SANDIA REPORT

SAND2004-8055
Unlimited Release
Printed February 2004

Revisiting Asynchronous Parallel Pattern Search for Nonlinear Optimization

Tamara G. Kolda

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of Energy's
National Nuclear Security Administration under Contract DE-AC04-94AL85000.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865)576-8401
Facsimile: (865)576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.doe.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800)553-6847
Facsimile: (703)605-6900
E-Mail: orders@ntis.fedworld.gov
Online order: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2004-8055
Unlimited Release
Printed February 2004

Revisiting Asynchronous Parallel Pattern Search for Nonlinear Optimization

Tamara G. Kolda*
Computational Sciences and Mathematics Research Department
Sandia National Laboratories
Livermore, CA 94551–9217

ABSTRACT

We present a new asynchronous parallel pattern search (APPS) method which is different from that developed previously by Hough, Kolda, and Torczon. APPS efficiently uses parallel and distributed computing platforms to solve science and engineering design optimization problems where derivatives are unavailable and cannot be approximated. The original APPS was designed to be fault-tolerant as well as asynchronous and was based on a peer-to-peer design. Each process was in charge of a single, fixed search direction. Our new version is based instead on a manager-worker paradigm. Though less fault-tolerant, the resulting algorithm is more flexible in its use of distributed computing resources. We further describe how to incorporate a zero-order sufficient decrease condition and handle bound constraints. Convergence theory for all situations (unconstrained and bound constrained as well as simple and sufficient decrease) is developed. We close with a discussion of how the new APPS will better facilitate the future incorporation of linear and nonlinear constraints.

Keywords: asynchronous parallel optimization, pattern search, direct search, distributed computing, generating set search.

*Email: tgkolda@sandia.gov. This work was supported by the Mathematical, Information, and Computational Sciences Program of the U.S. Department of Energy, under contract DE-AC04-94AL85000 with Sandia Corporation.

This page intentionally left blank.

1. Introduction. Asynchronous parallel pattern search (APPS) is a variation on parallel pattern search that uses parallel resources more efficiently by eliminating synchronization [8]. Pattern search methods [17, 21, 16, 14] and, more generally, generating set search (GSS) methods [10, 11] are geared toward solving science and engineering optimization problems that lack derivative information. These problems are typically characterized by objective functions based on complex and expensive computer simulations. GSS methods are provably convergent to a stationary point if the underlying objective function is suitably smooth; further, GSS methods often "work well" in practice (with some theoretical justification; see, e.g., [1]) even on non-smooth problems.

The original APPS algorithm is described in [8], and analysis follows in [12, 13]. The motivation for an asynchronous version of parallel pattern search has not changed from that described in [8]:

> A single synchronization step at the end of every iteration... is neither appropriate nor effective when any of the following factors holds: function evaluations finish in varying amounts of time (even on equivalent processors), the processors employed in the computation possess different performance characteristics, or the processors have varying loads.

However, another driving motivation for the original APPS was the need for a method that was tolerant to various types of failures that might cause synchronous parallel pattern search to completely fail or be extremely slow to converge. To facilitate fault-tolerance, the original APPS algorithm was based on a peer-to-peer model and used PVM [7] as the communication architecture. The new APPS is based instead on a manager-worker paradigm, sacrificing some fault-tolerance in exchange for greater simplicity and flexibility. Further, the new version is based on MPI [20], which many users seem to prefer to PVM. (Is should be noted that some fault-tolerant versions of MPI do exist [4] but such functionality is still rare.)

The sacrifices in terms of fault-tolerance are minimal since checkpointing to disk in the manager-worker version can be used in lieu of a peer-to-peer design. The checkpoint data is small, consisting of only the current best point and corresponding function value. The primary difference between peer-to-peer and checkpointing manager-worker implementations is that the checkpoint version requires some mechanism for restarting (either manual or automated) after a failure, whereas the peer-to-peer continues without any intervention.

In the original APPS, there were multiple agents (i.e., the peers), each of which owned part of the logic of the search. These agents had to correspond with one another regarding algorithmic events (new best, single direction convergence, and overall convergence), not to mention different types of faults; see [8] for more details. With a single manager process controlling all the logic of the search, these complexities are eliminated. Since the number of worker processes is typically very small (1-100 workers) and each communicates infrequently and asynchronously with the manager, it is unlikely that there will be any sort of communication bottleneck at the manager process.

While presenting the new APPS, we present additional modifications for a zero-order sufficient decrease condition and for bound constraints. The adaptation of a zero-order sufficient decrease condition to pattern search has been discussed in several contexts [10, 19], including a different take on peer-to-peer asynchronous parallel pattern search [6]. In particular, the generalization of pattern search to GSS in [10]

was motivated by the desire to incorporate generic globalization strategies, including sufficient decrease, into the framework. The use of a sufficient decrease condition yields greater flexibility in the selection of search directions at each iteration. Handling bound constraints for pattern search has also be the subject of several papers [15, 18, 10]. Some problematic numerical results in the original APPS paper [8, Table 5.6] are the result of not appropriately handling the bound constraints.

The organization of this paper is as follows. In §2, we review the parallel pattern search algorithm and variants that can be used for sufficient decrease and/or bound constraints; known convergence results are summarized in §2.4. The new APPS algorithm is presented in §3, along with its own corresponding variants. An illustrative example of the new APPS algorithm is presented in §4. Convergence theory follows in §5. We conclude with commentary on the algorithm and associated theory, pointers to its implementation and some numerical results, and ideas for future work in §6. For those familiar with the original APPS, a discussion of the evolution from the old APPS to this one is discussed in Appendix A.

For the purposes of this text, we consider both the unconstrained and bound-constrained nonlinear optimizations problems. The unconstrained problem is given by

$$\min \quad f(x). \tag{1.1}$$

Here $f : \mathbb{R}^n \to \mathbb{R}$ and $x \in \mathbb{R}^n$. The bound constrained problem is given by

$$\begin{array}{ll} \min & f(x) \\ \text{subject to} & \ell \leq x \leq u. \end{array} \tag{1.2}$$

The function f is the same as for the unconstrained problem. The upper and/or lower bounds are optional on an element-by-element basis; specifically, ℓ is an n-vector with entries in $\mathbb{R} \cup \{-\infty\}$ and u is an n-vector with entries in $\mathbb{R} \cup \{+\infty\}$. The set Ω denotes the feasible region; i.e.,

$$\Omega = \{x : \ell \leq x \leq u\}.$$

The unconstrained problem can be thought of as a special case of the bound constrained problem. In other words, $\Omega = \mathbb{R}^n$ in the unconstrained problem.

2. Review of Parallel Pattern Search. We briefly review parallel pattern search (PPS) with simple decrease and its extensions for sufficient decrease and bound constraints. We refer throughout to pattern search though it might be more accurate to refer to GSS; recall that the generalization of pattern search to GSS was motivated by the desire to bring in different globalization strategies, including sufficient decrease [10]. We conclude by presenting unified convergence results. This review lays the groundwork for the description of the asynchronous methods.

The generic algorithm is presented in Figure 2.1, and the notation used is as follows. Subscripts denote the iteration index. The vector $x_k \in \mathbb{R}^n$ denotes the *best point* (i.e., the point with the smallest known function value) at the beginning of iteration k. The set

$$\mathcal{D}_k = \left\{ d_k^{(1)}, \ldots, d_k^{(p_k)} \right\},$$

denotes the set of *search directions* at iteration k, and the number of search directions in $\mathcal{D}_k$ is denoted by p_k. Superscripts denote the *direction index*, which ranges between

1 and p_k at iteration k. The value Δ_k denotes the *step length* at iteration k, and the values $\tilde{\Delta}_k^{(i)} \in [0, \Delta_k]$, for $i = 1, \ldots, p_k$, denote the corresponding *pseudo step lengths*. The function $\rho(\cdot)$ denotes the *forcing function*.

Initialization.

Let $\Delta_{\text{tol}} > 0$ be the step length convergence tolerance.

Set x_0 to be some initial guess.

Set $\mathcal{D}_0 = \left\{ d_0^{(1)}, \ldots, d_0^{(p_0)} \right\}$ to be the initial set of search directions.

Set $\Delta_0 > \Delta_{\text{tol}}$ to be the initial value of the step length.

Iteration. For $k = 0, 1, \ldots$

STEP 1. Generate a set of trial points corresponding to the search directions; i.e.,

$$\mathbf{X}_k = \left\{ x_k + \tilde{\Delta}_k^{(i)} d_k^{(i)} : 1 \leq i \leq p_k \text{ and } \tilde{\Delta}_k^{(i)} \geq \Delta_{\text{tol}} \right\}$$

Send all points in $\mathbf{X}_k$ to the evaluation queue.

