

ARGONNE NATIONAL LABORATORY
9700 South Cass Avenue
Argonne, Illinois 60439

**A Batch, Derivative-Free Algorithm for
Finding Multiple Local Minima¹**

Jeffrey Larson and Stefan M. Wild

Mathematics and Computer Science Division

Preprint ANL/MCS-P5228-1114

November 2014 (*Revised December 2015*)

To appear as:

J. Larson and S. M. Wild. A Batch, Derivative-Free Algorithm for Finding
Multiple Local Minima. *Optimization and Engineering*,
doi:10.1007/s11081-015-9289-7.

¹This work was supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research program under contract number DE-AC02-06CH11357.

A Batch, Derivative-Free Algorithm for Finding Multiple Local Minima ^{*}

Jeffrey Larson Stefan M. Wild

December 10, 2015

Abstract

We propose a derivative-free algorithm for finding high-quality local minima for functions that require significant computational resources to evaluate. Our algorithm efficiently utilizes the computational resources allocated to it and also has strong theoretical results, almost surely starting a finite number of local optimization runs and identifying all local minima. We propose metrics for measuring how efficiently an algorithm finds local minima, and we benchmark our algorithm on synthetic problems (with known local minima) and two real-world applications.

1 Introduction

In this paper, we are interested in finding multiple, high-quality local minima for the bound-constrained, nonlinear optimization problem

$$\begin{aligned} & \underset{x \in \mathbb{R}^n}{\text{minimize}} && f(x) \\ & \text{subject to: } && x \in \mathcal{D} = \{x : -\infty < l \leq x \leq u < \infty\} \end{aligned} \tag{1}$$

when the function f is expensive to evaluate and w concurrent evaluations of f are possible.

Why multiple, high-quality local minima? In settings where problems such as (1) are stated without a given nominal design or starting point $x_0 \in \mathcal{D}$, a typical practitioner desires a *globally optimal solution*, meaning an $x^* \in \mathcal{D}$ with $f(x^*) \leq f(x)$ for all $x \in \mathcal{D}$. On the other hand, whether driven by theoretical guarantees or practical time constraints, many algorithmic approaches for solving (1) focus on efficiently finding *local minimizers*. Such a local minimizer \bar{x}^* has the desirable property that no improvements in the objective can be obtained by small, feasible perturbations of \bar{x}^* . In the context of global

^{*}This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, SciDAC and applied mathematics programs under Contract DE-AC02-06CH11357.

optimization, “high quality” would thus refer to those local minimizers whose corresponding function values are small, the highest-quality minimizer(s) corresponding to the global minimizer(s).

Finding multiple, high-quality local minimizers, however, is not just a surrogate for finding an approximate global minimizer. For example, in many physical systems one is interested not just in the ground state (that which globally minimizes energy) but also in nearby (also in terms of energy) excited states; see [7] for another application where all local minima are desired.

In engineering design optimization, such a task is also frequent when one has not just the quantifiable objective f but also additional nonquantifiable or nonordinal metrics that are used to compare points within the domain. For example, the objective f may be the cost of producing a structure, but the aesthetics of the design may also matter in a way that cannot be captured in terms of cost or some other ordinal objective. In this context, “high quality” would refer to the trade-off between the objective function value of different minima and their nonordinal objective value. In this work we focus on f as a measure of quality in order to demonstrate our proposed approach and to test this approach’s ability to find all minima of (1). This approach naturally produces a list of points to be evaluated by nonordinal metrics.

Why concurrent evaluations? For many engineering design problems, evaluating the objective f involves running a complex numerical simulation. To reduce the overall wall time required by the simulation, researchers increasingly use parallel resources (cores and/or nodes). However, savings in time from such parallelism is useful only up to a point (see, e.g., [12]); beyond this point, more resources do not decrease the time required to evaluate the simulation at a single input. When the computational resources that can be effectively utilized by a simulation are less than the available computational resources, the only way to further decrease the time needed to solve the design problem (without fundamentally altering the simulation) is to evaluate the objective function (and, hence, the simulation) concurrently at multiple inputs.

In many situations with computationally expensive simulations, the number of concurrent evaluations possible, w , is relatively small; this is often because the simulation is run on a computational system that is not multiple orders of magnitude larger than the resources that the simulation can efficiently use. Furthermore, when the time required to evaluate the simulation is relatively homogeneous throughout the domain \mathcal{D} , it is natural to think of such evaluations as occurring in synchronous batches of size w . Although this batch assumption will be relaxed in future work (see, e.g., [12] and [1] for cases when such asynchronous behavior is useful), here we focus exclusively on the synchronous batch case.

Given the above context, a natural approach for identifying multiple local minima is a batch, *multistart* method. In a serial paradigm, such methods iteratively identify a point \hat{x} that will be improved by using a local optimization run and then, on subsequent iterations, decide whether to continue improving

\hat{x} or to find a new point to start improving. In a parallel, batch paradigm, the batches of w points could consist of some combination of the next point in a serial local optimization run, points randomly sampled from the domain in order to determine points from which to start future local optimization runs, and a group of points in a concurrent local optimization run. Here, we do not consider the third class of points; instead we focus on the use of readily available local optimization solvers that perform sequential evaluations of the objective. Intuitively, handling multiple local optimization runs simultaneously allows one to make quicker progress in identifying more local minima than does sequentially completing local optimization runs; this situation is especially true when considerable evaluations are expended in the “tail” of a local optimization run.

In addition to these practical concerns, we would like to establish theoretical properties of our algorithm, especially those that have implications on the algorithm’s empirical performance. In particular, we will show that our algorithm finds every local minimum while starting only a finite number of local optimization runs; we view these local optimization runs as the chief expense of multistart methods. Since our algorithm is stochastic, these results can at best be in the *almost sure* sense. That is, the algorithm finds all minima while starting a finite number of local optimization runs with probability 1.

Although the analysis holds when derivatives are available, in this paper we focus on the case where derivatives of f are unavailable. In particular, we show that the framework is effective when the local solver employed does not require derivative information. Our algorithm can be viewed as a *master* giving points to w *workers* of two types: workers that are randomly sampling points in the domain and workers that are performing a local optimization run. We ensure that these w workers are occupied in every batch iteration; for example, if no local optimization run requires function evaluations, then w randomly sampled points will be evaluated in that iteration.

An outline of the paper and its contribution are as follows. Section 2 contains background information on previous methods for solving (1) and other necessary information. In Section 3, we propose a new, batch parallel multistart algorithm, which prioritizes and begins local optimization runs according to the availability of computational resources. Our analysis of the algorithm shows that it finds all local minima almost surely and that it starts finitely many local optimization runs almost surely. Section 4 discusses performance measures, and we propose a new convergence test to measure how efficiently an algorithm identifies a set of local minima. We use this test in Section 5 on synthetic test problems and two engineering applications and show that the proposed algorithm reliably finds multiple, high-quality local minima. Furthermore, our numerical results show that these minima are found without significantly sacrificing efficiency in finding a global minimum, often surpassing other algorithms designed with this single goal in mind. A summary of the paper and a discussion of future work in Section 6 conclude the paper.

2 Background

We first discuss previous algorithms and then develop some notation and background material necessary for our algorithm.

2.1 Algorithms for finding multiple local minima

Multistart methods are a broad category of algorithms that aim to find the best local optima by starting local optimization runs from different points in the domain. This multistart mechanism can be added on top of any local optimization method, and thus many abstractions of multistart algorithms do little more than determine starting points for local optimization runs. Consequently, such multistart algorithms must be implemented with great care in settings where local optimization algorithms converge only asymptotically and computational resources are limited.

One such multistart algorithm is multilevel single linkage (MLSL) [16, 17], a stochastic method with the theoretical property that all local minima are found almost surely and that only a finite number of local optimization runs are started almost surely. The GLOBAL algorithm [2] is a sequential implementation of MLSL in the derivative-free case. In each iteration, GLOBAL randomly samples the domain and then sequentially performs local optimization runs, refining randomly sampled points until all are assigned to known clusters of points.

In addition to algorithms that start multiple local optimization runs, methods for finding a global minimum of (1) may also identify local minima, even if the methods were not designed with this goal in mind. Without additional assumptions on f , any algorithm with a theoretical guarantee of identifying a global minimum must generate a dense set of iterates in the domain [20]. Thus, any global optimization algorithm with a convergence guarantee will eventually evaluate points arbitrarily close to the local minima of (1). The points evaluated by the algorithm could easily be processed to find points in the domain that are better than all nearby points.

