first auction, we do not have to bid in the second auction, and have to pay the amount of the second highest bidder in the first auction. Because we must win one auction, if we lose the first auction, we must win the second auction, and pay the amount of the second highest bidder in the second auction.

Because we assume that the bids of the second highest bidder are independently identically distributed, the costs of winning the right number of auctions after we win or lose is independent of our first bid. Because we know the distribution, for any bid x, we can find the estimated cost for winning at that bid. Because the distribution of second highest bidders is independently identically distributed, the total cost, based on the first bid is

$$\min\Big(\text{(chance to win)} \times \text{(average cost to win)} + \text{(chance to lose)} \times \text{(average cost to lose)}\Big). \tag{14}$$

The probability of winning for a bid x is $\frac{x}{10}$. The cost when you win is the average cost, less than the bid, or $\frac{x}{2}$. The probability of losing is $\frac{10-x}{10}$. When you lose, you must win the second auction. Therefore you must bid 10, and the average cost when you bid 10 is 5. We

compute the estimated cost by minimizing the following quadratic,

$$\min_{x\in[0,10]}\Big(\frac{x}{10}\times\frac{x}{2}+\frac{10-x}{10}\times 5\Big) \tag{15}$$

. We can easily get an estimated cost of 3.75 at an optimal first bid of 5.

Next, we expand the equation for a more general case of r out of n bids to win. Denote the expected cost to win all r out of n auctions as $E(r,n)$.

We can never win more auctions than the number remaining, so $n \geq r$. If $n = r$, then we need to win all remaining auctions, so $E(n,n) = 5n$. If we do not need to win anything, we can bid 0, so $E(0,n) = 0$.

By (14) we get

$$\min_{x\in[0,10]}\frac{x}{10}\Big(\frac{x}{2}+E(r-1,n-1)\Big)+\frac{10-x}{10}E(r,n-1). \tag{16}$$

Therefore, because this is quadratic,

$$E(r,n)=\frac{-1}{20}\big(E(r-1,n-1)-E(r,n-1)\big)^2+E(r,n-1). \tag{17}$$

We notice that this is a recursive formula, which is computationally expensive but mathematically accurate.

5.2 The Binomial Distribution Case

We consider next the binomial case, when the highest other bids either 0 or 1, each in 50% of all cases. In this case there are only two possible bids: bidding to win everything, or bidding to only win when the cost is zero. Because it is always possible to have at least r of the remaining highest other bids be zero, we always bid to only win when the cost is zero until we must win everything else.

We calculate the estimated cost to win r out of n auctions using the formula (14) above. It is useful to arrange these values in a triangle, in the same shape as Pascal's triangle. Denote the estimated cost to win r out of n auctions $E(r, n)$. Arrange the triangle of numbers with $E(r, n)$ in the n^{th} row, r^{th} from the right.

			0				/1
		1		0			/2
	4		1		0		/4
12		5		1		0	/8
32	17		6		1	0	/16
80	49	23		7	1	0	/32

Subtract each value from the value to the right of it, effectively calculating

$$E(r, n) - E(r - 1, n). \tag{18}$$

		1			/2
	3		1		/4
7		4		1	/8
15	11		5	1	/16
31	26	16		6	1	/32

Subtract each value from the value to the right of it again, effectively calculating

$$\big(E(r, n) - E(r - 1, n)\big) - \big(E(r - 1, n) - E(r - 2, n)\big). \tag{19}$$

	2			/4
3		3		/8
4	6		4	/16
5	10	10	5	/32

If this pattern continues, then we can build the following equation:

$$\begin{aligned}
\big(E(r,n) - E(r-1,n)\big) - \big(E(r-1,n) - E(r-2,n)\big) &= \binom{n}{r-1}/2^n \\
\big(E(r,n) - 2E(r-1,n) + E(r-2,n)\big) &= \binom{n}{r-1}/2^n \\
E(r,n) &= \binom{n}{r-1}/2^n + 2E(r-1,n) - E(r-2,n).
\end{aligned} \tag{20}$$

By using substitution,

$$\begin{aligned}
E(r,n) =& \binom{n}{r-1}/2^n + 2E(r-1,n) - E(r-2,n) \\
=& \binom{n}{r-1}/2^n + \left(2\binom{n}{r-2}/2^n + 2E(r-2,n) - E(r-3,n)\right) - E(r-2,n) \\
=& \binom{n}{r-1}/2^n + 2\binom{n}{r-2}/2^n + 3E(r-2,n) - 2E(r-3,n) \\
& \ldots \\
=& \sum_{i=1}^{r-1}\left[(r-i)\binom{n}{i}/2^n\right] + (r)E(1,n) - (r-1)E(0,n).
\end{aligned} \tag{21}$$

We know that $E(0,n)$ is zero for all n. We also know that $E(1,n) = \frac{1}{2^n}$ for all n, because if any of them are zero, then from our bidding strategy we pay zero, and there is a $\frac{1}{2^n}$ chance of them all being 1. Therefore, $r \times E(1,n) = \frac{r}{2^n} = \frac{(r-0)\binom{n}{0}}{2^n}$. We know that $(r-i)\binom{n}{i}/2^n$ is zero when $i = r$ Therefore, if our earlier assumption that the pattern would hold is true,

$$E(r,n) = \sum_{i=0}^{r}(r-i)\binom{n}{i}/2^n. \tag{22}$$

We now want to show that this holds true. First we generalize to an arbitrary probability that the second highest bidder bids 1, denoted by p. Rather than $p = \frac{1}{2}$, the highest other bidder has a p chance to bid 1, and a $1-p$ chance to bid 0. Based on similarities with the binomial choice probability, we assume the following equation.

