Rank and Impression Estimation in a Stylized Model of Ad Auctions

Jordan Berg, Carleton Coffrin, Amy Greenwald, and Eric Sodomka
Department of Computer Science, Brown University
{jberg, cjc, amy, sodomka}@cs.brown.edu

Abstract

Empirical research on ad auctions often aims to estimate model parameters (e.g., click probabilities) from aggregate data (e.g., average positions, total impressions, etc.). This is not surprising, because aggregate data are what the search engines reveal. But it has recently been pointed out that aggregate data can produce biased estimates [1]. In this work, we construct a disaggregated model of positions and impressions from aggregate data. Previous work has demonstrated the feasibility and benefits of disaggregation [13]. We extend the previous approach by formulating the disaggregation process as two intertwined problems, *rank* and *impression estimation*. We solve these problems (and a natural simplification) using dynamic, mathematical, and constraint programming techniques. We then evaluate the merits of our solution techniques in a simulated ad auction environment, the Ad Auctions division of the annual Trading Agent Competition.

1 Introduction

Most Internet advertising space is sold via ad auctions, which operate as follows: Sometime in advance, a publisher (e.g., a search engine) solicits bids from advertisers on potential queries. Then, when a user submits one of those queries to a publisher, the user is shown a set of sponsored search results (i.e., ads) alongside so-called *organic* results based purely on relevance and importance, not on sponsorship. The placement (also called slot or position) of each ad is determined on-the-fly by the publisher, who runs an auction based on the previously submitted bids. Then, if/when the user clicks on an ad, the corresponding advertiser pays the publisher an amount determined by the auction—a so-called *cost-per-click* (CPC). Higher slots tend to cost more because they are more likely to generate clicks.

In this environment, both publishers and advertisers face difficult decision problems. The publishers must determine CPCs, and in what order to display the winning ads, depending on bids. The advertisers in turn must decide what queries to bid on and the maximum CPC they are willing to pay. Correspondingly, work on sponsored search can be divided into two categories: ad auction design, which addresses the publishers' problem (e.g., [2; 3; 5; 7; 12; 15]), and bidding strategy design, which addresses the advertisers' problem (e.g., [4; 10; 13]). This paper falls into the latter category.

Adopting a common assumption in past research (e.g., [8; 13]),[1] we view the decision problem the advertisers face as decision-theoretic, not game-theoretic. As such, we assume the advertiser builds a model of the world around him, which predicts such things as the future bids and budgets of other advertisers and anticipated user behavior.[2] Then, taking this model as input, the advertiser makes its decisions.

Most existing empirical research on ad auctions aims to estimate models from aggregate data (e.g., average position, total number of impressions, etc.). This is not surprising, because aggregate data are what the search engines reveal. But it has recently been pointed out that aggregate data can produce biased estimates; for example, predicted click-through-rates can exceed actual [1]. Consequently, one must either figure out a way to adjust for potential bias in their estimates, or try to infer a disaggregated model of the intra-day variation in average positions, total number of impressions, etc. before estimating model parameters.

In this paper, we take the latter route. That is, we attempt to construct a disaggregated picture of slot dynamics over the course of a day—meaning which advertiser was in which slot when and for how long—from aggregate statistics. Previous work [13] has demonstrated the feasibility and benefits of disaggregation. We extend the previous approach by formulating the disaggregation process as two intertwined problems, *rank* and *impression estimation*. The impression estimation problem (IE) is to estimate the number of impressions that

[1]See Wellman *et al.* [16] for a discussion of the validity of this assumption.

[2]In most auctions, the rules are known. In real-world ad auctions, however, the rules are often not known. Hence, in general, an advertiser should incorporate predictions about the auction rules themselves into its model of the world.

each bidder sees in each position. The rank and impression estimation problem (RIE) is to estimate each bidder's rank as well, when sorted in (squashed) bid order. This decomposition is advantageous for several reasons, including that it allows us to clearly state and prove mathematical properties about *cascades*, the data structure we use to represent our solutions to these problems.

Using our theoretical insights, we develop a mathematical programming formulation of the rank and impression estimation problem, which can be solved with commercial integer programming solvers. We compare and contrast our mathematical programming formulation with a problem-specific tree search, similar to that which was developed previously [13]. We evaluate the merits of these two approaches using a simulated, experimental ad auction platform, the Ad Auctions division of the annual Trading Agent Competition (TAC AA) [9], for which both aggregate statistics and disaggregated data are readily available. In ongoing work, we are further evaluating the impact of these two competing approaches on our techniques for predicting things like opponents' (squashed) bids and budgets.

This paper is organized as follows. In Sec. 2, we describe the stylized ad auctions model from which this work will be based. In Sec. 3, we define a data structure we call a *cascade*, which stores the number of impressions each advertiser sees in each slot. Further, we present a dynamic program (DP) that populates a cascade, given the total number of impressions seen by all advertisers and their ranks. We then state several key properties that characterize cascades. In Sec. 4, we define the rank and impression estimation problems, and we describe our integer and constraint programming solutions to IE, as well as a least discrepancy search which we wrap around these solutions to solve RIE. In Sec. 5, we report on the performance of our solutions in TAC AA relative to ground truth, which we calculate using our DP.

2 A Stylized Model

Imagine a set of N advertisers, each of which is interested in displaying its ad in one of M slots alongside the organic search results for a given query. At the start of each time period, each advertiser submits a bid and spending limit to the publisher. The submitted bids are then ranked by either bid or revenue, or something in between, as determined by a squashing parameter [11]. When a user searches for the given query, the ads that are displayed are those of the advertisers with the top M (squashed) bids, in order, from highest to lowest, of (squashed) bid. Advertisers pay the publisher as users click on their ads according to standard [7] generalized second price (GSP) auction rules (i.e., they pay the minimum amount they would have had to bid to maintain their assigned slot). Once an advertiser has paid the publisher so much that another click would put it over its spending limit, it drops out of all subsequent auctions, and the remaining advertisers are re-ranked. In particular, the advertisers previously ranked below that advertiser move up one position in the ranking. This process continues until either all agents exhaust their budgets, or no further users place search queries.

At the end of the time period, each advertiser receives the following aggregate data from the publisher: its total number of impressions (i.e. the number of times its ad was displayed) and the average position of all advertisers that participated in the day's auctions.

In the remainder of the paper, we use the term *agent* in place of advertiser or bidder.