STEP 2. Wait until all trial points in in the evaluation queue have been evaluated. Collect those points in the set $\mathbf{Y}_k$.

STEP 3. If there exists a trial point in $y_k \in \mathbf{Y}_k$ such that $f(y_k) < f(x_k) - \rho(\Delta_k)$, then goto Step 4; else goto Step 5.

STEP 4. The iteration is successful:

- Set $x_{k+1} = y_k$.
- Choose a new $\mathcal{D}_{k+1} = \left\{ d_{k+1}^{(1)}, \ldots, d_{k+1}^{(p_{k+1})} \right\}$.
- Set $\Delta_{k+1} = \Delta_k$.
- Go to Step 1.

STEP 5. The iteration is unsuccessful:

- Set $x_{k+1} = x_k$.
- Set $\mathcal{D}_{k+1} = \mathcal{D}_k$ (and $p_{k+1} = p_k$).
- Set $\Delta_{k+1} = \frac{1}{2}\Delta_k$
- If $\Delta_{k+1} < \Delta_{\text{tol}}$, **terminate**; else, goto Step 1.

FIG. 2.1. *PPS algorithm*

In Step 1 of Figure 2.1, a set of trial points is generated, denoted by $\mathbf{X}_k$. The method for choosing the pseudo step lengths is discussed in detail in the subsections that follow. In general, $\tilde{\Delta}_k^{(i)} = \Delta_k$ unless constraints are involved.

In Step 2 of Figure 2.1, the trial points are evaluated, and the results are collected in $\mathbf{Y}_k$. For PPS, $\mathbf{Y}_k = \mathbf{X}_k$ for all k; however, this will not be the case for the asynchronous version in §3. Step 2 is where parallelism may be employed, in which

case the p_k function evaluations are executed in parallel. The algorithm does not go on to the next step until all evaluations have completed, so this is the point of synchronization. Furthermore, this is typically the most computationally expensive step because p_k function evaluations must be computed.

In Step 3 of Figure 2.1, the decrease condition is evaluated. The choice of the function $\rho(\cdot)$ is discussed in detail in §2.1 and §2.2. If the decrease condition is satisfied, then the iteration is termed *successful*; otherwise, it is *unsuccessful*.

As an aside, we make the following remark. If multiple points in $\mathbf{Y}_k$ produce decrease, any one can be chosen as y_k without impacting the convergence theory in §2.4. However, from a practical perspective, a point that yields the smallest function value should be selected.

The algorithm executes Step 4 of Figure 2.1 if the iteration is successful. The next iterate x_{k+1} is updated to be the trial point that produced decrease in the function, y_k. A new set of search directions may also be selected at this point. The search directions must be chosen in a particular way in order to guarantee that the algorithm will converge. The criteria are detailed in the subsections that follow. For simplicity, a choice that always works is the set of plus and minus unit vectors, i.e., $\mathcal{D}_k = \{\pm e_1, \pm e_2, \ldots, \pm e_n\}$ and $p_k = 2n$ for $k = 1, 2, \ldots$.

The algorithm executes Step 5 of Figure 2.1 if the iteration is unsuccessful. In this case, the step length Δ_k is reduced by a factor of two. Termination of the method is controlled by the step length.

2.1. PPS with Simple Decrease. Let us consider PPS with simple decrease for the unconstrained optimization problem (1.1). The term *simple decrease* means that only $f(y_k) < f(x_k)$ is required in Step 2. In other words, the function $\rho(\cdot)$ is assumed to be identically zero.

Below, we describe the four conditions that specialize the algorithm in Figure 2.1 to be PPS with simple decrease (in the unconstrained case).

The first two conditions have to do with the selection of the search directions, $\mathcal{D}_k$. It is useful to decompose the set of search directions as $\mathcal{D}_k = \mathcal{G}_k \cup \mathcal{H}_k$. The set $\mathcal{G}_k$ is the core set of search directions (the poll set), while the set $\mathcal{H}_k$ is a possibly empty set of additional search directions (the search set) [3, 10].

Condition 1 requires that the *cosine measure* of the subset $\mathcal{G}_k$ be uniformly bounded; see [10] for a discussion of cosine measure.

> CONDITION 1. Every $\mathcal{G}_k$ positively spans $\mathbb{R}^n$. Furthermore, there exists a constant $c_{\min} > 0$, both independent of k, such that $\kappa(\mathcal{G}_k) \geq c_{\min}$ for all k, where
>
> $$\kappa(\mathcal{G}_k) \equiv \min_{v \in \mathbb{R}^n} \max_{d \in \mathcal{G}_k} \frac{v^T d}{\| v \| \, \| d \|}.$$

Condition 2 requires that the search directions in $\mathcal{G}_k$ be uniformly bounded in length.

> CONDITION 2. There exist $\beta_{\min} > 0$ and $\beta_{\max} > 0$, independent of k, such that for all k the following holds:
>
> $$\beta_{\min} \leq \| d \| \leq \beta_{\max}, \qquad \text{for all } d \in \mathcal{G}_k.$$

Parts (a)–(c) of Condition 3 set more specific conditions for selecting the search directions; these conditions are important in the simple decrease case. Essentially, all search directions must be derived from a fixed, finite set $\mathbf{G}$. Part (c) explains how $\mathcal{H}_k$ may be formed. Condition 3 also requires that the forcing function is identically zero in part (d) and that the pseudo step lengths are chosen appropriately in part (e).

CONDITION 3. (Rational Lattice)
(a) There exists a finite set $\mathbf{G} = \{d^{(1)}, \ldots, d^{(p)}\}$ such that every vector $d^{(i)} \in \mathbf{G}$ is of the form $d^{(i)} = Bc^{(i)}$ where $B \in \mathbb{R}^{n\times n}$ is a nonsingular matrix and $c^{(i)} \in \mathbb{Q}^n$.
(b) All search directions in $\mathcal{G}_k$ are chosen from $\mathbf{G}$; i.e., $\mathcal{G}_k \subseteq \mathbf{G}$ for all k.
(c) All search directions in $\mathcal{H}_k$ are integer combinations of the elements of $\mathbf{G}$;. i.e., $\mathcal{H}_k \subset \left\{\sum_{i=0}^{p} \xi^{(i)} d^{(i)} \mid \xi^{(i)} \in \{0, 1, 2, \ldots\}\right\}$ for all k.
(d) The forcing function is identically zero, i.e., $\rho(t) \equiv 0$.
(e) All pseudo step lengths $\tilde{\Delta}_k^{(i)} \in [0, \Delta_k]$ satisfy either $\tilde{\Delta}_k^{(i)} = 0$ or $\tilde{\Delta}_k^{(i)} = \Delta_k$.

Conditions 1–3 are not difficult to satisfy; consider, for example,

$$\mathcal{D}_k = \{\pm e_1, \pm e_2, \ldots, \pm e_n\} \text{ for all } k.$$

Since we are only considering the unconstrained problem in this subsection, we further assume that the pseudo step lengths are always equal to the step length, i.e.,

$$\tilde{\Delta}_k^{(i)} = \Delta_k \text{ for } i = 1, \ldots, p_k,\ k = 1, 2, \ldots$$

This is formalized in the discussion of bound constraints as Condition 6.

2.2. PPS with Sufficient Decrease. Let us consider PPS with sufficient decrease for the unconstrained optimization problem (1.1). In this case, $\rho(\cdot)$ is a non-zero function, in contrast to the simple decrease case.

There are four conditions that specialize the algorithm in Figure 2.1 to be PPS with sufficient decrease (in the unconstrained case). As before, Conditions 1, 2, and 6 are imposed. Condition 3 is replaced instead by the following.

CONDITION 4. (Forcing Function)
(a) The function $\rho(t)$ is a nonnegative continuous function on $t \in [0, +\infty)$.
(b) The function $\rho(t)/t$ monotonically decreases to zero as $t \downarrow 0$.

A common choice that satisfies Condition 4 is

$$\rho(\Delta) = \alpha\Delta^2,$$

where α is some fixed, positive constant. For a complete discussion of forcing functions for GSS, see [10] and references therein.

2.3. PPS with Bound Constraints. Let us consider PPS for the bound constrained optimization problem (1.2). Adapting PPS for bound constraints affects the choice of search directions and the choice of the pseudo step lengths. The adaptation is largely independent of the choice of simple or sufficient decrease, except for the particulars of choosing the pseudo step lengths.

Three conditions specialize the algorithm in Figure 2.1 to be PPS with bound constraints.

In the bound constrained case, the search directions must "conform" to the geometry of the nearby boundary, so Condition 5 requires that $\mathcal{G}_k$ be the coordinate search directions [15]. Condition 5 replaces Condition 1 and Condition 2 since these conditions are trivially satisfied by this choice of $\mathcal{G}_k$. More general selection criteria may be employed; see the requirements on choosing search directions for general linear constraints in [10, 11].