Many algorithms utilize the components of multistart and global algorithms. For example, the “Global and Local Optimization using Direct Search” (GLODS) algorithm [3] starts several direct searches in regions of interest, possibly merging local optimization runs that approach each other. The Multilevel Coordinate Search algorithm [13] is a modification of the global optimization algorithm DIRECT [14] to also include some local optimization components. Each of these three methods is purely sequential: each evaluates a single point in \mathcal{D} on every iteration.

An example of a global optimization method that is not sequential is the parallel implementation of DIRECT called pVTdirect [11, 10]. This algorithm uses parallel function evaluations to solve derivative-free optimization problems and can approximate a batch algorithm if it is allocated $w + 1$ MPI ranks (one additional rank to coordinate the w workers evaluating points of interest). By design, pVTdirect does not attempt to always evaluate w points concurrently when w is small. Rather, the algorithm was implemented to avoid bottlenecks

that could arise from a master giving each of w workers a point to evaluate on every iteration.

Largely because of its theoretical results regarding the number of local optimization runs started, we base our algorithm on MLSL. We follow the spirit of the sequential MIPE algorithm from [21, Ch. 6] and use all previously evaluated points when determining where to start a local optimization run. This is a key distinction; to our knowledge, all other methods consider starting runs only from randomly sampled points.

Since we assume that computational resources are limited, we want a theoretical multistart framework that does not start optimization runs arbitrarily. Rather, our framework will start runs only when computational resources are available. This distinction is glossed over by many multistart methods but is important for ensuring that a multistart algorithm can be implemented in practice while retaining desired asymptotic properties.

In Section 5, we compare an implementation of our proposed algorithm with implementations of several existing algorithms

2.2 Notation

We take \mathcal{X}^* to denote the set of all local minima of f within \mathcal{D} . We make the following assumption about the problem (1).

Assumption 1. *Assume that the function f and its local minima satisfy the following:*

- (A) f is twice continuously differentiable,
- (B) $\mathcal{X}^* \subset \text{int}(\mathcal{D})$, and
- (C) there exists $\epsilon_{x^*} > 0$ such that $\|x^* - y^*\| > \epsilon_{x^*}$ for all $x^*, y^* \in \mathcal{X}^*$.

Let $f_{(i)}^*$ be the i th smallest value in $\{f(x_{(i)}^*) : x^* \in \mathcal{X}^*\}$, and let $x_{(i)}^*$ be the corresponding $x^* \in \mathcal{X}^*$ (breaking ties arbitrarily). Denote the set of workers \mathcal{W} and its constituent sets of random-sampling and local-optimization workers by \mathcal{W}_S and \mathcal{W}_L , respectively. Denote their respective sizes by $|\mathcal{W}| = |\mathcal{S}| + |\mathcal{L}|$.

Denote the cumulative history of points evaluated after batch k by $\mathcal{H}_k = \mathcal{S}_k \cup \mathcal{L}_k$ where \mathcal{S}_k are randomly sampled points and \mathcal{L}_k are local optimization points. Let $\mathcal{A}_k \in \mathcal{L}_k$ be the subset of points that are within active local optimization runs, and let \mathcal{X}_k^* be the set of local optimizers identified after batch k . When necessary, define $x_{k,\ell}$ to be the point evaluated by worker ℓ on batch k . Let $D_{i,j} = \|x_i - x_j\|$ for all points in \mathcal{H}_k . Let $B(x; \delta)$ denote the ball of radius δ around $x \in \mathbb{R}^n$.

An algorithm is said to identify all local minima almost surely if

$$\mathbb{P} \left[\lim_{k \rightarrow \infty} \mathcal{X}_k^* = \mathcal{X}^* \right] = 1.$$

Table 1: Example logical conditions to determine when to start a local optimization run.

- (L1) $\hat{x} \in \mathcal{L}_k$ and $\nexists x \in \mathcal{L}_k$ with $[\|\hat{x} - x\| \leq r_k \text{ and } f(x) < f(\hat{x})]$
 - (S1) $\hat{x} \in \mathcal{S}_k$ and $\nexists x \in \mathcal{L}_k$ with $[\|\hat{x} - x\| \leq r_k \text{ and } f(x) < f(\hat{x})]$
 - (L2) $\hat{x} \in \mathcal{L}_k$ and $\nexists x \in \mathcal{S}_k$ with $[\|\hat{x} - x\| \leq r_k \text{ and } f(x) < f(\hat{x})]$
 - (S2) $\hat{x} \in \mathcal{S}_k$ and $\nexists x \in \mathcal{S}_k$ with $[\|\hat{x} - x\| \leq r_k \text{ and } f(x) < f(\hat{x})]$
 - (L3) $\hat{x} \in \mathcal{L}_k$ has not started a local optimization run
 - (S3) $\hat{x} \in \mathcal{S}_k$ has not started a local optimization run
 - (L4) $\hat{x} \in \mathcal{L}_k$ is at least a distance μ from the domain boundary
 - (S4) $\hat{x} \in \mathcal{S}_k$ is at least a distance μ from the domain boundary
 - (L5) $\hat{x} \in \mathcal{L}_k$ is at least a distance ν from any known local min
 - (S5) $\hat{x} \in \mathcal{S}_k$ is at least a distance ν from any known local min
 - (L6) $\hat{x} \in \mathcal{L}_k$ is not in an active local optimization run
 - (L7) $\hat{x} \in \mathcal{L}_k$ has not been ruled stationary
 - (L8) $\hat{x} \in \mathcal{L}_k$ is on an r_k -descent path in \mathcal{H}_k for some $x \in \mathcal{S}_k$ satisfying (S2-S5)
-

Similarly, an algorithm is said to start finitely many local optimization runs almost surely if

$$\mathbb{P} \left[\lim_{k \rightarrow \infty} (\text{Local optimization runs started in batches 1 to } k) < \infty \right] = 1.$$

2.3 Previous MLSL results

A fundamental task of a multistart method is deciding where to start local optimization runs. The simplest methods perform random sampling and start a local optimization run from every randomly sampled point. They continue this procedure until a computational budget is exhausted. More complex methods, naturally requiring more overhead, are more judicious in their selection of starting points. Table 1 outlines some tests that may be used by a multistart method for determining whether a local optimization run should be started from a previous randomly sampled point in \mathcal{S}_k or a point in \mathcal{L}_k evaluated during some previous local optimization run.

The conditions (L1), (S1), (L2), and (S2) depend on a critical distance r_k that determines when no better points are sufficiently close to a point \hat{x} . Although many forms for r_k are possible, we focus on the sequence from the original MLSL paper. In iteration k , let

$$r_k = \frac{1}{\sqrt{\pi}} \sqrt[2]{\Gamma \left(1 + \frac{n}{2} \right) \text{vol}(\mathcal{D}) \frac{\sigma \log |\mathcal{S}_k|}{|\mathcal{S}_k|}}, \quad (2)$$

where $\Gamma(\cdot)$ is the gamma operator, $|\mathcal{S}_k|$ is the total number of points randomly sampled in the first k iterations, and σ is a positive constant. It is evident that if sampling is guaranteed, $|\mathcal{S}_k|$ goes to infinity, and r_k goes to zero.

The conditions (L4), (S4), (L5), and (S5) are largely to show theoretical results for the algorithm. (L6) is a computational construct of the algorithm. The condition (L8) is needed in order to bound the number of local optimization runs started. The conditions (L3), (S3), and (L7), are designed to prevent clearly redundant local optimization runs.

Whether implicitly (e.g., (S2)) or explicitly (e.g., (L8)), several of the conditions in Table 1 are based on the notion of an r_k -descent path. An r_k -descent path in a set \mathcal{B} from a point $x_0 \in \mathcal{B}$ is a subset of points $\{x_i \in \mathcal{B}\}_{i=0}^h$ such that $D_{i,i+1} \leq r_k$ and $f(x_i) > f(x_{i+1})$ for $i = 0, \dots, h-1$.

Algorithm 1: Original sequential MLSL

```

for  $k = 1, 2, \dots$  do
    Evaluate  $f$  at  $N$  points uniformly sampled from  $\mathcal{D}$ 
    Compute  $r_k$  using (2)
    Start a local optimization run at any previously evaluated point
    satisfying
        (S2), (S3), (S4), (S5)
end

```

The conditions in Table 1 allow us to succinctly define the original, sequential MLSL algorithm in Algorithm 1. Although the presentation of this algorithm is simple, many fundamental questions about MLSL must be addressed when trying to implement it when the number of possible concurrent function evaluations is limited. In order to prove that MLSL almost surely starts finitely many local optimization runs and almost surely identifies all local minima, the original MLSL analysis [16] made the following assumption about the local optimization method.