3 Cascade Calculations

The first problem we present is a simplification of rank and impression estimation in which we know not only the rank of all agents, but the total number of impressions seen by each agent as well. We originally studied this problem because its solution is necessary to compute ground truth and is hence essential for experimentation with our constraint programming solvers for the IE and RIE problems. But in doing so, we identified mathematical properties of cascades which gave rise to integer programming solutions to IE and RIE.

3.1 The Cascade Data Structure

We represent the slot dynamics that arise in our stylized model using a data structure we call a *cascade*. In nature, a cascade is a structure in which water flows down a stair case shaped rock formation. For our purposes, a cascade is a data structure in which impressions flow *up* an inverted stair case shaped formation. The stairs in our cascades arise whenever one agent exhaust its budget, at which point the impressions the other agents see flow up from lower slots to higher ones.

Given a ranked list of agents $1, \ldots, N$ and the total number of impressions each agent j sees, $T_j \geq 0$, a cascade can be populated as follows. First, agent 1 sees T_1 impressions in the first slot. Second, if $T_2 \leq T_1$, then agent 2 sees T_2 impressions in the second slot; or, if $T_2 > T_1$, then agent 2 sees T_1 impressions in the second slot, and $T_2 - T_1$ impressions in the first slot. In general, the number of impressions that agent j sees in slot j is bounded above by the number of impressions agents 1 through $j-1$ saw in slot $j-1$. Similarly, the number of impressions that agent j sees in slot $j-1$ is is bounded above by the number of impressions agents 1 through $j-1$ saw in slot $j-2$. And so on.

We formalize this algorithm as a dynamic program in the next section. We then use our formalization to characterize the mathematical properties of cascades.

3.2 Dynamic Program

Given a set of n agents in rank order, a cascade is described by a vector I of length n. The jth entry in this vector is itself a vector of length j. Intuitively, $I_{j,k}$ denotes the number of impressions agent j sees in the kth slot, for $k \in \{j, \ldots, 1\}$.

We calculate cascade values (i.e., the entries in "matrix" I) in terms of two auxiliary matrices S and R: $S_{j,k}$

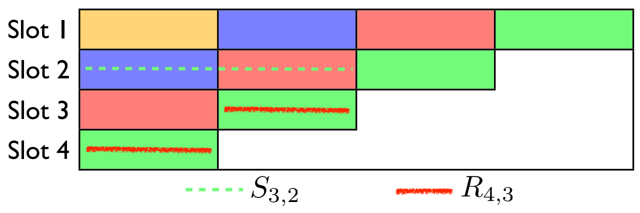


Figure 1: $S_{3,2}$ is the sum of the values of the cells with green dotted lines running through them: i.e., the number of impressions the agents ranked 1 through 3 (the orange, blue, and red agents) see in slot 2. $R_{4,3}$ is the sum of the values of the cells with the red splotchly lines running throught them: i.e., the number of impressions agent 4 (the green agent) sees in slots 4 and 3.

records the total number of impressions the agents with ranks 1 through j see in slot k and $R_{j,k}$ records the total number of impressions agent j sees in slots j through k (see Figure 1). Mathematically, $S_{j,k} = \sum_{j'=1}^{j} I_{j',k}$ and $R_{j,k} = \sum_{k'=k}^{j} I_{j,k'}$.

Our definition of cascade values is a mutually recursive one, where I is computed in terms of S and R, and vice versa. More specifically, given a ranked list of agents $1, \ldots, N$ and the total number of impressions each agent j sees, $T_j \geq 0$, S, R, and ultimately I can be defined inductively as follows:

- for all $j \in \{1, \ldots, N\}$ and for all $k \in \{j, \ldots, 1\}$,

$$I_{j,k} = \begin{cases} T_j, & \text{if } j = 1 \\ \min\{S_{j-1,k-1} - S_{j-1,k}, T_j - R_{j,k+1}\}, & \text{otherwise} \end{cases} \quad (1)$$

- for all $j \in \{0, \ldots, N\}$ and for all $k \in \{j+1, \ldots, 1\}$,

$$S_{j,k} = \begin{cases} 0, & \text{if } k = j+1 \\ S_{j-1,k} + I_{j,k}, & \text{otherwise} \end{cases} \quad (2)$$

- for all $j \in \{1, \ldots, N\}$ and for all $k \in \{j+1, \ldots, 1\}$,

$$R_{j,k} = \begin{cases} 0, & \text{if } k = j+1 \\ R_{j,k+1} + I_{j,k} & \text{otherwise} \end{cases} \quad (3)$$

These equations form the basis of the dynamic program for computing cascade values presented in Algorithm 1. Initially, agent 1 sees T_1 impressions in slot 1. Subsequently, in the nested for loops, we calculate the number of impressions seen by agents 2 through N in slots $k = j$ through 1. Define R_j as the total number of impressions agent j has seen thus far, and S_k as the total number of impressions agents ranked above j have seen thus far in slot k. If $S_{k-1} - S_k$ (i.e., the difference between the total number of impressions seen thus far by the agents ranked above j in slots $k-1$ and k) is less than the number of impressions agent j has yet to see (i.e., $T_j - R_j$), then a portion of the aforementioned difference is alloted to agent j, but no more than $T_j - R_j$. The quantity S_0 is initialized to ∞ so that $T_j - R_j$ is always (i.e., for all agents j and for all slots k) less than $S_0 - S_k$. That way, if an agent sees any impressions at all in slot 1, it sees all impressions it has yet to see in that slot.

