Dynamic Segmentation for Large-Scale Marketing Optimization

Tyler Lu TL@CS.TORONTO.EDU
Department of Computer Science, University of Toronto
and Granata Decision Systems, Toronto

Craig Boutilier CEBLY@CS.TORONTO.EDU
Department of Computer Science, University of Toronto
and Granata Decision Systems, Toronto

Abstract

Marketing organizations often design multiple marketing campaigns to meet a variety of objectives. Marketing resources must be allocated carefully across campaigns to minimize conflicts and maximize organizational value (e.g., impact on customer lifetime value). While data-driven analytics has greatly improved the ability to predict the effect of marketing actions on individual customers, such fine-grained discrimination of customers has made multi-campaign optimization much more difficult. We propose a new *dynamic segmentation* approach for linear programming in large-scale, multi-campaign targeting optimization. Using on a novel form of column generation, our method produces segments of customers who are *provably* treated identically in the optimal solution of the underlying (unsegmented) problem. The resulting compression allows problems involving hundreds of campaigns and millions of customers to be solved optimally in fractions of a second. We also describe how the data-intensive components of the algorithm can be distributed to take advantage of modern cluster-computing frameworks.

1. Introduction

The marketing landscape has evolved dramatically over the past few years (Adobe Systems, 2013). Sophisticated marketers have increasingly come to rely on large amounts of data about consumer behavior, and advances in machine learning and predictive analytics, to increase the effectiveness of their marketing activities. Data might include: customer profiles; product ownership; transaction history; surveys; responses to past marketing approaches; behavioral data (e.g., browsing, mobile app usage); social media (e.g., blogs, reviews, tweets, Facebook likes); social network connections; and a variety of others. Coupled with advances in predictive analytics, machine learning and distributed data processing, these rich data sources enable extremely fine-grained predictions of customer behavior, ranging from propensity to act, to responses to marketing activities, to the expected value of such responses.

†Patent Pending.

Taking advantage of such insights requires some form of *marketing optimization* to allocate the resources required to undertake specific marketing activities. These optimizations are very complex when considering the delivery of *multiple* marketing campaigns. Large marketing organizations routinely run dozens or hundreds of simultaneous campaigns, each designed with a specific objective in mind. For example, a retail bank might run concurrent campaigns to to cross-sell new financial products to eligible customers; to reduce the risk of customer defection (or *churn*); and to improve brand awareness. Each campaign requires resources, including budget for its design and execution, and access to the marketing channels that are likely to be most effective given the nature of the campaigns and its targeted customers.

Most critically, each campaign places demands on *customer attention*. Sending multiple (perhaps conflicting, inconsistent) messages to a customer usually leads to lower response rates and negative impact on goodwill. To ensure customers are not inundated with multiple messages, it is essential to limit customer contacts. Doing so effectively requires the use of optimization to determine how best to allocate customer attention to specific campaigns.

These optimizations are often modeled using linear programs (LPs), mixed integers programs (MIPs), or other mathematical programming models. In this work, we focus on LP formulations. While state-of-the-art solvers scale polynomially with the number of decision variables, they typically cannot handle problems involving more than a handful of campaigns and channels, and a few million customers in practice. One practical approach to alleviating this bottleneck to create customer *segments* based on (observable, predicted, or derived) customer attributes. By treating all customers within a segment as indistinguishable, one can dramatically reduce the number of decision variables, making optimization feasible. Unfortunately, there is no principled way to segment the customer base *a priori* that will admit an optimal solution. Customers will be grouped for whom very different predictions of behavior, response, or value are made. This effectively means that one is losing the value derived from the sophisticated analytics used to produce these predictions in the first place. Unless this grouping is done in a way *that respects the ultimate objective of the optimization process*, the lost value can be significant.

In this work, we show how to get the best of both worlds. We develop a dynamic segmentation technique, called *dynamic cell abstraction (DCA)*, that creates a small set of customer segments (or *cells*) based on predictive models and marketing objectives. Our segmentations have a very small number of cells, rendering optimization tractable—indeed, able to support *real-time optimization*. At the same time, these cells are *provably optimal*; i.e., all customers in a cell would be treated the same even if we had the computational power to optimally personalize the approach to each *without segmentation*.

DCA essentially discovers the customer attributes that are *most relevant to solving the optimization problem*, allowing us to abstract away *irrelevant distinctions* in order to render the optimization problem tractable. Our procedure is based on a form of *column generation* a classic technique for tackling LPs and MIPs with large numbers of variables (Lübbecke & Desrosiers, 2005). However, because we need to both introduce multiple columns at each stage, and remove redundant columns, we must modify column generation scoring: we adapt the method introduced by (Walsh et al., 2010), in the context of online advertising, to our marketing optimization problem.

2. Problem Formulation

We begin by describing the basic marketing optimization problem tackled in this work.