Assumption 2. *Assume the following:*

- (A) *The local optimization method is “strictly descent” for a path contained in \mathcal{D} . That is, starting from any point $x \in \mathcal{D}$, the local optimization method generates a sequence of points $x_{k'}$, with $x_{k'+1} = x_{k'} + p_{k'}$, which converges to a local minimum x^* such that $f(x_{k'} + \beta_1 p_{k'}) \leq f(x_{k'} + \beta_2 p_{k'})$ for all k' and all β_1, β_2 satisfying $0 \leq \beta_1 \leq \beta_2 \leq 1$.*
- (B) *If the local optimization method is applied to a point that is within distance ν of a local minimizer \bar{x}^* , then it will recognize \bar{x}^* as such.*

Under these assumptions, the original MLSL algorithm has the following theoretical results; the first result is proved in [16, Theorem 8], and the second result is proved in [16, Theorem 12].

Theorem 1. *If the problem in (1) satisfies Assumption 1, the local optimization method satisfies Assumption 2, and r_k is defined by (2) with $\sigma > 4$, then Algorithm 1 will start a finite number of local optimization runs and will identify every local minimum.*

3 A batch, derivative-free algorithm for finding multiple minima

The simplest way that Algorithm 1 can be adjusted to the concurrent evaluation, derivative-free setting is to exploit the natural parallelism offered by having N randomly sampled points and possibly starting multiple local optimization runs in each iteration. However, such an approach raises three problems, which motivate our algorithmic developments.

First, the algorithm may request more or fewer local optimization runs than there are available resources. If the local optimization method is a sequential derivative-free algorithm, evaluating one point at a time, and $|\mathcal{W}|$ function evaluations can occur concurrently, then computational resources will not be used if fewer than $|\mathcal{W}|$ local optimization runs are called for. If more than $|\mathcal{W}|$ local optimization runs are requested, then such runs would need to be prioritized.

Second, even if exactly $|\mathcal{W}|$ local optimization runs are requested, it is unlikely that each run will stop after the same number of function evaluations. Rather, one must wait for the local optimization method requiring the most function evaluations to terminate before proceeding to the next iteration.

Third, one iteration of Algorithm 1 may request only N evaluations, but the next will request N plus however many local optimization function evaluations are performed. This situation can make estimating the required computational resources difficult.

We propose a batch algorithm for finding multiple local minima in Algorithm 2 that efficiently uses the available computational resources by always evaluating $|\mathcal{W}|$ evaluations on every iteration. The algorithm also uses Algorithm 3 to explicitly state how local optimization runs should be prioritized. Note that we must use r_{k-1} when deciding where to start local optimization runs since only $|\mathcal{S}_{k-1}|$ random sample points have been evaluated at batch k .

Aside from the batch/sequential nature, Algorithm 2 and Algorithm 1 have two key distinctions that we highlight. First, whereas Algorithm 1 bases its decisions solely on randomly sample points (as indicated by the letter “S” in the conditions (S2), (S3), (S4), (S5)), Algorithm 2 employs the entire history of evaluations thus far (with decisions based on points from local optimization runs indicated by the letter “L” in the conditions in Table 1). Second, the batch iteration counter k in Algorithm 2 is fundamentally different from the iteration counter in Algorithm 1. The counter in Algorithm 2 is directly tied to the number of function evaluations performed, with exactly $k|\mathcal{W}|$ evaluations performed in the first k iterations. This decision has practical benefits (the number of iterations now directly relates with wall time) but will complicate the analysis.

Algorithm 3 starts local optimization runs at the best points satisfying the conditions in Table 1 and never stops giving computational resources to an active local optimization run until it is completed. After the last point in a local optimization run has been evaluated, the local optimization method will identify the run as completed. This check ensures that Algorithm 3 has total

Algorithm 2: Batch algorithm for finding multiple local minima

Given $|\mathcal{W}|$ workers, a function f , and a bound-constrained domain \mathcal{D}
Set $\sigma > 0$, $\mathcal{X}_k^* = \emptyset$
for $w_\ell \in \mathcal{W}$ **do**
| Give w_ℓ a uniformly sampled point $x_{0,\ell}$ to evaluate
end
 $k = 1$
while true do
| Receive $f(x_{k-1,\ell})$ -values from each $w_\ell \in \mathcal{W}$ and update \mathcal{H}_k
| Give \mathcal{H}_k to the local optimization method to see if next point in any
| run is
| already evaluated and/or mark any worker w_ℓ as having no active
| run
| Possibly update \mathcal{A}_k , and \mathcal{X}_k^*
| Compute r_{k-1} using (2)
for $w_\ell \in \mathcal{W}$ **do**
| $x_{k,\ell} = \text{point_generator}(w_\ell, \mathcal{H}_k, \mathcal{A}_k, r_{k-1})$ (see Algorithm 3)
| Give $x_{k,\ell}$ to worker w_ℓ to evaluate
end
| Increment k
end

Algorithm 3: point_generator for worker w_ℓ

if $w_\ell \in \mathcal{W}_S$ **then**
| Draw $x_{k,\ell}$ uniformly from \mathcal{D}
else
| **if** w_ℓ has an active local optimization run **then**
| | Set $x_{k,\ell}$ to be the next point determined by
| | the local optimization method
else
| | Identify the point $\hat{x} \in \mathcal{H}_k$ satisfying the list of conditions in
| | Table 1 with the smallest function value (breaking ties arbitrarily)
| | **if** No such $\hat{x} \in \mathcal{H}_k$ exists **then**
| | | Draw $x_{k,\ell}$ uniformly from \mathcal{D}
| | | **else**
| | | Start a local optimization run at \hat{x} , set $x_{k,\ell}$ as the first point
| | | evaluated by this run, mark w_ℓ as having an active local
| | | optimization run, and mark \hat{x} (and the associated point in \mathcal{S}_k
| | | from condition (L8)) as having started a run
| | | **end**
| | **end**
end
end

knowledge of what runs are completed and ensures that any local optimization points have not been previously evaluated.

For the analysis of the algorithm, we assume the following about the local optimization method employed in Algorithm 2.

Assumption 3. *The local optimization algorithm has the following features:*

- (A) *It is strictly descent.*
- (B) *Started from any $x \in \mathcal{D}$, it will evaluate some point within a distance ν of some local minimizer \bar{x}^* in a finite number of function evaluations, for arbitrary $\nu > 0$.*
- (C) *Once within a distance ν of \bar{x}^* (ν sufficiently small) the local optimization algorithm will identify \bar{x}^* as a minimizer in a finite number of function evaluations.*

Assumption 3(A) is the MLSL Assumption 2(A) and Assumption 3(C) assumption is a more explicit statement of what is required in practice for Assumption 2(B), namely that MLSL “recognizes a local minimizer as such,” to be satisfied. Assumption 3(B) is necessary because our algorithm explicitly bounds the number of active local optimization runs. This can be relaxed if one assumes (as done in the analysis of other multistart methods) that local optimization runs can be started arbitrarily without concern for available resources.

We make the following assumption about the number of workers of each type.

Assumption 4. *Assume that $|\mathcal{W}_{\mathcal{L}}| > 0$ for infinitely many batch iterations.*

This assumption guarantees that any local optimization run will eventually be given the computational resources it needs. Note that if no randomly sampled points are available to start a local optimization run and if $|\mathcal{W}_{\mathcal{S}}| = 0$, then Algorithm 3 will naturally generate $|\mathcal{W}|$ sample points to be used in the next iteration.

Using these assumptions, we can prove that Algorithm 2 will almost surely start a finite number of local optimization runs. We then use this result to show that every local minimum will be identified almost surely by Algorithm 2 (and computational resources will be available to perform the local optimization run).

Theorem 2. *If the problem in (1) satisfies Assumption 1, the local optimization method satisfies Assumption 3, and r_k is defined by (2) with $\sigma > 4$, then Algorithm 2 will start a finite number of local optimization runs almost surely.*

Proof. The result follows from establishing that the conditions that must be satisfied to start a local optimization run in Algorithm 2 are more restrictive than the conditions required to start a point in Algorithm 1.