$$E(r,n) = \sum_{i=0}^{r}(r-i)\binom{n}{i}p^{n-i}(1-p)^i. \tag{23}$$

We now want to recursively show that this correctly estimates the total cost based on the bidding strategy we know to be correct. Fitting the formula at $r = 0$ and $r = n$ is trivial. At $0 < r < n$, we want to recursively show, based on (14) that

$$E(r,n) = p \times E(r,n-1) + (1-p) \times E(r-1,n-1) \tag{24}$$

$$\begin{aligned}
&= p \times E(r,n-1) + (1-p) * E(r-1,n-1) \\
&= p \times \sum_{i=0}^{r}(r-i)\binom{n-1}{i}p^{n-i-1}(1-p)^i + (1-p)\sum_{i=0}^{r-1}(r-i-1)\binom{n-1}{i}p^{n-i-1}(1-p)^i \\
&= p \times \sum_{i=0}^{r}(r-i)\binom{n-1}{i}p^{n-i-1}(1-p)^i + (1-p)\sum_{i=1}^{r}(r-i)\binom{n-1}{i-1}p^{n-i}(1-p)^{i-1} \\
&= \sum_{i=0}^{r}(r-i)\binom{n-1}{i}p^{n-i}(1-p)^i + \sum_{i=1}^{r}(r-i)\binom{n-1}{i-1}p^{n-i}(1-p)^i \\
&= r\binom{n-1}{0}p^n(1-p)^0 + \sum_{i=1}^{r}(r-i)(\binom{n-1}{i-1} + \binom{n-1}{i})p^{n-i}(1-p)^i \\
&= r \times 1 \times p^n + \sum_{i=1}^{r}(r-i)\binom{n}{i}p^{n-i}(1-p)^i \\
&= \sum_{i=0}^{r}(r-i)\binom{n}{i}p^{n-i}(1-p)^i \\
&= E(r,n)
\end{aligned} \tag{25}$$

Thus the equation correctly predicts the estimated total cost to win r out of n auctions.

5.3 Multinomial Distributions

In the binomial case, $r - i$ is the minimum cost when there are i auctions where the highest other bidder bids less than zero. $\binom{n}{i}p^{n-i}(1-p)^i$ is the probability that there are i auctions where the highest other bidder bids zero. We can calculate those values for multinomial distributions similarly. However, when the probability is not binomial, the minimum cost is different from the expected cost.

In the trinomial case, there is a p_1 chance that the cost is α_1, a p_2 chance that the cost is α_2, and a p_3 chance that the cost is α_3. Without loss of generality, assume $\alpha_1 < \alpha_2 < \alpha_3$. The cost when there are i auctions where the next highest bidder bids α_1 is $i \times \alpha_i$ plus the cost needing to win $r - i$ of the remaining $n - i$ auctions, for none of which the next highest bid is α_1. That cost can be determined the same way, by summing all the probabilities and costs of having j auctions with the next highest bid α_2.

The probabilities are simple binomial probabilities, though the probability that $i = r$ is the binomial probability that $i \geq r$. Therefore the expected cost is

$$\sum_{i=0}^{r-1}\binom{n}{i}p_1^i(1-p_1)^{n-i}\Big[\alpha_1 i + \sum_{j=0}^{r-i-1}\binom{n-i}{j}\Big(\frac{p_2}{1-p_1}\Big)^j\Big(1-\frac{p_2}{1-p_1}\Big)^{n-i-j}(\alpha_2 j + \alpha_3(r-i-j))$$
$$+ \sum_{j=r-i}^{n-i}\binom{n-i}{j}\Big(\frac{p_2}{1-p_1}\Big)^j\Big(1-\frac{p_2}{1-p_1}\Big)^{n-i-j}(\alpha_2(r-i))\Big]$$
$$+ \sum_{i=r}^{n}\binom{n}{i}(p_1^i(1-p_1)^{n-i})(\alpha_1 r). \tag{26}$$

We were also able to show that for arbitrary p_1, p_2, p_3, r, n

$$\begin{aligned}
&\sum_{i=0}^{r}\sum_{j=0}^{r-i}\frac{n!}{i!j!(n-i-j)!}(p_1)^i(p_2)^j(p_3)^{n-i-j}\\
=&\sum_{k=0}^{r}\sum_{l=0}^{k}\frac{n!}{l!(k-l)!(n-k)!}(p_1)^l(p_2)^{k-l}(p_3)^{n-k}\\
=&\sum_{k=0}^{r}\sum_{l=0}^{k}\frac{n!k!}{l!k!(k-l)!(n-k)!}(p_1)^l(p_2)^{k-l}(p_3)^{n-k}\\
=&\sum_{k=0}^{r}\frac{n!}{k!(n-k)!}(p_3)^{n-k}\sum_{l=0}^{k}\frac{k!}{l!(k-l)!}(p_1)^l(p_2)^{k-l}\\
=&\sum_{k=0}^{r}\frac{n!}{k!(n-k)!}(p_3)^{n-k}(p_1+p_2)^k.
\end{aligned} \tag{27}$$