Algorithm 1 Dynamic Program

```
Input: a ranked list of agents $1, \ldots N$
Input: the total number of imp'ns $T_j$, for all agents $j$
Output: the number of impressions $I_{j,k}$ that each agent $j$ sees in each slot $k$
  $S_0 = \infty$
  $I_{1,1} = R_1 = S_1 = T_1$
  for $j = 2$ to $N$ do
    $R_j = 0$
    for $k = j$ down to 1 do
      $S_k = 0$
      $I_{j,k} = \min\{S_{k-1} - S_k, T_j - R_j\}$ {the number of impressions agent $j$ sees in slot $k$}
      $S_k = S_k + I_{j,k}$ {the total number of impressions agents see in slot $k$}
      $R_j = R_j + I_{j,k}$ {the total number of impressions agent $j$ sees}
```

In fact, the inductive definition and dynamic program presented above are only valid when the number of slots M is greater than or equal to the number of agents N. But more commonly, $M < N$. In this more common case, we can proceed as above for all agents with ranks $1, \ldots, M$. But to handle the agents ranked below M (i.e., $i \in \{M+1, \ldots, N\}$), we create imaginary slots $k \in \{M+1, \ldots, N\}$. The number of impressions these agents see in imaginary slots is unconditionally set as follows:

$$I_{j,k} = S_{j-1,k-1} - S_{j-1,k}, \quad \text{if } k \in \{M+1, \ldots, N\} \quad (4)$$

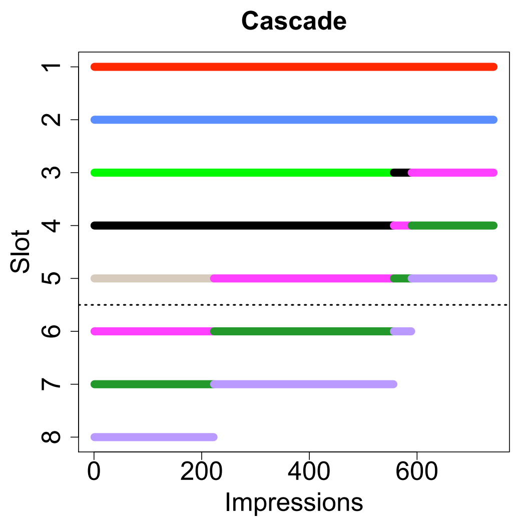
The number of impressions they see in real slots is determined almost as before, except that a comparison is made not simply to $T_j - R_{j,k+1}$, but rather to this same quantity offset by $R_{j,M+1}$, the number imaginary impressions that agent j sees, because the term $R_{j,k+1}$ includes both real and imaginary impressions:

$$I_{j,k} = \min\{S_{j-1,k-1} - S_{j-1,k}, T_j - (R_{j,k+1} - R_{j,M+1})\}, \quad \text{if } k \in \{1, \ldots, M\} \quad (5)$$

Figure 2 depicts a sample cascade that was constructed using our dynamic program given the data presented in the corresponding table, assuming 8 agents and 5 slots, as in TAC AA.

3.3 Properties of Cascades

Cascades can be characterized by the following essential properties: 1. For all $j \in \{1, \ldots, N\}$ and $k \in \{j, \ldots, 1\}$, $I_{j,k} \geq 0$. That is, no agent sees a negative number of impressions in any slot. 2. Impressions run out only once. By this we mean the following: For all $k \in \{M_j, \ldots, k^* + 1\}$, $I_{j,k} = S_{j-1,k-1} - S_{j-1,k}$, until some critical slot k^* at which point $I_{j,k^*} = T_j - R_{j,k^*+1}$.



Agent (i)	Impressions (T_i)
1	742
2	742
3	556
4	589
5	222
6	520
7	186
8	153

Figure 2: Sample input to the dynamic program, and corresponding cascade when $N = 8$ and $M = 5$.

Thereafter (i.e, for all $k \in \{k^* - 1, \ldots, 1\}$), $I_{j,k} = 0$. Here, $M_j = \min\{j, M\}$.

Equivalently, for all $j \in \{1, \ldots, N\}$ and $k \in \{j, \ldots, 1\}$,

1. $I_{j,k} \geq 0$.

2. $R_{j,k+1} \leq S_{j-1,k}$.

   This property captures the essence of the cascade data structure. In words, the total number of impressions agent j has seen so far (i.e., through slot $k+1$) is always *less than or equal to* the total number of impressions all agents above him in the ranking have seen in slot k.

3. $I_{j,k} > 0 \Rightarrow R_{j,k+1} = S_{j-1,k}$.

   In words, if agent j sees any impressions at all in slot k, then it must be the case that the total number of impressions agent j has seen so far (i.e., through slot $k+1$) is *precisely equal to* the total number of impressions all agents above him in the ranking have seen in slot k. It cannot be more, or the aforementioned property would be violated; and it cannot be less, because otherwise there would be an opportunity for agent j to see more impressions in slot $k+1$.

4 Rank and Impression Estimation

En route to solving the rank and impression estimation problem, we first set out to solve only the impression problem (IE). That is, we assume that the agents' ranks are still given, but that total number of impressions seen by each agent other than oneself is not known. Instead, for each agent j, agent i knows only agent j's average position μ_j, defined as:

$$\mu_j = \frac{\sum_{k=1}^{M} I_{j,k} k}{T_j} = \frac{\sum_{k=1}^{M} I_{j,k} k}{\sum_{k=1}^{M} I_{j,k}} \qquad (6)$$

These equations describe N constraints in the IE problem. But recall that agent i also knows the total number of impressions it sees, T_i. This information leads to an $N+1$th constraint: $T_i = \sum_{k=1}^{M_i} I_{i,k}$. IE is a constraint satisfaction problem (CSP): the entries in the cascade matrix are the variables, and the $N+1$ aforementioned constraints together with the cascade properties comprise the constraints.

Unfortunately, the set of cascades that corresponds to an arbitrary instance of IE is not a singleton. For example, if there are 3 agents, and we, agent 3, see 100 impressions, while the average positions for agents 1, 2, and 3 respectively are 1.0, 2.0, and 3.0, then agent 2 may see any number of impressions greater than 100, and agent 1 may see any number of impressions greater than the number agent 2 sees.

Indeed, the T_j values, for $j \neq i$, are not sufficiently constrained. Hence, we enhance the IE CSP with an objective function that helps select among the various feasible T_j's. In principle, the objective function is straightforward: an agent should choose the most likely T_j's given its information, including current observations and past models (i.e., its past predictions of the bids and budgets of the other agents that gave rise to the current cascade). That is, an agent should choose a solution that maximizes $P(\mathcal{T}_j' s \mid \text{observations}, \text{models}) = P(\mathcal{T}_1, \ldots, \mathcal{T}_{i-1}, \mathcal{T}_{i+1}, \ldots, \mathcal{T}_N \mid T_i, \vec{\mu}, \text{models})$, where $\mathcal{T}_j$ is a random variable ranging over the total number of impressions agent j sees.

But there are two problems with this approach. The first is that the IE and RIE problems are meant to precede modeling, so it is circular to take models themselves as inputs to these problems. We address this problem in our experimental section, where we vary the models that our algorithms take as input, considering both no models and models based on previous cascade predictions.