In order to start a local optimization run in Algorithm 2 from a random sample point in \mathcal{S}_k , the conditions in Table 1 that must be satisfied are a superset

of the conditions used to determine when to start a local optimization run in Algorithm 1. In order to start a local optimization run in Algorithm 2 from a local optimization point in \mathcal{L}_k , condition (L8) ensures that the run is associated with a random sample point that would have satisfied the subset of conditions required to start a local optimization run in Algorithm 1. Therefore, since Algorithm 2 starts no more runs than does Algorithm 1 and since Algorithm 1 starts finitely many local optimization runs almost surely by Theorem 1, so does Algorithm 2. \square \square

We now use the fact that the number of local optimization runs is finite almost surely to guarantee almost surely that a worker will be available for Algorithm 2 to start a local optimization run that will identify every local minimum.

Theorem 3. *If the function f satisfies Assumption 1, the local optimization method satisfies Assumption 3, the set of workers satisfies Assumption 4, and $\lim_{k \rightarrow \infty} r_k = 0$, then every local minimum will be found by Algorithm 2 within a finite number of batches.*

Proof. Let $x_{(i)}^*$ be any local minimum that has not yet been identified by Algorithm 2 before batch iteration \bar{k} (defined below). Define $Q_\mu \subset \mathcal{D}$ to be the set of all points within a distance μ of the boundary of \mathcal{D} . For $x_{(i)}^* \in \mathcal{X}^*$, let $\psi(x_{(i)}^*) > 0$ be the largest scalar such that $B(x_{(i)}^*; \psi(x_{(i)}^*))$ satisfies the following:

1. $B(x_{(i)}^*; \psi(x_{(i)}^*)) \cap Q_\mu = \emptyset$, and
2. $x_1, x_2 \in B(x_{(i)}^*; \psi(x_{(i)}^*))$ with $\|x_1 - x_{(i)}^*\| < \|x_2 - x_{(i)}^*\| \leq \psi(x_{(i)}^*)$ implies $f(x_1) < f(x_2)$.

Let $\psi = \min_{x_{(i)}^* \in \mathcal{X}^*} \{\psi(x_{(i)}^*)\}$. That is, ψ is the largest Euclidean distance around any local minima such that no points are within a distance μ of the boundary of \mathcal{D} and f is increasing.

The existence of such a ψ satisfying the first condition is guaranteed by $\mathcal{X}^* \subset \text{int}(\mathcal{D})$ from Assumption 1(B) for μ sufficiently small. The existence of such a ψ further satisfying the second condition is guaranteed by Assumption 1(C) ensuring an $\epsilon_{x^*} > 0$ such that $\|x_{(i)}^* - x_{(j)}^*\| \geq \epsilon_{x^*}$ for all $x_{(i)}^*, x_{(j)}^* \in \mathcal{X}^*$ and the boundedness of \mathcal{D} ensuring that $|\mathcal{X}^*|$ is finite.

Since $\text{vol}(B(x_{(i)}^*; \psi)) > 0$, the uniform random sampling within Algorithm 2 will almost surely produce a point $\hat{x} \in B(x_{(i)}^*; \psi)$ at some batch iteration \bar{k} when $r_{\bar{k}} \leq \psi$.

Since the total number of local optimization runs started is almost surely finite by Theorem 2 and by Assumption 3(B), each of these runs will evaluate a point within a distance ν of a local minimizer in a finite number of function evaluations. By Assumption 3(C), the local optimization algorithm will identify that minimizer as such in a finite number of function evaluations. By

Assumption 4, these runs will all eventually be given the needed computational resources. Therefore, almost surely, a worker $w_{\mathcal{L}} \in \mathcal{W}_{\mathcal{L}}$ eventually will be available to start the local optimization run at batch iteration \bar{k} .

Therefore at batch iteration \bar{k} , we will start a run either at \hat{x} or at a point $\hat{x}' \in \mathcal{L}_{\bar{k}}$ on an $r_{\bar{k}}$ -descent path in $\mathcal{H}_{\bar{k}}$ from \hat{x} . By construction of ψ , \hat{x}' must also be in $B(x_{(i)}^*; \psi)$. In either case, since the local optimization procedure is strictly descent by Assumption 3(A), the local optimization run started from \hat{x} or \hat{x}' will identify $x_{(i)}^*$ as a local minimizer. \square \square

4 Measuring performance

Plots of the best-found function value at batch k or the minimum distance from an evaluated point to a given minimum at batch k may sufficiently display a solver’s performance on a single problem. Examining such plots for a collection of problems is much less straightforward. We use the performance and data profiles developed in [15] to more succinctly compare the performance of a set of numerical methods \mathcal{M} on the collection of problems \mathcal{P} . These profiles require a measure of convergence (parameterized by a constant τ) to determine whether or when an algorithm has “solved” a given problem. We will measure convergence using the following tests.

4.1 Within a relative distance τ of the j best local minima

Measuring convergence to a set of local minima is not straightforward. Monitoring only the function values observed by an algorithm is irrelevant without also ensuring that the points producing those function values are close to a minimum of interest. However, just monitoring the minimum distance from a local minimum to any point evaluated by an algorithm is also unsatisfactory since it is relatively harder to find a point within a sphere of radius τ of a local minimizer in higher dimensions. For example, a ball in \mathbb{R}^n with radius 0.1 around a local minimizer has a volume that is 3.1% of the volume of the unit cube when $n = 2$, but $4.2 \times 10^{-5}\%$ of the volume of the unit cube when $n = 7$. Therefore, in our estimation the more appropriate metric will keep a constant probability (independent of dimension) of a point drawn uniformly from the domain being sufficiently close (in the Euclidean sense) to a local minimizer.

Before defining how we measure how close an algorithm is to the j best local minimizers, we first use the fact that $f_{(i)}^* \leq f_{(i+1)}^*$ to define $\bar{j} \geq j$ and $\underline{j} \geq j$ to be the largest and smallest indices such that $f_{(\bar{j})}^* = f_{(j)}^* = f_{(\underline{j})}^*$. These definitions are necessary in order to handle the case when multiple local minima exist with the function value $f_{(j)}^*$. For example, if we want to monitor progress to the three best minima, but the five best local minima all have the same value, an algorithm should receive credit for finding any of those five minima.

We consider a method to have found the j best local minima at a level $\tau \geq 0$

after k batches if

$$\left| \left\{ x_{(1)}^*, \dots, x_{(\underline{j}-1)}^* \right\} \cap \left\{ x_{(i)}^* : \exists x \in \mathcal{H}_k \text{ with } \|x - x_{(i)}^*\| \leq \gamma_n(\tau) \right\} \right| = \underline{j} - 1$$

and

$$\left| \left\{ x_{(\underline{j})}^*, \dots, x_{(\bar{j})}^* \right\} \cap \left\{ x_{(i)}^* : \exists x \in \mathcal{H}_k \text{ with } \|x - x_{(i)}^*\| \leq \gamma_n(\tau) \right\} \right| \geq \bar{j} - \underline{j} + 1, \quad (3)$$

where $\gamma_n(\tau) = \sqrt[n]{\frac{\tau \text{vol}(\mathcal{D}) \Gamma(\frac{n}{2} + 1)}{\pi^{n/2}}}$. Here, τ is interpreted as being a fraction of the domain. This test has the desirable properties of being equally likely to be satisfied by a point drawn uniformly from \mathbb{R}^n independent of n . We note that this measure of convergence requires adequate knowledge of the minimizers $\{x_{(1)}^*, \dots, x_{(\bar{j})}^*\}$. We note that the definition of $\gamma_n(\tau)$ depends on the domain \mathcal{D} and therefore depends on how the domain is scaled.

4.2 Within a relative distance τ of the best function value

In addition to measuring when an algorithm finds points close to a set of local minima, we want a test that measures when an algorithm finds function values close to a global minimum. We consider an algorithm to find a global minimum at a level τ after k batches if it has found a point $x \in \mathcal{H}_k$ satisfying

$$f(x_0) - f(x) \geq (1 - \tau)(f(x_0) - f_G), \quad (4)$$

where x_0 is the starting point for problem p and f_G is an estimate of the global minimum (e.g., the known value $f_{(1)}^*$ or the least function value computed by any solver in \mathcal{M}). The quantity $1 - \tau$ denotes the fraction of possible decrease found.

4.3 Performance profiles

We now describe how to construct performance profiles to monitor how effectively an algorithm is solving problems in \mathcal{P} relative to other methods in \mathcal{M} for a given convergence test. Let $t_{p,m}$ be the smallest number of batches required for method m to satisfy the given convergence test for problem $p \in \mathcal{P}$. Then the performance profile of each solver m is the fraction of the problems in \mathcal{P} where the ratio

$$\phi_{p,m} = \frac{t_{p,m}}{\min_{m' \in \mathcal{M}} \{t_{p,m'}\}}$$

is at most $\alpha \geq 1$. That is,

$$\rho_m(\alpha) = \frac{|\{p \in \mathcal{P} : \phi_{p,m} \leq \alpha\}|}{|\mathcal{P}|}.$$

Some relevant values of $\rho_m(\alpha)$ are given in Table 2.