Using equation (27) to simplify expression (26), we get that the expected cost is

$$\begin{aligned}
&\alpha_1 np_1\sum_{i=0}^{r-1}\frac{(n-1)!}{(i-1)!(n-i)!}p_1^{i-1}(1-p_1)^{n-i}\\
+&\alpha_2 np_2\sum_{i=0}^{r-1}\sum_{j=1}^{r-i-1}\frac{(n-1)!}{i!(j-1)!(n-i-j)!}(p_1)^i(p_2)^{j-1}(1-p_1-p_2)^{n-i-j}\\
+&\alpha_3 n(1-p_1-p_2)\sum_{i=0}^{r-1}\sum_{j=0}^{r-i-1}\frac{(n-1)!}{i!j!(n-i-j-1)!}(p_1)^i(p_2)^j(1-p_1-p_2)^{n-i-j-1}\\
+&\alpha_3(r-n)\sum_{i=0}^{r-1}\sum_{j=0}^{r-i-1}\frac{n!}{i!j!(n-i-j)!}(p_1)^i(p_2)^j(1-p_1-p_2)^{n-i-j}\\
+&\alpha_2 r\sum_{i=0}^{r-1}\sum_{j=r-i}^{n-i}\frac{n!}{i!j!(n-i-j)!}(p_1)^i(p_2)^j(1-p_1-p_2)^{n-i-j}\\
-&\alpha_2 np_1\sum_{i=1}^{r-1}\sum_{j=r-i}^{n-i}\frac{(n-1)!}{(i-1)!j!(n-i-j)!}(p_1)^{i-1}(p_2)^j(1-p_1-p_2)^{n-i-j}\\
+&\alpha_1 r\sum_{i=r}^{n}\frac{n!}{i!(n-i)!}p_1^i(1-p_1)^{n-i}.
\end{aligned} \tag{28}$$

Continuing on this path did not lead to any simpler or easier to calculate results, so we moved on to other approaches.

6 General Distributions

We consider using a general distribution, $P(x)$ as the distribution of highest other bidders.

6.1 Optimal Bids

We show that that for all probability distributions, the bid, $X(r,n)$ that minimizes $E(r,n)$ equals $E(r,n-1)-E(r-1,n-1)$. Let $C(n)$ be the highest bid which is not ours when n auctions remain. Consider the costs, knowing $C(n)$. If we lose the first auction the cost is $E(r,n-1)$. If we win, the cost is $E(r-1,n-1)+C(n)$. If $C(n) < E(r,n-1)-E(r-1,n-1)$, then winning costs less than losing. If $C(n) > E(r,n-1)-E(r-1,n-1)$, then winning costs more than losing. If $C(n) = E(r,n-1)-E(r-1,n-1)$, then the costs are the same. A bid of $E(r,n-1)-E(r-1,n-1)$ means that we will always win when it is less expensive to win, and always lose when it is less expensive to lose. This is ideal because bidding at this value always leads to the result which causes us to bid less. All we need to do is find it.

6.2 Estimated Costs and the Recursive Algorithm

Because we can find the optimal bid based on the estimated costs of future auctions, and we can find the estimated costs based on the optimal bid, we can create a recursive algorithm to determine the optimal bid. Let $P(x)$ be the probability that the highest other bidder bids less than x, and let $\mu_{C|C<x}$ be the mean bid for the highest other bidder when he bids less than x. We have

$$X(r,n) = \begin{cases} 0 & r = 0 \\ \infty & r = n \\ E(r,n-1) - E(r-1,n-1) & else \end{cases}.$$

$$E(r,n) = min\big(\text{ chance to win} \times \text{average cost to win} + \text{chance to lose} \times \text{average cost to lose}\big)$$

$$E(r,n) = P(X(r,n)) \times [\mu_{C|C<X(r,n)} + E(r-1,n-1)] + \big(1 - P(X(r,n))\big) \times E(r,n-1), \tag{29}$$

therefore, given $P(x)$, we can recursively find the estimated costs for all r and n, and can also find the optimal bid for all r and n.

6.3 The Faster Algorithm

When we are calculating the amount to bid, we do not necessarily need to calculate the full estimated cost for every auction. The amount we need to bid is $E(r,n-1) - E(r-1,n-1)$. Using the previous estimation method, we expect $E(r,n-1)$ to be the sum of the r smallest values of $n-1$ samples. We expect $E(r-1,n-1)$ to be the sum of the $r-1$ smallest values of $n-1$ samples. The difference, therefore, is the r^{th} lowest value, or the mean of the r order-statistic of $n-1$ trials. Therefore $X(r,n) = \mathbb{E}[X_{(r)}]$.

This is much simpler to calculate than what is done in previous algorithms. Note that when $r = n$ we must win everything, and must bid higher than the maximum of the distribution. We start by calculating the cumulitive distribution function (cdf) for the r order-statistic, $X_{(r)}$.

$$\begin{aligned} \mathbb{P}[X_{(r)} \leq x] &= \mathbb{P}[\text{there are at most } n-1-r \text{ observations greater than } x \text{ out of } n-1 \text{ trials}] \\ &= \sum_{j=0}^{n-1-r} \binom{n-1}{j} \mathbb{P}[X > x]^j \mathbb{P}[X \leq x]^{n-1-j}. \end{aligned} \tag{30}$$

From the cdf of the order statistic, we can calculate its mean, which is the amount we want to bid, because we know that for a random variable Y, $\mathbb{E}[Y] = \int_0^\infty \mathbb{P}[Y > x]dx$. Thus,

$$\mathbb{E}[X_{(r)}] = \int_0^\infty 1 - \sum_{j=0}^{n-1-r} \binom{n-1}{j} \mathbb{P}[X > x]^j \mathbb{P}[X \leq x]^{n-1-j} dx. \tag{31}$$

Since r is often significantly smaller than n, it can be faster to calculate 1-cdf directly as $\mathbb{P}[X_{(r)} > x]$,

$$\mathbb{E}[X_{(r)}] = \int_0^\infty \sum_{j=0}^{r} \binom{n-1}{j} \mathbb{P}[X \leq x]^j \mathbb{P}[X > x]^{n-1-j} dx. \tag{32}$$

