Maximum Likelihood Optimization via Parallel Estimating Gradient Ascent

Yining Wang
Warrington College of Business, University of Florida, Gainesville, FL 32601, USA
Quanquan Liu
Department of Economics, University of Pittsburgh, Pittsburgh, PA 15260, USA

Global optimization without access to gradient information is a central task to many econometric applications as the tool to obtain maximum likelihood estimators for very complicated likelihood functions. The estimating gradient descent framework is particularly popular, which uses local functional evaluation to build gradient estimates and perform gradient descent from multiple initial points. In this work, we study the problem of coordination between the multiple "threads" of estimating gradient descent in order to pause or terminate unpromising threads early. The high-level idea is to make predictions, either conservative or aggressive, on the potential progress of each estimating gradient descent threads and to compare them with the progress on other threads. We also test our proposed methodology on both synthetic data and real airline pricing data, and compare with competitive methods including the genetic algorithm and the pattern search algorithm. The numerical results show the effectiveness and efficiency of our proposed approach.

1. Introduction

Maximum-likelihood (ML) estimation is the workhorse for a wide range of inferring tasks of econometric modeling. Mathematically, given an econometrical model $p(x;\theta)$ with unknown parameter of interest $\theta \in \Theta \subseteq \mathbb{R}^d$ and collected data sample $\{x_i\}_{i=1}^N$, the ML estimation problem can be formulated as

$$\max_{\theta\in\Theta} \frac{1}{N}\sum_{i=1}^{N} \log p(x_i;\theta). \tag{1}$$

In the rest of this paper, we also abbreviate $F_i(\theta) := \log p(x_i;\theta)$ and $F(\theta) := \frac{1}{N}\sum_{i=1}^N F_i(\theta)$. The optimization question of Eq. (1) is then equivalent to $\max_{\theta\in\Theta} F(\theta)$.

The primary focus of this paper is on the *computation* of approximate solutions of the ML estimates (1) under challenging scenarios when the underlying model $p(\cdot;\theta)$ is very complicated and has several undesirable properties. Some important challenges include but are not limited to:

1. **Non-concavity and non-unimodality of log-likelihood**: the log-likelihood function $\log p(x_i;\cdot)$ might not be concave or even uni-modal with respect to the unknown parameter θ,

making "local search" type methods such as mountain-climbing or gradient ascent difficult to find *global* optima of Eq. (1). Instead, it is very likely that these methods would stuck in local minima or saddle points;

2. **Inaccessible first-order information**: many optimization methods, such as gradient ascent or Newton's method, requires access to first-order or even second-order derivatives $\nabla_\theta \log p(x_i;\theta), \nabla^2_\theta \log p(x_i;\theta)$. Unfortunately, for many complicated econometric models, even if such derivatives exist, they still cannot be easily computed in closed forms. This prevents straightforward adoptions of famous continuous optimization algorithms to solve Eq. (1);

3. **Noisy zeroth-order evaluation**: in some scenarios, even the log-likelihood function $\log p(x_i;\theta)$ itself cannot be evaluated or computed without error, given data x_i and a hypothetical parameter θ. For example, in econometric models involving game theoretical process with unknown parameters, the computation of $\log p(x_i;\theta)$ requires multiple Monte-Carlo samples and cannot achieve arbitrary levels of accuracy;

4. **Curse of dimensionality for multiple parameters**: while nonparametric estimation of the log-likelihood function naturally leads to an approximate optimization algorithm for Eq. (1), such an approach suffers from the *curse of dimensionality* when there are more than one parameters (i.e., $d > 1$). Even when d is moderately large (e.g., $d \in [5, 10]$), estimating the entire log-likelihood function could already lead to unacceptable level of computations.

In this paper, we propose an algorithm framework which we call "parallel estimating gradient ascent". Our algorithm has the following properties:

(a) **Parallelism**: the algorithm runs *in parallel* several computing threads simultaneously, with vastly different initial points. This partially avoids the problem of being stuck in local optima/saddle points, as the different computing threads could well lead/converge to different local minima;

(b) **Gradient estimation**: our method *estimates* the first-order derivative of the log-likelihood function only with access to noisy evaluation of $\log p(x_i;\theta)$, making the proposed method applicable to the widest range of econometric models;

(c) **Coordination of computing threads**: instead of running all parallel computing threads with the same pace, we design "selection rules" and "stopping rules" to carefully *coordinate* the different computing threads, so that promising computing threads are devoted with more computing resource and unpromising threads are terminated early without wasting more computing time.

The rest of this paper is organized as follows: In Sec. 2 we give accounts to related works. The description of our proposed algorithm, as well as its several components and convergence guarantees, are given in Sec. 3.

2. Related works

The idea of using iterative methods to solve stochastic optimization questions without first-order information is a well-studied topic in the literature of mathematical optimization, machine learning and computational statistics. The estimating gradient descent/ascent originates from the works of Kiefer & Wolfowitz (1952), Blum (1954), which was later studied and explored in (Nemirovsky & Yudin 1983, Flaxman et al. 2005, Besbes et al. 2015, Agarwal et al. 2010, Shamir 2017). Our proposed approach also resembles "zeroth-order trust region" algorithms, studied in the works of Bandeira et al. (2014), Billups et al. (2013), Spall (1992), Powell (2003), Conn et al. (2009), Chen et al. (2018). The majority of this line of work assumes the objective function to be optimized is *convex* (or concave, for maximization problems). An exception is the work of Ghadimi & Lan (2013), which considered non-convex objectives and studied how fast iterative methods converge to *stationary points* of the said objectives.

Apart from iterative or estimating gradient type methods, many other heuristic algorithms are also well-known for such global optimization questions considered in this paper. Examples include the genetic algorithm (Koza 1997, Whitley 1994), simulated annealing (Van Laarhoven & Aarts 1987), pattern search (Torczon 1997, Lewis & Torczon 1999), as well as modern approaches such as Bayesian optimization (Snoek et al. 2012) and hierarchical optimistic optimization (Bubeck et al. 2011, 2009).

The idea of running several computing threads coordinating them appropriately has also been explored in (Agarwal et al. 2016) in which bandit optimization problems are solved by running multiple candidate algorithms in parallel, and in (Lykouris et al. 2018) to solve multi-armed bandit problems subject to an unknown amount of adversarial corruption.

3. Algorithm description

The pseudo-code description of the proposed algorithm framework is given in Algorithm 1. The algorithm consists of three major components: ThreadCoordination, GradientEstimation and ThreadStopping, which we describe in further details below and in subsequent sections:

- ThreadCoordination: this component aims at the selection (at iteration t) of an active thread $j \in \mathcal{A}_t$, where $\mathcal{A}_t$ is the subset of all active threads at time t (see also the description of ThreadStopping for the interpretation and construction of active subsets). While the uniform distribution $U(\mathcal{A}_t)$ is the most widely used (which spreads the computing resources evenly among all remaining active threads), other distributions could be developed that favor more "promising" threads. Further details and discussion are given in Sec. 3.1.

Algorithm 1 The meta-algorithm framework.

