ECLIPSE: An Extreme-Scale Linear Program Solver for Web-Applications

Kinjal Basu [*1] **Amol Ghoting** [*1] **Rahul Mazumder** [*2] **Yao Pan** [*1]

Abstract

Key problems arising in web applications (with millions of users and thousands of items) can be formulated as Linear Programs (LP) involving billions to trillions of decision variables and constraints. Despite the appeal of LP formulations, solving problems at these scales appears to be well beyond the capabilities of existing LP solvers. Often ad-hoc decomposition rules are used to approximately solve these LPs, which have limited optimality guarantees and lead to a sub-optimal performance in practice. In this work, we propose a distributed solver that solves a perturbation of the LP problems at scale via a gradient-based algorithm on the smooth dual of the perturbed LP with computational guarantees. The main workhorses of our algorithm are distributed matrix-vector multiplications (with load balancing) and efficient projection operations on distributed machines. Experiments on real-world data show that our proposed LP solver, ECLIPSE, can solve problems with 10^{12} decision variables – well beyond the capabilities of current solvers.

1. Introduction

Large-Scale Linear Programs arise naturally in several applications across various scientific and industrial disciplines. In this paper, we focus on problems arising in the internet industry. Many key problems in this domain, can be formulated via a Linear Program (LP). Examples include optimizing the volume of emails to be sent to users (Gupta et al., 2016; 2017), balancing multiple business metrics in a ranking or recommendation system (Agarwal et al., 2012; 2014; 2015), optimizing internal promotions (Gupta et al., 2019); and matching entities (Azevedo & Weyl, 2016) such as riders to drivers in ride-sharing apps (Zheng & Wu, 2017), items to buyers in an e-commerce platform (Linden et al., 2003; Schafer et al., 1999), jobs to potential job-seekers (Borisyuk et al., 2017; Kenthapadi et al., 2017), restaurants to users, etc.

While most of these problems can be formulated as an LP, solving them becomes difficult as the scale of the problem increases. Consider, for instance, the task of matching items to users—the scale of the problem can easily range from hundreds of millions to trillions of variables—they are well beyond the capabilities of generic LP solvers[1]. One common approach used in web-applications, is to translate the primal problem to a dual problem, solve the dual problem and derive the primal solution as a function of the dual variables (Agarwal et al., 2011; 2012). Due to the scalability limitations of current generic LP solvers, solving the dual problem is also extremely challenging.

Some ad-hoc techniques are often used in practice resulting in sub-optimal solutions. One approach is to split the problem into several smaller problems, which makes the system sub-optimal. Another approach is to not solve the optimization problem at all, but rely on the hyperparameter tuning to estimate these dual variables in order to maximize business metrics (Agarwal et al., 2018; Letham et al., 2019; Du et al., 2019). Although few specific problems can be framed as above, tuning multiple dual variables through an online hyperparameter tuning problem is an incredibly difficult task (Snoek et al., 2012; Letham & Bakshy, 2019).

In this paper, we introduce ECLIPSE: An **E**xtreme **Sc**ale **Li**near **P**rogam **S**olv**er**, which is a perturbation based approach (Mangasarian & Meyer, 1979) to solve the LPs at an unprecedented scale. Our approach leads to a dual that has Lipschitz continuous gradient (Nesterov, 2005). We present gradient-based methods on the dual – with guaranteed convergence properties. This allows for a distributed implementation that can tackle several challenging problems in the internet industry. We compare ECLIPSE with some of the other large-scale solvers using simulated and real data. Using this system in production for solving extreme-scale problems we have observed significant improvements in metrics.

The efficiency of the method introduced in this paper, de-

[*]Equal contribution [1]LinkedIn Corporation [2]MIT. Correspondence to: Kinjal Basu <kbasu@linkedin.com>, Rahul Mazumder <rahulmaz@mit.edu>.

Proceedings of the 37^{th} International Conference on Machine Learning, Vienna, Austria, PMLR 119, 2020.

[1]Traditional LP solvers (roughly speaking) are based on simplex or interior point methods. They become prohibitively expensive for large scale instances. This necessitates the use of specialized solvers that can exploit problem-structure.

pends on the special structure of the problem. However, this special structure is abundant in real-life problems especially in the web-applications which also brings out the need for an extreme scale, and that is the specific problem we wish to solve in this paper.

The rest of the paper is organized as follows. We begin by describing the problem formulation and traditional solutions in Section 2. Our proposed solution with theoretical guarantees is shown in Section 3. In Section 4 we describe several applications and Section 5 discusses the system architecture. Experimental results are shown in Section 6 before concluding in Section 7.

2. Problem Setup

To describe the problem setup, let us first begin with some notations. Any web-scale application has users, which we denote by i, and items which we denote by j. Any association between (i, j) is denoted by x_{ij} which is our main variable of interest. For example, x_{ij} can denote the probability of showing item j to user i. We denote the vectorized version as $x = (x_1, \ldots, x_I)$ where $x_i = \{x_{ij}\}_{j=1}^{J}$. Here, I denotes the total number of users and J denotes the total number of items. Throughout this paper we consider problems of the following form:

$$\min_x \; c^T x \quad \text{s.t.} \quad Ax \leq b, \;\; x_i \in \mathcal{C}_i, \; i \in [I], \tag{1}$$

where, $A_{m\times n}, b_{m\times 1}, \mathcal{C}_i$ are problem data; $x_{n\times 1}$ is the optimization variable; and $[I]$ is shorthand for $1, \ldots, I$.

In order to achieve the extreme-scale arising from problems in web-applications we focus on a particular structure of the above problem. We assume the following:

- $n = IJ$ can range in 100s of millions to 10s of trillions (and potentially can be unbounded).
- A is sparse and highly structured, i.e. multiplying A with a vector can be done efficiently[2]. A is of the form: $[A^{(1)}; A^{(2)}]$ where $A^{(1)}_{m_1\times n}$ with $m_1 = O(1) \ll n$; and

$$A^{(2)} = \begin{pmatrix} D_{11} & \ldots & D_{1I} \\ & \ldots & \\ D_{m_2 1} & \ldots & D_{m_2 I} \end{pmatrix}$$

  where, D_{ij} are $J \times J$ diagonal matrices.
- The diagonal of each D_{ij} is sparse as well, with maximum of K non-zero entries, where $K < J$. Thus overall, the number of non-zero entries of A is given by

$$nnz(A) = m_1 IJ + m_2 IK \tag{2}$$

[2]This takes into account the actual flop count and other load-balancing considerations arising in our distributed implementation.

- $\mathcal{C}_i, i \in [I]$ are "simple" constraints (i.e., it is efficient to compute the Euclidean projection into $\mathcal{C}_i$)—examples include the non-negative orthant, box constraints or the unit simplex (Parikh & Boyd, 2014b, Ch 6).

While our assumptions regarding problem-structure makes (1) a bit specific, we shall show in Section 4, that several important problems arising in the internet industry (that we focus on in this paper) belong to this family.