Let $\mathcal{N}(\mu, \sigma, x)$ be the probability that a normal distribution with mean μ, and standard deviation σ is less than x, the cdf of the normal distribution. Let $P(x)$ be the probability for a single sample to be less than or equal to x, the cdf of the distribution of second highest bids. Then we have

$$\begin{aligned}
\mathbb{E}[X_{(r)}] &= \int_0^\infty \sum_{j=0}^{r} \binom{n-1}{j} \mathbb{P}[X \leq x]^j \mathbb{P}[X > x]^{n-1-j} dx \\
&= \int_0^\infty \sum_{j=0}^{r} \binom{n-1}{j} P(x)^j (1 - P(x))^{n-1-j} dx \\
&\approx \int_0^\infty \mathcal{N}((n-1)P(x), (n-1)P(x)(1-P(x)), r) dx.
\end{aligned} \tag{33}$$

This formula gives us a very close approximation of the optimal bid in any scenario. This is also much faster to calculate than the recursive algorithm. This is the formula that we reccomend to Chitika for use for bidding for Cidewalk. By using this formula at every iteration, they will always be able to fulfill the customer's request, and will spend less money over time to fill those requests.

7 Implementation

7.1 Principles of Computer Science Used

In this section, we explain basic computer science terminology and technique used in the implementation of our model.

7.1.1 Programming Language concepts

Programming Language Used We use MATLAB for each case's simulations. We also use MATLAB to determine which numerical method is best for our model in production. For the code embedded into Cidewalk's existing module, we use Erlang and C.

Fundamental Concepts Throughout the simulation and implementation stage, we follow fundamental concepts in writing code. These include commands, definitions, values, names, evaluations, structures, conditional expressions, parameter calling modes, functions and routines.

It is also important to be familiar with following basic concepts and be able to apply them in code: data types, control structures, data structures, syntax, and tools.

7.1.2 Documentation

We documented our code to satisfy following expectations:

To identify attributes, characteristics, capabilities, and qualities of a system. This is to explain well what has been implemented.

To overview software, to check relations to an environment and construction principles to be used in design of software components

To document actual code, interface, and algorithms.

7.1.3 Testing

Testing is an important part of software development. We test software to find errors by executing a program. By testing we can improve the quality of the code and validate our software.

The techniques we use to test our implementation are:

We use unit testing to determine whether usage procedures, operating procedures, source code units, and modules together with associated control data are appropriate for use.

We also test the function of the code by conducting functional testing, which is done by feeding input and checking output.

We test compatibility of code embedded to Cidewalk's Campaign module. [14][15]

7.1.4 Software Engineering Principles

Separation of concerns Is it a principle for separating a computer program into different units, such that each unit is given a separate concern. By following this principle, we are able to deal with two crucial concerns of data structure: basic functionality and support for data integrity.

Modularity We separate our model into components based on functionality and responsibility by following the principle of modularity.

Abstraction We separate the behavior of software components from their implementation by following the principle of abstraction by examine what the codes does, and how it does it.[21][22]

7.2 Computational Efficiency

7.2.1 The Case of Uniform Distribution

In the case of the uniform distribution, we have equation

$$E(r,n) = \frac{-1}{20}\big(E(r-1,n-1) - E(r,n-1)\big)^2 + E(r,n-1). \tag{34}$$

Here the base case is $E(0,0)$, and $E(r,0)$ is a linear recursion with $E(0,0)$ and $r > n$. Imagine that we form a recursion tree. Each level would have the same n, since the function always calls itself with $n-1$. Now analysing the tree and separating it, we obtain a triple recursion. After $n = 0$ we get a branches of linear recursion.

By analyzing the triple recursion, we can find the complexity by using a simple method. Assume that each node uses one time unit. Since each level has 3 times the nodes' times of the level before, that the first level has one node and the tree has n levels. The total time is the summation of all nodes. Thus we get the $totaltime = Summ(0,n)$ of 3^k. This can be given by the closed formula for a geometric progression, so the complexity should be $\mathcal{O}(\frac{(1-3^n+1)}{-2})$ which is $\mathcal{O}(3^n)$.

7.2.2 The Binomial Distribution

In the case of the binomial distribution, we have equation

$$E(r,n) = \sum_{i=0}^{r}(r-i)\binom{n}{i}p^{n-i}(1-p)^i. \tag{35}$$

We extract terms that do not have $\mathcal{O}(1)$, then we have the term

$$\sum_{i=0}^{r}\binom{n}{i}. \tag{36}$$

$n!$ can be computed in $\mathcal{O}(n)$.

$\sum_{i=0}^{r}$ can be computed in $\mathcal{O}(r)$. We know that $r < n$, so this is $\mathcal{O}(n)$.

Therefore, the total computation cost is $\mathcal{O}(n^2)$.

7.2.3 Using Order Statistics

In the case of order statistics, we have,

$$\mathbb{E}[X_{(r)}] \approx \int_0^\infty \mathcal{N}[(n-1)P(x), (n-1)P(x)(1-P(x)), r]dx. \tag{37}$$

To calculate the cdf of the normal distribution, we can use erf functions built in most of the programming languages. The complexity of erf function is $\mathcal{O}(1)$, and we use numerical methods to approximate an integral whose complexity is $\mathcal{O}(1)$ as well. Therefore, for the order statistics method, the complexity is $\mathcal{O}(1)$.

7.3 Ideas for Implementation

Since the order statistics method has the complexity of a constant, it is reasonable to use this method for implementation. We use the built-in function erf in Erlang to calculate the cdf of the normal distribution, and we use numerical methods that have the smallest error for a cdf function to approximate the actual integral.