1: Let $\widehat{\theta}_0^{(1)}, \widehat{\theta}_0^{(2)}, \cdots, \widehat{\theta}_0^{(J)}$ be the initial parameter estimates of J computing threads; initialize also $\mathcal{J} = \{1, 2, \cdots, J\}$ as the set of active threads;
2: **for** $\tau = 0, 1, 2, \cdots$ **do**
3: Select $j_\tau \in \mathcal{J}$ using the ThreadCoordination component;
4: Compute $g_\tau = \widehat{\nabla}_\theta F(\widehat{\theta}_\tau^{(j_\tau)})$ using the GradientEstimation component;
5: Perform *projected ascent* step for thread j_τ: $\widehat{\theta}_{\tau+1}^{(j_\tau)} = \mathcal{P}_\Theta(\widehat{\theta}_\tau^{(j_\tau)} - \eta_\tau \widehat{\nabla}_\theta F(\widehat{\theta}_\tau^{(j_\tau)}))$, where η_τ is a certain step size and $\mathcal{P}_G(\cdot) = \arg\min_{\theta \in \Theta} \|\cdot - \theta\|_2$;
6: Eliminate threads in $\mathcal{J}$ using the ThreadStopping component;
7: **end for**

- GradientEstimation: this component aims at the (approximate) computation of the first-order derivative $\nabla_\theta F(\theta)$ using only noisy evaluations of $F(\theta + \delta)$, where $\delta \in \mathbb{R}^d$ is a small d-dimensional perturbation vector. The computation is enabled by first-order Taylor expansions of the objective function F centered at θ. Further details and discussion are given in Sec. 3.2.

- ThreadStopping: this component identifies unpromising threads which are not possible to lead to or converge to good solutions. Such unpromising threads are identified through comparison with the results from other threads, and are subsequently removed from consideration in future iterations. The subset $\mathcal{A}_t$ is defined as the set consisting of all active threads at iteration t, which forms the support of the thread distribution used in the ThreadCoordination component. Further details and discussion are given in Sec. 3.3.

3.1. The ThreadCoordination component

The ThreadCoordination component, as suggested by its name, determines (potentially randomly) at each time stamp t the particular computing thread j to be pursued for the next iteration.

For clarity, we assume the algorithm is currently at time t (i.e., a total number of t observations have already been collected), and thread $j \in [J]$ is at parameter estimate $\widehat{\theta}_t^{(j)}$. We also use $\mathcal{T}_{j,t}(\Delta_t)$ to denote the time stamps for the previous Δ_t times thread j is selected, up to time stamp t. We shall also assume that t is not too small. (When t is very small, meaning that the optimization algorithm just starts, there is very little information regarding the performance of each computing threads and hence the threads to pursue should be selected uniformly at random from all J threads.) We use the following three aspects to decide on which computing thread is to be pursued next:

1. *Current performance*: the *current* performance of each computing thread is of the utmost importance, since a thread that already performs well (i.e., attaining parameter estimates with

good objective values) is likely to further push the performance of the entire algorithm/system. For a particular thread j, its current performance can be estimated as the average objective values over the last Δ_t time stamps at which thread j is selected, or more specifically

$$\mathsf{CP}(j,t) := \frac{1}{\Delta_t} \sum_{\tau \in \mathcal{T}_{j,t}(\Delta_t)} \widehat{F}(\widehat{\theta}_\tau^{(j)});$$

2. *Estimated progress*: a thread that can progress fast (i.e., rapidly increasing the objective value of the parameter estimates) should be pursued more frequently since they will likely deliver fast progress for the overall algorithm/system performance as well. The estimated progress of thread j can be obtained by comparing the its performance for the previous Δ_t and $2\Delta_t$ time stamps during which thread j is selected, or more specifically

$$\mathsf{EP}(j,t) := \frac{1}{\Delta_t} \sum_{\tau \in \mathcal{T}_{j,t}(\Delta_t)} \widehat{F}(\widehat{\theta}_\tau^{(j)}) - \frac{1}{\Delta_t} \sum_{\tau \in \mathcal{T}_{j,t}(2\Delta_t) \setminus \mathcal{T}_{j,t}(\Delta_t)} \widehat{F}(\widehat{\theta}_\tau^{(j)});$$

3. *Volatility*: the "volatility" of a thread measures how stable/volatile of the quality of the parameter estimates obtained by the thread. It can be estimated by the sample standard deviation of the performance for the pervious Δ_t time stamps during which thread j is selected, or more specifically

$$\mathsf{V}(j,t) := \sqrt{\frac{1}{\Delta_t} \sum_{\tau \in \mathcal{T}_{j,t}(\Delta_t)} (\widehat{F}(\widehat{\theta}_\tau^{(j)}) - \mathsf{CP}(j,t))^2}.$$

Once the statistics $\mathsf{CP}(j,t), \mathsf{EP}(j,t), \mathsf{V}(j,t)$ are computed for each thread j, an aggregated statistic $\mathsf{A}(j,t)$ is calculated as

$$\mathsf{A}(j,t) = \kappa_{\mathsf{CP}} \mathsf{CP}(j,t) + \kappa_{\mathsf{EP}} \mathsf{EP}(j,t) + \kappa_{\mathsf{V}} \mathsf{V}(j,t),$$

where $\kappa_{\mathsf{CP}}, \kappa_{\mathsf{EP}}, \kappa_{\mathsf{V}} \geq 0$ are pre-determined weight parameters that carefully balance the three aspects we discussed above. The larger the value of $\mathsf{A}(j,t)$ is, the more promising the thread j is deemed by our algorithm at time t. The thread selection/coordination rule is then designed as

$$\Pr(\text{select thread } j \text{ at time } t) = \frac{\exp\{\mathsf{A}(j,t)\}}{\sum_{j' \leq J} \exp\{\mathsf{A}(j',t)\}},$$

which selects computing thread j at random with probability positively correlated with its score $\mathsf{A}(j,t)$.

3.2. The GradientEstimation component

The Gradient Estimation component aims at the approximate computation of the first-order derivative $\nabla_\theta F(\theta) = \frac{1}{N} \sum_{i=1}^N \nabla_\theta \log p(x_i;\theta)$. A pseudo-code description of GradientEstimation is given in Algorithm 2.

Algorithm 2 The GradientEstimation component/procedure.

1: **Input**: solution point $\theta \in \mathbb{R}^d$, probing radius $\delta > 0$, number of probing points $m \in \mathbb{N}$;
2: **Output**: $\widehat{\nabla}_\theta F(\theta) \in \mathbb{R}^d$, an estimate of $\nabla_\theta F(\theta)$.
3: Sample $u_1, \cdots, u_m$ uniformly at random from the unit sphere $\{u \in \mathbb{R}^d : \|u\|_2 = 1\}$;
4: For each probing vector u_i, $i \in \{0, 1, \cdots, m\}$, collect noisy function evaluation y_j, y_j' at θ and $\theta + \delta u_i$, respectively such that $\mathbb{E}[y_j|\theta, x_j] = F(\theta)$ and $\mathbb{E}[y_j'|\theta, x_j] = F(\theta + \delta u_j)$;
5: Find $\widehat{\nabla}_\theta F(\theta)$ as the least-squares estimation

$$\widehat{\nabla}_\theta F(\theta) = \arg\min_{g \in \mathbb{R}^d} \frac{1}{m} \sum_{j=1}^{m} \Big| \frac{y_j' - y_j}{\delta} - \langle u_j, g \rangle \Big|^2.$$

As motivated in the introduction, such first-order derivatives $\nabla_\theta F(\theta)$ cannot be directly computed in closed forms. Instead, given a hypothetical parameter θ' (possibly different from θ at which the derivative $\nabla_\theta F(\theta)$ is sought), one can compute a *noisy* evaluation of $F(\theta') = \frac{1}{N} \sum_{i=1}^{N} \log p(x_i; \theta')$. With many such noisy evaluations at different "probing" positions θ', a local linear model can be constructed via first-order Taylor expansions and least-squares estimators are employed to find an estimator of $\nabla_\theta F(\theta)$.

The following lemma gives an upper bound on the estimation error of $\widehat{\nabla}_\theta F(\theta)$ under the assumption that the gradients of the log-likelihood to be estimated, $\nabla F(\cdot)$, is Lipschitz continuous.