Customers and Attributes. We assume a set of *customers* $\mathcal{S}$ who are potential targets for the marketing campaigns under consideration. Each customer is described by a set of features or *attributes* $\mathcal{A}$. We assume each attribute $A^j \in \mathcal{A}$ has finite domain $Dom(A^j) = \{a_1^j, \ldots, a_{n^j}^j\}$ of possible values.[2] Attributes describe specific features of customers that are used to predict customer behavior, value, or responses (such as those outlined above).[3]

[2]This is for ease of exposition only; our methods can easily be adapted to handle continuous features as we discuss below.

[3]We describe our methods assuming approaches are directed to a fixed set of *identifiable* customers. However, our methods can be applied directly to settings in which customers are unknown *a priori*, but the joint distribution of customer attributes is available (see (Walsh et al., 2010)).

Campaigns, Channels and Responses. We assume a set of *marketing campaigns* $\mathcal{C}$ designed to meet specific marketing objectives. Such campaigns may be intended to acquire new customers, cross-sell new products/services to existing customers, upsell new product features to existing customers, retain existing customers (overall or w.r.t. specific products), improve brand/product awareness, etc. A customer can be *targeted* or *approached* with zero or more campaigns. Each campaign must be delivered using a specific marketing *channel* (e.g., direct mail; email; inbound or outbound telemarketing; online or app-specific advertising, coupons, or offers; POS couponing or communication; broadcast media). We assume a set of channels $\mathcal{H}$.

Each campaign-channel pair $j \in \mathcal{C}, h \in \mathcal{H}$, if presented to customer $i \in \mathcal{S}$, induces a response r. Let $R_{j,h}$ denote the set of possible responses to campaign-channel pair. For example, if j is a campaign that offers an extension of the term for a discounted credit card interest-rate via a telemarketing call j, possible responses might be: no response; accepts offer, and churn probability over 12mo. horizon reduced by 25%; rejects offer/churn probability reduced by 15%; rejects offer/churn probability unaffected; rejects offer/drops credit card; rejects offer/leaves bank.

We model campaign and channel *costs* as follows. We assume a unit cost u_{jh} for campaign-channel pair $\langle j, h\rangle$; this reflects the cost of approaching a single customer with campaign j via channel h. Often this will depend on the channel and not the campaign itself. In certain cases, costs may depend on responses as well, but we capture this below in our *value models*.[4] Finally, we assume a capacity limit L_h for channel h, restricting how many approaches (across campaigns and customers) can be made using channel h over the time period in question.

Predictive Models, Customer Values. We assume a set of *predictive models* used to predict customer behaviors, campaign responses, and value to the marketer. A *response model* provides a probabilistic prediction $p_{ijh}(r)$ of customer i's response to (campaign-channel) approach $\langle j, h\rangle$. A *value model* $v_{ijh}(r)$ predicts the value of i's response r to approach $\langle j, h\rangle$. The *expected value* of $\langle j, h\rangle$ to i is:

$$v_{ijh} = \sum_{r \in R_{j,h}} v_{ijh}(r) p_{ijh}(r).$$

Generally, predictive models are learned from data, and depend only on the values of a (possibly small) set of customer attributes specific to a campaign and channel. Note that the v_{ijh} terms can be computed directly from these models. Such models are sometimes further decomposed

[4]Fixed costs f_j for campaign j, incurred if j is executed (i.e., delivered one or more customers), are discussed below.

into more fine-grained models that predict specific customer behaviors that must be aggregated to determine comprehensive value and response models. For instance, i's response probability for $\langle j,h\rangle$ may be decomposed into a *channel propensity probability* (i.e., probability i is reachable via channel h) and a "received offer" response model (i.e., odds of response conditional on successful contact). Similarly, value may be determined as a function of several "joint responses;" e.g., if campaigns are measured using impact on customer lifetime value (LTV), then the value of an approach may depend on the impact it has on immediate sales, as well as the impact it has on churn prevention.

We assume that approaches do not *interfere* with one another: if i is receives multiple approaches, her response to each is independent of the others.[5]

Comparing Different Objectives. While campaigns are often designed with different objectives in mind, their impact on customer value must be phrased in terms of some "common numeraire" against which they can all be measured. This numeraire can be flexible and can even account for weighted combinations of specific subobjectives corresponding to the aims of different campaigns. For instance, consider campaigns designed for cross-sell, upsell, and customer retention in financial services. Cross-sell and upsell campaigns are often comparable using the revenue generated by any net new sales; or LTV over some horizon might be appropriate. Customer retention campaigns usually have a different objective—preventing customers from dropping specific products, or dropping the firm entirely. In this case, we might measure effectiveness using the (expected) number of customers that a campaign prevents from defecting. However, even comparing two retention campaigns is problematic, e.g., if each is designed for a different service—how can one compare the value of a prevented defection of one product vs. the other? One obvious metric for comparison is again impact on revenue or LTV. Similarly, cross-sell and upsell campaigns may have ancillary benefits w.r.t. customer retention. These too can be measured by focusing not on only on the new revenue from new product sales, but on the *total* effect on LTV, which can serve as a common numeraire.