Table 2: Relevant values of ρ_m and their interpretations.

Value	Interpretation
$\rho_m(1)$	Fraction of problems method m solves fastest among the methods in \mathcal{M} .
$\rho_m(\alpha)$	Fraction of problems method m solves in fewer than α times the batches required by the fastest method in \mathcal{M} .
$\lim_{\alpha \rightarrow \infty} \rho_m(\alpha)$	Fraction of problems method m solves (within the computational budget allotted).

4.4 Data profiles

Performance profiles are a relative metric: an algorithm may perform better or worse depending on the other methods it is being compared with. Data profiles are an absolute metric to compare the performance of a set of solvers \mathcal{M} on a set of problems \mathcal{P} . Data profiles can be especially useful for analyzing algorithms for optimizing computationally expensive functions. While performance profiles consider a fixed budget of evaluations, data profiles measure the fraction of problems solved as a function of the number of batches. That is, the data profile of a method $m \in \mathcal{M}$ is

$$d_m(\alpha) = \frac{\left| \left\{ p \in \mathcal{P} : \frac{t_{p,m}}{n_p+1} \leq \alpha \right\} \right|}{|\mathcal{P}|},$$

where n_p is the dimension of problem p . In other words, for a given budget of $\alpha(n_p + 1)$ function evaluations for each problem $p \in \mathcal{P}$, $d_m(\alpha)$ is the fraction of problems that method m will solve within $\alpha(n_p + 1)$ evaluations. This definition mirrors that in [15] and considers a problem’s “difficulty” to grow linearly with the problem dimension.

5 Numerical experiments

We now compare the performance of a few optimization routines on a collection of benchmark problems. The problems (explained in greater detail below) include a family of synthetic functions and two real-world applications, one pertaining to microscopy and the other to biometrics. The microscopy problem has three variables, which is few enough to allow us to have some understanding of its local minima within the domain. This is not possible for the 57-variable biometrics problem. The synthetic benchmark problems are generated randomly; but each problem comes with a complete list of local minima, thereby allowing us to monitor how effectively each optimization algorithm identifies them.

5.1 Algorithms considered

We compare BAML (a Matlab prototype of Algorithm 2), the serial global/local search method GLODS [3], and two implementations of DIRECT (a serial ver-

sion written in Matlab [5], which we will call “Direct,” and a parallel implementation in Fortran95 [11] denoted “pVTdirect”). We also include a Random Sampling method that uniformly samples points from \mathcal{D} . We consider all methods to be batch methods that evaluate $|\mathcal{W}|$ points concurrently, with the restriction that $|\mathcal{W}| = 1$ for the serial algorithms GLODS and Direct. We also examine the idealized performance of Direct, GLODS, and pVTdirect in the Results section.

The performance of BAMLM critically depends on the local optimization routine used to generate points in \mathcal{L}_k . Example properties of a local optimization method that would be desirable in BAMLM include the following:

1. Accepting a history of previously evaluated points to efficiently utilize past function evaluations.
2. Honoring a given starting point by not moving to the best point in the history.
3. Honoring the bound constraints in (1).
4. Accepting additional constraints on where new points will be evaluated (e.g., avoiding areas being explored by other local optimization runs).
5. Having the ability to generate anywhere from 1 to $|\mathcal{W}|$ points for concurrent evaluation, and indicating how useful to the local optimization each of these points is expected to be.

We are unaware of an implementation of a local optimization routine that has all these properties. Algorithm 2 was developed to evaluate only a single point from a given local optimization run, in part because of the dearth of methods that satisfy the last characteristic. In the tests that follow, the local optimization runs were performed with ORBIT [22], a trust-region method that uses radial basis function models. ORBIT honors bound constraints and uses a history of previously evaluated points to help construct local models; it thus satisfies some of the desired properties listed above.

ORBIT was run in our computational experiments with its default termination criteria. In engineering applications, selection of an efficient solver and its associated termination criteria (which define what it means to be a “local solution” in that application) are key to ensuring that the proposed algorithm performs effectively.

BAMLM was run with $|\mathcal{W}| = 4$ workers, three possible local optimization workers and one worker dedicated to uniform random sampling. Each pVTdirect run was given five MPI ranks, one master rank telling four worker ranks which points to evaluate. Random Sampling evaluates points in batches of four as well. Since the two DIRECT implementations evaluate a predetermined set of $2n + 1$ points, independent of the function, we also ensure that GLODS, BAMLM, and Random Sampling initialize with these points. Therefore, all methods start by first evaluating the centroid of \mathcal{D} and agree on their first $2n + 1$ evaluations. BAMLM completes its initialization by evaluating f at an additional $8n - 1$ points drawn uniformly over the domain (so $10n$ points are always evaluated

before deciding where to start the first local optimization run); if less sampling is desired, fewer initial points can be evaluated.

All algorithms other than pVTdirect have Matlab implementations, so we can easily monitor which points are being evaluated in each batch. Since pVTdirect does not provide a history of points evaluated, we monitor its progress by having the objective function print out every x being evaluated, $f(x)$, the MPI rank doing the evaluation, and the time of the evaluation. The objective function pauses for a fixed amount of time, allowing us to determine which evaluations are occurring simultaneously; similar steps were taken to benchmark pVTdirect in [8, 9]. However, pVTdirect is not a batch algorithm, so each rank is not evaluating a point at every iteration, and these inactive ranks must be accounted for. Since these inactive ranks can consume over 20% of the allocated budget, we also plot an “idealized pVTdirect” for the simulation-based engineering problems using the raw output grouped into batches of four, thus eliminating inactive ranks.

The default GLODS parameters specify a deterministic 2^n -centers-search step to generate points that are asymptotically dense in the domain. When running GLODS, we use this setting on all problems except for the biometrics problem, where merely generating the first set of 2^{57} points is prohibitive. For the biometrics problem alone, the search step points are generated uniformly over the domain. The GLODS results presented for the biometrics problem are the average of 10 such runs. Otherwise, the only changes to the default GLODS parameters are decreasing the initial step size from 1 to 0.1 (since all the problems are scaled to the unit cube turning off the stopping criteria based on the step size parameter, so that the computational budget is exhausted), and setting the initial set of points to mirror the Direct and pVTdirect.

For each synthetic problem and the microscopy problem, algorithms were run for 5,000 total function evaluations, or 1,250 four-evaluation batches. The local optimization runs within BAMLML were given at most 200 function evaluations for these problems. For the biometrics problem, each method was allocated 10^5 function evaluations (or 2.5×10^4 batches), and the local runs in BAMLML were limited to 10^3 evaluations. The stochastic algorithms (BAMLML and Random Sampling) were run on each synthetic problem with 5 different random seeds, run with 40 different seeds on the microscopy problem, and run with 10 different seeds on the biometrics problem.

5.2 Problems with known minima

We first constructed test functions with known local minima by minor modifications to the GKLS problem generator [6]. We have intentionally chosen these problems with known local minima, not because any of the tested algorithms require knowledge of the location or number of minima, but because knowing the minima allows us to measure how efficiently each algorithm finds them. Each problem consists of a convex quadratic augmented by polynomials to introduce local minima at random locations.

The GKLS problem generator requires that r^* , the distance from the quadratic

vertex to the (unique, by construction) global minimum, satisfies $0 < r^* < \frac{1}{2} \min_i \{u_i - l_i\}$ and that GKLS’s radius of attraction around the global minimum be at most $\frac{r^*}{2}$. For a problem on the unit cube in \mathbb{R}^7 , the volume of this basin is less than $\frac{\pi^{7/2}}{\Gamma(\frac{7}{2}+1)} \frac{1}{4}^7 \approx 0.00029$. Even if the entire basin were to be contained in \mathcal{D} (which is not guaranteed), we find this fraction of the domain to be too small relative to the volume bound given for problems in \mathbb{R}^2 . We therefore modify the upper bound on r^* to be $\frac{n}{2}$ for problems on the unit cube. This modification allows for the possibility of placing the global minimum outside the domain; but we remove the restriction that only 100 problems can be generated, and we then take the first 100 problems with all feasible local minima. As a result, we obtain 100 continuously differentiable functions in each of $\mathbb{R}^2, \mathbb{R}^3, \dots, \mathbb{R}^7$, and each with unit cube domains and 10 local minima.