Lemma 1 *Suppose $\nabla_\theta F(\cdot)$ is L-Lipschitz continuous, meaning that $\|\nabla_\theta F(\theta) - \nabla_\theta F(\theta')\|_2 \le L\|\theta - \theta'\|_2$ for all θ, θ'. Suppose also that $\mathrm{Var}(y_j), \mathrm{Var}(y_j') \le \sigma^2$ for some $\sigma > 0$, and $m \ge 8d \ln(d/\delta)$ for some $\delta \in (0, 1/2)$. Then with probability at least $1 - \delta$, the estimation error $\widehat{\nabla}_\theta F(\theta) - \nabla_\theta F(\theta)$ can be decomposed as*

$$\widehat{\nabla}_\theta F(\theta) - \nabla_\theta F(\theta) = \beta + \zeta,$$

where $\|\beta\|_2 \le L\delta$ almost surely and $\zeta \in \mathbb{R}^d$ is a random vector satisfying $\mathbb{E}[\zeta|\theta] = 0$ and $\mathbb{E}[\zeta^\top \zeta|\theta] \le 4d^2/(m\delta^2)$.

Proof of Lemma 1. For notational simplicity denote $y_j = F(\theta) + \varepsilon_j$ and $y_j' = F(\theta + \delta u_j) + \varepsilon_j'$ where $\mathbb{E}[\varepsilon_j|\theta, u_j] = \mathbb{E}[\varepsilon_j'|\theta, u_j] = 0$ and $\mathrm{Var}[\varepsilon_j|\theta, u_j], \mathrm{Var}[\varepsilon_j'|\theta, u_j] \le \sigma^2$. By the mean-value theorem, there exists $\widetilde{u}_j = \lambda \delta u_j$ for some $\lambda \in (0, 1)$ such that

$$\begin{aligned} y_j' - y_j &= F(\theta + \delta u_j) - F(\theta) + (\varepsilon_j - \varepsilon_j') = \langle \nabla_\theta F(\theta + \widetilde{u}_j), \delta u_j \rangle + (\varepsilon_j' - \varepsilon_j) \\ &= \delta \langle \nabla_\theta F(\theta), u_j \rangle + \delta \langle \nabla_\theta F(\theta + \widetilde{u}_j) - \nabla_\theta F(\theta), u_j \rangle + (\varepsilon_j' - \varepsilon_j). \end{aligned}$$

Dividing both sides of the above equality by δ, we obtain

$$\frac{y'_j - y_j}{\delta} = \langle \nabla_\theta F(\theta), u_j \rangle + \underbrace{\langle \nabla_\theta F(\theta + \widetilde{u}_j) - \nabla_\theta F(\theta), u_j \rangle}_{:=b_j} + \underbrace{\delta^{-1}(\varepsilon'_j - \varepsilon_j)}_{:=s_j}. \tag{2}$$

Next, define $X = (u_1; u_2; \cdots, u_m) \in \mathbb{R}^{m \times d}$ as an $m \times d$ matrix with each row corresponding to a probing vector u_j; $z, b, s \in \mathbb{R}^n$ as n-dimensional vectors with $z_j = (y'_j - y_j)/\delta$, $b_j = \langle \nabla_\theta F(\theta + \widetilde{u}_j) - \nabla_\theta F(\theta), u_j \rangle$, $s_j = (\varepsilon'_j - \varepsilon_j)/\delta$ for $j = 1, 2, \cdots, n$. The estimate $\widehat{\nabla}_\theta F(\theta)$ can then be written as $\widehat{\nabla}_\theta F(\theta) = (X^\top X)^{-1}(X^\top z)$. In addition, $z = X \nabla_\theta F(\theta) + b + s$ thanks to Eq. (2). Subsequently,

$$\widehat{\nabla}_\theta F(\theta) - \nabla_\theta F(\theta) = (X^\top X)^{-1} X^\top (b + s) = (X^\top X)^{-1} X^\top b + (X^\top X)^{-1} X^\top s. \tag{3}$$

Define $\beta := (X^\top X)^{-1} X^\top b$. By Lemma 5, we know that $\|(X^\top X)^{-1}\|_{\text{op}} \le 2d/m$ with probability $1 - \delta$. Therefore,

$$\|(X^\top X)^{-1} X^\top b\|_2 \le \|(X^\top X)^{-1}\|_{\text{op}} \|X\|_{\text{op}} \times \sqrt{m} \|b\|_\infty \le \frac{2d}{m} \times \sqrt{m} \times \sqrt{m} \times \|b\|_\infty \le 2d \|b\|_\infty,$$

with probability $1 - \delta$, where $\|b\|_\infty = \max_j |b_j|$. Because $\nabla_\theta F(\theta)$ is L-Lipschitz continuous, $|b_j|$ can be upper bounded by

$$\begin{aligned} |b_j| &= \left| \langle \nabla_\theta F(\theta + \widetilde{u}_j) - \nabla_\theta F(\theta), u_j \rangle \right| \le \|\nabla_\theta F(\theta + \widetilde{u}_j) - \nabla_\theta F(\theta)\|_2 \times \|u_j\|_2 \\ &\le L \times \|\widetilde{u}_j\|_2 \times \|u_j\|_2 \le L\delta, \end{aligned}$$

where the last inequality holds because $\|\widetilde{u}_j\|_2 = \lambda\delta \|u_j\|_2 \le \delta$ since $\|u_j\|_2 = 1$ and $0 < \lambda < 1$. Subsequently, we have with probability 1 that

$$\|\beta\|_2 = \|(X^\top X)^{-1} X^\top b\|_2 \le 2Ld\delta. \tag{4}$$

We next establish a co-variance upper bound on $\zeta = (X^\top X)^{-1} X^\top s$. It should be noted that each $s_j = (\varepsilon_j - \varepsilon'_j)/\delta$ is a centered, independent random variable with variance upper bounded by $\mathbb{E}[s_j^2 | \theta] \le 2/\delta^2$. Subsequently, we have

$$\mathbb{E}[\zeta^\top \zeta | \theta] = \mathbb{E}[s^\top X (X^\top X)^{-2} X^\top s | \theta] = \frac{2}{\delta^2} \text{tr}\left[X (X^\top X)^{-2} X^\top\right] \le \frac{2}{\delta^2} \times d \times \frac{2d}{m} = \frac{4d^2}{m\delta^2}. \tag{5}$$

Combining Eqs. (4,5) we complete the proof of Lemma 1.

3.3. The ThreadStopping component

In the ThreadStopping component, we discuss rules for *stopping* a computing thread j if it is deemed to be not promising, either unable or too time-expensive to converge to a good-quality parameter estimate. While the proposed rules are heuristics in nature, we prove under certain local concavity assumptions that these proposed stopping rules are *conservative* in the sense that, with high probability, they will not remove a promising computing thread by mistake.

We describe the two major stopping rules considered in the ThreadStopping component, which can be categorized at a higher level as "first-order" and "second-order" rules.

3.3.1. First-order stopping rule Suppose thread j is currently at a parameter estimate θ_j, with gradient estimate $\widehat{\nabla}F(\theta_\tau^{(j)}) \approx \nabla F(\theta_\tau^{(j)})$. Suppose also that an estimate $\widehat{F}(\theta_\tau^{(j)}) \approx F(\theta_\tau^{(j)})$ is obtained by simply averaging all observations of y_j in Algorithm 2. The thread j should be terminated, then, if there exists another thread $j' \neq j$ such that

$$\textbf{Stopping rule 1:} \quad \widehat{F}(\theta_\tau^{(j)}) + D \times \|\widehat{\nabla}F(\theta_\tau^{(j)})\|_2 \leq \widehat{F}(\theta_\tau^{(j')}).$$

Here, $D > 0$ is a tuning parameter for the stopping rule and also the optimization algorithm, with larger D values indicating more aggressive (and hence less "safe") stopping rule for computing threads.

Intuitively, the stopping rule obtains an "over-estimate" of the likelihood function F on the solution thread j can potentially converge to, on the left-hand side of the stopping rule. The intuition is that if $\widehat{F}(\theta_\tau^{(j)})$ is already small, it means that the function will change very slowly in a neighborhood of $\theta_\tau^{(j)}$ and therefore the estimating gradient ascent procedure is unlikely to advance/improve the likelihood objective significantly in thread j.