CONDITION 5. For all k, we have $\mathcal{G}_k = \{\pm e_1, \ldots, \pm e_n\}$.

The second condition is either Condition 3 or Condition 4, depending on the choice of simple or sufficient decrease.

The final condition is the one we have already referred to, having to do with the choice for pseudo step lengths. Special choices for these values are required in the case of bound constraints. There are several ways that $\tilde{\Delta}_k^{(i)}$ can be chosen so long as Condition 6, which states that the full step is used if possible, is satisfied.

CONDITION 6. If $x_k + \Delta_k d_k^{(i)} \in \Omega$, then $\tilde{\Delta}_k^{(i)} = \Delta_k$.

Three possible strategies for choosing admissible values for $\tilde{\Delta}_k^{(i)}$ are described in [11] for the case of general linear constraints; we present two here. The simplest choice is the following.

$$\tilde{\Delta}_k^{(i)} = \begin{cases} \Delta_k & \text{if } x_k + \Delta_k d_k^{(i)} \in \Omega, \\ 0 & \text{otherwise.} \end{cases} \tag{2.1}$$

A more sophisticated choice may be employed in the sufficient decrease case: taking the longest possible feasible step. Define $\tilde{\Delta}_k^{(i)}$ as the solution to

$$\begin{array}{ll} \max & \tilde{\Delta} \\ \text{subject to} & 0 \le \tilde{\Delta} \le \Delta_k, \\ & x_k + \tilde{\Delta}\, d_k^{(i)} \in \Omega. \end{array} \tag{2.2}$$

2.4. PPS Convergence Theory. Before discussing convergence theory, we present some useful definitions and assumptions.

In any practical situation, $\Delta_{\text{tol}} > 0$. However, for the purposes of studying the asymptotic behavior of the algorithm, $\Delta_{\text{tol}} = 0$.

The following assumptions on the function are employed later theorems.

ASSUMPTION 1. The set $\mathcal{L}_f(x_0) = \{x \in \Omega : f(x) \le f(x_0)\}$ is bounded.

ASSUMPTION 2. The function f is bounded below on Ω.

ASSUMPTION 3. The function f is continuously differentiable on $\mathcal{L}_f(x_0)$.

ASSUMPTION 4. The gradient ∇f is Lipschitz continuous with constant M on $\mathcal{L}_f(x_0)$.

In constrained optimization, we can measure progress to a KKT point using the following analogue of $\| \nabla f(x) \|$. For $x \in \Omega$, define

$$\chi(x) = \max_{\substack{x+w \in \Omega \\ \| w \| \leq 1}} -\nabla f(x)^T w.$$

The function χ is particularly suitable for the analysis of pattern search (and GSS) methods [10, 11]. It has the following three properties [2]: $\chi(x)$ is continuous, $\chi(x) \geq 0$, and $\chi(x) = 0$ if and only if x is a KKT point. Note that $\chi(x) \equiv \| \nabla f(x) \|$ if $\Omega = \mathbb{R}^n$.

Now that the assumptions and notation have been established, we can present the convergence results for PPS.

THEOREM 2.1 ([10] and references therein). *Consider the optimization problem (1.1), satisfying Assumptions 1–4. Let the PPS algorithm in Figure 2.1 satisfy Conditions 1, 2, either 3 or 4, and 6. Then*

$$\liminf_{k \to +\infty} \| \nabla f(x_k) \| = 0.$$

THEOREM 2.2 ([10, 11] and references therein). *Consider the optimization problem (1.2), satisfying Assumptions 1–4. Let the PPS algorithm in Figure 2.1 satisfy Conditions either 3 or 4, 5, and 6. Then*

$$\liminf_{k \to +\infty} \chi(x_k) = 0.$$

3. APPS. The premise of APPS is that greater efficiency in parallel processor utilization will enable faster solution of many problems. The original peer-to-peer version has indeed demonstrated faster execution times [8]. Here we present a new manager-worker design; a comparison between the manager-worker and peer-to-peer approaches is presented in Appendix A.

As mentioned in §2, the synchronization point in pattern search occurs in Step 2 of Figure 2.1, where the algorithm is required to wait until the evaluation of every trial point is complete before continuing. The difference between the synchronous and asynchronous version is that the asynchronous version need not wait until all function evaluations complete before moving on to the decision step (Step 3). Instead, the points with incomplete function evaluations are stored in a queue, and the algorithm moves ahead based on the best information available to it.

The flexibility of APPS necessitates a small amount of additional bookkeeping, as observed in [8]. Each trial point must "remember" how it was generated; more specifically, a trial point generated at iteration k using direction i stores the following additional information:

1. its parent x_k,
2. its parent's function value $f(x_k)$,
3. its direction index i, and
4. its step length $\Delta_k^{(i)}$ (defined below).

It is not necessary that the actual parent be stored; instead, a unique identifier is sufficient. In terms of implementation, the additional storage is negligible.

The manager-worker APPS algorithm, presented in in Figure 3.1, has the same structure as PPS in Figure 2.1. We discuss the major differences.

The notation is the same as for PPS, with the following exceptions and additions. There is no longer a single step length Δ_k at step k; instead, there is a step length associated with each direction, denoted by $\Delta_k^{(i)}$. As before, we assume that $\tilde{\Delta}_k^{(i)} = \Delta_k^{(i)}$ in the unconstrained case. We introduce a minimum step length, $\Delta_{\min}$, defined by the integer $\Gamma_{\min}$. There is an evaluation queue which may not be completely emptied in each iteration. Correspondingly, we introduce the set $\mathcal{A}_k$ containing the indices of the search directions that, at the start of iteration k, are "active"; in other words, those directions that have an associated trial point in the evaluation queue. Further, we define $q_{\max}$ to be the maximum number of points the queue holds.

In Step 1 of Figure 3.1, the trial points are generated. The selection criteria for generating new trial points has changed slightly and now takes into account whether or not a given search direction is "active". The set $\mathcal{A}_{k+1}$ is set during this step, and it may be reset or modified in Step 4 or Step 5.

In Step 2 of Figure 3.1, a set of evaluated trial points, denoted $\mathbf{Y}_k$, is collected. In contrast to Step 2 of Figure 2.1, this step does not wait until all trial points have been evaluated before moving on. Thus, it may be the case that $\mathbf{Y}_k \neq \mathbf{X}_k$ and further that $\mathbf{Y}_k \not\subseteq \mathbf{X}_k$.

Step 3 of Figure 3.1 now selects a *subset* of the trial points to consider for a simple decrease comparison with respect to the current best point. The subset includes those points that satisfy a sufficient decrease condition *with respect to their corresponding parent function values*. The specific criteria are presented in Figure 3.2 and discussed in more detail in §3.1 and §3.2.

In the case of a successful iteration (Step 4), the primary difference between APPS (Figure 3.1) and PPS (Figure 2.1) is the step length update. In both cases, the step length is updated to be the same as the step length that produced y_k. In PPS, this is simply Δ_k. However, in APPS, the step used to produce y_k is "remembered" as part of the extra bookkeeping described above. All p_{k+1} step lengths are reset to the larger of either this "remembered" step length or the quantity $\Delta_{\min}$. If $\Delta_{\min} \leq \Delta_{\text{tol}}$, this has no affect in practice, but it is important in the convergence theory (where it cannot be less than Δ_{tol} since $\Delta_{\text{tol}} = 0$). A successful iteration clears the active directions, so $\mathcal{A}_{k+1}$ is reset to the empty set. At this point, the evaluation queue needs to be pruned to prevent it from growing too large; such a measure has analytical (see Condition 9) as well as practical benefits. Any or all points may be pruned.

In the case of an unsuccessful iteration (Step 5 in Figure 3.1), the step lengths are reduced individually depending on the trial points in $\mathbf{Y}_k$. Specifically, each evaluated trial point is considered, and if the trial point's parent is not x_k, then it is discarded. (Recall that keeping track of the parent is part of the bookkeeping described above.) Otherwise, the corresponding step is reduced by a factor of two. The correct step is identified by the direction index that was used to generate the trial point (also part of the bookkeeping). Termination is essentially the same, except that there are now p_{k+1} steps, all of which must be less than the specified tolerance before the algorithm terminates.

3.1. APPS with Simple Decrease. In the simple decrease version of APPS, Conditions 1–3 are imposed as in the synchronous version discussed in §2.1. Condition 6 is replaced with Condition 7 (see §3.3); the new condition handles the multiple,

Initialization.