While small instances of Problem (1) can be handled via off-the-shelf LP solvers, they quickly become prohibitively expensive for large scale instances. We propose to use first-order methods (Nesterov, 2004) to solve (1). However, performing a projection onto the polyhedral constraint in (1) is cumbersome — hence, projected gradient descent methods on the primal LP are not a good choice. We apply a perturbation method (Mangasarian & Meyer, 1979) with old roots in optimization: we add a small ridge regularization into the objective, and then consider solving the dual of the perturbed problem. This dual is smooth (i.e, has Lipschitz continuous gradient) and is hence amenable to first-order methods in smooth continuous optimization (Nesterov, 2013). Our proposed approach is discussed in Section 3.

2.1. Current Approaches

One of the most common methods to solve LPs is via the (primal or dual) simplex algorithm–they are implemented in many software packages (commercial and noncommercial), and they work well in practice for small-moderate instances (Bertsimas & Tsitsiklis, 1997; Gearhart et al., 2013; Bixby, 2002). Interior point methods are also used for solving LPs (Boyd & Vandenberghe, 2004) though they usually have large memory requirements due to matrix factorizations. To solve large-scale instances, it is necessary to exploit the problem-specific structure. For example, Gondzio & Sarkissian (2003) present impressive specialized parallel interior point methods for solving large-scale LPs where the constraint matrix has special block-angular like structure. This requires solving large-scale linear systems using specialized numerical linear algebra routines. Our approach is based on first order methods that mainly require distributed matrix-vector multiplications. The structure of our problem is different than what is considered in Gondzio & Sarkissian; and our approach scales to much larger instances.

Well-known techniques for solving large scale LPs include Dantzig Wolfe decomposition which is closely related to column generation (Bertsimas & Tsitsiklis, 1997). Roughly speaking, these methods express an LP in a suitable basis that admits a sparse solution (few nonzero coefficients). Owing to the anticipated sparsity of the solution, these methods often require a case-by-case analysis and have slow convergence. Moreover, as these methods rely on the simplex method, they may be difficult to parallelize. A related

method is Bender's decomposition (Bertsimas & Tsitsiklis, 1997).

Splitting methods such as ADMM (Boyd et al., 2011) can also be used to approximately solve LPs; though they often involve expensive intermediate steps for solving large linear systems. There are variants of ADMM such as linearized ADMM (Boyd et al., 2011) or linearized Bregman Osher et al., that avoid solving these large linear systems. Unlike the basic version of ADMM, these linearized algorithms converge under carefully chosen step-sizes (which may be hard to compute for large-scale instances). There are improved versions of ADMM that form the basis of state-of-the-art large scale conic optimization solvers such as SCS (O'Donoghue et al., 2016) and they work very well in practice. Such operator splitting or dual decomposition methods can be performed in a distributed fashion (Boyd et al., 2011; Parikh & Boyd, 2014a) though they have limited convergence guarantees (Chen et al., 2016).

Pock & Chambolle (2011) present diagonal preconditioning methods for first order primal-dual algorithms (Chambolle & Pock, 2011) for a family of convex problems with a known saddle-point structure. The authors discuss how to solve LPs arising in total variation denoising, graph cuts and minimal partitioning problems. See also Goldstein et al. (2013) for related algorithms. Similar primal-dual algorithms may also be used to solve the LPs considered in this paper, provided we exploit problem-structure in a manner similar to what we have done in this paper.

Current practice in Industry: In practice, solving the dual problem to find the optimal dual variable $\hat{\lambda}$ is often challenging due to the extreme-scale in industrial problems. One ad-hoc technique that is used in industry (as available in the published literature) (Gupta et al., 2016) is to split the data according to different groups of members into K machines and solve a proxy problem in each machine. In the k-th machine we solve a (tractable) sub-problem of the form:

$$\min_x \; c_k^T x \quad \text{s.t.} \quad A_k x \leq b_k, \quad x_i \in \mathcal{C}_i, i \in S_k. \tag{3}$$

where $x \in \mathbb{R}^{n_k}$ with $n_k = IJ/K$, A_k, c_k are the components of the original matrix $A = [A_1 : \ldots, A_K]$ and vector $c = (c_1, \ldots, c_K)$. b_k is a proxy such that $b = \sum_{k=1}^K b_k$. Finally, S_k denotes the set of members in the k-th machine. This is a much smaller problem, which we can solve and obtain an estimate of $\hat{\lambda}_k$ from each machine for $k \in [K]$ and finally consider $\hat{\lambda} = \sum_{k=1}^K \hat{\lambda}_k / K$. This can only give us a sub-optimal solution as we split the constraints into artificial sub-constraints.

In many practical situations, where the above approach is costly as well, practitioners often rely on hyper-parameter tuning to get the optimal $\hat{\lambda}$. Practitioners choose different values of λ and use it to evaluate different business metrics by serving the users with $\hat{x}(\lambda)$. The λ that provides the best business metrics is chosen as the final optimal $\hat{\lambda}$. This solution can be extremely challenging to employ due to large amount of noise in the system (Letham et al., 2019; Agarwal et al., 2018), potential bad user experience, and high dimension of λ.

3. ECLIPSE: Extreme-Scale LP Solver

We present our framework to solve problem (1) at unprecedented scales, without the ad-hoc splitting schemes discussed in Section 2.1, which appears to be the state-of-the-art in industry.

A perturbation of the LP: As alluded to in Section 2, the basic idea behind our approach is to consider a perturbed version (Mangasarian & Meyer, 1979) of the LP (1) and propose algorithms to maximize its (smooth) dual. To this end, consider the following ℓ_2-squared regularized version of (1) — this leads to the quadratic program (QP):

$$\min_x \; c^T x + \frac{\gamma}{2} x^T x \quad \text{s.t.} \quad Ax \leq b, \;\; x_i \in \mathcal{C}_i, i \in [I] \tag{4}$$

where, $\gamma > 0$ is a regularization parameter that controls (a) how well Problem (4) approximates (1) (cf Lemma 1); and (b) the smoothness of the dual (cf Lemma 3).

Problem (4) is a large scale QP—as written this is not amenable to first-order optimization methods due to the (complicated) nature of the inequality constraints $Ax \leq b$ which is difficult to project onto. The constraints $x_i \in \mathcal{C}_i$ for $i \in I$ on the other hand, separate across i; and it is simple to project onto the set $\mathcal{C}_i$. Thus motivated, we dualize the complicating polyhedral constraint $Ax \leq b$, but retain the (simple) constraints $x_i \in \mathcal{C}_i$ as implicit[3]. This leads to a certain Lagrangian dual given by:

$$g_\gamma(\lambda) = \min_{x \in \mathcal{C}} \; \left\{ c^T x + \frac{\gamma}{2} x^T x + \lambda^T (Ax - b) \right\}, \tag{5}$$

where $\mathcal{C} = \Pi_{i=1}^I \mathcal{C}_i$. Note that, by strong duality (Boyd & Vandenberghe, 2004) the optimum objective g_γ^* of the dual

$$g_\gamma^* := \max_{\lambda \geq 0} \; g_\gamma(\lambda) \tag{6}$$

is the minimum of (4).