Below we state a local concavity condition and shows that, with the condition held and the estimates $\widehat{F}, \widehat{\nabla}F$ being accurate, the proposed stopping rule is "safe" in the sense that it will only remove computing threads impossible to obtain better parameter estimates. This is further accomplished by showing that, $\widehat{F}(\theta_\tau^{(j)}) + D \times \|\widehat{\nabla}F(\theta_\tau^{(j)})\|_2$, under the considered circumstances, is an upper bound on how large $F(\theta_j^*)$ could potentially be.

Condition 1 (Local concavity) *For every computing thread j let θ_j^* be the parameter the thread converges to. There exists a convex neighborhood U_j containing θ_j^* with diameter $\sup_{x,x' \in U_j} \|x - x'\|_2 \leq D$, such that F is* concave *in U_j, meaning that*

$$F(\lambda x + (1-\lambda)x') \geq \lambda F(x) + (1-\lambda)F(x'), \quad \forall x, x' \in U_j,\ \lambda \in [0,1].$$

Lemma 2 *Suppose F is twice differentiable, condition 1 holds and $\theta_\tau^{(j)} \in U_j$. Then $F(\theta_j^*) \leq F(\theta_\tau^{(j)}) + D\|\nabla F(\theta_\tau^{(j)})\|_2$.*

Proof of Lemma 2. Because F is twice differentiable and locally concave on U_j, we know that $\nabla^2 F(\theta) \preceq 0$ for all $\theta \in U_j$. Using second-order Taylor expansion of $F(\theta_j^*)$ at $\theta_\tau^{(j)}$ with Lagrangian remainders, it holds that

$$F(\theta_j^*) = F(\theta_\tau^{(j)}) + \langle \nabla F(\theta_\tau^{(j)}), \theta_j^* - \theta_\tau^{(j)} \rangle + \frac{1}{2}(\theta_j^* - \theta_\tau^{(j)})^\top \nabla^2 F(\widetilde{\theta})(\theta_j^* - \widehat{\theta}_\tau^{(j)}),$$

where $\widetilde{\theta} = \lambda\theta_j^* + (1-\lambda)\theta_\tau^{(j)}$ for some $\lambda \in (0,1)$, and $\widetilde{\theta} \in U_j$ since U_j is a convex domain. This implies that $\nabla^2 F(\widetilde{\theta}) \preceq 0$ and therefore $(\theta_j^* - \theta_\tau^{(j)})^\top \nabla^2 F(\widetilde{\theta})(\theta_j^* - \widehat{\theta}_\tau^{(j)}) \leq 0$. Subsequently,

$$F(\theta_j^*) \leq F(\theta_\tau^{(j)}) + \|\nabla F(\theta_\tau^{(j)})\|_2 \times \|\theta_j^* - \theta_\tau^{(j)}\|_2 \leq F(\theta_\tau^{(j)}) + D\|\nabla F(\theta_\tau^{(j)})\|_2,$$

where the first inequality is by the Cauchy-Schwarz inequality and the second inequality holds because $\|\theta_j^* - \theta_\tau^{(j)}\|_2 \leq D$.

3.3.2. Second-order stopping rule The first stopping rule we developed in the previous section could be strengthened if the likelihood objective F has finer properties locally around θ_j^*. In this section we consider a second-order stopping rule, which stops a particular thread j if there exists another thread $j' \neq j$ such that

$$\textbf{Stopping rule 2:} \quad \widehat{F}(\theta_\tau^{(j)}) + \frac{1}{2\alpha} \times \|\widehat{\nabla} F(\theta_\tau^{(j)})\|_2^2 \leq \widehat{F}(\theta_\tau^{(j')}).$$

Comparing the stopping rule 2 with stopping rule 1, the major difference is the *squared* ℓ_2-norm of the estimated gradients of F at $\widehat{\theta}_\tau^{(j)}$. Inuitively speaking, this rule is more "aggressive" than stopping rule 1, since when thread j approaches θ_j^* it converges to, the gradients would be close to zero and therefore $\|\widehat{\theta}_\tau^{(j)}\|_2^2$ would be much smaller than $\|\widehat{\theta}_\tau^{(j)}\|_2$.

Below we state a local *strong-concavity* condition, which is stronger than Condition 1 for stopping rule 1. We then show, in Lemma 3 below, that under the stronger condition a "clearner" version of stopping rule 2 will not remove computing threads that are still relevant.

Condition 2 (local strong-concavity) *For every computing thread j let θ_j^* be the parameter the thread converges to. There exists a convex neighborhood U_j containing θ_j^* with diameter $\sup_{x,x' \in U_j} \|x - x'\|_2 \leq D$, such that F is* α-strongly concave *in U_j, meaning that*

$$\nabla^2 F(\theta) \preceq -\alpha I, \quad \forall \theta \in U_j.$$

To see why Condition 2 is strong than Condition 1, recall that a twice-differentiable function f is concave on U if $\nabla^2 f(x) \preceq 0$ for all $x \in U$. This is weaker than Condition 2 with some $\alpha > 0$.

Lemma 3 *Suppose F is twice differentiable, condition 2 holds and $\theta_\tau^{(j)} \in U_j$. Then $F(\theta_j^*) \leq F(\theta_\tau^{(j)}) + \frac{1}{2\alpha}\|\nabla F(\theta_\tau^{(j)})\|_2^2$.*

Proof of Lemma 3. Using second-order Taylor expansion of $F(\theta_j^*)$ at $\theta_\tau^{(j)}$ with Lagrangian remainders, it holds that

$$F(\theta_j^*) = F(\theta_\tau^{(j)}) + \langle \nabla F(\theta_\tau^{(j)}), \theta_j^* - \theta_\tau^{(j)} \rangle + \frac{1}{2}(\theta_j^* - \theta_\tau^{(j)})^\top \nabla^2 F(\widetilde{\theta})(\theta_j^* - \widehat{\theta}_\tau^{(j)}),$$

where $\widetilde{\theta} = \lambda\theta_j^* + (1-\lambda)\theta_\tau^{(j)}$ for some $\lambda \in (0,1)$, and $\widetilde{\theta} \in U_j$ since U_j is a convex domain. Because $\nabla^2 F(\widetilde{\theta}) \preceq -\alpha I$ and $|\nabla F(\theta_\tau^{(j)}), \theta_j^* - \theta_\tau^{(j)})| \leq \|\nabla F(\theta_\tau^{(j)})\|_2 \times \|\theta_j^* - \theta_\tau^{(j)}\|_2$ thanks to the Cauchy-Schwarz inequality, we conclude that

$$F(\theta_j^*) \leq F(\theta_\tau^{(j)}) + \|\nabla F(\theta_\tau^{(j)})\|_2 \times \|\theta_j^* - \theta_\tau^{(j)}\|_2 - \frac{\alpha}{2}\|\theta_j^* - \theta_\tau^{(j)}\|_2^2.$$

Completing the squares, we obtain

$$\begin{aligned} F(\theta_j^*) - F(\theta_\tau^{(j)}) &\leq -\frac{\alpha}{2}\|\theta_j^* - \theta_\tau^{(j)}\|_2^2 + \|\nabla F(\theta_\tau^{(j)})\|_2 \times \|\theta_j^* - \theta_\tau^{(j)}\|_2 \\ &= -\frac{\alpha}{2}\left(\|\theta_j^* - \theta_\tau^{(j)}\|_2 - \frac{\|\nabla F(\theta_\tau^{(j)})\|_2}{\alpha}\right)^2 + \frac{\|\nabla F(\theta_\tau^{(j)})\|_2^2}{2\alpha} \\ &\leq \frac{\|\nabla F(\theta_\tau^{(j)})\|_2^2}{2\alpha}. \end{aligned}$$

Re-arranging the terms we have $F(\theta_j^*) - F(\theta_\tau^{(j)}) \leq \frac{\|\nabla F(\theta_\tau^{(j)})\|_2^2}{2\alpha}$, which is to be demonstrated.