The second problem with this approach is that it is unwieldy to calculate the desired probability. We address this problem by simplifying the objective function: first, we (unreasonably) assume independence among all the $\mathcal{T}_j$ and then we (reasonably) ignore any dependencies among the T_j's and the μ_j's. The probability is then: $\prod_{j \neq i} P(\mathcal{T}_j = T_j \mid \text{models})$. Finally, when using historical priors, we assume each random variable $\mathcal{T}_j$ is a Gaussian, whose mean value is an exponential moving average $\overline{T}_j$ over the past non-zero total number of impressions predictions for agent j, and whose standard deviation σ_j is $\overline{T}_j$ multiplied by a large constant, because our priors are not necessarily reliable.

In summary, we define the IE problem as follows: given constants T_i for agent i and μ_j for all agents j, together with variables $I_{j,k}$ for all agents j and slots k and T_j for all agents $j \neq i$,

$$\arg\max_{I_{j,k}, T_j} \prod_{j \neq i} P(\mathcal{T}_j = T_j \mid \text{models}) \tag{7}$$

subject to:

$$I_{j,k} \geq 0 \quad \forall j, k \tag{8}$$
$$R_{j,k+1} \leq S_{j-1,k} \quad \forall j, k \tag{9}$$
$$I_{j,k} > 0 \Rightarrow R_{j,k+1} = S_{j-1,k} \quad \forall j, k \tag{10}$$
$$\mu_j = \frac{\sum_{k=1}^{M_j} I_{j,k} k}{\sum_{k=1}^{M_j} I_{j,k}} \quad \forall j \tag{11}$$
$$T_j = \sum_{k=1}^{M_j} I_{j,k} \quad \forall j \tag{12}$$

These constraints are exactly the properties described in Sec. 3.3, along with the average position and total impression constraints.

We tackle this problem using two solution techniques: integer linear programming (ILP) and constraint programming (CP). In the former, we simply feed ILPs into a commercial solver; in the latter, we design and implement our own problem-specific tree search.

4.1 Integer Linear Programming

An integer linear program cannot incorporate even the simplified objective function in Equation 7 because it is multiplicative, and hence non-linear. Consequently, we approximate this objective function with one that is additive: $\sum_j \left| \frac{T_j - \overline{T}_j}{\sigma_j} \right|$. This objective function can be represented in a linear fashion as follows:[3] We add two constraints per agent: $T_j \leq \overline{T}_j + \sigma_j x_j$ and $T_j \geq \overline{T}_j - \sigma_j x_j$. The objective is then to minimize the total number of standard deviations between T_j and $\overline{T}_j$: $\min \sum_j x_j$.

Our additive, linear objective function approximates the multiplicative one in the following senses: (i) setting T_j close to the mean of the prior is preferred by both objectives; and (ii) if the standard deviation corresponding to a variable T_j is large, then neither objective function is particularly sensitive to the setting of T_j. Figure 3 depicts the two objective functions—the one in Equation 7, and the ILP's linear approximation.

Aside from the conditional constraints, incorporating the other IE constraints into an ILP is straightforward. Consider the conditional constraint $A \rightarrow B$. This is equivalent to $\neg A \vee B$, and can be broken down into two new constraints, of which only one needs to be satisfied. To allow for only one of these constraints to be satisfied, we introduce a boolean variable, say z, and then add a large constant times z to one of the new constraints, and a large constant times $1 - z$ to the other. Then, depending on whether the boolean variable is true or false, one of the constraints becomes trivial to satisfy, leaving only the other constraint.[4]

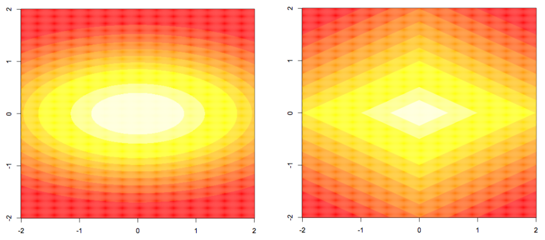


Figure 3: A visualization of the objective function described in Equation 7 for a two-agent cascade (left), and the ILP's simplified linear objective function (right).

In addition to experimenting with this ILP, we also experiment with a LP relaxation of this ILP—a mixed-integer linear program, in which the $I_{j,k}$ (and hence, the T_j) are real- rather than integer-valued.

4.2 Constraint Programmming

In addition to using an industrial-strength ILP solver to solve IE (and RIE), we also solve these problems using constraint programming (CP), by which we mean problem-specific tree search. CSPs are often solved using CP solvers (such as Comet [6]). The tree searches these solvers initiate take advantage of the compositionality of well understood constraints and their specialized satisfaction algorithms. Since our applications involve only cascade constraints, which are not particularly well studied, we solve these CSPs using our own light-weight CP solver that is only equipped to solve cascade satisfaction problems.

Like our ILP, our CP solver searches for $I_{j,k}$: i.e., the number of impressions each agent sees in each slot. It begins by assigning agent 1 (equivalently, the agent with average position 1.0) some constant number of impressions, say L, somewhere in the range of 1 and a fixed upper bound T on the maximum total number of impressions (i.e., the maximum number of impressions seen in slot 1). If agent 2's average position μ_2 is fractional, then μ_2 is necessarily less than 2.0, meaning that agent 2 sees some impressions in slot 1. By the properties of a cascade, agent 2 sees L impressions in slot 2. Given this information, the number of impressions, call it x, that agent 2 sees in slot 1 is fully determined: i.e., $x = \frac{(\mu_2 - 2)L}{(1 - \mu_2)}$, because agent j's fractional average position $\mu_2 = \frac{2L + 1x}{L + x}$.

More generally, when the agent with rank j has a fractional average position, the number of impressions it could have seen in slots 2 through j is constrained by the existing cascade: i.e., for all $k = j, \ldots, 2$, $I_{j,k} \leq S_{j-1,k-1} - S_{j-1,k} \equiv L_k$. Hence, we fill in impressions

[3]See http://www.aimms.com/aimms/download/manuals/aimms3om_linearprogrammingtricks.pdf.