It may be difficult to assess the value of brand exposure, or retaining a customer, in a way that is commensurate with a numeraire like LTV. We describe below how *scenario analysis* can be used without committing to precise tradeoff values. Our model readily supports eligibility and opt-out constraints as well.

Optimization Models. We describe a straightforward LP formulation of the campaign optimization problem without fixed costs. We assume a *contact limit* L (i.e., at most L approaches per customer);[6] channel capacities (or limits) L_h on the usage of each channel h; campaign budgets B_j limiting the cost incurred by campaign $j \in \mathcal{C}$; a global budget B limiting total cost; and lead limits L_j restricting the number of customers that can be contacted by $j \in \mathcal{C}$. Let binary variable x_{ijh} denote that customer i receives approach $\langle j,h\rangle$. We can solve the optimization using the MIP:

$$\max_{x_{ijh}} \quad \sum_{i,j,h} x_{ijh}(v_{ijh} - u_{jh}) \tag{1}$$

$$\text{s.t.} \quad \sum_{j,h} x_{ijh} \le L \qquad \forall i \in \mathcal{S} \tag{2}$$

$$\sum_{i,h} x_{ijh}u_{jh} \le B_j \qquad \forall j \in \mathcal{C} \tag{3}$$

$$\sum_{i,j,h} x_{ijh}u_{jh} \le B \tag{4}$$

$$\sum_{i,h} x_{ijh} \le L_j \qquad \forall j \in \mathcal{C} \tag{5}$$

$$\sum_{i,j} x_{ijh} \le L_h \qquad \forall h \in \mathcal{H} \tag{6}$$

$$x_{ijh} \in \{0,1\} \qquad \forall i \in \mathcal{S}, j \in \mathcal{C}, h \in \mathcal{H} \tag{7}$$

It is not hard to show that the variables x_{ijh} can be relaxed so the problem can be solved as an LP (setting $x_{ijh} \in [0,1]$). Other constraints can be accommodated (e.g., limiting customers to one contact per campaign).

If we include fixed costs f_j for using a campaign j (or channel h), then we must include (non-relaxable) integer variables: a 0-1-indicator variable I_j for each $j \in \mathcal{C}$ indicating if campaign j has been delivered to any customers.[7]

3. Dynamic Segmentation

A key difficulty with LP (and MIP) approaches to the multi-campaign, multi-channel optimization models above is the sheer number of decision variables: the x_{ijh} variables grow with the product $|\mathcal{S}||\mathcal{C}||\mathcal{H}|$ of the number of customers, campaigns and channels.[8] Fixed costs exacerbate the problems by introducing integer variables.

We now describe a *dynamic segmentation* method that segments the the customer population into *cells* of various

[5]Such effects can be modeled using, e.g., random utility or stochastic choice models (Shepard, 1959; Luce, 1959; Louviere et al., 2000), which can require more complicated formulations due to (a) the combinatorics of choosing *sets of offers*; and (b) the typically nonlinear nature of these effects. We leave the integration of such models into our DCA framework to future work.

[6]The contact limit need not be constant, but can vary with customer attributes, channel or campaign.

[7]Our methods can be applied to the LP relaxation of this MIP, providing "approximate" cells for the MIP; we leave evaluation of this approach to an extended paper.

[8]Eligibility restrictions reduce this number: we can eliminate any x_{ijh} where i is ineligible for j or h. But even if customers are eligible for a constant fraction of approaches, this offers only a constant factor reduction in decision variables.

sizes, and optimizes the approach for each cell rather than for individual customers. If the number of cells is kept small, scalability is no longer an issue. At the same time, unless cells are crafted carefully, significant value may be sacrificed. Our technique, *dynamic cell abstraction (DCA)*, provides an *anytime approach* that gradually refines cells and is guaranteed to converge to a set of cells such that no value is lost. Furthermore, in practice, it finds very high value or optimal solutions with a very small number of cells by finding *just those customer distinctions required to make optimal decisions* and nothing more.

Customer Segments. DCA creates a set of abstract *customer cells* that distinguish customers based on one or more attributes. These may include observable attributes such as those discussed above, or the values associated with customers by specific predictive models. Here we focus on cells created using derived values.

We discuss *dynamic cell generation* below, but first detail how the optimization model exploits such cells. Suppose we partition the customer population $\mathcal{S}$ into a set of K covering and exclusive *cells* $\mathcal{S} = \cup_{k \leq K} \mathcal{S}_k$, s.t. $\mathcal{S}_k \cap \mathcal{S}_{k'} = \emptyset$ for any $k \neq k'$. We call such a partitioning a *segmentation*. Let $z_k = |\mathcal{S}_k|$ denote the size of cell k. For the purposes of optimization, we treat all customers in a cell as if they have the same *expected value* to any approach $\langle j, h \rangle$, namely, the average value across all customers in the cell, i.e., as if we randomly picked a customer s_k from that cell to approach. Letting s_k be a generic customer from cell $\mathcal{S}_k$, we define:

$$v_{jh}^k = v_{jh}(s_k) = \frac{1}{z_k} \sum_{i \in \mathcal{S}_k} v_{ijh}.$$

Approximate Optimization with Segments. Targeting customer segments rather than individual customers simply requires that we replace the targeting variables x_{ijh} for individual customers in LP (1) with variables x_{jh}^k corresponding to specific cells: here $x_{jh}^k \in [0, 1]$ is the fraction of customers in cell k that receive approach $\langle j, h \rangle$:

$$\max_{x_{jh}^k} \quad \sum_{k,j,h} x_{jh}^k z_k (v_{jh}^k - u_{jh}) \tag{8}$$

$$\text{s.t.} \quad \sum_{j,h} x_{jh}^k \leq L \qquad \forall k \leq K \tag{9}$$

$$\sum_{k,h} x_{jh}^k u_{jh} \leq B_j \qquad \forall j \in \mathcal{C} \tag{10}$$

$$\sum_{k,j,h} x_{jh}^k u_{jh} \leq B \tag{11}$$

$$\sum_{k,h} x_{jh}^k \leq L_j \qquad \forall j \in \mathcal{C} \tag{12}$$

$$\sum_{k,j} x_{jh}^k \leq L_h \qquad \forall h \in \mathcal{H} \tag{13}$$

$$x_{jh}^k \in [0, 1]. \qquad \forall i \in \mathcal{S}, j \in \mathcal{C}, h \in \mathcal{H} \tag{14}$$

The optimization LP (8) assumes that each approach assigned to a cell k will attain the expected value of that approach, taking expectation over all customer $i \in k$, i.e., assuming each approach will be assigned uniformly at random to some i. Obviously, this random assignment is feasible and by linearity of expectation attains the objective value of the LP in expectation.[9]

Notice however that the abstract LP *underestimates* the value attainable by the assignment given by assuming a random allocation. If approach $\langle j, h \rangle$ is assigned m customers from cell k where m is significantly less than the cell size z_k, the allocation could be realized using customers $i \in \mathcal{S}_k$ that have higher value than the mean. With multiple approaches assigned to the same cell, the optimal assignment (i.e., the optimal packing) may have much greater than the mean value for each approach. As a simple example, suppose that the values of two approaches $\langle j, h \rangle$ and $\langle j', h' \rangle$ are perfectly negatively correlated on a cell k (i.e., any $i \in \mathcal{S}_k$ with high value for $\langle j, h \rangle$ has low value for $\langle j', h' \rangle$ and vice versa). Even if the two approaches consume the entire capacity of k, we can pack them so that each gets only high value customers. Based on this insight we recognize that we often can improve the objective value of the LP by splitting certain cells. In the example above, splitting cell k into two new cells k' and k'' so that high value customers for approach $\langle j, h \rangle$ fall into one cell and high value customers for approach $\langle j', h' \rangle$ fall into the other will offer dramatic improvements in objective value.

Dynamic Segment Generation. To discover useful cell splits reflecting the intuitions above, we adapt the method of (Walsh et al., 2010), originally developed in the context of display advertising. It is based on the well-known *column generation* technique. Our general approach is to use the *reduced costs* produced in the solution of the abstract LP to estimate the value of splitting a cell. Column generation is useful for LPs with large numbers of variables: since only a small subset of the variables are active in the optimal solution, the goal is to solve the problem using only a few variables, to iteratively estimate which variables are active, and gradually add them to the LP, resolving a slightly larger LP at each iteration.

Each iteration in the standard column generation procedure proceeds as follows: suppose we solve an LP with

[9]Two small caveats: First, if contact limit $L > 1$, the random assignment must be over all L-tuples of approaches that have a positive assignment to k. For instance, if $L = 2$ and three approaches a, b, c are assigned fractions p_a, p_b, p_c (resp.), then the random assignment of the approach *pair* (a, b) to fraction $p_a \cdot p_b$ of cell k (similarly for (a, c) and (b, c)) ensures satisfaction of the contact limit constraint while attaining the LP objective value in expectation. Second, if allocated fractions are non-integral, small rounding errors may arise; but since our cell sizes tend to be in the many thousands, these are negligible.

only a subset of its variables (but assume all constraints are present). This corresponds to solving an LP with the columns (in the objective vector and constraint matrix) for the missing variables deleted. Equivalently, we can view this as a basic feasible solution of the LP with all missing variables treated as nonbasic (i.e., set to zero), much like in simplex. In column generation we assess the value of adding a new variable to the reduced LP much like we would the value of entering a non-basic variable into the basis in simplex. Specifically, define the *reduced cost* of a (missing) variable x to be:

$$rc(x) = v_x - \mathbf{c}\boldsymbol{\pi}$$

where v_x is the objective function coefficient associated with x in the (unreduced) LP, $\mathbf{c}$ is the column vector of constraint coefficients associated with x in the (unreduced) LP, and $\boldsymbol{\pi}$ is the vector of dual variables (or shadow prices) at the optimal solution of the *reduced* LP. The reduced cost $rc(x)$ corresponds to the marginal increase in objective value per unit increase in (nonbasic variable) x if one were to add x to the LP.