4. Numerical results on synthetic data

In this section we report numerical results of our proposed algorithm on the optimization task of a synthetic function. We consider the problem of maximizing a 5-dimensional non-concave function with multiple local maxima and saddle points. To construct such a function, we use the probability-density function of a Gaussian Mixture Model (GMM). More specifically, we consider the following objective function

$$f(x) = \sum_{k=1}^{5} \frac{1}{\sqrt{2\pi\sigma_k^2}} \exp\left\{-\frac{\|x - \mu_k\|_2^2}{2\sigma_k^2}\right\},$$

with $\mu_k = e_k \in \mathbb{R}^5$ being the coordinate basis functions, $\sigma_k = 1.0$ for $k \in \{1,2,3,4\}$ and $\sigma_k = 0.5$ for $k = 0.5$. The construction of the objective function f ensures that it has at least five local minima with similar values, with the local minima tilting towards the last component $k = 5$ being comparably higher due to its smaller variance. Hence, if an (estimating) gradient descent algorithm is initialized near the first for components it will be attracted to the first four local maxima first before escaping and turning towards the final global maxima near the last component.

In Figure 1 we report the convergence of our proposed algorithm with five threads, initialized to solutions close to each of the component centers μ_k defined in the objective function. Both

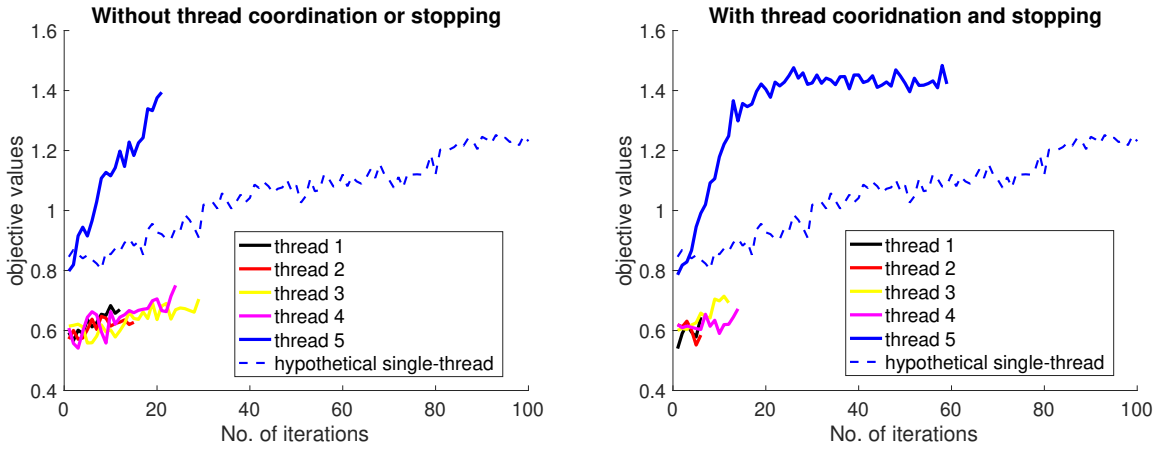

Figure 1 **Convergence of our proposed algorithm with five threads. Details of the figures and the algorithms being implemented are given in the main text.**

algorithms are run for a total of $T = 100$ gradient evaluations (total number of time periods across all 5 threads), with each gradient evaluation taking $m = 20$ random samples with $\delta = 0.1$ probing radius. In the left panel of Figure 1, both the thread coordination and thread stopping components are disabled (meaning that each time we select a thread uniformly at random, and no thread is terminated early); in the right panel of Figure 1, the thread coordination (thread sampling) component is activated with parameters $\kappa_{\mathsf{CP}} = \kappa_{\mathsf{EP}} = \kappa_{\mathsf{V}} = 1$ with history window $\Delta_t = 100$, and the thread stopping component is activated with rule $\widehat{F}(\theta_\tau^{(j)}) + 0.5\|\theta_\tau^{(j)}\|_2 \leq \widehat{F}(\theta_\tau^{(j')})$. In both plots of Figure 1, we also report in the dashed blue curve the objective function values of a hypothetical single-thread estimating gradient descent method, initialized at a point close to a sub-optimal component center μ_k, $k < 5$.

From Figure 1, it is clear that our proposed algorithm with multiple threads (more precisely 5 threads in this experiment) outperform its single-thread version significantly, with the same number of function value/gradient evaluations ($T = 100$). Furthermore, the right panel of Figure 1 shows that our thread coordination and thread stopping components could quickly identify sub-optimal threads (those marked with red, yellow, black and magenta curves) and stop them, while the same multi-thread optimization algorithm without thread coordination or stopping is forced to almost evenly distribute the computation among the five threads, wasting computation on unpromising threads and thereby slowing the overall progress of the algorithm.

We also report the global (overall) convergence of the proposed algorithms in Figure 2. For the single-thread curve (the dashed black curve) Figure 2 coincides with Figure 1. For the other two curves corresponding to five threads, we report the best objective value the algorithm attains after *a total of* $t = 1, 2, \cdots, 100$ gradient/function value evaluations are made. This gives the reader a

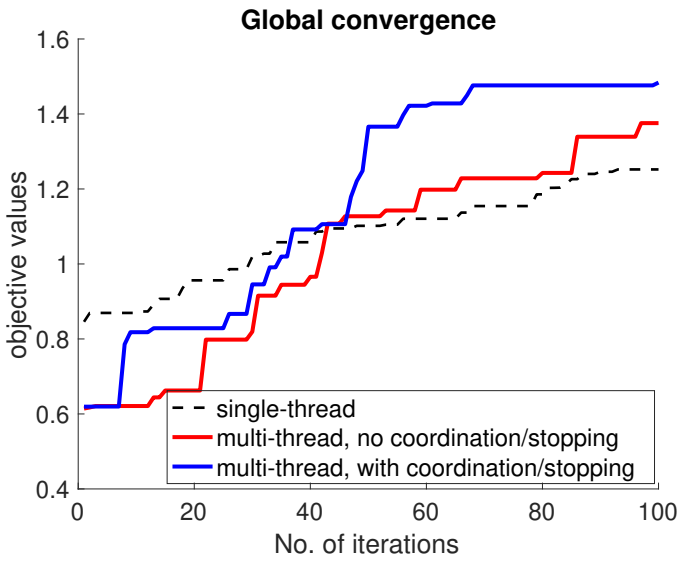

Figure 2 Global (overall) convergence of the proposed algorithm for single-thread, five-thread without coordination/stopping and five-thread with coordination and stopping. Further details are given in the main text.

more intuitive picture of the efficiency of the proposed optimization algorithms. As we can see, the algorithm with both thread coordination and thread stopping components activated (the solid blue curve) converges to higher objective values much faster than the same algorithm with the same number of threads, but with neither thread coordination nor thread stopping rules (the solid red curve). This is because with thread coordination (sampling the more promising threads more frequently) and thread stopping (terminating sub-optimal threads early to not waste more computation time on these threads), the algorithm allows more samples/computation to be spent on the promising thread so that the convergence speed of the algorithm is much faster.

5. Numerical results on airline pricing data

In this section, we apply our proposed optimization method to a real-world airline pricing dataset and compare its performance with benchmark heuristics optimization algorithms, including the *genetic algorithm* and the *pattern search* algorithm. Our proposed algorithm is implemented in C++, while both benchmark methods are implemented in Matlab.