In order to find out which numerical method is best suited for our model based on the use of order statistics, we ran MATLAB code to see which approximation method is the best fit to the model.

Let $n = 100,000$, $r = 1,000$, and $P(x) = 1 - e^{-x^2}$. We choose this formula for $P(x)$ because it is an increasing function from 0 to infinity, and it is bounded by 0 and 1. However, this is a very naive choice of $P(x)$, and we will discuss about a better way to estimate $P(x)$ later. Therefore

$$\begin{aligned} \mathbb{E}[X_{(r)}] &\approx \int_0^\infty \mathcal{N}[(n-1)P(x), (n-1)P(x)(1-P(x)), r]dx \\ &= \int_0^\infty \frac{1}{2}\big[1 + \operatorname{erf}(\frac{r-(n-1)P(x)}{(n-1)P(x)\sqrt{2}})\big]. \end{aligned} \tag{38}$$

In order to approximate the integral, we test 4 numerical methods for integration and choose one that best fits this method. Table 7.3 gives the comparison of our results using Trapezoidal rule, composite Simpson rule, Romberg integration, and Gaussian quadrature for different value of step size m. The value of $\mathbb{E}[X_{(r)}]$ in each case is displayed in each cell in Table 7.3. See the appendix for the code.

In the table m represents the step size for Trapezoidal rule and composite Simpson rule. d represents the steps for Romberg integration, and k represents the Gaussian quadrature at k, $(k = 2, \ldots, 5)$ points.

Table 7.3

Trapezoid Rule:

m	10	100	500	1000
Trap	0.180898164895	0.190492820225	0.190495173747	0.190495254291

Composite Simpson Rule:

m	10	100	500	1000
Simp	0.174925256950	0.190491969478	0.190495281139	0.190495281139

Romberg Integration:

d	2	4	6	7
Romb	0.167792240496	0.189105593708	0.190505235258	0.190494970118

Gaussian Quadrature:

k	2	3	4	5
Gauss	0	0	0	0

We choose to implement using trapezoidal rule because it is converge to desired result fastest.

7.4 Feasibility

We test the run-time of the calculation of the formula in Erlang language which is used by Chitika primarily for its campaign module. See the appendix for the code. In the code

first we calculate the cdf using the built in erf function, and then we get the approximation of the integral of the cdf using Trapezoidal rule with 150 steps. The total run time was 250 microseconds. We have 80 milliseconds in total between receiving the request and the deadline to bid on an auction. Therefore this algorithm is completely feasible.

7.5 Actual Implementation & Results

⟨ Redacted by sponsor ⟩

7.5.1 Estimating the Number of Available Auctions to Bid

As discussed in chapter 4, we use n to denote the umber of acceptable ad request arrivals sequentially, which gives the number of auctions available for us to bid on. We create a look up table for the value of n using the number of requests we have received in the past. The keys are zip code, time interval, and date of the week. To provide an illustrative example, if we need a value of n on Wednesday from 9am to 10am, it is easy and quick to obtain that desired value of n because we only need to search through hash table.

8 Result

We implemented our method with Chitika, and ran it live in a real world setting. We did not yet have a method to estimate $P(x)$, so we created a naive guess on a similar scale to previous data. We created two campaigns to each run for an hour, one in Boston and one in New York City. Each of these had a target of 15000 impressions. We also ran identical campaigns in the same places as control groups. One of those used the algorithm they currently used (Default). The other used an algorithm which had been developed by a Chitika employee recently (PS).

BOS DEFAULT:	100%	$1.03	(cost per thousand)
BOS PS:	100%	$0.92	(cost per thousand)
BOS WPI:	100%	$0.84	(cost per thousand)

NY DEFAULT:	100%	$1.02	(cost per thousand)
NY PS:	98%	$0.61	(cost per thousand)
NY WPI:	0.5%	$0.96	(cost per thousand)

Unfortunately, we guessed a $P(x)$, and the $P(x)$ we used causes the module to bid very low. For the campaign in Boston, it worked well and we believe if we could obtain a better estimation of $P(x)$ the result can be substantially improved. However, for the campaign in New York, the module bid too low, therefore only 0.5%, or 88 impressions, are met.

9 Conclusions

Based on the work we have done, we have found a fast algorithm for estimating the optimal bids in the situation that Cidewalk is in. When we need to win r out of n remaining auctions, all with an independent identically distributed random highest other bid, with bid distribution cdf $P(x)$, the algorithm tells us that the amount we want to bid is

$$\int_0^\infty \mathcal{N}\big[(n-1)P(x), (n-1)P(x)(1-P(x)), r\big]dx. \tag{39}$$

We also found an algorithm for that situation which would give a more accurate optimal bid, much more slowly. This mutually recursive algorithm calculates the optimal bid, $X(r,n)$, and the estimated cost $E(r,n)$ at all smaller values to get its answer,

$$X(r,n) = \begin{cases} 0 & r = 0 \\ \infty & r = n \\ E(r,n-1) - E(r-1,n-1) & \text{else} \end{cases}$$

$$E(r,n) = P(X(r,n)) \times (\mu_{C_i|C_i<X(r,n)} + E(r-1,n-1)) + P(X(r,n)) \times E(r,n-1). \tag{40}$$

10 Further Work

Moving forward, there are several issues the model does not address. When the number of auctions that need to be won is close to the number of auctions available to win (i.e, $r \approx n$), the model assumes that it must win the remaining auctions at any cost. Another model which considers the value of meeting the customer's goals would help keep high bids from that scenario in check.