In column generation, we typically add the variable that has maximum reduced cost. Solving this *pricing subproblem* requires some intelligence and exploitation of problem structure to evaluate which variable has maximum reduced cost from a large (often exponential or infinite) set of choices, and is often formulated as a subsidiary optimization problem. Note also that if all columns have nonpositive reduced costs, then the solution to the reduced LP is in fact the optimal solution to the full LP; hence this approach can be used to prove the optimality of the reduced/relaxed solution.

In our setting, when we split a cell k into two new cells k_1 and k_2, we are not adding a single variable to the LP. Rather we are: (a) adding a *set* of variables to the LP, all those of the form $x^{k_1}_{jh}$ and $x^{k_2}_{jh}$; and (b) removing a set of variables, all those of the form x^{k}_{jh}. Removing variables doesn't (negatively) impact the LP, since every assignment that can be accomplished with cell k can also be accomplished with cells k_1 and k_2. But adding these new variables is somewhat problematic because certain constraints are not present in the reduced LP. Specifically, the "supply constraint" Eq. (9) associated with the new cells k_1 and k_2 are not present in the reduced LP. Since we haven't explicitly modeled these two new cells, we do not have dual variables associated with their constraints, and we don't have columns for the new k_1 and k_2 allocation variables. This makes pricing of these columns difficult. However, following (Walsh et al., 2010), we can prove that the solution to the reduced LP is also an optimal solution to the LP that is obtained by adding the supply constraints for k_1 and k_2 to the reduced LP (assuming one maintains the old x^{k}_{jh} variables, but does not introduce the new $x^{k_1}_{jh}$ and $x^{k_2}_{jh}$ variables). Furthermore, we can show that the corresponding dual variables π_{k_1} and π_{k_2} are zero. As a result, we can accurately score a column corresponding to a new split cell k_1 or k_2 using only the shadow prices produced by the original reduced LP.

More precisely, suppose cell k' is a descendent (one side of the split) of a cell k that is under consideration. We define the reduced cost of the variable $x^{k'}_{jh}$ to be:

$$rc(x^{k'}_{jh}) = v^{k'}_{jh} - \mathbf{c}^{k'}_{jh}\boldsymbol{\pi} \tag{15}$$

where $\mathbf{c}^{k'}_{jh}$ is the column vector in the constraint matrix corresponding to $x^{k'}_{jh}$, and $\boldsymbol{\pi}$ is the vector of dual variables (shadow prices) in the reduced LP. The non-zero terms in $\mathbf{c}^{k'}_{jh}\boldsymbol{\pi}$ are the following (assuming formulation LP (8)):

- The supply constraint corresponding to cell k, with dual variable value π_k and coefficient 1 for $x^{k'}_{jh}$. This is the only supply constraint in which $x^{k'}_{jh}$ participates.
- The budget constraint for every campaign $j \in \mathcal{C}$, with dual variable value π_{B_j} and coefficient u_{jh} for $x^{k'}_{jh}$.
- The global budget constraint with dual variable value π_B and coefficient u_{jh} for $x^{k'}_{jh}$.
- The lead constraint for each campaign $j \in \mathcal{C}$, with dual variable value π_{L_j} and coefficient 1 for $x^{k'}_{jh}$.
- The capacity constraint for each channel $h \in \mathcal{H}$, with dual value π_{L_h} and coefficient 1 for $x^{k'}_{jh}$.

Taken together, we have:

$$rc(x^{k'}_{jh}) = v^{k'}_{jh} - \pi_k - u_{jh}\pi_{B_j} - u_{jh}\pi_B - \pi_{L_j} - \pi_{L_h}. \tag{16}$$

Finally, we need to consider how to score the actual *split* of a cell k into k_1 and k_2, given that it introduces a number of columns which replace a number of others. Suppose we ignore budget, lead and capacity constraints, and focus on the simple problem that uses only supply constraints. If we split k into k_1 and k_2, all customers in new cell k_1 will go to the approach $\langle j^*, h^*\rangle$ that has highest expected value $v^{k_1}_{j^*h^*}$ for that new cell. Therefore the true improvement in expected value will in fact be $rc(x^{k_1}_{j^*h^*})z_{k_1}$. Thus we score splits as follows:

$$score(k, k_1, k_2) = \max_{j\in\mathcal{C},h\in\mathcal{H}}\{rc(x^{k_1}_{jh})z_{k_1}\} + \max_{j\in\mathcal{C},h\in\mathcal{H}}\{rc(x^{k_2}_{jh})z_{k_2}\}. \tag{17}$$

Searching for Splits. Apart from evaluating splits, we require methods to search through the space of potential splits, scoring these splits to determine which ones to adopt. An ideal "split language" is compact enough to allow all splits to be effectively evaluated during optimization, while also offering the expressiveness and flexibility to find near-optimal solutions with relatively few cells.