5.2.1 Results

We ran Random Sampling, BAMLM, Direct, pVTdirect, and GLODS on the 3,000 synthetic problem instances (100 different problems in each of the 6 dimensions between \mathbb{R}^2 and \mathbb{R}^7 , each with 5 different random seeds). Since the Direct, pVTdirect, and GLODS implementations are deterministic, their performance on the 600 unique problems was duplicated 5 times.

We can use the convergence test defined by (3) to measure how efficiently each algorithm locates the j best local minima. Similarly, using the convergence test defined by (4), we can compare how effectively the algorithms identify the global minimum. As a point of reference, we also include a hypothetical “idealized Direct” that takes the points evaluated by Direct and then groups them into batches of four points. This would be the result of a perfectly scalable Direct implementation, meaning that the same set of points is evaluated, but the total time required would be a factor four less because of the fourfold concurrency.

In Fig. 1, we see the relative performance of the set of algorithms at identifying the j best minima to a level τ . Fig. 1(a) shows that BAMLM finds (for a very restrictive τ and within 1,250 batch iterations) the three best minima on over 60% of the problems. Fig. 1(b) shows that Random Sampling is as good as any method at evaluating points close to the three best minima for a less restrictive τ ; this result suggests that finding points within a level $\tau = 10^{-2}$ of the three best minima is relatively easy and that the convergence test defined by (3) is sensitive to the value of τ .

For a level $\tau = 10^{-3}$, Figs. 1(c)–1(d) show that BAMLM is successful at identifying the best minima ((c) and (d) indicating whether four or seven minima were desired, respectively) within the computational budget for most of the problems. The fact that BAMLM does not identify 100% of the minima is unsurprising given that the radius of the basins for the nonglobal minima can be small and thus difficult to randomly sample within a limited budget of evaluations.

Although BAMLM is effective at identifying many high-quality local minima, this additional effort does not hinder its progress toward the global minimum on

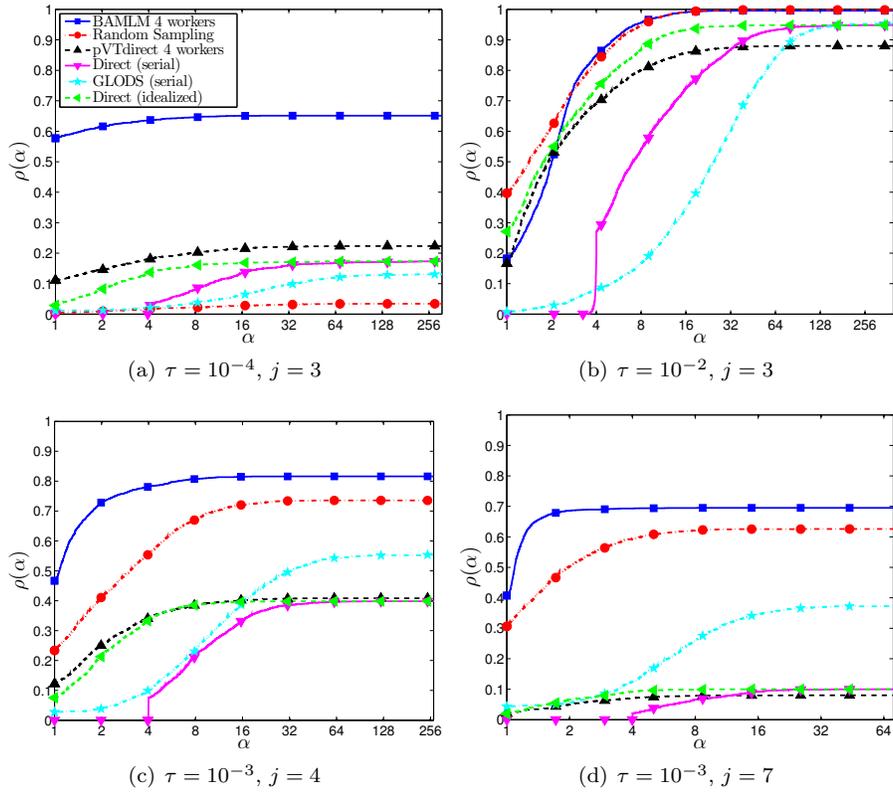


Figure 1: Performance profiles using convergence test (3) to measure how six methods identify the j best minima within a level τ on 3,000 synthetic problem instances from Section 5.2.

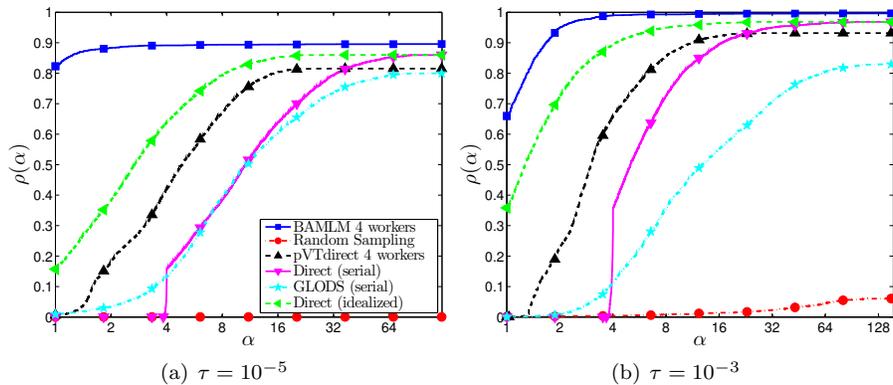


Figure 2: Performance profiles using convergence test (4) to measure how six methods identify the possible decrease from the starting point to the global minimum on 3,000 synthetic problem instances from Section 5.2.

these problems. In Fig. 2, we see that nearly all the methods (aside from Random Sampling) are effective at identifying a point within 99.9% of the possible decrease from the starting point to the global minimum. The same is true for finding 99.999% of the possible decrease, but we see that BAMLML takes fewer function evaluations to do so compared with the other algorithms considered.

In Fig. 2, the idealized version of Direct behaves as expected. It is exactly four times better than the serial version of Direct, and the performance of pVTdirect falls between the idealized and serial version. Since pVTdirect is not perfectly efficient at evaluating four function values on every iteration, we see that it does not converge fast enough to the global minimum on approximately 5% of the 3,000 problems before the budget of 1,250 batches is exhausted. On the other hand, we note from Fig. 1 that pVTdirect is better than the idealized version of Direct at finding many of the local minima.

In Fig. 3, we present the data profiles for the same values of j and τ as presented in Fig. 1 and Fig. 2 with performance profiles. The results of the performance and data profiles are nearly identical; these results support the previous conclusions and ensure that the three implementations of DIRECT are not “stealing from each other” in the performance profiles.

We note that the performance of BAMLML, as measured by either of the proposed convergence tests, would not decrease if more concurrent function evaluations occurred. If $|\mathcal{W}|$ were larger, then more local optimization runs could occur simultaneously, a situation that would never increase the number of batches required to find a given local minimum. Fig. 2(b) suggests that for $\tau = 10^{-3}$ and $|\mathcal{W}| = 4$, BAMLML is roughly twice as efficient at identifying the global minimum for these problems as is an idealized implementation of Direct, and at least four times as efficient as an actual Direct implementation. Therefore, $|\mathcal{W}|$ would have to at least be doubled, and BAMLML see no improvement from an increase in workers, in order for even a hypothetical, perfectly scaling