We show in Lemma 3 that $\lambda \mapsto g_\gamma(\lambda)$ is differentiable and the gradient is Lipschitz continuous with parameter $L < \infty$

[3]Similar ideas where a complicated polyhedral constraint is dualized with separable (simple) constraints remaining implicit appear in classic Dantzig Wolfe decomposition (Bertsimas & Tsitsiklis, 1997) methods for solving large scale LPs. These methods are usually embedded within a simplex like framework for solving LPs. Our approach differs in that we use first-order methods to solve the QP (4)—making it amenable to distributed matrix-vector multiplications (Section 5). See also (Boyd et al., 2011) for further discussions on dual decomposition methods.

i.e., $\|\nabla g_\gamma(\lambda) - \nabla g_\gamma(\lambda')\| \leq L\|\lambda - \lambda'\|$ for all λ, λ'. In addition, it is simple to project onto the constraint set of (6). Hence (6) is amenable to first-order methods for smooth optimization (Nesterov, 2013) provided the gradient $\nabla g(\lambda)$ can be computed efficiently. As we shall see shortly (cf Lemma 3), this is indeed the case — computing the gradient does not require expensive memory-intensive matrix factorizations, which is a well-known bottleneck in ADMM based methods (Boyd et al., 2011). As we will see, computing $\nabla g_\gamma(\lambda)$ in our case requires performing

(i) a distributed matrix-vector multiplication and
(ii) a projection onto the set $\mathcal{C}$.

Due to the separable structure of the set $\mathcal{C}$ across users, and the simplicity of projecting onto the set $\mathcal{C}_i$, operation (ii) can be performed in parallel across multiple executors (See Section 5). While we are able to compute $\nabla g_\gamma(\lambda)$ efficiently in practice (especially, considering the scale of the problems we are dealing with)—this constitutes the dominant cost of our proposed framework.

Note that, when $\gamma = 0$, we get a dual of the LP (1). If g_0^* is the optimal objective value of (5) for $\gamma = 0$, by LP strong duality this equals the optimal objective of (1). The following Lemma gives an uniform bound between the perturbed dual objective $g_\gamma(\lambda)$ to the unperturbed objective—this immediately leads to a $O(\gamma)$-bound on the difference in their optimal objective values as discussed in the following Lemma (Nesterov, 2005):

Lemma 1. *The dual $g_\gamma(\lambda)$ satisfies the (uniform) bound:*

$$g_\gamma(\lambda) - \gamma\vartheta/2 \leq g_0(\lambda) \leq g_\gamma(\lambda) \;\; \forall\lambda$$

where, $\vartheta = \max_{x\in\mathcal{C}} x^T x$. The optimal objective value of the LP (i.e., g_0^*), lies in the interval $[g_\gamma^* - \gamma\vartheta/2, g_\gamma^*]$, where, g_γ^* is the optimal objective (6).

The proof of Lemma 1 (which appears in the Appendix) also provides simple bounds on the constant ϑ. From the above result, it follows that as $\gamma \to 0+$, the solution of Problem (4) converges to the minimum ℓ_2-norm solution of Problem (1). This result can be strengthened further (Mangasarian & Meyer, 1979; Friedlander & Tseng, 2008) in light of what is known as the exact regularization for LPs:

Lemma 2. *There exists a $\bar\gamma > 0$ such that for all $\gamma \in [0, \bar\gamma]$ the minimizer x_γ^* of the perturbed primal (4) is an optimal solution to the LP (1).*

Lemma 2 states that for sufficiently small (but, strictly positive) values of γ, an optimal solution to the perturbed QP is also an *optimal* solution to the LP — this occurs even if there is a positive gap (i.e., of order $O(\gamma)$ as in Lemma 1) between the optimal objective values of the QP and the LP. Note however, that Lemma 2 offers an existential result—the choice of $\bar\gamma$ (Mangasarian & Meyer, 1979) requires knowing an optimal primal-dual pair of solutions to (1) and a variant of the QP (4).

Smooth optimization on the dual: We proceed to show that for any $\gamma > 0$, the function $\lambda \mapsto g_\gamma(\lambda)$ has Lipschitz continuous gradient. This follows from the fundamental observation that the minimization task w.r.t. x in (5) admits a unique solution due to the strong convexity of the objective (w.r.t x) (Bertsekas, 1999). For a dual variable λ, the corresponding primal variable $\hat x(\lambda)$ can be obtained by solving the inner optimization problem (5). Due to the block-structure of $\mathcal{C} = \prod_{i=1}^I \mathcal{C}_i$ the ith block of the primal vector $\hat x(\lambda)$ is given by:

$$\hat x_i(\lambda) = \Pi_{\mathcal{C}_i}[-\tfrac{1}{\gamma}(A^T\lambda + c)_i] \tag{7}$$

where $\Pi_{\mathcal{C}_i}(\cdot)$ is the Euclidean projection operator onto $\mathcal{C}_i$; and for a vector a with the same dimensions as the decision variable x, we let a_i denote the sub-vector with indices corresponding to user i. For our applications (See Sections 2 and 4), $\Pi_{\mathcal{C}_i}$ can be computed in nearly closed form. We obtain $\hat x(\lambda)$ by a stacking operation: $\hat x(\lambda) = \{\hat x_i(\lambda)\}_{i=1}^I$.

In addition to the above (nearly closed form) expression of $\hat x(\lambda)$, the gradient of $g_\gamma(\lambda)$ and an upper bound on its Lipschitz constant can be computed by using standard results in convex analysis (Bertsekas, 1999; Nesterov, 2005):

Lemma 3. *$g_\gamma(\lambda)$ is differentiable with $\nabla g_\gamma(\lambda) = A\hat x(\lambda) - b$ where, where $\hat x(\lambda) = \{\hat x_i(\lambda)\}_{i=1}^I$. Furthermore, $g_\gamma(\lambda)$ has Lipschitz continuous gradient with parameter $L = \|A\|_S^2/\gamma$ where, $\|A\|_S$ is the largest singular value of A.*