Set x_0 be some initial guess.
Set $\mathcal{D}_0 = \left\{d_0^{(1)}, \ldots, d_0^{(p_0)}\right\}$ to be the initial set of search directions.
Let $\Delta_{\text{tol}} > 0$ be the step-length convergence tolerance.
Set $\Delta_0^{(i)} = \Delta_0 > \Delta_{\text{tol}}$ for $i = 1, \ldots, p_0$ to be the initial step lengths.
Let $\Gamma_{\min} \in \mathbb{Z}$ with $\Gamma_{\min} \geq 0$. Set $\Delta_{\min} = 2^{-\Gamma_{\min}}\Delta_0$.
Set $\mathcal{A}_0 = \emptyset$. Let $q_{\max}$ be the evaluation queue size.

Iteration. For $k = 0, 1, \ldots$

STEP 1. Generate a (possibly empty) set of trial points

$$\mathbf{X}_k = \left\{x_k + \tilde{\Delta}_k^{(i)} d_k^{(i)} \ : \ 1 \leq i \leq p_k, \ \ i \notin \mathcal{A}_k, \text{ and } \tilde{\Delta}_k^{(i)} > \Delta_{\text{tol}}\right\}.$$

Then, send the set of points $\mathbf{X}_k$ to the evaluation queue.
Set $\mathcal{A}_{k+1} = \{i \ : \ \tilde{\Delta}_k^{(i)} > \Delta_{\text{tol}}\}$.

STEP 2. Collect a non-empty set $\mathbf{Y}_k$ of evaluated trial points.

STEP 3. Let $\bar{\mathbf{Y}}_k \subseteq \mathbf{Y}_k$ be the subset of trial points that satisfy the sufficient decrease condition (see Figure 3.2). If there exists a trial point $y_k \in \bar{\mathbf{Y}}_k$ such that $f(y_k) < f(x_k)$, then goto Step 4; else goto Step 5.

STEP 4. The iteration is successful.

- Set $x_{k+1} = y_k$.
- Choose a new $\mathcal{D}_{k+1} = \left\{d_{k+1}^{(1)}, \ldots, d_{k+1}^{(p_{k+1})}\right\}$.
- Let $\hat{\Delta}$ denote the "remembered" step length for y_k.
- Set $\Delta_{k+1}^{(i)} = \max\{\hat{\Delta}, \Delta_{\min}\}$ for $i = 1, \ldots, p_{k+1}$.
- Reset $\mathcal{A}_{k+1} = \emptyset$.
- Prune the evaluation queue to $(q_{\max} - p_{k+1})$ or fewer entries.
- Go to Step 1.

STEP 5. The iteration is unsuccessful.

- Set $x_{k+1} = x_k$.
- Set $\mathcal{D}_{k+1} = \mathcal{D}_k$ (and $p_{k+1} = p_k$).
- Let $\mathcal{I}_k$ denote the corresponding "remembered" direction indices of trial points in $\mathbf{Y}_k$ whose "remembered" parent is x_k.
- Set $\Delta_{k+1}^{(i)} = \begin{cases} \frac{1}{2}\Delta_k^{(i)}, & \text{if } i \in \mathcal{I}_k \\ \Delta_k^{(i)}, & \text{if } i \notin \mathcal{I}_k \end{cases}$ for $i = 1, \ldots, p_{k+1}$.
- Update $\mathcal{A}_{k+1} \leftarrow \mathcal{A}_{k+1} \setminus \mathcal{I}_k$.
- If $\Delta_{k+1}^{(i)} < \Delta_{\text{tol}}$ for $i = 1, \ldots, p_{k+1}$, **terminate**. Else, goto Step 1.

FIG. 3.1. *Manager-Worker APPS Algorithm*

For each $y \in \mathbf{Y}_k$

– Let $f(\hat{y})$ denote the "remembered" function value of the parent of y.

– Let $\hat{\Delta}$ denote the "remembered" step length for y.

– Define the set $\bar{\mathbf{Y}}_k = \left\{ y \in \mathbf{Y}_k \ : \ f(y) < f(\hat{y}) - \rho(\hat{\Delta}) \right\}$.

FIG. 3.2. *Sufficient Decrease Condition for Step 3 in APPS (Figure 3.1)*

possibly different step lengths.

In the simple decrease case, we can assume, without loss of generality, that $\bar{\mathbf{Y}}_k = \mathbf{Y}_k$ in Step 3. The reasoning is that it cannot be the case that a trial point y satisfies $f(y) < f(x_k)$ but not $f(y) < f(\hat{y})$ (where $\hat{y}$ is the parent of y). Since $\hat{y}$ is a previous "best point", it must be true that $f(x_k) \leq f(\hat{y})$.

3.2. APPS with Sufficient Decrease. In the sufficient decrease version of APPS, Conditions 1, 2, 4, and 7 (the replacement for Condition 6) are enforced.

Implementing sufficient decrease in an asynchronous environment adds a layer of difficulty because the sufficiency condition is with respect to the parent of the trial point. There is no assurance that x_k is the parent of the trial point, as in the synchronous case. Consequently, in the asynchronous case, determining whether or not an evaluated trial point is a new "best point" becomes a two step process. First, the point is checked to see if it satisfies a sufficient decrease condition with respect to its parent's function value (see Figure 3.2). Second, it is assessed to see if simple decrease with respect to the current x_k is satisfied.

For example, consider an evaluated trial point y at iteration k. Let $f(\hat{y})$ be the "remembered" parent function value, and let $\hat{\Delta}$ be the "remembered" step length. In order to be a candidate for new "best point", y must satisfy

$$f(y) < \min\{f(\hat{y}) - \rho(\hat{\Delta}), f(x_k)\}.$$

3.3. APPS with Bound Constraints. Bound constraints are handled essentially the same as before. Now, however, Condition 6 must be modified to reflect the p_k independent step lengths. Condition 7 results.

CONDITION 7. If $x_k + \Delta_k^{(i)} d_k^{(i)} \in \Omega$, then $\tilde{\Delta}_k^{(i)} = \Delta_k^{(i)}$.

Similarly, the step calculations in (2.1) and (2.2) need to be modified. The following choice is suitable for either simple or sufficient decrease [11]:

$$\tilde{\Delta}_k^{(i)} = \begin{cases} \Delta_k^{(i)}, & \text{if } x_k + \Delta_k^{(i)} d_k^{(i)} \in \Omega, \\ 0, & \text{otherwise.} \end{cases} \tag{3.1}$$

In the sufficient decrease case, taking the longest possible feasible step is an alternative [11]. Define $\tilde{\Delta}_k^{(i)}$ as the solution to

$$\begin{array}{ll} \max & \tilde{\Delta} \\ \text{subject to} & 0 \leq \tilde{\Delta} \leq \Delta_k^{(i)}, \\ & x_k + \tilde{\Delta}\, d_k^{(i)} \in \Omega. \end{array} \tag{3.2}$$

4. An illustrated example of APPS. A two-dimensional example is presented in Figures 4.1–4.2. The contour plot of the objective function uses darker shading to indicate lower function values. Each figure represents the state of the algorithm at an iteration. The square denotes the best point (i.e., x_k) at that iteration, and the circles denote points in the evaluation queue after Step 1 is completed. The lines denote the search directions. For simplicity, we use the same set of search directions throughout: $\mathcal{D}_k = \{e_1, e_2, -e_1, -e_2\}$. The points are labeled with letters, and the algorithm is initialized with the starting point $x_0 = \mathbf{a}$ and an initial step length of $\Delta_0 = 1$.

Before we continue, it is important to note the following. At each iteration, the set of evaluated trial points returned in Step 2 could be any non-empty subset of points in the evaluation queue — the choice of this subset is not controlled by the APPS algorithm. Thus, the set $\mathbf{Y}_k$ at each iteration can be interpreted as the result of random chance. (In truth, we have crafted the selection in this example to demonstrate certain features of the algorithm.) The algorithm makes no assumption that the points in the evaluation queue finish being evaluated in any particular order.

A couple of algorithmic choices also influence our example. In Step 2, we assume that no sufficient decrease criteria is employed (i.e., $\rho \equiv 0$) so that $\bar{\mathbf{Y}}_k \equiv \mathbf{Y}_k$ for all k. In Step 4, we assume that $q_{\max} = 6$.

Iteration 0 illustrates an unsuccessful iteration. We assume that only two evaluations (**b** and **e**) are returned in Step 2. Neither **b** nor **e** improve the function value, so the iteration is unsuccessful. The parent of both **b** and **e** is $x_0 = \mathbf{a}$ and their corresponding direction indices are 0 and 3, thus $\mathcal{I}_0 = \{0, 3\}$ in Step 5. The step lengths corresponding to those directions are reduced by a factor of 2. Note that points **c** and **d** remain in the evaluation queue.