[4]Again, see http://www.aimms.com/aimms/download/manuals/aimms3om_integerprogrammingtricks.pdf.

almost exactly as we did in the dynamic program. In the DP, for each agent j (in rank order) and for each slot $k = j \dots 1$, we assign agent j all L_k impressions unless doing so would result in agent j seeing more than T_j impressions in total. This incremental assignment procedure is guaranteed to terminate because the total number of impressions agent j sees is monotonically non-decreasing in the number of impressions seen. In the CP solver, we assign agent j all L_k impressions unless doing so would result in agent j's average position falling below μ_j. Again, this incremental assignment procedure is guaranteed to terminate because average position is monotonically non-increasing in the impressions agent j sees in higher slots (i.e., slots with lower numbers).

In summary, our CP solvers rely on a subroutine which, for agents with fractional average positions, populates the cascade almost exactly as the dynamic program does. Recall the formula, $I_{j,k} = \min\{S_{j-1,k-1} - S_{j-1,k}, T_j - R_{j,k+1}\}$. The term $T_j - R_{j,k+1}$ represents the number of impressions j has yet to see before reaching its total. Here, $I_{j,k} = \min\{S_{j-1,k-1} - S_{j-1,k}, \bar{I}_{j,k}\}$, where $\bar{I}_{j,k} = \left(\mu_j \sum_{i=k+1}^{j} L_i - \sum_{i=k+1}^{j} iL_i\right)/(k - \mu_j)$. The term $\bar{I}_{j,k}$ represents the number of impressions agent j would have to see in slot k in order to reach its average position. Hence, agent j sees either $S_{j-1,k-1} - S_{j-1,k}$ impressions in slot k, or if seeing that many would result in an average position that is below μ_j, it sees only $\bar{I}_{j,k}$.

While the number of impressions each agent j with a fractional number of impressions sees in each slot is constrained by the properties of the cascade, the number of impressions agents with whole number average positions see in each slot is unconstrained. Hence, the search within our CP solver is over the number of impressions agents with whole number average positions (e.g., agent 1) see in each slot.

When our CP solver uses historical priors, the search begins at the mean (i.e., the expected number of total impressions), and then zig zags back and forth around the mean with ever-increasing magnitude. When our CP solver operates without historical priors, for agent 1 (with average position 1.0), it searches integers in the range of 1 and T in increments of Δ, the discretization factor; for all other agents j (with whole number average positions greater than 1.0), it initializes its search by assigning j $S_{j-1,j-1}$ impressions in slot j and decrementing from there (again, according to Δ).

Having explained how our CP solver generates cascades (i.e., candidate solutions), it remains to explain how our solver evaluates the quality of these solutions, so that it can return the best one it finds (and prune based on quality). Because we discretize the search space, our CP solver is not guaranteed to satisfy Constraints 11 or 12. Instead, for each candidate solution, we calculate $\hat{\mu}_j = \frac{\sum_{k=1}^{M_j} I_{j,k}k}{\sum_{k=1}^{M_j} I_{j,k}}, \forall j$ and $\hat{T}_i = \sum_{k=1}^{M_i} I_{i,k}$. Consequently, the probability of interest here is: $P(\mathcal{T}_1, \dots, \mathcal{T}_{i-1}, \mathcal{T}_i, \mathcal{T}_{i+1}, \dots, \mathcal{T}_N, \mathcal{M}_1, \dots, \mathcal{M}_N, \mid T_i, \vec{\mu}, \text{models})$, where $\mathcal{M}_j$ is a random variable ranging over agent j's average position. As above, we simplify this objective by ignore any dependencies among the T_j's and the μ_j's. The new objective function is then: $\left(\prod_{j \neq i} P(\mathcal{T}_j = T_j \mid \text{models})\right) P(\mathcal{T}_i = \hat{T}_i \mid T_i) \left(\prod_j P(\mathcal{M}_j = \hat{\mu}_j \mid \mu_j)\right)$. Finally, when using historical priors, we assume each random variable $\mathcal{M}_j$ (and $\mathcal{T}_i$) is a Gaussian, with mean μ_j (or T_i) and standard deviation equal to μ_j (or T_i) multiplied by a small constant, because the μ_j's (and T_i) are reliable statistics.

Our CP solver does not have a time limit; instead time is indirectly controlled by Δ. But in the interest of time, we do prune cascades whenever agent i's error in average position exceeds that of the best solution found so far.

4.3 Least Discrepancy Search

We solve the rank and impression estimation problem (RIE) in two ways: (1) using an extension of our ILP that searches over rankings as well (which we omit here due to space constraints), and (2) wrapping a least discrepancy (LDS) search around our CP solver, which fixes a ranking of the agents, thereby reducing RIE to IE.

The LDS starts with an initial guess at the ranking. This initial guess is a ranking in which any agent with a whole number average position is ranked in that position, and all other agents are greedily filled into the remaining open positions, with the next lowest available slot being assigned to the remaining agent with the next lowest average position. This initial guess is passed to the CP, which solves the corresponding IE problem. The LDS records the ensuing cascade and its objective value. All two-swaps of this initial ranking are then added to a priority queue, with priority being defined as the sum of the absolute values of the differences in average positions between swapped agents. In this manner, the next guess is the ranking in the priority queue that minimizes the discrepancy between itself and the initial guess. Whenever a ranking is evaluated by the CP, all two-swaps from this ranking are subsequently added to the priority queue (if they have not already been considered). The LDS terminates after evaluating a fixed number of rankings, and the ranking and corresponding cascade with the highest objective value seen is returned.

5 Evaluation

In this section, we evaluate how well our algorithms estimate cascades. We evaluate our algorithms' performance on both the IE sub-problem, where bid ranking is known, and the RIE problem, where bid ranking is unknown.

Our evaluation is done in the context of the TAC AA market game, a simulated ad auction environment that was released as part of the annual Trading Agent Competition (TAC). The TAC Ad Auctions (AA) game [9] captures many of the intricacies faced by advertisers in real ad auctions, such as how to bid on multiple queries and set budgets in the presence of noisy, delayed, and incomplete information. Furthermore, it enables interested parties to benchmark various ad auction designs and bidding strategies in a quasi-realistic, competitive environment without incurring the monetary risks associated with real-world experimentation.

5.1 Prediction Challenge

To evaluate performance, we created what we call a "prediction challenge" using logs from previously played TAC AA games, similar to the approach taken in other TAC games [14]. The games we run on are a subset of the 2010 TAC AA finals. For each game, auction, day, and agent, if the agent participated in the auction, we look at the aggregate data that the agent received on that day (its own total impressions, and all agents' average positions), and make predictions about the resulting cascade from that agent's perspective. Since each agent receives different summary statistics, we can make separate predictions for each auction participant.