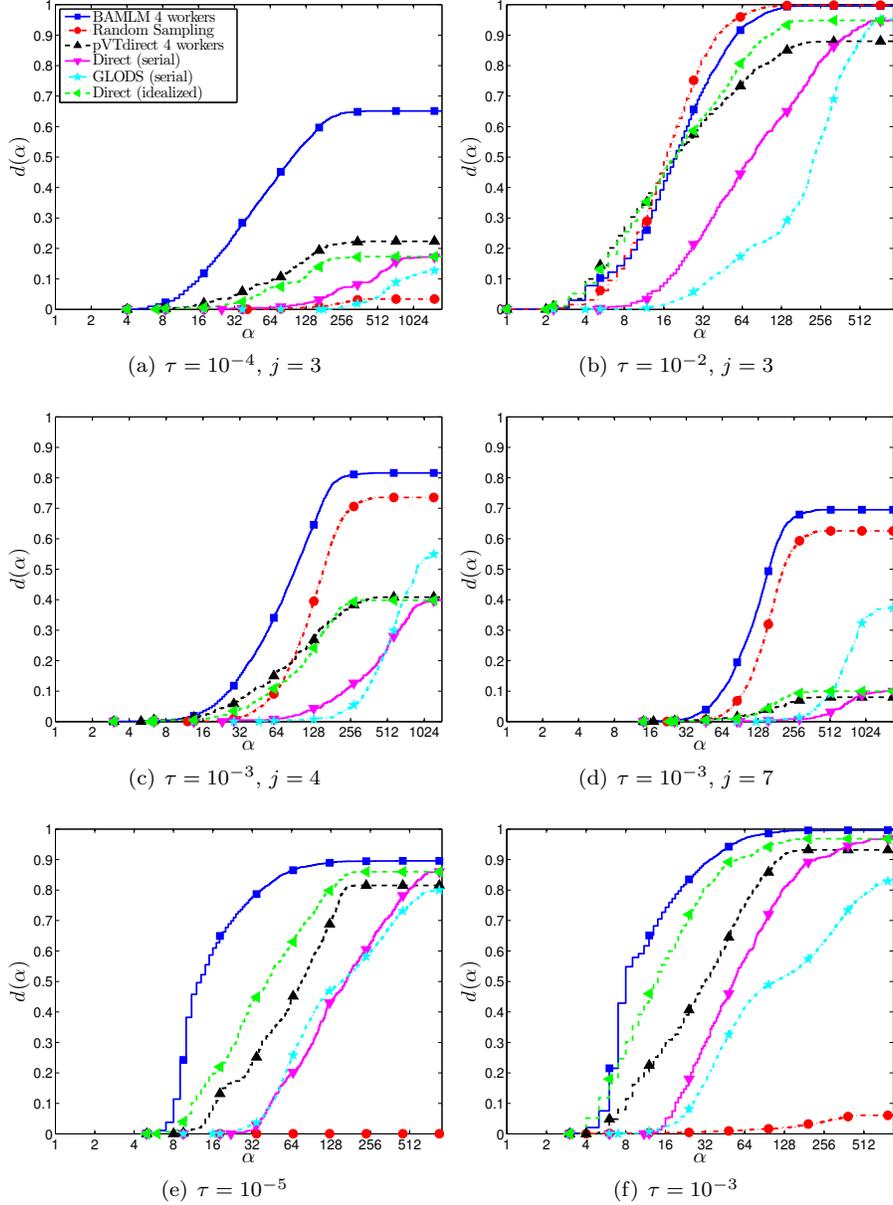


Figure 3: Data profiles. Figs. 3(a)–3(d) use convergence test (3) to measure how six methods identify the j best minima within a level τ on 3,000 synthetic problem instances from Section 5.2. Figs. 3(e)–3(f) use convergence test (4) to measure how six methods identify the possible decrease from the starting point to the global minimum.

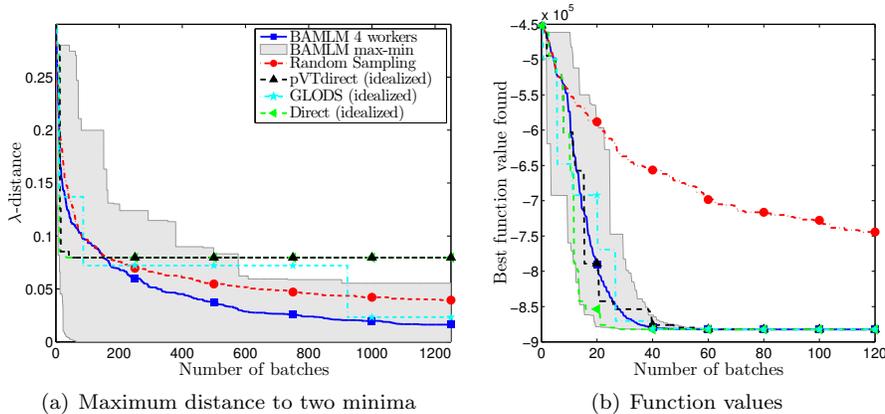


Figure 4: Trajectories of the minimum function value and maximum distance from an evaluated point to both local minima for various algorithms for the microscopy problem in Section 5.3.1.

implementation of Direct to perform as well as BAMLML.

5.3 Simulation-based problems

We now evaluate the performance on BAMLML on two problems from engineering applications.

5.3.1 Microscopy problem

This three-dimensional, simulation-based problem that we refer to as the “microscopy problem” entails improving the quality of an image obtained from a scanning transmission electron microscope. The goal is to minimize the variance in an image by adjusting the defocus parameter and two astigmatism parameters. Computing the image quality is significantly cheaper than recording an image with such a microscopy with a given set of parameters. Since the repeated capturing of images can degrade or destroy the sample, the variance should be minimized in as few function evaluations as possible. The problem is explained in detail in [19, 18] and the citations therein.

Here we work within the bounded domain $[-700, 300]^3$, which contains two known local minima with non-negligible basins. As with the other problems, this domain was scaled to the unit cube to more consistently compare the algorithms (with their default parameter values). The better of the two minima is located close to the origin and has a function value of -8.8×10^5 ; the nonglobal minima has a value of -4.3×10^5 , which is still better than the average function value over \mathcal{D} , which we estimate to be approximately -3.57×10^5 . We will denote these two points $\tilde{x}_{(1)}^*$ and $\tilde{x}_{(2)}^*$ to signify that they approximate $x_{(1)}^*$ and $x_{(2)}^*$, respectively.

We can then measure how effectively a method m in a set of solvers \mathcal{M} finds these two points by observing

$$\lambda_m(k) = \max \left\{ \min_{\substack{1 \leq j \leq k \\ 1 \leq \ell \leq |\mathcal{W}|}} \|x_{j,\ell} - \tilde{x}_{(1)}^*\|, \min_{\substack{1 \leq j' \leq k \\ 1 \leq \ell' \leq |\mathcal{W}|}} \|x_{j',\ell'} - \tilde{x}_{(2)}^*\| \right\}. \quad (5)$$

That is, $\lambda_m(k)$ is the larger of $\delta_1(k)$ and $\delta_2(k)$, where $\delta_j(k)$ is the minimum distance between points evaluated by method m through batch k and $\tilde{x}_{(j)}^*$.

We performed 40 runs of the stochastic algorithms (Random Sampling and BAMLML) and show the function value trajectories in Fig. 4(b). We see that, on average, BAMLML finds the global minimum as fast as pVTdirect, but Fig. 4(a) shows that even the worst instance of BAMLML locates both minima faster than does either of the implementations of Direct. GLODS takes considerably more batches before evaluating a point near the second minimum. Even under perfect scaling (which is unrealistic), GLODS, Direct, and pVTdirect do not significantly outperform BAMLML in converging to the global minimum. Their idealized performance remains worse in identifying the two best local minima (as measured by the λ -distance from (5)).

5.3.2 Biometrics problem

The last problem we consider is a 57-dimensional biomechanical control problem whose local minima are not precisely known, although Easterling et al. [4] report a best-known global solution of $f = 1222.05$; finding this value required significant computational expense. The problem appears to be highly multimodal and noisy; see [4, Fig. 2]. The variables of the problem are torque levels for three joints (hips, knee, and ankle) at each of nineteen 100-millisecond intervals in a human musculoskeletal model. The optimization problem arises from trying to keep the model stable (without stepping) after a quick displacement of the supporting surface occurs. The objective function contains a weighted sum of desired properties, including movement of the model center of mass and the change between activation levels at consecutive time steps for each given joint.

In Fig. 5, we plot the minimum function value found by each algorithm as each progresses through 10^5 function evaluations (or 2.5×10^4 batches). The average of 10 runs is shown for the three stochastic algorithms: Random Sampling, BAMLML, and GLODS. We plot the idealized versions of pVTdirect, Direct, and GLODS by grouping their first 10^5 function evaluations into batches of four. We see that BAMLML is, on average, better than all the other algorithms at finding a smaller function value. Even the worst performing of the 10 BAMLML runs is comparable to the (unrealistic) idealized pVTdirect and Direct implementations. The best BAMLML run finds a function value half as small as the other algorithms. (Note that BAMLML performs well on this noisy problem in spite of the problem not satisfying Assumption 1(A), and possibly neither Assumption 1(B) nor Assumption 1(C). This empirical robustness is an indication that

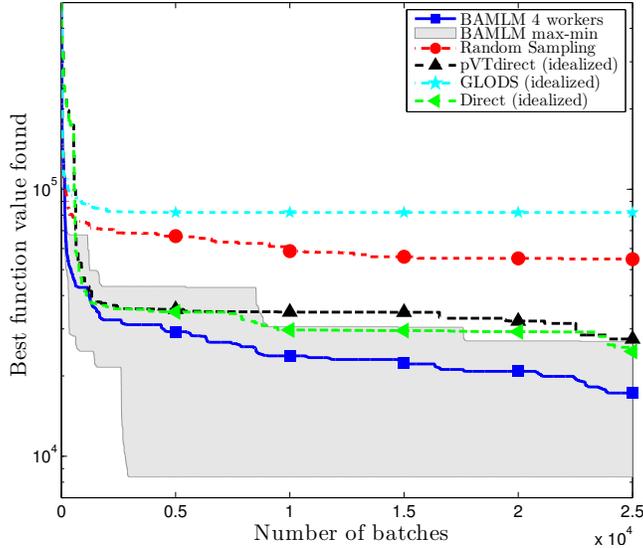


Figure 5: Trajectories of the minimum function value for various algorithms on the biometrics problem in Section 5.3.2.