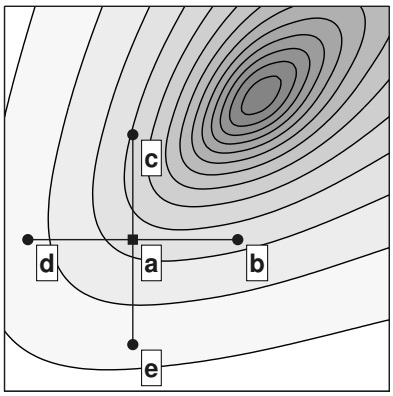
Iteration 1 illustrates a successful iteration. In Step 1, this iteration only generates two new trial points (**f** and **g**) because Directions 1 and 2 are already active (i.e., $\mathcal{A}_1 = \{1, 2\}$). In Step 2, we assume points **f** and **g** are returned. Since **f** reduces the function value, this iteration is successful. All step lengths for the next iteration are reset to the length of the step length that produced **f** (i.e., $\hat{\Delta} = \frac{1}{2}$). No pruning of the queue is necessary.

Iteration 2 illustrates "disconnected" points in the evaluation queue and a successful iteration that results from one of these disconnected points. Two points remain in the evaluation queue, and four new trial points are generated and added in Step 1. Because $x_2 \neq x_1$, the older points in the queue are no longer connected to the current best point and so are referred to as disconnected. In Step 2, we assume the evaluation for the point **c** is finally returned (along with **j** and **h**) and results in another successful step. All step lengths for the next iteration are set to the step length that produced **c** (i.e., $\hat{\Delta} = 1$), and this step is from Iteration 0. This time, the evaluation queue is pruned by removing the oldest point, **d**.

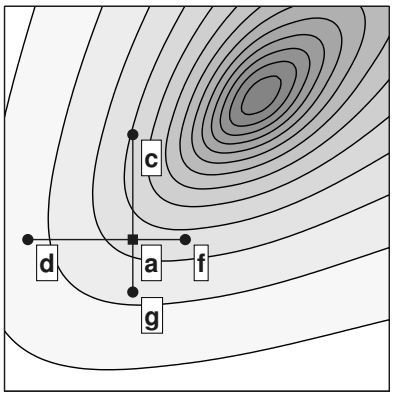
Iteration 3 illustrates two points with improved function values returning simultaneously (**i** and **l**). As with synchronous parallel pattern search, we will assume that we take the best one, though this is not strictly necessary in terms of the theory.

Iteration 4 illustrates an unsuccessful iteration that results in no changes and thus no new trial points in the next iteration. Here, points **k** and **n** finish their evaluations and the result is an unsuccessful iteration. However, since both points are disconnected (i.e., neither has **l** as its parent), no step lengths are reduced in Step 5.

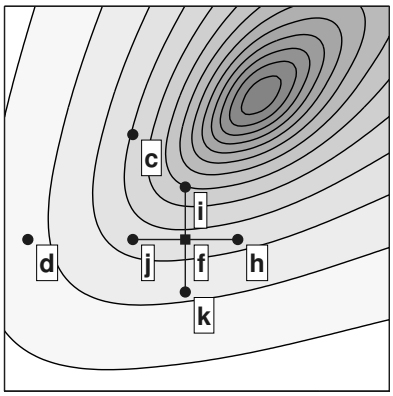
At the beginning of Iteration 5, no new trial points are generated, and four points remain in the evaluation queue. The process continues from there, marching toward a local minimizer.



Iteration 0
$x_0 = \mathrm{a}$
$\Delta_0^{(0)} = \Delta_0^{(1)} = \Delta_0^{(2)} = \Delta_0^{(3)} = 1$
$\mathcal{A}_0 = \emptyset$
$X_0 = \{\mathrm{b}, \mathrm{c}, \mathrm{d}, \mathrm{e}\}$ Queue $= \{\mathrm{b}, \mathrm{c}, \mathrm{d}, \mathrm{e}\}$
$Y_0 = \{\mathrm{b}, \mathrm{e}\}$ Queue $= \{\mathrm{c}, \mathrm{d}\}$
Unsuccessful ($\mathcal{I}_0 = \{0, 3\}$)



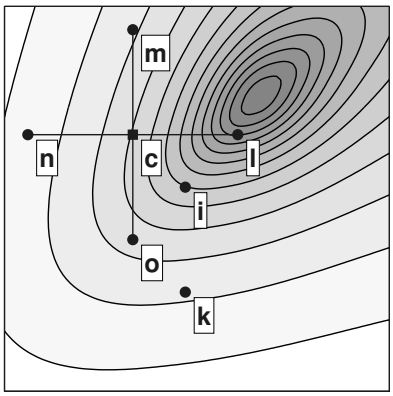
Iteration 1
$x_1 = \mathrm{a}$
$\Delta_1^{(0)} = \Delta_1^{(3)} = \frac{1}{2}, \Delta_1^{(1)} = \Delta_1^{(2)} = 1$
$\mathcal{A}_1 = \{1, 2\}$
$X_1 = \{\mathrm{f}, \mathrm{g}\}$ Queue $= \{\mathrm{c}, \mathrm{d}, \mathrm{f}, \mathrm{g}\}$
$Y_1 = \{\mathrm{f}, \mathrm{g}\}$ Queue $= \{\mathrm{c}, \mathrm{d}\}$
Successful (f) Pruned Queue $= \{\mathrm{c}, \mathrm{d}\}$



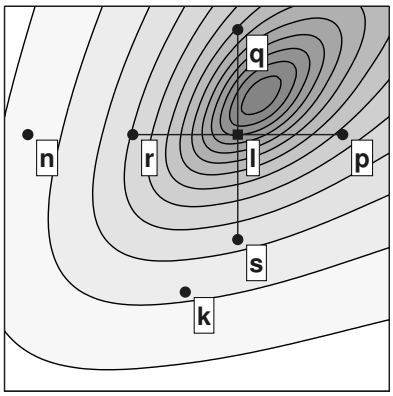
Iteration 2
$x_2 = \mathrm{f}$
$\Delta_2^{(0)} = \Delta_2^{(1)} = \Delta_2^{(2)} = \Delta_2^{(3)} = \frac{1}{2}$
$\mathcal{A}_2 = \emptyset$
$X_2 = \{\mathrm{h}, \mathrm{i}, \mathrm{j}, \mathrm{k}\}$ Queue $= \{\mathrm{c}, \mathrm{d}, \mathrm{h}, \mathrm{i}, \mathrm{j}, \mathrm{k}\}$
$Y_2 = \{\mathrm{c}, \mathrm{j}, \mathrm{h}\}$ Queue $= \{\mathrm{d}, \mathrm{i}, \mathrm{k}\}$
Successful (c) Pruned Queue $= \{\mathrm{i}, \mathrm{k}\}$

FIG. 4.1. *Example APPS Iterations: Part 1*

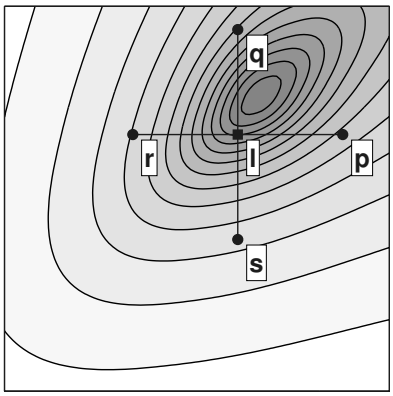
5. APPS Convergence Theory. We develop convergence theory for APPS, concluding with results analogous to Theorems 2.1 and 2.2. The analysis borrows heavily from [12, 10, 11]. We begin in §5.1 by determining bounds on $\| \nabla f(x_k) \|$ and $\chi(x_k)$ in terms of the step lengths. Next in §5.2, we present some results showing that a subsequence of the step lengths go to zero. Finally, in §5.3, we give the convergence results.



Iteration 3
$x_3 = \mathrm{c}$
$\Delta_3^{(0)} = \Delta_3^{(1)} = \Delta_3^{(2)} = \Delta_3^{(3)} = 1$
$\mathcal{A}_3 = \emptyset$
$X_3 = \{\mathrm{l}, \mathrm{m}, \mathrm{n}, \mathrm{o}\}$ Queue $= \{\mathrm{i}, \mathrm{k}, \mathrm{l}, \mathrm{m}, \mathrm{n}, \mathrm{o}\}$
$Y_3 = \{\mathrm{i}, \mathrm{l}, \mathrm{m}, \mathrm{o}\}$ Queue $= \{\mathrm{k}, \mathrm{n}\}$
Successful (l) Pruned Queue $= \{\mathrm{k}, \mathrm{n}\}$



Iteration 4
$x_4 = \mathrm{l}$
$\Delta_4^{(0)} = \Delta_4^{(1)} = \Delta_4^{(2)} = \Delta_4^{(3)} = 1$
$\mathcal{A}_4 = \emptyset$
$X_4 = \{\mathrm{p}, \mathrm{q}, \mathrm{r}, \mathrm{s}\}$ Queue $= \{\mathrm{k}, \mathrm{n}, \mathrm{p}, \mathrm{q}, \mathrm{r}, \mathrm{s}\}$
$Y_4 = \{\mathrm{k}, \mathrm{n}\}$ Queue $= \{\mathrm{p}, \mathrm{q}, \mathrm{r}, \mathrm{s}\}$
Unsuccessful ($\mathcal{I}_4 = \emptyset$)



Iteration 5
$x_5 = \mathrm{l}$
$\Delta_5^{(0)} = \Delta_5^{(1)} = \Delta_5^{(2)} = \Delta_5^{(3)} = 1$
$\mathcal{A}_5 = \{0, 1, 2, 3\}$
$X_5 = \emptyset$ Queue $= \{\mathrm{p}, \mathrm{q}, \mathrm{r}, \mathrm{s}\}$
...