As $\lambda \mapsto \nabla g_\gamma(\lambda)$ is Lipschitz continuous, one can use projected gradient ascent (Nesterov, 2004; 2013) to solve (6). This leads to the update sequence

$$\begin{aligned}\lambda^{k+1} &\in \operatorname{argmin}_{\lambda\geq 0} \|\lambda - (\lambda^k + \eta\nabla g(\lambda^k))\|_2^2 \\ &= (\lambda^k + \eta\nabla g(\lambda^k))_+\end{aligned} \tag{8}$$

where, the step-size $\eta \leq 1/L$ and $(a)_+ = \max\{a, 0\}$ denotes a component-wise operation for a vector a. This has a sub-linear convergence rate of $g_\gamma^* - g_\gamma(\lambda^k) \leq O(1/k)$ where, g_γ^* is defined in (6). The gradient method can be accelerated by using Accelerated Gradient (AGD) methods (Beck & Teboulle, 2009) with updates (for $k \geq 0$):

$$\begin{aligned}\lambda^k &= \left(\xi^k + \eta\nabla g_\gamma(\xi^k)\right)_+ \\ \xi^{k+1} &= \lambda^k + \beta_k(\lambda^k - \lambda^{k-1})\end{aligned} \tag{9}$$

where, $\beta_k = (t_k - 1)/t_{k+1}$ and $t_{k+1} = (1 + \sqrt{1 + 4t_k^2})/2$ with initialization $t_1 = 1$ and $\xi^1 = \lambda^0$. AGD leads to a worst-case convergence rate of $g_\gamma^* - g_\gamma(\lambda^k) \leq O(1/k^2)$, an improvement over the sublinear convergence rate of the proximal gradient method. In the special case when $\beta_k \equiv 0$, updates (9) reduce to the proximal gradient updates (8).

Choice of γ: Observe that Lemma 1 shows a $O(\gamma)$ approximation quality of the perturbed problem vs the LP, while Lemma 3 shows $\nabla g_\gamma(\lambda)$ is $O(1/\gamma)$-Lipschitz, which dictates the convergence speed of gradient descent. Practitioners often have a rough idea of the allowable approximation quality that can be used to choose γ. If this is unknown and/or the γ value is too small, one can apply a continuation approach: solve (6) for a sequence of decreasing γ values (large to small) with warm-starts.

Computing $\|A\|_S$ and step-size selection: Recall that convergence of updates (8) or (9) occur for $\eta \leq \gamma/\|A\|_S^2$. Note that, for improved convergence, a line-search may be used (Beck & Teboulle, 2009)—but this may lead to increased computational costs (and communication overhead) due to distributed matrix-vector multiplications. We thus use a constant step-size η. We derive an easy-to-compute upper bound on the largest singular value of $A = [A^{(1)}; A^{(2)}]$ making use of its special structure (see Appendix). This requires computing $\|A^{(1)}\|_S$ and $\|A^{(2)}\|_S$, which can be computed (in a distributed fashion) with total cost $O(n)$.

The overall algorithm is given as Algorithm 1.

Algorithm 1 ECLIPSE: Extreme-Scale LP Solver

Input: $A_{m\times n}$, $\{\mathcal{C}_i\}_{i=1}^I$, b, c, γ and initialization λ^0.
Output: Dual Solution λ^*, Primal Solution x^*

1: Compute $L = \gamma/\|A\|_S^2$; and repeat the following steps till convergence
2: For a dual iterate λ^k, compute a primal solution via (7); and compute the gradient $\nabla g(\lambda^k) = A\hat{x}(\lambda^k) - b$ (cf Lemma 3). Both operations are performed in parallel over the executors (cf Section 5).
3: Update the dual variable λ^k via the AGD update (9). (The AGD update can be replaced by a proximal gradient update as in (8)).
4: Return (approximate) dual $\lambda^* = \lambda^k$ and primal $x^* = \hat{x}(\lambda^k)$ solutions.

4. Applications

We specialize the framework presented above to example problems arising in industrial applications; and discuss solution methods.

Multi-objective optimization problems (Agarwal et al., 2011; Marler & Arora, 2004) appear frequently in machine learning and web applications. In internet applications, multi-objective optimization problems can be framed by considering one of the objectives as primary and others as constraints. Typical examples of constraints are:

1. **Global constraints**: the constraint potentially involves all the variables of interest.
2. **Cohort-level constraints**: the constraints are over either a cohort of members, a cohort of items or both.
3. **User or item level constraints**: the constraints are on an individual item, individual user, or both.

For each of the problems described below, we explain how such constraints arise, how to formulate the problem as an LP and solve the corresponding dual (6).

In all of these applications, it is of paramount interest to get the accurate dual, because the primal can be generated from it via (7). Especially for the new items that are coming into the system, we can get the primal decision variable x_{ij} without even solving the optimization problem. This allows us to work in a low-latency environment as required by most internet applications. We can efficiently obtain $\hat{x}_i(\lambda)$ because A and c can be generated very quickly due to fast online model scoring pipelines and projection onto $\mathcal{C}_i$ is efficient.

Thus, the mechanism of solving such problems in industry is to first solve an extreme-scale problem to generate the duals and then use the duals in a low-latency environment to recover the primal, without the need of solving any optimization problem for every new item that is coming into the ecosystem.

Remark 1. *The above method for re-using the dual variable works as long as the score distribution of the new items matches that of the old items which were used to solve the Problem (4). To prevent staleness, in practice, the optimization problem is solved at a regular cadence.*

Volume Optimization: A core problem in any internet industry is to send out emails and notifications to the users either for marketing purposes, for gaining user attention, or to bring users back to the platform. They help in the quick distribution of information, but too many emails or notifications are not preferable as they may lead bad user experience (Gupta et al., 2016; 2017; Gao et al., 2018).

Let the formal problem be to maximize the overall sessions, such that the total number of emails or notifications that are sent is bounded, the overall click rate is above a threshold and the total disable rate is bounded as well. To frame this problem mathematically, let x_{ij} denote the probability of sending the j-th email or notification to the i-th user. We can build machine learning models for the various quantities of interest:

- *Sessions Model*: p_{ij}^1 predicts if user will come back to the platform if this email/notification is sent (Yuan et al., 2019).
- *Click Model*: p_{ij}^2 predicts if user will click on the item.
- *Disable Model*: p_{ij}^3 predicts if user will disable the email/notification if sent.

We can thus formulate an LP:

$$\begin{aligned}
\max_x \quad & x^T p^1 && \text{(Total Sessions)} && \text{(10a)}\\
\text{s.t.} \quad & x^T 1 \le c_1 && \text{(Sends are Bounded)} && \text{(10b)}\\
& x^T p^2 \ge c_2 && \text{(Clicks above a threshold)} && \text{(10c)}\\
& x^T p^3 \le c_3 && \text{(Disables below a threshold)} && \text{(10d)}\\
& 0 \le x \le 1 && \text{(Probability Constraint)} && \text{(10e)}
\end{aligned}$$

which is an instance of (1) with $m_1 = 3$, $m_2 = 0$ and $\mathcal{C} = \prod_{i=1}^{I}[0,1]^J$. Here, $n \approx 10^9$ with 100's of millions of users and 1000s of potential email/notifications. Such a problem can be solved using the methods we propose in Section 3—both matrix-vector multiplication and projection onto $[0,1]^J$ are cheap.