In this work, we focus on splits over *derived customer values*.[10] Specifically, assume only that we have the derived values v_{ijh} associated with each customer i and approach (j, h), provided in unstructured form (null values, e.g., denoting ineligibility, are permitted). We treat each approach (j, h) as a real-valued *feature* over the set of customers $\mathcal{S}$. Splits are defined over these features. We limit possible splits to *quantiles* for each such feature. Focusing on binary splits,[11] given a cell k, it can be split into a pair of cells k^1 and k^2, where $k^1 = \{i \in k : v_{ijh} \leq \tau\}$ and $\tau = v_{i'jh}$ for some $i' \in k$ and approach (j, h); and $k^2 = k \setminus k^1$. Given a cell of size n, there are $(n-1)|\mathcal{C}||\mathcal{H}|$ splits to evaluate, far too many to be practical for cells of significant size. For this reason, in practice we restrict splits a small number of quantiles. For instance, we may permit splitting only at the nine decile boundaries, 10%, 20%, …, 90%); or we may consider a single split point per feature (e.g., the median).

If we allow m splits per cell-approach pair, the evaluation of a cell's splits requires computing the reduced cost score Eq. 17 for $m|\mathcal{C}||\mathcal{H}|$ splits. If the DCA algorithm currently has c cells, then determining the split with maximum score requires $O(cm|\mathcal{C}||\mathcal{H}|)$ time, subject to two key assumptions: score computation takes constant time; and the appropriate split points are computable in constant time. Neither of these hold in practice. Score computation for a cell split on v_{ijh} requires computing the mean of v_{ijh} (over i) in the two new cells—this scales linearly with the size of the cell. Computing quantiles of a large data set can also be accomplished in linear time (and approximated readily). While linear-time computation is viable in some settings, for large customer sets this may not be acceptable. Fortunately, computation of these statistics can be distributed to exploit modern cluster-computing frameworks.

Distributed Implementation. Let a "customer record" for $i \in \mathcal{S}$ refer to any relevant customer data together with values v_{ijh} for all approaches (j, h). We assume that customer records contain *cell indicators* which evolve during DCA. To distribute the computation of quantiles within and cell mean values v^k_{jh}, we partition customer records across M compute nodes in a cluster computing framework, with (roughly) $|\mathcal{S}|/M$ records per node. The number of compute nodes can be chosen to meet specific performance demands (e.g., to allow all data to fit and be processed in memory).

[10]If predictive models and direct customer attributes are provided, one can search for splits over this attribute space directly. This can be especially effective if predictive models exhibit structure (e.g., each model uses only a small number of customer attributes). However, we will often not have access to the models themselves (e.g., for privacy or proprietary reasons). Instead we may have only the actual response, value, or other predictions; or perhaps just the derived values for each customer. In this work, we assume only access to the latter.

[11]Multiway splits can be realized by a sequence of binary splits.

Computing the mean value v^k_{jh} of approach (j, h) for cell k can be accomplished in $O(|\mathcal{S}|/M)$ time, by passing sum and count data from each compute node to the master node (which coordinates DCA and solves the optimization). Computing quantiles in a distributed fashion is somewhat less straightforward, but various methods for approximating quantiles in distributed settings (including sampling-based methods) can be used (Shrivastava et al., 2004). Note that computing exact quantiles is not essential to the performance of DCA; even crude approximations perform very well (as we see below).

The most computationally intensive aspect of DCA is not optimization itself—the LPs generated by DCA will remain very small. Rather it is the data processing required to compute quantiles and means within cells. The fact that these operations are readily distributable means that practical scalability can be achieved by the addition of commoditized computing resources/compute nodes.[12] We have carefully designed DCA to work in a distributed environment so that the algorithm can scale with problem size, both in terms of computation time and memory. We use *Apache Spark* (Zaharia et al., 2010), a general-purpose cluster computing engine, on top of Hadoop's distributed filesystem containing customer records. DCA is implemented using as a series of map-reduce operations that allow the parallel computation of customer values and their aggregate statistics for the relevant cells. Exploration of the DCA tree and reduced-cost computation for the splits is managed using mean value computations, and caching of these results. Using Spark allows us to cache the repeatedly accessed customer values and avoid both redundant re-computation and persistence to disk, improving performance w.r.t. systems like Hadoop MapReduce.

4. Scenario Analysis and Refinement

Our empirical results below suggest that DCA offers very effective optimization scaling. Indeed, it can find near-optimal solutions with remarkably few, well-chosen cells. Since LP solution times scale with the number of cells, optimization times are extremely fast. However, the iterative DCA process remains data-intensive, even with large-scale parallelization. This can be a concern when decision makers want to explore different settings of the optimization parameters. For instance, one may want to explore the impact of varying global or campaign-specific budget levels, lead limits, eligibility criteria, or even the effect of varying the LP objective. In such a case, we would like to avoid re-running DCA if possible; but the cells constructed for one LP may not support the solution of another.

[12]Data can also be distributed "horizontally" by distributing data for different (sets of) approaches.