FIG. 4.2. *Example APPS Iterations: Part 2*

It is implicitly assumed in the discussion of the asymptotic behavior that $\Delta_{\mathrm{tol}} = 0$ in Figure 3.1.

We make explicit the bound on the number of search directions in $\mathcal{D}_k$ in Condition 8. This is an implicit assumption in PPS.

CONDITION 8. There exists $p_{\max}$, independent of k, such that for all k, $p_k \leq p_{\max}$.

We also need to ensure that a trial point cannot languish in the evaluation queue indefinitely. This is also an implicit assumption in PPS.

CONDITION 9. If a trial point is submitted to the evaluation queue at iteration k, either its evaluation will have completed or it will have been pruned from the evaluation queue by iteration $k + \eta$.

Condition 9 is not saying that every function evaluation requires η iterations; instead, this is an upper bound on the number of iterations. In fact, the value of η may be quite large. A sticky point here is that an iteration does not necessarily correspond to a unit of time, so it is difficult to specify a maximum number of iterations for a function evaluation. However, if we assume that there is a unit of time associated with an iteration, this assumption can be enforced as follows. Without loss of generality, let the minimum iteration time correspond to 1 time unit. Now, suppose that there are w workers available for computing function evaluations and that the maximum number of time units required to compute a single function evaluation on a single worker is ϕ. Next, assume that trial points submitted to the evaluation queue are sent to the workers in order (although there is no assumption that the function evaluations finish in order). Finally, assume the maximum queue size is $q_{\max} \geq p_{\max}$ and is always pruned to a size no greater than $(q_{\max} - p_{k+1})$ for any successful iteration. Then, η can explicitly be computed as

$$\eta = \phi \left\lceil \frac{q_{\max}}{w} \right\rceil .$$

From an implementation point of view, the critical requirement is that the evaluation queue cannot be allowed to grow too large, and so the pruning in Step 4 is necessary for enforcing Condition 9.

5.1. Bounding the Measure of Stationarity. Theorem 5.1, below, applies to the unconstrained case and bounds the norm of the gradient as a function of the step length. This result and its proof are nearly identical to Theorem 3.3 in [10]. The difference is identifying those iterations for which such a bound can be shown. The necessary condition is that there must have been at least one contraction in every direction since the last successful iteration.

THEOREM 5.1. *Consider the optimization problem (1.1), satisfying Assumptions 3–4. Let the APPS algorithm in Figure 3.1 satisfy Conditions 1, 2, either 3 or 4, and 7. For every k such that*

$$\hat{\Delta}_k \equiv \max_{1 \leq i \leq p_k} \left\{ 2\Delta_k^{(i)} \right\} \leq \Delta_{\min}, \tag{5.1}$$

we have

$$\| \nabla f(x_k) \| \leq \frac{1}{c_{\min}} \left[M \hat{\Delta}_k \beta_{\max} + \frac{\rho(\hat{\Delta}_k)}{\hat{\Delta}_k \beta_{\min}} \right]. \tag{5.2}$$

Proof. By hypothesis (5.1), $\Delta_k^{(i)} < \Delta_{\min}$ for all $i = 1, \ldots, p_k$. This implies that there has been at least one contraction along each direction since that last successful iteration, so

$$0 \leq f(x_k + 2\Delta_k^{(i)} d_k^{(i)}) - f(x_k) + \rho(2\Delta_k^{(i)}) \text{ for } i = 1, \ldots, p_k. \tag{5.3}$$

The value of $2\Delta_k^{(i)}$ comes in because the current value of $\Delta_k^{(i)}$ is half of that for which the contraction was done. Also note that it is assumed $\tilde{\Delta}_k^{(i)} = \Delta_k^{(i)}$ by Condition 7.

Since, by hypothesis, Condition 1 is satisfied, there exists an $\bar{\imath} \in \{1, \ldots, p_k\}$ such that

$$c_{\min} \| \nabla f(x_k) \| \| d_k^{(\bar{\imath})} \| \leq -\nabla f(x_k)^T d_k^{(\bar{\imath})}. \tag{5.4}$$

Employing Assumption 3, the mean value theorem can be applied to (5.3) for $i = \bar{\imath}$ to conclude that there exists $\bar{\alpha} \in [0, 1]$ such that

$$f(x_k + 2\Delta_k^{(\bar{\imath})} d_k^{(\bar{\imath})}) - f(x_k) = 2\Delta_k^{(\bar{\imath})} \nabla f(x_k + \bar{\alpha} 2\Delta_k^{(\bar{\imath})} d_k^{(\bar{\imath})})^T d_k^{(\bar{\imath})}. \tag{5.5}$$

Combining (5.3) and (5.5), dividing through by $2\Delta_k^{(\bar{\imath})}$, and subtracting $\nabla f(x_k)^T d_k^{(\bar{\imath})}$ from both sides yields

$$\begin{aligned} -\nabla f(x_k)^T d_k^{(\bar{\imath})} &\leq \left(\nabla f(x_k + \bar{\alpha}\, 2\Delta_k^{(\bar{\imath})} d_k^{(\bar{\imath})}) - \nabla f(x_k) \right)^T d_k^{(\bar{\imath})} + \frac{\rho(2\Delta_k^{(\bar{\imath})})}{2\Delta_k^{(\bar{\imath})}} \\ &\leq \| \nabla f(x_k + \bar{\alpha}\, 2\Delta_k^{(\bar{\imath})} d_k^{(\bar{\imath})}) - \nabla f(x_k) \| \| d_k^{(\bar{\imath})} \| + \frac{\rho(2\Delta_k^{(\bar{\imath})})}{2\Delta_k^{(\bar{\imath})}}. \end{aligned}$$

Using (5.4) to replace the left hand side and dividing through by $\| d_k^{(\bar{\imath})} \|$, we now have

$$c_{\min} \| \nabla f(x_k) \| \leq \| \nabla f(x_k + \bar{\alpha} 2\Delta_k^{(\bar{\imath})} d_k^{(\bar{\imath})}) - \nabla f(x_k) \| + \frac{1}{\| d_k^{(\bar{\imath})} \|} \frac{\rho(2\Delta_k^{(\bar{\imath})})}{2\Delta_k^{(\bar{\imath})}}. \tag{5.6}$$

Since ∇f is Lipschitz (Assumption 4), the norm of any search direction is bounded (Condition 2), and $\bar{\alpha} \in [0, 1]$, it follows that

$$\| \nabla f(x_k + \bar{\alpha} 2\Delta_k^{(\bar{\imath})} d_k^{(\bar{\imath})}) - \nabla f(x_k) \| \leq M \left(\bar{\alpha}\, 2\Delta_k^{(\bar{\imath})} \| d_k^{(\bar{\imath})} \| \right) \leq M \hat{\Delta}_k \beta_{\max}. \tag{5.7}$$

Now, either ρ is identically zero (Condition 3) or $\rho(t)/t$ is monotonically decreasing as $t \downarrow 0$ (Condition 4). In either case,

$$\frac{1}{\| d_k^{(\bar{\imath})} \|} \frac{\rho(2\Delta_k^{(\bar{\imath})})}{2\Delta_k^{(\bar{\imath})}} \leq \frac{1}{\beta_{\min}} \frac{\rho(\hat{\Delta}_k)}{\hat{\Delta}_k}. \tag{5.8}$$

Note that the lower bound in Condition 2 is also employed in the above inequality.

Finally, combining (5.6), (5.7), and (5.8) and dividing by $c_{\min}$ yields (5.2). Hence, the claim. □

A similar result can be proved in the constrained case that is nearly identical to Theorem 4.4 in [11]. The same adaptations are used as in the unconstrained case, so the proof is left to the reader.