Problem (10) can be generalized by introducing some cohort-level constraints. For example, we can partition the member base into different cohorts S_k, for $k \in [K]$ (e.g. by activity levels such as daily, weekly, monthly, etc). Instead of placing a bound on the overall sends, we could have a separate bound for each such cohort S_k. Similar constraints can also hold for groups of notifications/emails. If we denote G_l, for $l \in [L]$ as the cohorts of items, we can enforce click constraints on each such group. We can even add individual user level constraints. For example, from a practical stand point, we would not want each user to be overwhelmed by a large number of emails/notifications. Combining the cohort and per-user level constraints, the modified problem replaces (10b), (10c), (10e) by

$$\begin{aligned}
\sum\nolimits_{i\in S_k}\sum\nolimits_j x_{ij} &\le c_{1k}\ \forall k\\
\sum\nolimits_i\sum\nolimits_{j\in G_l} x_{ij}p^2_{ij} &\ge c_{2l}\ \forall l\\
\sum\nolimits_j x_{ij} &\le c_i\ \forall i,\ \ 0 \le x \le 1
\end{aligned}$$

respectively. As an instance of (1), we have $m_1 = K+L+1$, $m_2 = 0$ and $\mathcal{C}_i = \{\alpha \in [0,1]^J : \sum_j \alpha_j \le c_i\}$. The projection $\Pi_{\mathcal{C}_i}$ can be computed easily (Parikh & Boyd, 2014b). As long as the number of cohorts K, L are small, this problem is computationally feasible.

Matching Problems: Matching problems (Azevedo & Weyl, 2016) are an important class of problems in the internet industry. Typical examples include matching items to users in an e-commerce platform, matching drivers to riders in ride-sharing apps, matching people to one another or even jobs to potential job seekers. While we describe a matching problem in the context of friend or connection recommendation, other generalizations are possible.

Suppose, we wish to maximize the total value members of a platform would receive when they form a new connection. At the same time, we would not want to overwhelm the influencers and famous people in the platform with a large number of connection invites. Let x_{ij} denote the probability of showing the j-th candidate to the i-th viewer. As before, we will have the following machine learning models

- *Value Model* p^1_{ij} denotes the probability of having a meaningful interaction if the connection is formed.
- *Connection Model* p^2_{ij} denotes if the i-th viewer will send a request to j-th candidate if the j-th candidate is recommended.

Based on these, we can write the matching problem as:

$$\begin{aligned}
\max_x \quad & x^T p^1 && \text{(Total Value)}\\
\text{s.t.} \quad & \textstyle\sum_i x_{ij}p^2_{ij} \le b_j,\ j \in [J],\ \ x_i \in \mathcal{C}_i,\ i \in [I] && \text{(11)}
\end{aligned}$$

where $\mathcal{C}_i$ denotes the simplex. Here the first constraint limits candidate j to at most b_j invitations and the simplex constraint forces an inherent ranking in the set of candidates for member i. We have $J \approx 10^4$ candidates, $I \approx 10^8$ users which generates $n \approx 10^{12}$ (1 Trillion) decision variables.

Remark 2. *Note that this problem is inherently different from the problems we discussed before, mainly because the first constraint is an item-level constraint (here the items are also members). We previously only considered global and cohort level constraints in our matrix A.*

In this setup we have $A^{(1)}$ as an empty matrix, $A^{(2)} = [D_{11} \dots D_{1I}]$, where $D_{1i} = \text{Diag}(p^2_i)$. For each user i, we get $\hat{x}_i(\lambda) = \Pi_{\mathcal{C}_i}(-\frac{1}{\gamma}(D_{1i}\lambda + c_i))$ where, $\mathcal{C}_i$ is the simplex. Since p^2_i is quite sparse, the overall computation is very fast. The exact computation details are given in Section 5.

The problem in (11) can be generalized by adding in further constraints. We can add global constraints for the total number of invitations being sent out–i.e., $\sum_{i,j} x_{ij}p^2_{ij} > \alpha_1$. We can also add cohort level constraints such as members belonging to a certain cohort gets a minimum number of invitations—i.e., $\sum_i \sum_{j\in S_k} x_{ij}p^2_{ij} > \alpha_k$. Here, we can write the matrix $A = [A^{(1)}; A^{(2)}]$ where $A^{(1)}$ corresponds to the global and cohort level constraints with $m_1 = 1 + K$ and $A^{(2)}$ corresponding to the per-item constraints having the diagonal structure as above. Since both of them have low-cost matrix-vector multiplication, the overall system can still be solved.

Overall, if we denote the dual variables as λ_1 and $\{\lambda^\ell_2\}_{\ell=1}^{m_2}$ corresponding to $A^{(1)}$ and $A^{(2)}$, the primal solution (7) for the ith user can be computed as: $\Pi_{\mathcal{C}_i}[-\frac{1}{\gamma}(A^{(1)T}_i\lambda_1 + \sum_{\ell=1}^{m_2}\text{Diag}(p^\ell_i)\lambda^\ell_2 + c_i)]$ where $A^{(1)}_i$ and c_i are the appropriately split versions of matrix $A^{(1)}$ and vector c for user i. From an implementation perspective, each of the $m_2 + 1$ matrix-vector multiplication operations can be done in parallel, combined together and finally projected onto $\mathcal{C}_i$.

Remark 3. *For this framework, we recommend to have a few dual variables and use per-member constraints that*

have efficient projection algorithms.

5. System Architecture

We now describe how we implemented the proposed algorithm on a distributed computing system. We discuss high-level system architecture as well as major implementation details. We will soon open-source the solver.

One key-factor limiting our ability to solve extreme-scale LPs on a single system is memory consumption. The storage space required for storing the input and intermediate results is well beyond the capability of a single machine. For this reason, we built the system on top of Apache Spark (Zaharia et al., 2016) that is optimized to handle large-scale iterative computations that are common in optimization algorithms. For iterative algorithms, Spark is capable of caching data in aggregate main memory, which can significantly reduce I/O cost. Communication across tasks is also very efficient reducing the overhead on iterative algorithms. Although some high performance computing architecture such as MPI (Jin et al., 2011) have better support for scientific computations, Spark has the following advantages: 1) Spark can be run on commodity clusters of machines that are commonplace, 2) many big data processing pipelines running at internet companies are implemented in Spark. Therefore, Spark-based solvers can be seamlessly integrated into an existing big data processing pipeline.

Data Representation: Our implementation is based on a customized DistributedMatrix API. Matrix A is composed of two parts: a dense flat matrix $A^{(1)}$ and a sparse matrix $A^{(2)}$. To facilitate distributed computing, we split the matrix into blocks. For $A^{(1)}$, we utilize the BlockMatrix API in Apache MLLib (Meng et al., 2016). For $A^{(2)}$, given its very special diagonal structure, the matrix-vector multiplication is equivalent to a vector-vector dot product. Therefore, we implemented a lightweight DistributedVector using an RDD of (index, vector). For all the remaining vectors in Algorithm 1, we use DistributedVector to represent them. The size of the block is chosen to be the same as the number of users.