Our model also assumes that the bids of the competitors are independent from our bid. If anyone else wants to win a certain number of auctions, then our bids will influence the market. Looking into how we influence the market and how that should change our bidding strategy may help save money. Also our model assumes an exact value for n, when in reality we will only have an approximation. Finding a way to model that could also save money. Finally when we used a guessed formula for $P(x)$ it did not work well for a campaign in New York City. Therefore it is very important to have an accurate estimate of $P(x)$ to obtain an optimal prediction of the price we want to bid.

References

[1] Robert J. Vanderbei. *Linear Programming: Foundations and Extensions*, Springer, 2007.

[2] Kuang-Chih Lee, Ali Jalali, and Ali Dasdan, *Real Time Bid Optimization with Smooth Budget Delivery in Online Advertising*. Arxiv Preprint, 2013.
`http://arxiv.org/abs/1305.3011`

[3] Herbert A. David. *Order Statistics*, Wiley, 1981.

[4] Zvi Drezner. *Computation of the Trivariate Normal Integral*, Mathmatics of Computation 62 (1994), 289-294 American Mathematical Society, 1994.

[5] Daniel Ralph and Steven Searby (Ed.). *Location and Personalisation: Delivering Online and Mobility Services*, Te Institution of Engineering and Technology, 2004.

[6] Matthew A. Carlton and Jay L. Devore. *Probability with Applications in Engineering, Science, and Technology*, Springer, 2014.

[7] Lan V. Truong. *A Novel Time-Varying Coding Scheme for the Gaussian Broadcast Channel with Feedback*, Arxiv Preprint, 2014.
`http://arxiv.org/abs/1404.2520`

[8] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*, Second Edition. MIT Press and McGraw-Hill, 2009.

[9] William J. Martin, *MA533/CS525M: Hardness vs. Randomness*, 2003. Lecture Notes.
`http://users.wpi.edu/~martin/TEACHING/533/2003/notes/notes.pdf`

[10] Donald Knuth. *The Art of Computer Programming: Fundamental Algorithms*, Third Edition. Addison-Wesley, 1997.

[11] J.H. Matthews. *Numerical Methods*. 2nd ed. United States of America: Prentice Hall. 1992

[12] Richard L. Burden and J.Douglas Fairs. *Numerical Analysis*, 9th Edition. Richard Stratton, 2010.

[13] Franceso Cesarini and Simon Thompson. *Erlang Programming*. O'Reilly. 2009.

[14] Michael Johnson. *Application Management: What you Need to Know For IT Operations Management*, 2011

[15] Khalid A. Buragga and Noor Zaman. *Software Development Techniques for Constructive Information Systems Design*. 2013.

[16] Joan Van Tassel and Lisa Poe-Hofield. *Managing Electronic Media: Making, Moving and Marketing Digital Content*. Focal Press, 2010

[17] Ariel Ortiz. *Teaching concurrency-oriented programming with Erlang*, SIGCSE '11 Proceedings of the 42nd ACM technical symposium on Computer science education, Pages 195-200, 2011.

[18] MathWorks. *Features of MATLAB*. Last time checked the website: 04-30-2015. `http://www.mathworks.com/products/matlab/features.html`

[19] Bruce Cameron Brown. *How to Use the Internet to Advertise, Promote and Market Your Business Or Web Site*. Atlantic Publishing Group, 2006.

[20] Roger S. Pressman. *Software Engineering. A Practioner's approach*, 6th Edition, McGraw-Hill, 2007

[21] Ian Sommerville. *Software Engineering*, 6th Edition, Pearson, 2000

[22] Vadim Yakovlev. *Numerical Methods for Calculus & DE. Course Web page*. Last time checked: 4-30-2015. `http://users.wpi.edu/~vadim/NM_I/MATLAB/B14_M-Scripts.html`

[23] Justin Boyer. *Gaussian Quadrature*. 2012. Lecture Notes. `http://www.math.utah.edu/~gustafso/s2012/2270/web-projects/LAGaussianQuadrature.pdf`

Appendices

A Related Terminologies

Here we explain some important advertising terminologies.

CPM: CPM represents cost per thousand advertisement impressions. To say an ad was purchased at 1 CPM means that if one thousand spaces at the same price would cost $1. Therefore the space cost $0.001.

Click: A click means a vistor interacts with an advertisement. This means the visitor actually clicks on an ad and goes to the advertiser's destination.

Targeting: "Targeting is purchasing ad space on Web sites that match audience and campaign objective requirements. " Brown (2006, p.82) [19]

Visit: A visit means a Web user with a unique address opening a Web site for the first time that day. "The number of visits is roughly equivalent to the number of different people that visit a site. " Brown (2006, p.329) [19]

B Sample simulation codes

B.1 Unifor Distribution Simulation

We use MATLAB to run simulation because it is intended primarily for numerical computing, and it is very easy for fast prototyping.

```
% x is our bid price for each auction
function x = x_test(r,n)

if r >= n
    x=2.5;
elseif r > 0
```

```
    x=1.5;
else
    x=0.5;
end

end


% p is randomly generated highest bid other than us
function p = p_uniform()
p = rand()*10;
end



% [Won, Cost] will output auctions we won and the total cost
function [Won,Cost] = A(r,n,p,x)
Won = 0;
Cost = 0;

while n > 0
        pt = p_test( );

       xt = x_test(r,n);


        if pt <= xt
            Won = Won +1;
            r = r-1;
            n = n-1;
            Cost = Cost + pt;
        else
            n = n-1;
        end
end
```