THEOREM 5.2. *Consider the optimization problem (1.2), satisfying Assumptions 1, 3, and 4. Let the APPS algorithm in Figure 3.1 satisfy Conditions either 3 or 4, 5, and 7. Let $\epsilon_\star > 0$ be given. Then there exists a constant c such that, for every k that satisfies,*

$$\hat{\Delta}_k \equiv \max_{1 \le i \le p_k} \left\{ 2\Delta_k^{(i)} \right\} \le \max \left\{ \Delta_{\min}, \frac{\epsilon_\star}{\beta_{\max}} \right\}, \tag{5.9}$$

we have

$$\chi(x_k) \le c \left[M \hat{\Delta}_k \beta_{\max} + \frac{\rho(\hat{\Delta}_k)}{\hat{\Delta}_k \beta_{\min}} \right]. \tag{5.10}$$

5.2. Globalization. Before we proceed to the globalization results, it is necessary to introduce some additional notation and assumptions.

We define $\Gamma_k^{(i)}$ for all k and $i = 1, \ldots, p_k$ as

$$\Gamma_k^{(i)} = -\log_2 \left(\frac{\Delta_k^{(i)}}{\Delta_0} \right). \tag{5.11}$$

We can conclude that $\Gamma_k^{(i)} \in \mathbb{Z}$ because any $\Delta_k^{(i)}$ is an integral power of 2 times the initial guess, i.e.,

$$\Delta_{k+1}^{(i)} = 2^{-\Gamma_k^{(i)}} \Delta_0.$$

The following Lemma 5.3 applies to APPS with a sufficient decrease condition. Because x_k is not necessarily the parent of x_{k+1}, the proof is somewhat different than its synchronous analogue, Theorem 3.4 in [10].

Additional notation is required for the proof. For any successful iteration k, a set of ancestors may be constructed for the point x_{k+1}. Let Π_k denote the iteration indices of the ancestors of x_{k+1} as well as $(k+1)$ itself, and let ℓ_k denote the number of ancestors. (The size of Π_k will be $\ell_k + 1$). To illustrate, consider again the example of §4. Iterations 1,2, and 3 are successful and yield the following ancestor sets:

$$\begin{array}{ll} \Pi_1 = \{0, 2\}, & \ell_1 = 1 \\ \Pi_2 = \{0, 3\}, & \ell_2 = 1 \\ \Pi_3 = \{0, 1, 4\} & \ell_3 = 2. \end{array}$$

It is important to note that 0 is necessarily in every set Π_k since x_0 is an ancestor to every point.

LEMMA 5.3. *Consider the optimization problem (1.1) or (1.2), satisfying Assumption 2. Let the APPS algorithm in Figure 3.1 satisfy Conditions 4, 8, and 9. Then there exists an index j and a set $\mathcal{K} \subset \{1, 2, \ldots\}$ such that*

$$\lim_{k \in \mathcal{K}} \Gamma_k^{(j)} = +\infty.$$

Proof. Suppose the lemma is false. Then there exists $\Gamma_\star$ such that $\Gamma_k^{(i)} < \Gamma_\star$ for all k and $i = 1, \ldots, p_k$. Consequently, the step lengths are bounded below:

$$\Delta_k^{(i)} \ge \Delta_\star = 2^{-\Gamma_\star} \Delta_0, \text{ for all } k \text{ and } i = 1, \ldots, p_k. \tag{5.12}$$

Then, by Condition 4, the forcing term is bounded below as well:

$$\rho(\Delta_k^{(i)}) \geq \rho_\star = \rho(\Delta_\star), \text{ for all } k \text{ and } i = 1, \ldots, p_k. \tag{5.13}$$

Suppose k is a successful iteration, and let $\Pi_k = \{i_1, i_2, \ldots, i_{\ell_k+1}\}$. Since each child-parent pair satisfies the sufficient decrease condition, we can apply a telescoping sum argument and (5.13) to obtain

$$f(x_{k+1}) - f(x_0) = \sum_{j=1}^{\ell_k} \left\{ f\left(x_{i_{j+1}}\right) - f\left(x_{i_j}\right) \right\} \geq \ell_k \, \rho_\star. \tag{5.14}$$

Another consequence of the lower bound on the step lengths in (5.12) is that each parent can only have a finite number of children. Specifically, a parent can have no more than $c = p_{\max}(\Gamma_\star + 1)$ children where the bound $p_{\max}$ comes from Condition 8. Thus, if iteration k is successful, x_{k+1} must have at least $\lceil k/c \rceil$ ancestors. Combining this information with (5.14) yields

$$f(x_{k+1}) \geq k \left(\frac{\rho_\star}{c} \right) + f(x_0).$$

Let $\mathcal{S}$ denote the subsequence of successful iterates. By Condition 9, the maximum number of iterations to evaluate a single trial point is bounded. This coupled with the method by which step lengths are updated implies that there must be infinitely many successful steps, i.e., $\mathcal{S}$ is infinite. Thus,

$$\lim_{k \in \mathcal{S}} f(x_{k+1}) \geq \lim_{k \in \mathcal{S}} k \left(\frac{\rho_\star}{c} \right) + f(x_0) = +\infty.$$

This contradicts Assumption 2. Hence, the claim. □

Before we can establish a result analogous to Lemma 5.3 for the simple decrease case, we first state a result regarding the structure of the iterates. It is a standard result, so no proof is provided here; see instead, e.g., [10].

PROPOSITION 5.4 ([12]). *Consider the optimization problem (1.1) or (1.2). Consider the APPS algorithm in Figure 3.1 satisfying Condition 3. Let $\Gamma > 0$ be a constant. Then, for any k with*

$$\Gamma \leq \Gamma_j^{(i)} \text{ for all } j \leq k, i = 1, \ldots, p_j$$

the following holds:

$$x_{k+1} = x_0 + 2^{-\Gamma} \, \Delta_0 \sum_{i=1}^{p} \zeta_k(i, \Gamma) \, d^{(i)}, \tag{5.15}$$

where $\zeta_k(i, \Gamma) \in \mathbb{Z}$ for each $i = 1, \ldots, p$ and $k = 0, 1, 2, \ldots$.

Given this result, the fact that the $\zeta_k(i, \Gamma)$ are integral, and the set $\mathbf{G}$ is as described in Condition 3, all iterates lie on on the lattice

$$\mathcal{M}(x_0, \Delta_0, \mathbf{G}, \Gamma) = \left\{ x_0 + 2^{-\Gamma} \Delta_0 \sum_{i=1}^{p} \zeta^{(i)} d^{(i)} \ : \ i \in \mathbb{Z} \right\}$$

We can now present our result.

LEMMA 5.5. *Consider the optimization problem (1.1) or (1.2), satisfying Assumption 1. Let the APPS algorithm in Figure 3.1 satisfy Conditions 3 and 9. Then, there exists an index j and a set $\mathcal{K} \subset \{1, 2, \ldots\}$ such that*

$$\lim_{k \in \mathcal{K}} \Gamma_k^{(j)} = +\infty.$$

Proof. Suppose not. Then there exists $\Gamma_\star$ such that $\Gamma_k^{(i)} < \Gamma_\star$ for all k and $i = 1, \ldots, p_k$. By Proposition 5.4, every iterate must lie on the lattice $\mathcal{M}(x_0, \Delta_0, \mathbf{G}, \Lambda_\star)$. On the other hand, by Assumption 1, every iterate must lie in the bounded set $\mathcal{L}_f(x_0)$. The intersection of $\mathcal{M}(x_0, \Delta_0, \mathbf{G}, \Gamma_\star)$ and $\mathcal{L}_f(x_0)$ is finite, so every successful iterate is drawn from a finite set. Next, observe that a successful point can only be successful once because Step 3 in Figure 3.1 requires strict improvement. Therefore, there can be only finitely many successful iterates; let $\hat{k}$ denote the last successful iterate.

After iteration $\hat{k}$, the set of search directions does not change. Further, by Condition 9, there is a contraction in the step length along each direction at least once per η iterations. Thus,

$$\lim_{k \to \infty} \max_{1 \le i \le p_k} \left\{ \Delta_k^{(i)} \right\} = 0.$$

So, necessarily, $\min\{\Gamma_k^{(i)}\} \to +\infty$. This contradicts our original assumption. Hence, the claim. ∎

Both Lemma 5.3 and Lemma 5.5 lead to the following general result regarding the step lengths. Additional notation is required for this proof. Define

$$\tilde{\Gamma}_k^{(i)} = \Gamma_k^{(i)} - \Gamma_{\min}.$$

This quantity is equal to the number of contractions required to go from $\Delta_{\min}$ to $\Delta_k^{(i)}$.