Estimating the Primal: The matrix multiplication of $A_i^{(1)T}\lambda_1$ is done in parallel with respect to i. To compute $\text{Diag}(p_i^\ell)\lambda_2^\ell$ we flatten the Diagonal matrix to a vector and do a vector-vector dot product, which makes the execution much faster. Moreover, they are done in parallel. The projection operation also takes place on the executors in parallel.

The computational complexity can be calculated as follows. Each vector x_i is of length J and $A^{(1)}$ is of dimension $m_1 \times IJ$. Moreover, let the sparsity of p_i^ℓ be bounded by K (i.e. the maximum number of non-zeros is less than K). Then to estimate $A_i^{(1)T}\lambda_1$ we need at the maximum $m_1 J$ operations. For the other part, we need K operations. Moreover, if we are performing the simplex projection it can be done in $O(J)$ steps. Thus the overall complexity of generating each $\hat{x}_i$ under simplex projection is $O(J)$. Since each of them can be done in parallel, we get $\hat{x}(\lambda)$ in $O(J)$ steps. Note that, matrix A will be constant across all iterations, while $\lambda_1, \{\lambda_2\}$ will be updated every iteration. We explicitly cache A in executor and broadcast $\lambda_1, \{\lambda_2\}$ to every executor. In this way, the communication cost is minimized.

Estimating the Dual: The most computationally expensive step to generate the dual is the matrix-vector multiplication $A\hat{x}(\lambda)$. Here $\hat{x}(\lambda)$ has length much larger than λ. Instead of actually combining individual $\hat{x}_i$ to get $\hat{x}(\lambda)$ we use the individual components to speed up the execution. We estimate in parallel $A_i^{(1)}\hat{x}_i$ which accounts for $m_1 J$ operations. Moreover, we get each of the $\text{Diag}(p_i^\ell)\hat{x}_i$ in K operations, (assuming the sparsity in p_i^ℓ to be K). Thus, each component of $A\hat{x}(\lambda)$ is computed in $O(J)$ steps, which are finally combined together by summing them all up, which takes $O(IJ)$ operations. The rest of the operations are all on the vector of size $m_1 + m_2 J$ which is rather small and can be easily handled.

Convergence Criteria: When we are able to estimate the duality gap (Boyd & Vandenberghe, 2004) (by having access to a primal feasible solution) our convergence criteria are set according to it. Specifically, we stop the algorithm if the relative duality gap is smaller than a threshold. In other situations, where projection to the feasible region is hard, we consider the relative difference in norm of the dual variable as the stopping criteria, i.e. $\|\lambda_{k+1} - \lambda_k\|/\|\lambda_k\| \leq \epsilon$.

The overall architecture is shown in Figure 1.

6. Experiments

We run extensive experiments in our system along with experiments with real datasets. Through this framework, we have been able to scale to problems which we had never tackled before. We first describe some of the simulated experiments followed by some real-world setups.

Simulated Experiments: We run simulated experiments for three purposes. First, to test the validity of our solution. Second, to compare the scalability of the solution as compared to other open-source solutions such as the Splitting Conic Solver (SCS) (O'Donoghue et al., 2016). Third, for extreme-scale problems, we compare our solution to the averaging technique as explained in Section 2.

There are many open-source or commercial solvers such as SDPT3, CPLEX, CLP, etc for solving linear programming problems. These solvers uses simplex (Bertsimas & Tsitsiklis, 1997) or interior-point algorithm (Boyd & Vandenberghe, 2004) and are implemented in C/C++. They

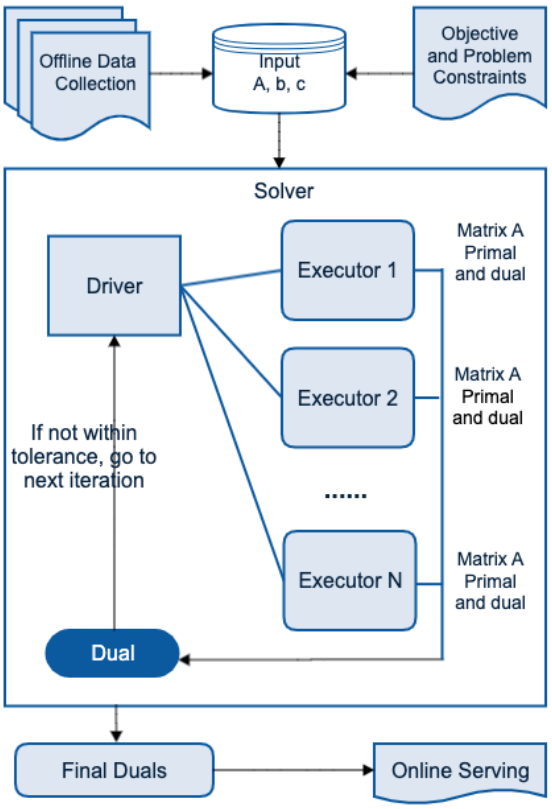

Figure 1. The Overall System Architecture for ECLIPSE.

are designed to run on a single machine, limiting the scale they can handle (a thousand to at most a million variables). Also, they are solving for high accuracy solution while our solver gives approximate solutions. It is not a fair and meaningful comparison. Moreover, our aim in this paper was not to introduce a new generic LP solver. We seek to solve structured LPs for web-applications at extreme-scale. Comparison with multiple off the shelf solvers that do not scale are hence excluded. The closest open-source solver we found is SCS (O'Donoghue et al., 2016) that could potentially scale to problem sizes comparable to the instances we consider.

To test the validity and scalability of the system, we consider a simple volume optimization problem, as described in Section 4. We let $n = IJ$ where I is the number of users and J is the number of items and solve the problem with two global constraints. We randomly simulate the vectors c, p and b, and solve the problem:

$$\max_x \; x^T c \quad \text{s.t.} \quad x^T 1 \le b_1,\; x^T p \le b_2,\; x \in [0,1]^n \quad (12)$$

To show the validity of our approach, we use $I = 10000$ and $J = 100$ for which we were able to solve the problem with SCS in a relatively short amount of time. We then use ECLIPSE to solve the problem and show how the primal and dual solutions are approaching the true solution of the problem. Here, we use the relative duality gap as the stopping criteria of our problem. Figure 2 shows the convergence of our method as iterations increase.

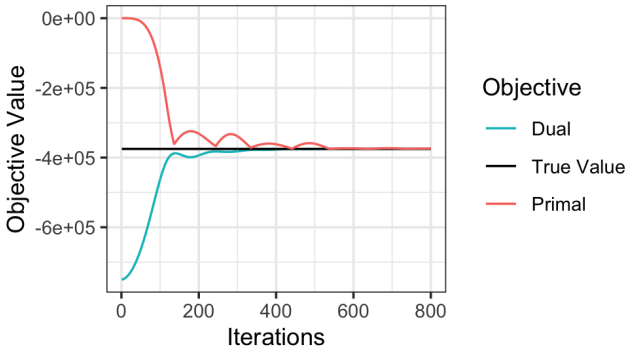

Figure 2. Duality Gap as iterations increase

To investigate the scalability of our system, we choose different values of I and J and report the results in Figure 3. We solve these problems using our method as well as SCS and in all cases, we have been able to match the solution to the SCS solver up-to a level of precision. We do multiple runs and the average running time is shown in Figure 3.