BAMLM might enjoy similar success on other engineering problems that may not satisfy all of Assumption 1.)

6 Conclusion

In this paper we presented a derivative-free, multistart algorithm for finding multiple, high-quality local minima for a bound-constrained optimization problems when relatively few concurrent functions are possible. If the function is sufficiently smooth and has nonconnected local minima, we showed that the algorithm almost surely starts finitely many local optimization runs while still finding all local minima. In addition to these theoretical results, a numerical implementation of the algorithm was shown to be efficient at evaluating points close to the local minima of problems from a large set of synthetic test problems. Our results on two engineering optimization problems show that in situations where the theoretical assumptions do not hold, our algorithm performs well.

Furthermore, the effort expended in identifying multiple minima on these problems does not appear to hinder the algorithm’s progress to a global minimum since it often outperforms existing methods (both currently parallel methods and sequential methods viewed under idealized parallel scaling conditions) in this metric. More rigorous tests are required to see whether this behavior is independent of problem dimension.

Many interesting avenues remain for extensions of this research. Our pro-

duction implementation of the proposed algorithm will target simulation-based problems on high-performance computers and address the associated difficulties. In particular, we plan to extend our theoretical framework from the batch paradigm to the case where function evaluations are completed asynchronously and possibly terminated before evaluation has completed. We are also interested in developing a local optimization method that is well suited for our framework, being able to usefully generate more than one point at a time, on an as-needed basis. Moreover, we wish to augment our algorithm so the local optimization runs can communicate with each other, preventing multiple runs from evaluating points within the same part of the domain.

Acknowledgements

We are grateful to Maria Rudnaya and Aswin Kannan for coding the microscopy problem and to Christine Shoemaker for valuable discussions on multistart methods. We gratefully acknowledge the computing resources provided on Blues, a high-performance computing cluster operated by the Laboratory Computing Resource Center at Argonne National Laboratory.

References

- [1] Audet, C., Dennis, Jr., J.E., Le Digabel, S.: Parallel space decomposition of the mesh adaptive direct search algorithm. *SIAM Journal on Optimization* **19**(3), 1150–1170 (2008). DOI 10.1137/070707518
- [2] Csendes, T., Pál, L., Sendín, J.O.H., Banga, J.R.: The GLOBAL optimization method revisited. *Optimization Letters* **2**(4), 445–454 (2007). DOI 10.1007/s11590-007-0072-3
- [3] Custódio, A.L., Madeira, J.F.A.: GLODS: Global and local optimization using direct search. *Journal of Global Optimization* pp. 1–19 (2014). DOI 10.1007/s10898-014-0224-9
- [4] Easterling, D.R., Watson, L.T., Madigan, M.L., Castle, B.S., Trosset, M.W.: Parallel deterministic and stochastic global minimization of functions with very many minima. *Computational Optimization and Applications* **57**(2), 469–492 (2014). DOI 10.1007/s10589-013-9592-1
- [5] Finkel, D.: DIRECT. URL http://www4.ncsu.edu/~ctk/Finkel_Direct/Direct.m
- [6] Gaviano, M., Kvasov, D.E., Lera, D., Sergeev, Y.D.: Algorithm 829: Software for generation of classes of test functions with known local and global minima for global optimization. *ACM Transactions on Mathematical Software* **29**(4), 469–480 (2003). DOI 10.1145/962437.962444

- [7] Gheribi, A.E., Robelin, C., Le Digabel, S., Audet, C., Pelton, A.D.: Calculating all local minima on liquidus surfaces using the factsage software and databases and the mesh adaptive direct search algorithm. *The Journal of Chemical Thermodynamics* **43**(9), 1323–1330 (2011). DOI 10.1016/j.jct.2011.03.021
- [8] He, J., Verstak, A., Sosonkina, M., Watson, L.: Performance modeling and analysis of a massively parallel DIRECT–part 2. *International Journal of High Performance Computing Applications* **23**(1), 29–41 (2009). DOI 10.1177/1094342008098463
- [9] He, J., Verstak, A., Watson, L., Sosonkina, M.: Performance modeling and analysis of a massively parallel DIRECT–part 1. *International Journal of High Performance Computing Applications* **23**(1), 14–28 (2009). DOI 10.1177/1094342008098462
- [10] He, J., Verstak, A., Watson, L.T., Sosonkina, M.: Design and implementation of a massively parallel version of DIRECT. *Computational Optimization and Applications* **40**(2), 217–245 (2007). DOI 10.1007/s10589-007-9092-2
- [11] He, J., Watson, L.T., Sosonkina, M.: Algorithm 897: VTDIRECT95: Serial and parallel codes for the global optimization algorithm DIRECT. *ACM Transactions on Mathematical Software* **36**(3), 1–24 (2009). DOI 10.1145/1527286.1527291
- [12] Hough, P.D., Kolda, T.G., Torczon, V.J.: Asynchronous parallel pattern search for nonlinear optimization. *SIAM Journal on Scientific Computing* **23**(1), 134–156 (2001). DOI 10.1137/S1064827599365823
- [13] Huyer, W., Neumaier, A.: Global optimization by multilevel coordinate search. *Journal of Global Optimization* **14**(1995), 1–28 (1999). DOI 10.1023/A:1008382309369
- [14] Jones, D.R., Perttunen, C.D., Stuckman, B.E.: Lipschitzian optimization without the Lipschitz constant. *Journal of Optimization Theory and Applications* **79**(1), 157–181 (1993). DOI 10.1007/BF00941892
- [15] Moré, J.J., Wild, S.M.: Benchmarking derivative-free optimization algorithms. *SIAM Journal on Optimization* **20**(1), 172–191 (2009). DOI 10.1137/080724083
- [16] Rinnooy Kan, A.H.G., Timmer, G.T.: Stochastic global optimization methods, part I: Clustering methods. *Mathematical Programming* **39**(1), 27–56 (1987). DOI 10.1007/BF02592070
- [17] Rinnooy Kan, A.H.G., Timmer, G.T.: Stochastic global optimization methods, part II: Multi level methods. *Mathematical Programming* **39**(1), 57–78 (1987). DOI 10.1007/BF02592071

- [18] Rudnaya, M.E., Kho, S.C., Mattheij, R.M.M., Maubach, J.M.L.: Derivative-free optimization for autofocus and astigmatism correction in electron microscopy. In: Proceedings of the 2nd International Conference on Engineering Optimization, pp. 1–10. Lisbon, Portugal (2010). URL http://www.dem.ist.utl.pt/engopt2010/Book_and_CD/Papers_CD_Final_Version/pdf/06/01059-01.pdf
- [19] Rudnaya, M.E., Van den Broek, W., Doornbos, R.M.P., Mattheij, R.M.M., Maubach, J.M.L.: Defocus and twofold astigmatism correction in HAADF-STEM. *Ultramicroscopy* **111**(8), 1043–54 (2011). DOI 10.1016/j.ultramic.2011.01.034
- [20] Törn, A., Zilinskas, A.: Global optimization. Springer-Verlag, New York (1989)
- [21] Wild, S.M.: Derivative-free optimization algorithms for computationally expensive functions. Ph.D. thesis, Cornell University (2009). URL <http://ecommons.cornell.edu/handle/1813/11248>
- [22] Wild, S.M., Regis, R.G., Shoemaker, C.A.: ORBIT: Optimization by radial basis function interpolation in trust-regions. *SIAM Journal on Scientific Computing* **30**(6), 3197–3219 (2008). DOI 10.1137/070691814

The submitted manuscript has been created by UChicago Argonne, LLC, Operator of Argonne National Laboratory (“Argonne”). Argonne, a U.S. Department of Energy Office of Science laboratory, is operated under Contract No. DE-AC02-06CH11357. The U.S. Government retains for itself, and others acting on its behalf, a paid-up, nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government.