As stated earlier, cascade predictions are inputs to other models, such as bid and budget models. But different models rely on different components of the cascade predictions. For example, a model that predicts the total number of searches may only need as input from the cascade the total number of impressions seen by all the agents. On the other hand, models of an opponent's bids and budgets may depend more heavily on the specifics of the cascade predictions; the time an opponent spent in each slot, for example, should impact predictions about how many clicks that opponent received and how much money it spent. To evaluate the usefulness of our predictions as inputs to various conceivable models, we consider three different forms of cascade predictions: $I_{j,k}$ values are predictions of how many impressions each agent sees in each slot; T_j values are predictions of how many total impressions each agent sees; $T = \sum_j T_{j,1}$ values are the total number of impressions seen during the auction on the given day.

Ground truth $I_{j,k}$, T_j, and T values are determined by manipulating data within the game logs. These logs contain the total number of impressions each agent saw (T_j) as well as the squashed bids of each agent. When each squashed bid is unique, there is a unique bid ordering, and we can infer the $I_{j,k}$ ground truth values using our dynamic programming algorithm. The total number of impressions across agents (T) can then be computed by summing up the impressions seen in the first slot. When each squashed bid is not unique, there can be rare cases where the ground truth values are unknown (for example, if two agents have average position 5.0 and each saw 10 impressions, we cannot determine which of these agents was shown first). We throw out such instances. Of the 9280 auctions that occurred in our data set of 10 games, 16 queries, and 58 days, we threw out 83 instances due to ambiguous squashed bid rankings.

For each type of impressions prediction we make, we look at the ground truth and compare it to the predicted value. We report the mean absolute difference between these predicted and actual $I_{j,k}$, T_j, and T values. Absolute instead of relative error is reported because ground truth impression per slot values can be zero, leading to undefined relative error, or they can be very small (e.g., when agents place probing bids), leading to artificially high relative error values. For ranking predictions, we report the fraction of instances in which the correct ranking is predicted for the RIE problem. All predictions are evaluated with respect to time; the CP is given different discretization factor (Δ) values which determine time spent, while the ILP and MIP are given a maximum amount of time that can be spent on a given instance.

Because agents are playing a repeated game and receive summary statistics at the start of each time period, they may be able to use their past cascade predictions to inform their current ones. To test this effect, in some experiments, we endow agents with historical priors: i.e., prior beliefs about each opponent's total number of impressions based on past predictions. In other experiments, we assume a uniform prior, which amounts to assuming agents have no prior beliefs about their opponents' impressions.

5.2 Algorithmic Performance

Figure 4 shows the results of the prediction challenge when run on both the IE and RIE problems from the point of view of our own agent, Schlemazl [4]. For all three metrics of impression prediction accuracy, the CP is the most accurate predictor for smaller time limits, until a certain threshold where the ILP becomes more accurate than the CP. The MIP is faster than the ILP but never more accurate than the CP, which suggests that integer decision variables are important for ILP accuracy. For TAC AA's specific time limits, the CP makes the most accurate impression predictions.

Among the predictions made for $I_{j,k}$, T_j, and T, error is lowest when predicting impressions per slot. This is a combination of two effects: first, $I_{j,k}$ values are smaller, so there is less room for error; second, $I_{j,k}$ predictions are often correct except for the number of impressions seen by the top agent in the first slot. When an agent in the first slot sees more impressions than anybody else, that agent could have seen an arbitrarily high number of impressions without affecting the impression per slot predictions for the other agents.

For all algorithms and error metrics, impression error is lower in the IE problem than the RIE problem. This is unsurprising, because incorrect ranking predictions in the RIE problem will lead to greater impression error. Figure 5 (Left) shows the fraction of instances of the RIE problem for which the correct ranking was predicted. Ranking accuracy improves with time, and like the impression accuracy results, the CP outperforms the ILP at smaller time limits. As the time limit increases, the LDS wrapped around the CP does not check more possible rankings, but its objective value for each ranking becomes more accurate, since its discretization factor for the resulting IE problem is smaller at larger time limits. We also see that the poor performance of the MIP can be explained by its incorrect ranking predictions. Allowing real-valued impression variables appears to greatly increase the space of feasible rankings, and the