5.1. Background: hidden city ticketing

Modern airlines operator primarily two types of flight networks: the hub-and-spoke network, which designated a handful of airports as *hubs* and route most of the flights from all airports to major hubs; and the fully-connected network, which operates direct flight in a point-to-point manner. The hub-and-spoke network is adopted by major airlines such as the United airlines and the Delta

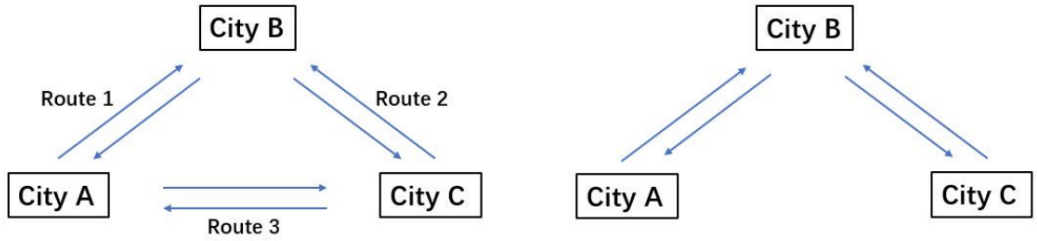

Figure 3 **Illustration of the fully-connected (FC) and hub-and-spoke (HS) structures among three airports.**

airlines, while the fully-connected approach is mostly used by smaller, low-cost carriers such as Southwest and JetBlue.

In hub-and-spoke network, many flights between non-hub airports are carried out using connecting flights. For example, the flight from Pittsburgh to Boston could be direct/non-stop, but most likely it needs connection at a New York airport. Naturally, connecting flights are priced (sometimes significantly) lower compared to direct flights due to the additional connection and extended travel time. In some extreme cases, the price of the indirect flight (e.g., Pittsburgh to Boston connecting via New York) might be even lower than the first-leg of the flight (Pittsburgh to New York non-stop). This creates the possibility of *hidden-city ticketing*, a practice that could significantly reduce the revenue/profits of the airlines.

More specifically, consider a traveller who wishes to travel from airport A to airport B. The practice of hidden-city ticketing is defined as the same traveller purchasing a ticket from airport A to airport C, connecting at airport B. The traveller would then proceed with only the first leg of her purchased ticket, essentially traveling from A to B on a non-stop flight. Clearly, such hidden-ticketing behavior is only practiced if the traveller knows about the practice, and furthermore the price of a flight ticket from A to C connecting at B is strictly lower than the price of a non-stop ticket directly from A to B.

5.2. Model formulation and maximum likelihood estimation

We collected commercial flights operated among the 133 busiest commercial service airports in the United States (identified by the FAA) with 96.34% of total passenger enplanements, 16,142 routes and 2,822,086 itineries.

For every three distinct airport tuples (A, B, C), with flights connecting (A, B) and (A, C), the flight network can be categorized as either *fully-connected (FC)*, if there are non-stop flights between (A, C), or *hub-and-spoke (HS)* if there are no direct flights between A and C. Figure 3

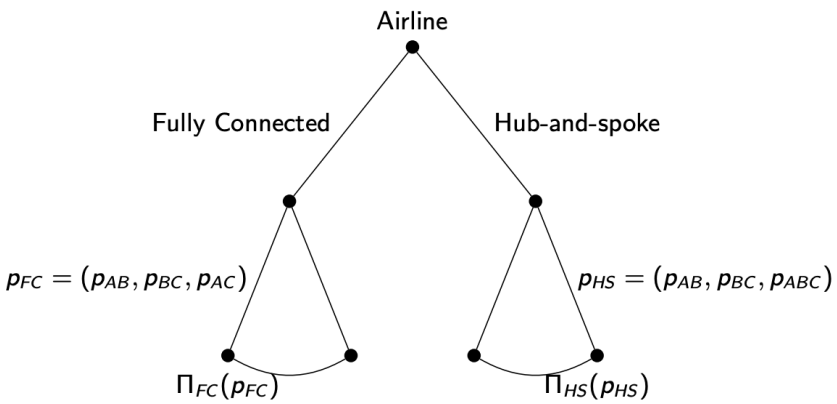

Figure 4 The Stackelberg game description of the airlines' network choices and travelers' riding behaviors.

gives a graphical illustration of the FC and/or HS network structures among the airports A, B and C. We use d_{AB}, d_{BC}, d_{AC} to denote the distances between pairs of airports, and p_{AB}, p_{BC}, p_{AC} for ticket prices of *direct* flights among the airports. Additionally, we use p_{ABC} for the ticket price of the indirect flight from A to C connecting at B.

We make the following assumptions on the supply side (the airlines):

1. There is only one airline serving the three cities, thus the firm charges monopoly airfares;

2. Aircrafts are assumed to have an unlimited capacity, thus there is one flight on each route. C_2 denote the airline's cost per mile on any route j;

3. Direct flight has a quality of q_h per mile and indirect flight has a quality of q_l per mile, with $0 < q_l < q_h < 1$.

We also assume the following on the demand side (the travelers):

1. Each individual i has a time preference parameter of λ_i, obtaining utility $C_1 e^{\lambda_i} q d - p$ from consuming a good of quality q, and 0 if he/she does not fly;

2. On each route j, the distribution of consumers' time preferences $\lambda_{ij} \sim \mathcal{N}(\theta_j, \sigma_1^2)$;

3. For passengers flying from A to B, the fraction of passengers being aware of hidden city opportunity is δ and the fraction of uninformed passengers is $1 - \delta$;

4. When hidden city opportunity exists (i.e., $p_{AB} > p_{ABC}$), informed passengers will pay p_{ABC} instead, while uninformed passengers will still pay p_{AB};

5. Amount of passengers on each route j are normalized to 1.

Based on the above assumptions, we use a Stackelberg game to characterize the airlines' network choices and the travelers' riding behaviors, as shown in Figure 4.

We next formulate the maximum likelihood estimation question that we aim to solve. The problem is centered on the parameter of interest δ, representing the portion of travelers who

are aware and would be willing to exploit the benefits of hidden city ticketing. Other nuisance parameters also needed to be estimated include C_1 (the travelers' average utility), C_2 (the airlines' average cost), σ_1 (standard deviation of λ_{ij}), σ_2 (standard deviation of $\theta_j \sim \mathcal{N}(\mu_j, \sigma_2^2)$), q_h and q_l (utility multipliers for different types of travelers). Mathematically, we use

$$\zeta = (\delta, C_1, C_2, \sigma_1, \sigma_2, q_h, q_l) \in \mathbb{R}^7$$

to denote a vector of parameter values.

For each airports tuple (A_i, B_i, C_i) in the collected data such that there are at least one flights between (A_i, B_i) and (B_i, C_i), respectively, we use

$$y_i \in \{0, 1\}$$

to denote the airline's actual choice of FC or HS structures (i.e., $y_i = 1$ if there are direct flights between (A_i, C_i) and $y_i = 0$ otherwise). The accessible data for airport tuple (A_i, B_i, C_i) are represented as

$$x_i = (p_{AB}, p_{BC}, p_{AC}, p_{ABC}, d_{AB}, d_{BC}, d_{AC}, \mu_B, \mu_C) \in \mathbb{R}^9,$$

where $p_{AB}, p_{BC}, p_{AC}, p_{ABC}$ are flight ticket prices, d_{AB}, d_{BC}, d_{AC} are distances between airports, and μ_B, μ_C are expected values of θ_B, θ_C which are obtained via analyzing the typical flows of flights/passengers for each of the 133 airports in the data set.