B.2 Order Statistics Simulation

```
% x is our bid price for each auction
function y = x_UniformOrderstats(r,n)


f = @(x) normcdf(r,n.*(x./10),n.*(x./10).*(1 - x./10));
y= integral(f,0,10);

end
end


% p is randomly generated highest bid other than us
function p = p_test( )
p=floor(rand()+1.5);
end

% [Won, Cost] will output auctions we won and the total cost
function [Won,Cost] = A(r,n,p,x)
Won = 0;
Cost = 0;

while n > 0
        pt = p_test( );

       xt = x_test(r,n);


        if pt <= xt
            Won = Won +1;
            r = r-1;
            n = n-1;
            Cost = Cost + pt;
        else
            n = n-1;
```

```
        end
end
```

C Numerical Methods to Approximate Integral

C.1 Composite Trapezoidal Rule

```
% [Won, Cost] will output auctions we won and the total cost
function I = trap(f,a,b,m,n,r)
% f is the function to be integrated
% a is lower bound
% b is upper bound
% m is the step size
% n is bids left
% r is bids we have to win


% p is the pdf, here we chose pdf to be 1-exp(-x^2),
% However, in order to change the interval of the integration from 0 to
% infinity, we need to change interval based on Monte Carlo methods, and
% original p(x) equals to:
p = @(x) 1-exp(-x^2/(x-1)^2);

% f is the cdf of normal distribution we get using central limit theorem

f = @(x) ((1/2).*(1+(erf((r-(n-1).*p(x))/((n-1).*p(x).*sqrt(2)))))).*(1/1-x^2);


%
% The function finds integral of f using composite trapezoid rule
%
h = (b-a)/m; S = feval(f,a);
%
```

```
for i = 1 : m-1
    x(i) = a + h*i
    S = S + 2*feval(f,x(i));
end
%
S = S + feval(f,b); I = h*S/2
```

C.2 Composite Simpson Rule

```
function I = Simp(f,a,b,m,n,r)
% f is the function to be integrated
% a is lower bound
% b is upper bound
% m is the step size
% n is bids left
% r is bids we have to win


% p is the pdf, here we chose pdf to be 1-exp(-x^2),
% However, in order to change the interval of the integration from 0 to
% infinity, we need to change interval based on Monte Carlo methods, and
% original p(x) equals to:
p = @(x) 1-exp(-x^2);

% f is the cdf of normal distribution we get using central limit theorem
f = @(x) ((1/2).*(1+(erf((r-(n-1).*p(x))/((n-1).*p(x).*sqrt(2))))));

%
% The function finds integral of f using composite simpson rule
%
h = (b-a)/m; S = feval(f,a);
%
for i = 1 : 2 : m-1
    x(i) = a + h*i
```

```
    S = S + 4*feval(f,x(i))
end
for i = 2 : 2 : m-1
    x(i) = a + h*i
    S = S + 2*feval(f,x(i))
end
%
S = S + feval(f,b); I = h*S/3
```

C.3 Romberg Integration

```
function [ W ] = Romb( f,a,b,d,n,r )
% f is the function to be integrated
% a is lower bound
% b is upper bound
% m is the step size
% n is bids left
% r is bids we have to win


% p is the pdf, here we chose pdf to be 1-exp(-x^2),
% However, in order to change the interval of the integration from 0 to
% infinity, we need to change interval based on Monte Carlo methods, and
% original p(x) equals to:
p = @(x) 1-exp(-x^2);

% f is the cdf of normal distribution we get using central limit theorem
f = @(x) ((1/2).*(1+(erf((r-(n-1).*p(x))/((n-1).*p(x).*sqrt(2))))));

%
%The function finds the integral of f on the interval [a,b] using
%d steps of Romberg integration (or accelerated Simpson's Rule).
%
T = zeros(d+1, d+1);
```

```
%
for k = 1 : d+1
    m = 2^k;
    T(1,k) = Simp(f,a,b,m,n,r);
end
%
for p = 1 : d
    q = 16^p;
    for k = 0 : d-p
        T(p+1, k+1) = (q*T(p, k+2) - T(p, k+1))/(q-1);
    end
end
%
for i = 1 : d+1
    table = T(i, 1 : d-i+2);
    disp(table)
end
%
W = T(d+1,1);

end
```

C.4 Gaussian Quadrature

```
function I = Gauss(f,a,b,k,n,r)

% f is the function to be integrated
% a is lower bound
% b is upper bound
% m is the step size
% n is bids left
% r is bids we have to win
```

```
% p is the pdf, here we chose pdf to be 1-exp(-x^2),
% However, in order to change the interval of the integration from 0 to
% infinity, we need to change interval based on Monte Carlo methods, and
% original p(x) equals to:
p = @(x) 1-exp(-x.^2);

% f is the cdf of normal distribution we get using central limit theorem

f = @(x) ((1/2).*(1+(erf((r-(n-1).*p(x))/((n-1).*p(x).*sqrt(2))))));


% The function finds integral of f on the interval [a,b] using
% Gaussian quadrature at k (k = 2, ..., 5) points
%
t = [-0.5773502692 -0.7745966692 -0.8611363116 -0.9061798459;
    0.5773502692 0.0 -0.3399810436 -0.5384693101;
    0.0 0.7745966692 0.3399810436 0.0;
    0.0 0.0 0.8611363116 0.5384693101;
    0.0 0.0 0.0 0.9061798459]
c = [1.0 0.5555555556 0.3478548451 0.2369268850;
    1.0 0.8888888889 0.6521451549 0.4786286705;
    0.0 0.5555555556 0.6521451549 0.5688888889;
    0.0 0.0 0.3478548451 0.4786286705;
    0.0 0.0 0.0 0.2369268850]
%
% Transformation of the interval of integration
x(1 : k) = 0.5*((b-1).*t(1:k,k-1) + b + a);
%
y = feval(f,x);
%
cc(1:k) = c(1:k,k-1);
cd = cc';
%
int = y*cd;
I = int*(b-1)/2
```