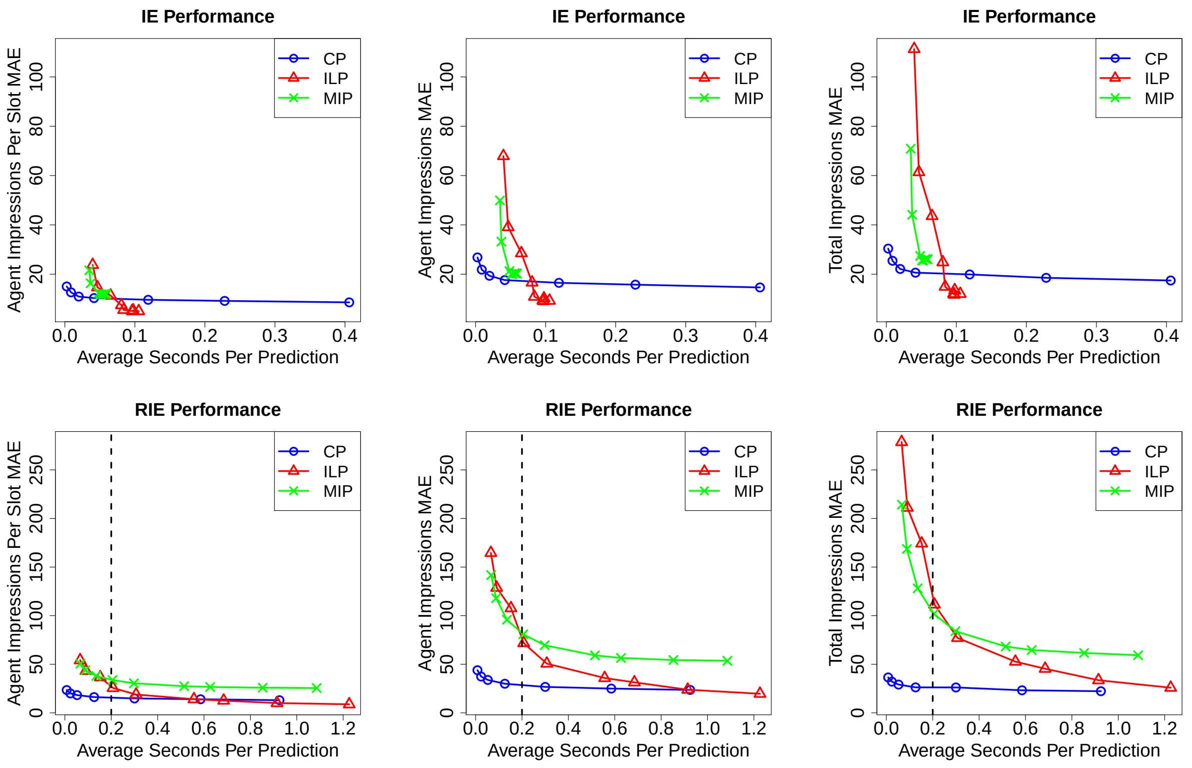


Figure 4: Accuracy of impression predictions for the IE (top row) and RIE (bottom row) problems. The left column shows predictions for agent impressions per slot, the middle column for agent impressions, and the right column for total impressions across agents. The black dashed line is approximately the amount of time an agent would have to build each of its per-query cascade predictions in TAC AA, to allow time for other predictions and optimization.

MIP assigns some of these rankings high objective values, but with integer-valued impression variables, these rankings are not considered.

Using historical priors led to mixed results, improving the CP's accuracy but not the ILP's or the MIP's. Historical priors are not based on ground truth but on past cascade predictions, so there is potential for a positive feedback loop which can cause predictions to become increasingly inaccurate. Similarly, accurate predictions will tend to reinforce themselves. Figure 5 (Right) illustrates how the error in predictions with historical priors compares favorably to the error in predictions with uniform priors for the CP. Historical priors are perhaps taking advantage of situations in which the day-to-day IE and RIE problems are underconstrained only sporadically. When a majority of consecutive instances have unique solutions, those solutions can be used to bootstrap predictions on more difficult instances: i.e., when many agents have whole number average positions.

5.3 Agent Behavior

We now examine prediction performance differences of individual agents for just one of the problems (RIE, uniform priors) and one of our algorithms (CP) (see Table 1). Surprisingly, some of our worst predictions occur when we take on the role of Mertacor and TacTex, two of the top three finishers in the 2010 TAC AA tournament. On the other hand, Mertacor does make the best total impressions predictions. Tau achieves relatively accurate predictions on all dimensions. Further analyzing individual agent behavior can lead to some insights about why performance varies across agents.

TacTex appears in more auctions than any other agent, and often receives a whole number average position, but is rarely in the first slot. Additionally, TacTex often drops out of the auction early, observing only a small amount of the day's total impressions. This means that TacTex is receiving daily summaries that are not particularly helpful for predicting total impressions—the agent's number of impressions typically constrains the space of feasible T values, but receiving only a few impressions does not provide a very tight lower bound. Additionally, agents that are in the auction for a longer period of time are more likely to have fractional average positions. A fractional average position together with a large total number of impressions often substantially limits the number of impressions that agent could have seen in each slot, which in turn constrains how many impressions the other agents could have seen in each slot.

However, it is worth noting that TacTex is at a disad-

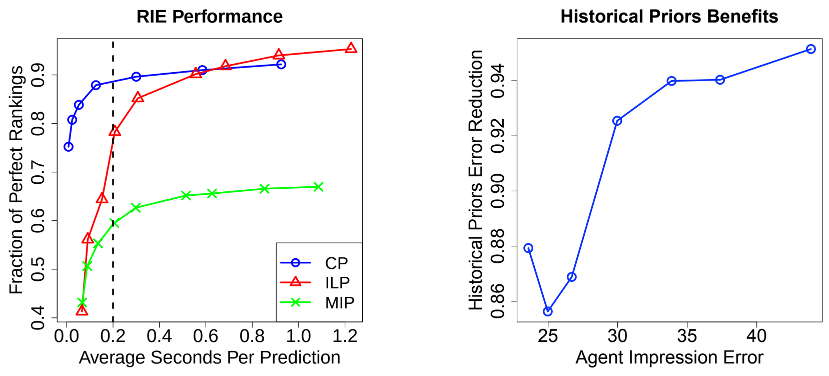


Figure 5: Left: Fraction of times the algorithms produce the correct rankings in the RIE problem. Right: Benefit from using historical priors with the CP algorithm. Due to the inherent feedback loop in historical priors, as agent impression error (with uniform priors) improves, historical priors become more effective.

Agent	Performance				Behavior				
	$\varepsilon(I_{j,k})$	$\varepsilon(T_j)$	$\varepsilon(T)$	rank	N	%F2	%int	%top	%seen
TacTex	18	39	60	.94	33686	69	71	12	26
Schlemazl	16	30	26	.88	14821	91	16	9	72
Mertacor	29	54	9	.83	8449	80	12	7	98
MetroClick	20	34	35	.89	14363	88	15	6	77
Nanda_AA	15	27	10	.89	12675	80	13	8	90
crocodileagent	17	27	29	.92	25451	89	17	9	67
tau	10	19	11	.92	19205	79	94	91	32
McCon	19	32	47	.95	29810	68	22	14	58

Table 1: Agent-specific prediction accuracy for the RIE problem with uniform priors using the CP algorithm. The best prediction of each type is shaded a light gray, while the worst prediction is shaded a dark gray. Descriptions of behavior columns, from left to right: number of predictions made, percentage of predictions made in the F2 focus level (a particular type of highly focused query in TAC AA), percentage of auctions where the agent had an integer average position, percentage of auctions where the agent was in the first slot, average percentage of impressions seen before the agent dropped out.

vantage in our prediction challenge because in it, agents are only required to make predictions when they actually participate in an auction, and TacTex participates in the most auctions. Seeing even a few impressions probably leads to better cascade predictions than seeing no impressions at all. Additionally, by dropping out of the auction early, TacTex causes other agents to have fractional average positions. While this improves cascade predictions overall, other agents are better able to capitalize because their own total impressions values are more informative. Even if TacTex's cascade predictions suffer when it drops out, its other models are likely improved because it sees cost information on a daily basis in a number of different queries.

Mertacor's strategy leads to notably different behavior than TacTex, as it appears in the smallest number of auctions. However, when it does appear in these auctions, it is frequently in them for the duration. This allows it to have nearly-perfect T predictions.