With $\theta_B \sim \mathcal{N}(\mu_B, \sigma_2^2)$ and $\theta_C \sim \mathcal{N}(\mu_C, \sigma_2^2)$ realized, the expected profit for the airline can be computed as follows. Let $\Phi(z; \mu, \sigma^2) = \int_0^z \frac{1}{\sqrt{2\pi\sigma^2}} \exp\{-\frac{(t-\mu)^2}{2\sigma^2}\} \mathrm{d}t$ be the cumulative density function (CDF) of $\mathcal{N}(\mu, \sigma^2)$. First, in the case of Fully-Connected (FC) network, the expected profit is

$$\Pi_{FC} = \Pi_{AB} + \Pi_{BC} + \Pi_{AC} \quad \text{where } \Pi_{XY} = p_{XY} \left[1 - \Phi\left(\ln\left(\frac{p_{XY}}{C_1 q_h d_{XY}} \right); \theta_Y, \sigma_1^2 \right) \right] - C_2 d_{XY}.$$

In the case of Hub-and-Spoke (HS) network, there are two cases. The first case is $p_{AB} < p_{ABC}$, in which there is no hidden-city ticketing opportunities. The airline's profit is then expressed as

$$\Pi_{HS} = \Pi_{AB} + \Pi_{BC} + R^l_{ABC} \quad \text{where } R^l_{ABC} = p_{ABC} \left[1 - \Phi\left(\ln\left(\frac{p_{ABC}}{C_1 q_l (d_{AB} + d_{AC})} \right); \theta_C, \sigma_1^2 \right) \right].$$

Note that, because the direct flight between A and C is no longer operated in a hub-and-spoke network, and therefore R^l_{ABC} is a pure profiting term.

Finally, in the case of $p_{AB} \geq p_{ABC}$, there is potential profit loss due to hidden-city ticketing. The airline's profit is

$$\Pi_{HS} = (1-\delta)\Pi_{AB} + \Pi_{BC} + R^l_{ABC} + \delta R^h_{ABC} \text{ where } R^h_{ABC} = p_{ABC} \left[1 - \Phi\left(\ln\left(\frac{p_{ABC}}{C_1 q_h d_{AB}} \right); \theta_B, \sigma_1^2 \right) \right],$$

Table 1 **Results for our proposed algorithm on the airline pricing data, with $D = 1/2\alpha \in \{1.0, 0.5, 0.3\}$. A $\times$ means that the particular thread is not active at the end of the optimization.**

	$D = 1/2\alpha$	log. likeli.	δ	C_1	C_2	σ_1	σ_2	q_h	q_l	running time (s)
Thread #1	1.0	-0.69	0.01	6.33	0.76	0.69	0.82	0.53	0.10	108.9
	0.5	-0.69	0.02	6.34	0.78	0.72	0.85	0.52	0.10	107.7
	0.3	-0.68	0.02	6.33	0.78	0.72	0.87	0.52	0.10	108.3
Thread #2	1.0	-0.73	0.49	11.8	0.95	0.35	0.86	0.54	0.12	108.9
	0.5	-0.71	0.50	11.8	0.94	0.35	0.93	0.52	0.12	107.7
	0.3	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$
Thread #3	1.0	-0.71	0.06	10.3	0.74	0.37	0.96	0.49	0.10	108.9
	0.5	-0.70	0.07	10.3	0.74	0.39	0.99	0.47	0.11	107.7
	0.3	-0.69	0.07	10.3	0.76	0.43	1.0	0.47	0.11	108.3
Thread #4	1.0	-0.68	0.50	10.9	0.95	0.72	0.83	0.34	0.10	108.9
	0.5	-0.69	0.49	10.9	0.96	0.72	0.82	0.33	0.12	107.7
	0.3	-0.67	0.48	10.9	0.91	0.73	0.86	0.33	0.12	108.3
Thread #5	1.0	-0.71	0.43	12.0	0.96	0.59	0.77	0.36	0.11	108.9
	0.5	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$
	0.3	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$

where $\delta \in (0,1)$ is the parameter of interest corresponding to the portion of travelers engaged in the hidden-city ticketing practice.

Given tuple x_i and the observed airline's network choice $y_i \in \{0,1\}$, the log-likelihood of y_i conditioned on x_i and parameter ζ can be written as

$$\log P(y_i|x_i;\zeta) = y_i \log \Pr_{\theta_{B_i},\theta_{C_i}}\left[\Pi_{FC} > \Pi_{HS}\right] + (1-y_i)\log \Pr_{\theta_{B_i},\theta_{C_i}}\left[\Pi_{HS} \geq \Pi_{FC}\right]. \tag{6}$$

The maximum-likelihood estimation problem is then formulated as

$$\arg\max_{\zeta} \frac{1}{N}\sum_{i=1}^{N} \log P(y_i|x_i;\theta), \tag{7}$$

where N is the total number of airport tuples (A,B,C) available in the data collected.

5.3. Results

Before presenting the computational results, we first mention some important implementation details. First, because the distances d_{XY} are measure in miles and could vary drastically, we adopt a re-normalization transform $d_{XY} \mapsto \sqrt{d_{XY}}$ to alleviate the scales of the distances. We also note that the log-likelihood in Eq. (6) cannot be evaluated directly because the $\Pr_{\theta_{B_i},\theta_{C_i}}\left[\Pi_{FC} > \Pi_{HS}\right]$ terms do not admit easy closed-form expression. Instead, we use Monte-Carlo sampling with M_{MC} samples to approximately compute the log-likelihood. Finally, because the collected data consist of

Table 2 Results for the genetic algorithm (`ga`) and the pattern search algorithm (`patternsearch`). Each algorithm terminates only when the designated time limit is reached. $M_{\mathrm{MB}} = M_{\mathrm{MC}}$ are the mini-batch sizes and the number of Monte-Carlo samples, respectively.

	M_{MB}	M_{MC}	log. likeli	δ	C_1	C_2	σ_1	σ_2	q_h	q_l	running time (s)
Genetic algorithm	200	200	-0.73	0.30	5.03	0.68	0.61	0.86	0.85	0.27	100
	200	200	-0.75	0.28	5.36	0.73	0.58	0.78	0.82	0.28	300
	500	500	-0.69	0.31	3.66	0.70	0.81	0.95	0.93	0.21	600
Pattern search	200	200	-0.82	0.13	7.5	0.10	0.10	0.85	1.0	0.10	100
	200	200	-1.01	0.0	8.0	0.10	0.10	0.60	1.0	0.10	300
	500	500	-0.76	0.35	0.34	0.35	0.51	0.76	0.58	0.16	600

many airport tuples (i.e., $N > 10^5$), we use a "mini-batch" approach when evaluating the objective function in Eq. (7). More specifically, we randomly sample $M_{\mathrm{MB}} \ll N$ airport tuples and use the average log-likelihood on the randomly sampled mini-batch data to approximate the log-likelihood of the objective function on the entire data set.

We also impose the following constraints on each of the parameters in ζ to be estimated: $\delta \in (0, 1/2]$, $C_1 \in [1, 20]$, $C_2 \in [0.1, 1]$, $\sigma_1 \in [0.1, 1]$, $\sigma_2 \in [0.1, 1]$ and $0.1 \leq q_l \leq q_h \leq 1$. These constraints are imposed to ensure the values of the unknown parameters are practically feasible and reasonable.

We first present computational results for our proposed parallel stochastic gradient ascent algorithm. Algorithmic parameters are set as $\kappa_{\mathsf{CP}} = \kappa_{\mathsf{EP}} = \kappa_{\mathsf{V}} = 1$ for the ThreadCoordination component, $m = 20$, $\delta = 0.05$ for the GradientEstimation component, and $D = 1/2\alpha \in \{1.0, 0.5, 0.3\}$ for the ThreadStopping component. The algorithm is run for $T = 200$ iterations, 5 initial threads with random initializations, and with the log-likelihood of the final solution of each active threads being reported in Tables 1.

As we can see from Table 1, at the end of the optimization there are 5 active threads if $D = 1/2\alpha = 1.0$, 4 active threads if $D = 1/2\alpha = 0.5$ and 3 active threads if $D = 1/2\alpha = 0.3$. This is intuitive because smaller $D = 1/2\alpha$ values indicate a more aggressive thread elimination policy, thereby leading to fewer active threads. In the core three threads (Threads #1, 3, 4) the log-likelihood is consistently below -0.7, indicating likely scenarios in the real world. Our algorithm takes roughly 100 seconds to complete an optimization.