To handle this issue, we augment DCA so that we don't solve a single LP during DCA, but instead solve multiple "representative" LPs. For instance, if we anticipate the need to optimize at a variety of budget levels, we can solve an LP for each possible budget level, or for several representative levels that "span" the anticipated range. We then use *the solutions of these multiple LPs* to assess the value of a split. Specifically, we score each split for each of the LPs using the usual method above. But since our goal is to find cells that support the solution of all LPs, we sum these scores to determine the overall score of a split.[13] This tends to create more cells; but once they are created, they support the optimization of a variety of different LPs.

If the solution of a new LP created during scenario analysis falls "out of range", the existing set of cells can be used as a starting point for *DCA refinement*: we refine the existing abstraction to accommodate the new LP. If the set of representative LPs provide any guidance for the new LP, we expect the number of refinement steps needed to solve the new LP to be few. Furthermore, the computational cost of cell splitting drops dramatically with cell size (as we show below). This means that additional iterations of DCA needed to refine a pre-existing abstraction will be much faster than the original iterations.

5. Empirical Analysis

We now describe experiments that illustrate the performance and value of DCA. All experiments are run on customer data sets of various sizes (250K, 500K, 750K, 1M, 2M, 4M, 6M, and 10M) using a set of 100 possible approaches (e.g., 20 campaigns and 5 channels, or 10/10). Data are generated randomly using a simulator developed using real marketing campaign predictive/value data. All DCA splits are binary and are restricted to the median (quantile 0.5) of any cell.[14] All optimizations use a state of the art, commercial LP solver on a high performance platform (dual Intel 2.6GHz processors, 244 Gb of RAM).

Fig. 1 shows the quality of the solution reached as a function of the number of DCA iterations (limited to 101 iterations)—since we only consider binary splits, the number of cells is the number of iterations (the first iteration processes the initial cell $\mathcal{S}$). For customer sizes under two million, solution quality is shown as a percentage of the optimal LP value (computed by solving the exact LP model Eq. 1 as discussed below). For the 2–10M instances, relative solution quality (w.r.t. the final value obtained at iteration 101) is shown only since the optimal benchmark is un-

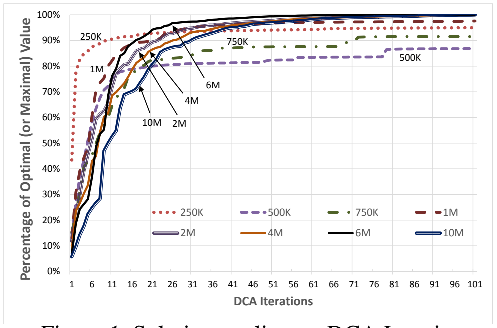

Figure 1. Solution quality vs. DCA Iterations.

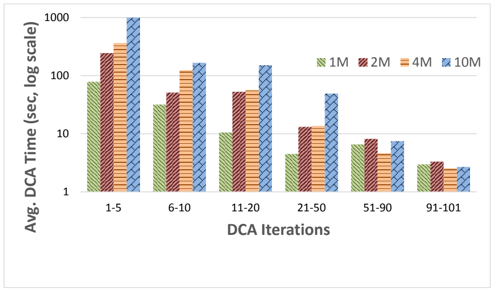

Figure 2. Avg. Time per DCA Iteration (binned, log scale).

available. The anytime profile is very similar to the smaller instances, suggesting convergence to a near-optimal result. We see that DCA is able to dynamically segment customers so that very few cells are required to achieve near-optimal results. This is true even with the extremely limited set of median-splits that we allow. Total run time (in seconds) for DCA on a 10-node cluster for different instance sizes is shown in the first row of the following table:

Size	.25M	.5M	.75M	1M	2M	4M	6M	10M
DCA time (s)	611	679	847	1437	3072	3636	6971	9129
DCA opt (s)	0.022	0.016	0.015	0.012	0.018	0.016	0.019	0.015
LP opt (s)	163	2458	3412	4434	–	–	–	–

Fig. 2 shows the average time to complete an iteration of DCA for various customer size ranges (we focus on larger problems only), binned by varying ranges of iterations. Not surprisingly, later iterations of DCA require significantly less time, since cell sizes decrease exponentially with iteration; indeed (noting the log-scale) we see that iteration times exhibit an exponential decrease. As discussed above, this is a critical property for the real-time cell refinement that supports scenario analysis. Indeed, the final five iterations require only a few seconds on average.

We tested a "typical" customer-attribute-based segmentation on the 1M-customer instance, segmenting using two sets of available customer attributes. The first segmentation gives a set of 738 cells; the second is a strict refinement of the first with 26198 cells. The first gives a solution that is only 51% of optimal (soln. time 0.1s.); the second is only 72.5% of optimal (4.3s.). This compares to the 97.6%-

[13]Other combinators can be used, e.g., weighted sums to account for the fact that some LPs have naturally higher objective values than others, or the maximum of the scores.

[14]Allowing a richer set of splits will provide better results with fewer cells, at the cost of further evaluation of the "cell tree".