[22]

D Run Time Check

D.1 Erlang Part

```
-module(wpi).
-export([n/2,n/3,bid/2,bid/3]).
-on_load(init/0).

-define(SQRT2, math:sqrt(2)).

% Trapezoid integration
% Guard statement (when ...) protects against invalid ranges
%   F   -> Function to integrate
%   N   -> Number of checks to perform across the range
%   A   -> Start of the range
%   B   -> End of the range
integrate(F, N, A, B) when N > 0, A < B ->
    % Determine the total range of our integration
    Range = B - A,

    % Add up the midpoints
    Sum = lists:foldl(fun(I, Sum) ->
                              % X is the point to check in this iteration
                              X = A + (Range * I / N),
                              Sum + F(X)
                      end, 0.0, lists:seq(1, N)),
```

```
    % Now add the endpoints
    Sum2 = Sum + (F(A) + F(B)) / 2.0,
    Sum2 * Range / N.

% Defines the CDF for a given R and N.
% Note: math:erf/1 is unavailable on Windows platforms
cdf(R, N) ->
    fun (X) ->
            PX      = p(X),
            Mu      = (N - 1) * PX,
            Sigma   = (N - 1) * PX * (1 - PX),
            0.5 * (1 + math:erf((R - Mu) / (Sigma * ?SQRT2)))
    end.

% Basic model for win rate of bids
p(X) ->
    fix(1.0 - math:exp(-(X * X))).

% Fix for rounding errors; lim x -> 0; lim x -> infinity
fix(X) when X < 0.00000001 -> 0.00000001;
%fix(X) when X == 1 -> 0.99999999;
fix(X) when X > 0.999999999 -> 0.999999999;
fix(X) when X > 0, X < 1 -> X.

% Integrate CDF(R,N) across X ranging from 0 to R
% This is an improper integral, as I've been told the proper integral
% should be from -infinity to R.
n(R, N) ->
    n(R, N, trunc(1.5 / 0.01)).

n(R, N, Iterations) ->
    integrate(cdf(R, N), Iterations, 0, 1.5).

% These functions lead to the NIF or C++ implementation
% of the above algorithm.
```

```
bid(R, N) ->
    bid(R, N, 1.5 / 0.01).

bid(R, N, It) ->
    do_bid(float(R), float(N), float(It)).

do_bid(_R, _N, _It) ->
    erlang:nif_error({not_loaded, wpi_nif}).

% Load the NIF library
init() ->
    PrivDir = case code:priv_dir(?MODULE) of
                  {error, bad_name} ->
                      EbinDir = filename:dirname(code:which(?MODULE)),
                      AppPath = filename:dirname(EbinDir),
                      filename:join(AppPath, "priv");
                  Path ->
                      Path
              end,
    erlang:load_nif(filename:join(PrivDir, wpi_nif), 0).
```

D.2 C part

```
#include <erl_nif.h>
#include <math.h>
#include <functional>

#define SQRT2 1.4142135623730951

// Erlang NIFs are supposed to use C, so we need
// to build like we're a C library.
extern "C" {
    // We really want to limit x as it approaches 0 and 1,
    // but we can only represent so many digits, so let's
```

```
// simulate the limit as best we can.
inline double fix(double x) {
    if (x < 0.0000000000001f) {
        x = 0.0000000000001f;
    } else if (x > 0.9999999999999f) {
        x = 0.9999999999999f;
    }

    return x;
}

// Basic win-rate estimation
inline double p(double x) {
    return fix(1.0 - exp(-(x * x)));
}

// Integrate with the trapezoidal rule using it iterations
double trapezoid(std::function<double(double)> f, double it, double from, double to)
    double sum = 0.0f,
           c = 1.0f,
           range = to - from;

    // We should really be iterating with an integer, but
    // then I would have to explicitly cast it to prevent
    // implicit conversion to integer during calculation.
    for (; c <= it; c += 1.0f) {
        double x = from + (range * c / it);
        sum += f(x);
    }

    sum += (f(from) + f(to)) / 2.0f;
    return sum * range / it;

}
```

```
// Our exported function. Grabs the supplied Erlang terms,
// converts them to our C types, and kicks the whole thing
// off.
static ERL_NIF_TERM wpi_nif_bid(ErlNifEnv* env, int argc, const ERL_NIF_TERM argv[])
    double r = 0,
           n = 0,
           i = 0;

    // If we receive anything but doubles for the arguments,
    // return badarg.
    if (!enif_get_double(env, argv[0], &r)) return enif_make_badarg(env);
    if (!enif_get_double(env, argv[1], &n)) return enif_make_badarg(env);
    if (!enif_get_double(env, argv[2], &i)) return enif_make_badarg(env);

    // This is our CDF function, implemented as a lambda so
    // that we can readily capture the r and n values.
    std::function<double(double)> cdf = [ r, n ] (double x) {
        double px       = p(x),
               mu       = (n - 1.0f) * px,
               sigma    = (n - 1.0f) * px * (1.0f - px),
               res      = (0.5f * (1.0f + erf((r - mu) / (sigma * SQRT2))));

        return res;
    };

    // Now we convert the result of integration back to an
    // Erlang term and return it to the VM.
    return enif_make_double(env, trapezoid(cdf, i, 0.0f, 1.5f));
}


// These inform the VM of the module we expect to be tied to
// and which functions in that module we replace.
static ErlNifFunc nif_funcs[] = {
    {"do_bid", 3, wpi_nif_bid}
```

```
    };

    ERL_NIF_INIT(wpi, nif_funcs, NULL, NULL, NULL, NULL)
}
```