We next compare the results obtained by our proposed algorithm with two baseline methods widely used in global optimization: the *genetic algorithm* and the *pattern search* algorithm. Both algorithms have been implemented in Matlab in standard packages, via the `ga` and the `patternsearch` routine. The results for both algorithms are reported in Table 2, with their corresponding running times. Note that the running times are set prior to each run, and the optimization

algorithms simply terminate once the time budgets are reached. Table 2 shows that, both the genetic algorithm and the pattern search algorithm take much longer time (around or even more than 10 minutes) to converge to less optimal solutions compared to the ones found by our proposed algorithm in 100 seconds. This demonstrates the effectiveness and efficiency of our proposed approach.

6. Conclusion

In this paper, we propose a general framework of optimizing functions with noisy function evaluations. The proposed framework is based on running multiple threads of stochastic estimating gradient ascent algorithms in parallel, and to carefully coordinate the different computing threads. Theoretical analysis and justifications are given for the stopping rules used in the proposed method, and numerical results on both synthetic data and a real airline industry data set are provided to corroborate the effectiveness and efficiency of our proposed methods.

Appendix: additional lemmas and proofs

The following lemma is a simplified version of Matrix Chernoff inequality, cited from (Tropp et al. 2015, Eq. (5.1.5)).

Lemma 4 *Let $X_1, \cdots, X_m$ be a sequence of $d \times d$ i.i.d. positive semi-definite (PSD) matrices satisfying $\|X_k\|_{\text{op}} \le L$ almost surely for all k. Then for any $t \in [0,1)$,*

$$\Pr\left[\lambda_{\min}(\sum_j X_j) \le t\mu_{\min}\right] \le d\exp\left\{-\frac{(1-t)^2\mu_{\min}}{2L}\right\},$$

where $\lambda_{\min}(\cdot)$ is the smallest eigenvalue of a PSD matrix and $\mu_{\min} = \lambda_{\min}(\sum_j \mathbb{E}X_j)$.

The next lemma upper bounds the operator norm of an inverse covariance matrix of random vectors uniformly distributed on a unit sphere.

Lemma 5 *Let $u_1, u_2, \cdots, u_m$ be i.i.d. d-dimensional vectors uniformly distributed on the unit sphere $\{x \in \mathbb{R}^d : \|x\|_2 = 1\}$. Suppose also that $m \ge 8d\ln(d/\delta)$ for some $\delta \in (0, 1/2]$. Then with probability at least $1-\delta$, $\|(\sum_j u_j u_j^\top)^{-1}\|_{\text{op}} \le 2d/m$.*

Proof of Lemma 5. Denote $\Lambda := \mathbb{E}[u_1 u_1^\top] \in \mathbb{R}^{d\times d}$. Because the distribution of u_1 is spherical invariant, we immediately have $\Lambda_{jk} = 0$ for all $j \ne k$ and $\Lambda_{jj} \equiv \lambda$ for all $j = 1, 2, \cdots, d$. Additionally, since $\mathbb{E}[u_1^\top u_1] = 1$ we have that $\text{tr}(\Lambda) = d\lambda = 1$. This implies $\Lambda = \frac{1}{d} I_{d\times d}$.

Next, invoke Lemma 4 with $L=1$, $\mu_{\min}=m/d$ and $t=1/2$. We then have

$$\Pr\left[\lambda_{\min}(\sum_j u_j u_j^\top) \le \frac{m}{2d}\right] \le d\exp\left\{-\frac{m}{8d}\right\}.$$

Under the condition that $m \ge 8d\ln(d/\delta)$, the right-hand side of the above inequality is upper bounded by δ. Lemma 5 is thus proved.

References

Agarwal, A., Dekel, O., & Xiao, L. (2010). Optimal algorithms for online convex optimization with multi-point bandit feedback. In *Proceedings of the Conference on Learning Theory (COLT)*, (pp. 28–40).

Agarwal, A., Luo, H., Neyshabur, B., & Schapire, R. E. (2016). Corralling a band of bandit algorithms. *arXiv preprint arXiv:1612.06246*.

Bandeira, A. S., Scheinberg, K., & Vicente, L. N. (2014). Convergence of trust-region methods based on probabilistic models. *SIAM Journal on Optimization*, *24*(3), 1238–1264.

Besbes, O., Gur, Y., & Zeevi, A. (2015). Non-stationary stochastic optimization. *Operations Research*, *63*(5), 1227–1244.

Billups, S. C., Larson, J., & Graf, P. (2013). Derivative-free optimization of expensive functions with computational error using weighted regression. *SIAM Journal on Optimization*, *23*(1), 27–53.

Blum, J. R. (1954). Multidimensional stochastic approximation methods. *The Annals of Mathematical Statistics*, (pp. 737–744).

Bubeck, S., Munos, R., Stoltz, G., & Szepesvári, C. (2011). X-armed bandits. *Journal of Machine Learning Research*, *12*(May), 1655–1695.

Bubeck, S., Stoltz, G., Szepesvári, C., & Munos, R. (2009). Online optimization in x-armed bandits. In *Advances in Neural Information Processing Systems*, (pp. 201–208).

Chen, R., Menickelly, M., & Scheinberg, K. (2018). Stochastic optimization using a trust-region method and random models. *Mathematical Programming*, *169*(2), 447–487.

Conn, A. R., Scheinberg, K., & Vicente, L. N. (2009). *Introduction to derivative-free optimization*, vol. 8. SIAM.

Flaxman, A. D., Kalai, A. T., Kalai, A. T., & McMahan, H. B. (2005). Online convex optimization in the bandit setting: gradient descent without a gradient. In *Proceedings of the annual ACM-SIAM symposium on Discrete algorithms (SODA)*, (pp. 385–394).

Ghadimi, S., & Lan, G. (2013). Stochastic first-and zeroth-order methods for nonconvex stochastic programming. *SIAM Journal on Optimization*, *23*(4), 2341–2368.

Kiefer, J., & Wolfowitz, J. (1952). Stochastic estimation of the maximum of a regression function. *The Annals of Mathematical Statistics*, *23*(3), 462–466.

Koza, J. R. (1997). Genetic programming.

Lewis, R. M., & Torczon, V. (1999). Pattern search algorithms for bound constrained minimization. *SIAM Journal on Optimization*, *9*(4), 1082–1099.

Lykouris, T., Mirrokni, V., & Paes Leme, R. (2018). Stochastic bandits robust to adversarial corruptions. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, (pp. 114–122). ACM.

Nemirovsky, A. S., & Yudin, D. B. (1983). *Problem complexity and method efficiency in optimization*. SIAM.

Powell, M. J. (2003). On trust region methods for unconstrained minimization without derivatives. *Mathematical Programming*, *97*(3), 605–623.

Shamir, O. (2017). An optimal algorithm for bandit and zero-order convex optimization with two-point feedback. *Journal of Machine Learning Research*, *18*(52), 1–11.

Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, (pp. 2951–2959).

Spall, J. C. (1992). Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE Transactions on Automatic Control*, *37*(3), 332–341.

Torczon, V. (1997). On the convergence of pattern search algorithms. *SIAM Journal on Optimization*, *7*(1), 1–25.

Tropp, J. A., et al. (2015). An introduction to matrix concentration inequalities. *Foundations and Trends® in Machine Learning*, *8*(1-2), 1–230.

Van Laarhoven, P. J., & Aarts, E. H. (1987). Simulated annealing. In *Simulated annealing: Theory and applications*, (pp. 7–15). Springer.

Whitley, D. (1994). A genetic algorithm tutorial. *Statistics and Computing*, *4*(2), 65–85.