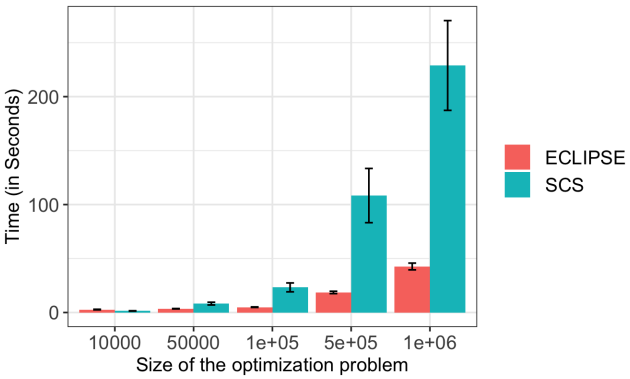

Figure 3. Running time of ECLIPSE vs SCS on Simulated Data

Note that we do not have an optimized algorithm for this simulation, while SCS uses a highly optimized algorithm written in C. In smaller problems, SCS is definitely faster but we can see that as the size of the problem increases, we converge much faster than SCS, this is due to the fact that we use fewer (structured) matrix-vector multiplications.

Finally, we compare our algorithm to the averaging technique as described in Section 2. We split the problem into different groups, solve them and use the average dual as the final solution. We compare it with our algorithm and report the objective values as well and the primal residuals for the different settings of the optimization problem in (12). We choose $J = 100$ and different values of I, with split sizes

$\{10^3, 10^4\}$. The results are shown in Table 1.

n	Method	Objective	Primal Residual
10^6	ECLIPSE	$\mathbf{3.751 \times 10^5}$	$\mathbf{6.91 \times 10^{-4}}$
	Average 1	3.748×10^5	3.73×10^{-3}
	Average 2	3.747×10^5	1.03×10^{-2}
10^7	ECLIPSE	$\mathbf{3.750 \times 10^6}$	$\mathbf{7.12 \times 10^{-4}}$
	Average 1	3.747×10^6	1.71×10^{-3}
	Average 2	3.747×10^6	3.73×10^{-3}
10^8	ECLIPSE	$\mathbf{3.750 \times 10^7}$	$\mathbf{6.56 \times 10^{-4}}$
	Average 1	3.747×10^7	1.17×10^{-3}
	Average 2	3.747×10^7	1.73×10^{-3}

Table 1. Comparison of our algorithm with the averaging method. Average 1 and 2 correspond to a split size of 10^3 and 10^4 respectively.

In all the above simulation experiments, it follows from (2) that the number of non-zero entries in A is $2IJ$, ($m_1 = 2, m_2 = 0$ in (12)). For further experimental details, data explanations and the code for all of these experiments please see the supplementary material. Note that, since we could not find open-source example datasets that exactly fit the problem structure, we excluded comparisons with common LP instances.

Real World Experiments: We have built ECLIPSE in Scala/Spark to scale to real-world problems. First, to validate the Scala/Spark implementation, we use a problem for which the solution is known, and we intentionally replicate the problem to drastically increase its scale. We ran such a problem using our implementation without explicitly using the separable nature. In all cases, as we scaled the problem, we achieved parity with the known solution up to the pre-specified level of precision. We then experimented with real-data sets. We considered the volume optimization problem with global constraints as given in (10), and the matching problem as given in (11). Table 2 shows us the running time and scale for solving such problems.

Problem	Scale n	Time(Hours)	
		ECLIPSE	SCS
Volume Optimization (10)	10^7	0.8	2.0
	10^8	1.3	>24
	10^9	4.0	>24
Matching Problem (11)	10^{10}	4.5	>24
	10^{11}	7.2	>24
	10^{12}	11.9	>24

Table 2. Running time (hrs) for extreme-scale Problems on real data.

The experiment was running in a development cluster in Spark 2.3 with up to 800 executors. In these instances, the volume optimization problem had $nnz(A) = 3n$ (see (10)), while the matching problem had $nnz(A) = IK = 10^{11}$ (with $I \approx 10^8$ and sparsity $K \approx 1000$). We also implemented SCS (indirect) method in Spark for comparison. However, SCS could not scale beyond $n = 10^8$ as solving time was longer than one day. This is primarily because SCS does not exploit the problem structure and hence the computing time for each iteration was rather long. On the contrary, even for a problem of one trillion scale, ECLIPSE converged within 12 hours. This running time satisfies most of the web applications' requirements.

7. Conclusion

In this paper, we designed a framework for solving extreme-scale LPs arising from several different kinds of web-applications. The framework is general enough to cater to almost any multi-objective optimization problem arising in the internet industry. Moreover, given the computation resources, the framework can potentially scale to arbitrarily large problems. We were able to scale the solution for a matching problem to a range of 1 trillion variables. If more compute power is available, this framework can easily scale to even larger problems.

Acknowledgements

We would sincerely like to thank Saathiya Keerthi, Lingjie Wang, Shipeng Yu, Hema Raghavan, Romer Rosales, Deepak Agarwal, Shaunak Chatterjee and Ankan Saha for their support and detailed insightful feedback during the development of this solver. We would also like to thank the anonymous reviewers for their helpful comments.

References

Agarwal, D., Chen, B.-C., Elango, P., and Wang, X. Click Shaping to Optimize Multiple Objectives. In *KDD*, pp. 132–140, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0813-7.

Agarwal, D., Chen, B.-C., Elango, P., and Wang, X. Personalized Click Shaping Through Lagrangian Duality for Online Recommendation. In *SIGIR*, pp. 485–494, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1472-5.

Agarwal, D., Chen, B.-C., Gupta, R., Hartman, J., He, Q., Iyer, A., Kolar, S., Ma, Y., Shivaswamy, P., Singh, A., and Zhang, L. Activity Ranking in LinkedIn Feed. In *KDD*, pp. 1603–1612, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2956-9.

Agarwal, D., Chen, B.-C., He, Q., Hua, Z., Lebanon, G., Ma, Y., Shivaswamy, P., Tseng, H.-P., Yang, J., and Zhang, L. Personalizing LinkedIn Feed. In *KDD*, pp. 1651–1660. ACM, 2015.

Agarwal, D., Basu, K., Ghosh, S., Xuan, Y., Yang, Y., and Zhang, L. Online Parameter Selection for Web-based Ranking Problems. In *KDD*, pp. 23–32, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5552-0.

Azevedo, E. M. and Weyl, E. G. Matching markets in the digital age. *Science*, 352(6289):1056–1057, 2016.

Beck, A. and Teboulle, M. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2(1):183–202, 2009.