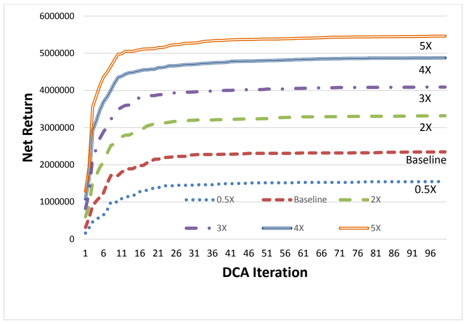

Figure 3. Solution Quality of Multiple LPs.

optimality attained by DCA with only 100 cells.

We note that DCA scales effectively with the number of compute nodes used to assess relevant cell statistics. For the 10M instance, the following table shows how total DCA time (101 iterations) reduces with the number of nodes. We see compute time reduces nearly linearly with the number of nodes, with a small 7–8% overhead.

Nodes	10	20	40
DCA time (min)	152.2 (100%)	87.2 (57.3%)	50.2 (33.0%)

One of the most important metrics is the time to solve the marketing optimization problems using the cells generated by DCA. Because the number of cells needed to support high-quality solutions is so small, optimization time is essentially negligible, on the order of hundredths of a second (see table above). By contrast the solution of the complete unabstracted LP was only feasible for problems of up to size 1M—beyond that memory limits are reached despite the high-performance platform used. The 1M instance took about 74min to solve. Indeed, with the exception of the 250K instance, unabstracted LP solution times are significantly longer than the time taken to run the *entire DCA process*. Of course, once DCA is run, the cells can be used for multiple optimizations; hence the overhead of DCA can be amortized over the analysis of multiple scenarios. We turn our attention to this now.

To test the ability to support scenario analysis, we applied DCA to the simultaneous solution of 1M-customer instances using six different LPs; the base level LP is the one used above, while five additional LPs were solved with different lead limits—0.5, 2, 3, 4 and 5 times the base level—with both global and campaign-specific limits adjusted by the same factor. Note that *a single collection of cells* is produced to solve all six optimization problems. Fig. 3 illustrates the change in the objective value of each LP as the number of cells increases. We see that a set of cells similar in size to that produced for the single original LP does well for all six. The objective value obtained using the cells produced by DCA are near-optimal for all six LPs: 96.0% (0.5X); 96.7% (Baseline); 98.0% (2X); 98.2% (3X); 98.5% (4X); and 98.5% (5X). The slight bias towards splits that favor the LPs with larger limits is due to the fact that we use an *unweighted* sum of objective values (since the larger limits offer greater total return on marketing spend). The difference in DCA time/iteration w.r.t. solving a *single* LP is negligible.

Finally, to the robustness of these cells, we used them to compute solutions for *new lead limits* different than those used to create the cells. With limits of 2.5X and 4.5X, the solutions produced using the cells above were 98.6% and 98.4% optimal, respectively. While a rather minimal test, this does suggest that cells produced for a well-chosen set LPs can generalize to new problems well.

6. Concluding Remarks

We have developed DCA, a dynamic segmentation method for marketing optimization problems that allows for the optimal allocation of marketing resources—including budget, channel access and customer attention—in problems significantly larger than those that can be addressed using standard models, and with solution quality guarantees that cannot generally be offered by techniques that use manual or purely statistical segmentation. Apart from producing optimal or near-optimal results, the data-intensive computation required can be distributed effectively to exploit modern parallel/cluster computing frameworks, ensuring extreme scalability. DCA also offers strong anytime performance, and can be used to support real-time scenario analysis. Future directions include incorporating additional contact-response models (e.g., sequential contacts, slates of offers that influence one another), and analyzing performance on MIP relaxations (e.g., using branch-and-price).

References

Adobe Systems, Inc. Digital distress: What keeps marketers up at night? 2013.

Louviere, J. J., Hensher, D. A., and Swait, J. D. *Stated Choice Methods: Analysis and Application*. Cambridge, 2000.

Lübbecke, M. E. and Desrosiers, J. Selected topics in column generation. *Operations Research*, 53(6):1007–1023, 2005.

Luce, R. D. *Individual Choice Behavior*. Wiley, 1959.

Shepard, R. N. Stimulus and response generalization: A stochastic model relating generalization to distance in psychological space. *Psychometrika*, 22(4):325–345, 1959.

Shrivastava, N., Buragohain, C., Agrawal, D., and Suri, S. Medians and beyond: new aggregation techniques for sensor networks. *2nd Intl. Conf. on Embedded Networked Sensor Systems (SenSys-04)*, pp.239–249, Baltimore, 2004.

Walsh, W. E., Boutilier, C., Sandholm, T., Shields, R., Nemhauser, G., and Parkes, D. C. Automated channel abstraction for advertising auctions. *24th AAAI Conf. on Art. Intel. (AAAI-10)*, pp.887–894, Atlanta, 2010.

Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., and Stoica, I. Spark: Cluster computing with working sets. *2nd USENIX HotCloud Workshop*, pp.1–7, Boston, 2010.