Tau never appears in F0 queries (not shown), which are the most general queries with the highest number of searches per day, and thus have the most potential for error. Additionally, we see that tau is very frequently in the first slot. The number of impressions seen by the agent in the first slot is often underconstrained, but tau does not run into problems making this prediction because its usually occupies this position itself.

The notable difference in performance of agents' cascade predictions brings up the issue of the coupled nature of predictions and decisions; specifically, the ability to make decisions not just to maximize profit given current predictions, but to improve predictions for future days. While some agents do take ad-hoc approaches to probing, explicitly calculating the value of information, as far as we know, is not typical of agents in TAC AA or the other TAC games. Our results suggest that such calculations could be important for improving agent performance. We leave this as a direction for future work.

6 Related Work

The problem of disaggregating average position data to predict advertiser rankings and impressions within the context of TAC AA was first studied by the TacTex team [13]. Their approach to the RIE problem utilized a tree-based search over three values for each advertiser: squashed bid rank, the order that spending limits were reached, and a GCD multiplier. The TacTex algorithm reduces the search space dramatically by searching over GCD multipliers and takes advantage of heuristics to further prune the space. Our paper is a natural extension of their work: we formalize the mathematical properties of cascades in a stylized model of GSPs and present an ILP based on these properties. We also present a CP solver that runs faster than our ILP. An additional contribution of our paper is a closer investigation of how an agent's average position and total number of impressions seen affect the accuracy of its cascade.

7 Conclusion

Recent work by Abhishek *et al.* [1] shows both analytically and empirically, with actual search engine data, that building models from average position data can lead to aggregation bias; for example, predicted click probabilities can exceed their actual values. Our work directly addresses the problems identified by these researchers, and could potentially be used to mitigate such bias.

In this paper, we develop algorithms for disaggregating ad auction data within a stylized version of the GSP: i.e., for unraveling an advertiser's daily average position into its actual positions throughout the day. We began by formulating and solving via dynamic programming a complete information version of the disaggregation problem. We then formalized the mathematical properties of cascades (i.e., solutions to this problem), and used the cascade properties to guide our design of a constraint programming algorithm and a mathematical programming model.

We show that (1) the ILP and CP are both well-suited to solving the rank and estimation problem, and the choice of which to use should depend on time constraints; (2) using historical priors on opponent impressions improved accuracy for the CP, despite a potential feedback loop; and (3) prediction accuracy differs depending on the agent's bidding behavior. This third point shows the potential importance of making decisions not just to maximize profit given current predictions, but also to improve predictions so that the inputs to the optimization routine are more accurate on subsequent days. As far as we know, no TAC AA agents make any explicit value of information calculations, though they might probe for information in an ad-hoc manner; our results suggest that this may be a fruitful direction for future research. Additional future research will investigate how the accuracy of our cascade predictions affect the accuracy of other predictions such as opponent bids and budgets.

TAC AA assumes a stylized model of generalized second price auctions, with more straightforward slot dynamics than most real-world ad auctions. For example, in TAC AA, the ads of top ranking advertisers are consistently displayed until those advertisers' budgets are exhausted; but some prominent Internet publishers do not show all the top ranking advertisers, because they attempt to control the rates at which advertisers spend their budgets. Hence, our algorithms would need to be extended before they could be used by advertisers in some important real-world ad auctions. Nonetheless, we believe that the present work is a necessary first step towards solving any real-world analogue of RIE, and hence potentially mitigating the aggregation bias reported by Abhishek *et al.* [1].

8 Acknowledgments

This research was supported by NSF Grant CCF–0905234 and the Centrum Wiskunde and Informatica (CWI).

References

[1] V. Abhishek, K. Hosanagar, and P.S. Fader. Identifying the Aggregation Bias in Sponsored Search Data. 2011.

[2] Gagan Aggarwal, Ashish Goel, and Rajeev Motwani. Truthful auctions for pricing search keywords. In *Proc. ACM EC*, pages 1–7, June 2006.

[3] S. Athey and G. Ellison. Position auctions with consumer search. *Working Paper*, 2007.

[4] Jordan Berg, Amy Greenwald, Victor Naroditskiy, and Eric Sodomka. A first approach to bidding in ad auctions. In *TADA '10: Workshop on Trading Agent Design and Analysis*, June 2010.

[5] Christian Borgs, Jennifer T. Chayes, Nicole Immorlica, Mohammad Mahdian, and Amin Saberi. Multi-unit auctions with budget-constrained bidders. In *Proc. ACM EC*, pages 44–51, 2005.

[6] Dynadec, Inc. Comet 2.1 User Manual. http://dynadec.com/, 2009.

[7] Benjamin Edelman, Michael Ostrovsky, and Michael Schwarz. Internet advertising and the generalized second price auction: Selling billions of dollars worth of keywords. NBER Working Paper No. 11765, 2005.

[8] A. Greenwald, V. Naroditskiy, and S. J. Lee. Roxybot-06: Stochastic prediction and optimization in TAC travel. *Artificial Intelligence Research*, 36:513–546, 2009.

[9] P.R. Jordan and M.P. Wellman. Designing an ad auctions game for the trading agent competition. In *Workshop on Trading Agent Design and Analysis*, July 2009.

[10] Brendan Kitts and Benjamin J. LeBlanc. Optimal bidding on keyword auctions. *Electronic Markets*, 14(3):186–201, 2004.

[11] Sébastien Lahaie and David M. Pennock. Revenue analysis of a family of ranking rules for keyword auctions. In *Proceedings of the 8th ACM conference on Electronic commerce*, EC '07, pages 50–56, New York, NY, USA, 2007. ACM.

[12] Aranyak Mehta, Amin Saberi, Umesh V. Vazirani, and Vijay V. Vazirani. Adwords and generalized on-line matching. In *Proc. FOCS*, pages 264–273, 2005.

[13] D. Pardoe, D. Chakraborty, and P. Stone. TacTex09: A champion bidding agent for ad auctions. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, 2010.

[14] David Pardoe and Peter Stone. The 2007 tac scm prediction challenge. In *AAAI 2008 Workshop on Trading Agent Design and Analysis*, 2008.

[15] Hal R. Varian. Position auctions. *International Journal of Industrial Organization*, 25(6):1163–1178, December 2007.

[16] Michael P. Wellman, Amy Greenwald, and Peter Stone. *Autonomous Bidding Agents: Strategies and Lessons from the Trading Agent Competition.* MIT Press, 2007.