Bertsekas, D. P. *Nonlinear Programming*. Athena Scientific, Belmont, Massachusetts, 2nd edition, 1999. ISBN 1886529000.

Bertsimas, D. and Tsitsiklis, J. N. *Introduction to linear optimization*. Athena Scientific Belmont, MA, 1997.

Bixby, R. E. Solving real-world linear programs: A decade and more of progress. *Operations Research*, 50(1):3–15, 2002.

Borisyuk, F., Zhang, L., and Kenthapadi, K. LiJAR: A system for job application redistribution towards efficient career marketplace. In *KDD*, pp. 1397–1406. ACM, 2017.

Boyd, S. and Vandenberghe, L. *Convex Optimization*. Cambridge university press, 2004.

Boyd, S., Parikh, N., Chu, E., Peleato, B., and Eckstein, J. Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011.

Chambolle, A. and Pock, T. A first-order primal-dual algorithm for convex problems with applications to imaging. *Journal of mathematical imaging and vision*, 40(1):120–145, 2011.

Chen, C., He, B., Ye, Y., and Yuan, X. The direct extension of ADMM for multi-block convex minimization problems is not necessarily convergent. *Mathematical Programming*, 155(1-2):57–79, 2016.

Du, S., Lee, J., and Ghaffarizadeh, F. Improve User Retention with Causal Learning. In *The 2019 ACM SIGKDD Workshop on Causal Discovery*, pp. 34–49, 2019.

Friedlander, M. P. and Tseng, P. Exact regularization of convex programs. *SIAM Journal on Optimization*, 18(4): 1326–1350, 2008.

Gao, Y., Gupta, V., Yan, J., Shi, C., Tao, Z., Xiao, P., Wang, C., Yu, S., Rosales, R., Muralidharan, A., and Chatterjee, S. Near Real-time Optimization of Activity-based Notifications. In *KDD*, pp. 283–292, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5552-0.

Gearhart, J. L., Adair, K. L., Detry, R. J., Durfee, J. D., Jones, K. A., and Martin, N. Comparison of open-source linear programming solvers. *Sandia National Laboratories, SAND2013-8847*, 2013.

Goldstein, T., Li, M., Yuan, X., Esser, E., and Baraniuk, R. Adaptive primal-dual hybrid gradient methods for saddle-point problems. *arXiv preprint arXiv:1305.0546*, 2013.

Gondzio, J. and Sarkissian, R. Parallel interior-point solver for structured linear programs. *Mathematical Programming*, 96(3):561–584, 2003.

Gupta, R., Liang, G., Tseng, H.-P., Holur Vijay, R. K., Chen, X., and Rosales, R. Email Volume Optimization at LinkedIn. In *KDD*, pp. 97–106, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450342322. doi: 10.1145/2939672.2939692.

Gupta, R., Liang, G., and Rosales, R. Optimizing Email Volume For Sitewide Engagement. In *CIKM*, pp. 1947–1955, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450349185. doi: 10.1145/3132847.3132849.

Gupta, R., Chen, G., and Yu, S. Internal Promotion Optimization. In *KDD*, pp. 2358–2366, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362016. doi: 10.1145/3292500.3330715.

Jin, H., Jespersen, D., Mehrotra, P., Biswas, R., Huang, L., and Chapman, B. High performance computing using MPI and OpenMP on multi-core parallel systems. *Parallel Computing*, 37(9):562–575, 2011.

Kenthapadi, K., Le, B., and Venkataraman, G. Personalized Job Recommendation System at LinkedIn: Practical Challenges and Lessons Learned. In *RecSys*, pp. 346–347, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4652-8. doi: 10.1145/3109859.3109921.

Letham, B. and Bakshy, E. Bayesian Optimization for Policy Search via Online-Offline Experimentation. *ArXiv*, abs/1904.01049, 2019.

Letham, B., Karrer, B., Ottoni, G., Bakshy, E., et al. Constrained bayesian optimization with noisy experiments. *Bayesian Analysis*, 14(2):495–519, 2019.

Linden, G., Smith, B., and York, J. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, 7(01):76–80, jan 2003. ISSN 1941-0131. doi: 10.1109/MIC.2003.1167344.

Mangasarian, O. L. and Meyer, R. Nonlinear perturbation of linear programs. *SIAM Journal on Control and Optimization*, 17(6):745–752, 1979.

Marler, R. T. and Arora, J. S. Survey of multi-objective optimization methods for engineering. *Structural and Multidisciplinary Optimization*, 26(6):369–395, 2004.

Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., Freeman, J., Tsai, D., Amde, M., Owen, S., et al. Mllib: Machine learning in apache spark. *The Journal of Machine Learning Research*, 17(1):1235–1241, 2016.

Nesterov, Y. *Introductory Lectures on Convex Optimization: A Basic Course*. Kluwer, Norwell, 2004.

Nesterov, Y. Smooth minimization of non-smooth functions. *Mathematical programming*, 103(1):127–152, 2005.

Nesterov, Y. Gradient methods for minimizing composite functions. *Mathematical Programming*, 140(1):125–161, 2013.

Osher, S., Mao, Y., Dong, B., and Yin, W. Fast linearized bregman iteration for compressive sensing and sparse denoising. *arXiv preprint arXiv:1104.0262*, 2011.

O'Donoghue, B., Chu, E., Parikh, N., and Boyd, S. Conic optimization via operator splitting and homogeneous self-dual embedding. *Journal of Optimization Theory and Applications*, 169(3):1042–1068, 2016.

Parikh, N. and Boyd, S. Block splitting for distributed optimization. *Mathematical Programming Computation*, 6(1):77–102, 2014a.

Parikh, N. and Boyd, S. P. Proximal Algorithms. *Foundations and Trends in optimization*, 1(3):127–239, 2014b.

Pock, T. and Chambolle, A. Diagonal preconditioning for first order primal-dual algorithms in convex optimization. In *2011 International Conference on Computer Vision*, pp. 1762–1769. IEEE, 2011.

Schafer, J. B., Konstan, J., and Riedl, J. Recommender systems in e-commerce. In *Proceedings of the 1st ACM conference on Electronic commerce*, pp. 158–166. ACM, 1999.

Snoek, J., Larochelle, H., and Adams, R. P. Practical Bayesian Optimization of Machine Learning Algorithms. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'12, pp. 2951–2959, 2012.

Yuan, Y., Zhang, J., Chatterjee, S., Yu, S., and Rosales, R. A State Transition Model for Mobile Notifications via Survival Analysis. In *WSDM*, pp. 123–131, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450359405. doi: 10.1145/3289600.3290981.

Zaharia, M., Xin, R. S., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., Rosen, J., Venkataraman, S., Franklin, M. J., et al. Apache Spark: A Unified Engine for Big Data Processing. *Communications of the ACM*, 59(11):56–65, 2016.

Zheng, H. and Wu, J. Online to offline business: urban taxi dispatching with passenger-driver matching stability. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pp. 816–825. IEEE, 2017.