THEOREM 5.6. *Consider the optimization problem (1.1) or (1.2), satisfying Assumptions 1–2. Let the APPS algorithm in Figure 3.1 satisfy Conditions either 3 or 4, 8, and 9. Then, there exists a set $\mathcal{K} \subset \{1, 2, \ldots\}$ such that*

$$\lim_{k \in \mathcal{K}} \left\{ \max_{1 \le i \le p_k} \Delta_k^{(i)} \right\} = 0.$$

Proof. By either Lemma 5.3 (using Assumption 2 and Conditions 4, 8, and 9) or Lemma 5.5 (using Assumption 1 and Conditions 3 and 9), we have that there exists an index j and set $\mathcal{K}$ such that

$$\lim_{k \in \mathcal{K}} \Gamma_k^{(j)} = +\infty.$$

Without loss of generality, assume that

$$\Gamma_k^{(j)} > \eta \, (\Gamma_{\min} + 1) \text{ for all } k \in \mathcal{K}, \tag{5.16}$$

where η is as defined in Condition 9.

Then if $k \in \mathcal{K}$, by (5.16), $\tilde{\Gamma}_k^{(i)} > 0$ and there has not been a success for at least $\tilde{\Gamma}_k^{(j)}$ iterations. On the other hand, by (5.16), $\lfloor \tilde{\Gamma}_k^{(j)} / \eta \rfloor > 0$ and there has been at least $\lfloor \tilde{\Gamma}_k^{(j)} / \eta \rfloor$ contractions in all other directions. Thus,

$$\tilde{\Gamma}_k^{(i)} \ge \lfloor \tilde{\Gamma}_k^{(j)} / \eta \rfloor \text{ for } k \in \mathcal{K}, 1, \le i \le p_k, i \ne j.$$

Thus,

$$\lim_{k \in \mathcal{K}} \left\{ \min_{1 \leq i \leq p_k} \Gamma_k^{(i)} \right\} = +\infty.$$

Hence, the claim. □

5.3. Convergence Results. Using the machinery built in §5.1–5.2, results following Theorems 2.1 and 2.2 are immediate.

THEOREM 5.7. *Consider the optimization problem (1.1), satisfying Assumptions 1–4. Let the APPS algorithm in Figure 3.1 satisfy Conditions 1, 2, either 3 or 4, 7, 8, 9. Then*

$$\liminf_{k \to +\infty} \| \nabla f(x_k) \| = 0.$$

THEOREM 5.8. *Consider the optimization problem (1.2), satisfying Assumptions 1–4. Let the APPS algorithm in Figure 3.1 satisfy Conditions either 3 or 4, 5, 7, 8, 9. Then*

$$\liminf_{k \to +\infty} \chi(x_k) = 0.$$

6. Conclusions. We have presented a new version of APPS based on a manager-worker paradigm. This algorithm encapsulates either simple or sufficient decrease as well as the ability to handle bound constraints. A nice feature of this version of APPS is that it closely mirrors PPS (at least as described here).

In fact, neither PPS nor APPS has been presented in its most general form. For example, these algorithms handle updating the step lengths in a particular way. At unsuccessful iterations, the contraction factor in Step 5 is hard-wired to $\frac{1}{2}$; in fact, this could be any fixed value. Similarly, an expansion factor could be used on successful iterations in Step 4. In both cases, these terms could be adaptive (i.e., different at each iteration). We also assume that the search directions are fixed between successful iterations. This is not required for PPS; however, we have presented it that way because it is required for APPS.

Likewise, some of the assumptions and conditions employed in the convergence analysis can be relaxed. We need not assume that the gradient is Lipschitz (Assumption 4); instead, continuous differentiability is sufficient (see the note at the end of §3.6 in [10]). Part (d) in Condition 3 can be changed to say that either ρ is identically zero or it satisfies Condition 4; in other words, the argument based on lattice structure is independent of the decrease condition. Part (e) in Condition 3 can be generalized to say that the pseudo step can be anything of the form $2^{-\Gamma}\Delta_0$ for $\Gamma \in \mathbb{Z}$. Part (b) in Condition 4 is more restrictive than necessary for PPS (which only needs that $\rho(t)$ monotonically decreases), but this more restrictive assumption is needed by APPS. Condition 5 can be weakened, but the resulting condition is much more complex (see Condition 1 in [11]).

The convergence theory borrows heavily from the analysis of GSS in [10, 11] as well as the previous version of APPS [12]. The convergence results presented in §5 are *weak* convergence results because it is possible that only a subsequence of the iterates will converge to a minimizer. Although strong convergence results are possible in the synchronous case [10], it is unclear whether or not such assurances can be made for

the asynchronous algorithm because strong convergence requires that the algorithm always take the best direction at each iteration. Local convergence results exist for PPS [10] but are left as a topic for future study for APPS.

One objective of the redesign of APPS is to enable easier incorporation of methods for handling linear constraints. In that case, the search directions must conform to the nearby boundary [16, 11]. Thus, the ability to change the search directions in Step 4 makes this relatively simple. Incorporating changing directions in the peer-to-peer version would be substantially more complex.

This version of APPS is implemented in APPSPACK 4.0 [9], and numerical results on a set of problems in groundwater flow are presented in [5].

Acknowledgments. I am very grateful to my collaborators on APPS and GSS who have inspired many of the ideas in this paper: Virginia Torczon, Michael Lewis, Patty Hough, and Genetha Gray. Thanks also to Genetha Gray and Virginia Torczon for reading earlier versions of this manuscript and offering many helpful comments.

Appendix A. Evolution of Peer-to-Peer to Manager-Worker. The switch from the peer-to-peer version [8] to manager-worker was gradual and largely the result of user requests. As mentioned in the introduction, peer-to-peer APPS is based on the concept of what are called agents. Each agent handles a single direction (and up to one corresponding trial point) and launches its own workers to actually execute the function evaluation. Thus, there is one agent per search direction and the number of search directions is necessarily fixed. The working assumption is that there is one direction per processor and one processor per machine.

The first step in the evolution to the manager-worker design is motivated by multiprocessor (i.e., SMP) machines. On a cluster of machines that each have, say, four processors, it is more efficient to have one agent (as opposed to four) for every four search directions. The peer-to-peer design remained intact, but a single agent could handle multiple search directions. The directions sharing a common agent also shared one common "best point." In fact, this is equivalent to the original peer-to-peer model with instantaneous communication between appropriate subsets of the agents. Once agents were designed and implemented to handle multiple directions, having one agent handle all directions was trivial.

The difference between this first manager-worker concept and the algorithm described here is the handling of the search directions. Having a fixed set of search directions is fairly crucial to the peer-to-peer design. In particular, it is implemented so that there is at most one function evaluation per search direction at any given time. The "disconnected" points described in §4 cannot exist. Though it would certainly be possible to design a peer-to-peer APPS that allows the search directions to change as the optimization progresses, it is much simpler to do this in a manager-worker context.

REFERENCES

[1] C. Audet and J. E. Dennis, Jr., *Analysis of generalized pattern searches*, SIAM Journal on Optimization, 13 (2003), pp. 889–903.

[2] A. R. Conn, N. I. M. Gould, and P. L. Toint, *Trust-Region Methods*, MPS-SIAM Series on Optimization 1, SIAM, 2000.

[3] J. E. Dennis and V. Torczon, *Managing approximation models in optimization*, in Multidisciplinary Design Optimization: State of the Art, N. M. Alexandrov and M. Y. Hussaini, eds., SIAM, Philadelphia, 1997, pp. 330–347.

[4] G. E. Fagg and J. Dongarra, *FT–MPI: Fault tolerant mpi, supporting dynamic applications in a dynamic world*, in Proceedings of the 7th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface, Lecture Notes In Computer Science, London, UK, 2000, Springer-Verlag, pp. 346–353.

[5] K. R. Fowler, J. P. Reese, C. E. Kees, D. J. E. Dennis, C. T. Kelley, C. T. Miller, C. Audet, A. J. Booker, G. Couture, R. W. Darwin, M. W. Farthing, D. E. Finkel, J. M. Gablonsky, G. Gray, and T. G. Kolda, *A comparison of optimization methods for problems involving flow and transport phenomena in saturated subsurface systems*. in preparation.

[6] U. M. García-Palomares and J. F. Rodríguez, *New sequential and parallel derivative-free algorithms for unconstrained minimization*, SIAM Journal on Optimization, 13 (2002), pp. 79–96.

[7] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. S. Sunderam, *PVM: Parallel Virtual Machine: A Users' Guide and Tutorial for Network Parallel Computing*, MIT Press, Cambridge, Massachusetts, 1994.

[8] P. D. Hough, T. G. Kolda, and V. J. Torczon, *Asynchronous parallel pattern search for nonlinear optimization*, SIAM Journal on Scientific Computing, 23 (2001), pp. 134–156.

[9] T. G. Kolda et al., *Appspack version 4.0*. http://software.sandia.gov/appspack/, 2004.

[10] T. G. Kolda, R. M. Lewis, and V. Torczon, *Optimization by direct search: New perspectives on some classical and modern methods*, SIAM Review, 45 (2003), pp. 385–482.

[11] ———, *Stationarity results for generating set search for linearly constrained optimization*, Tech. Rep. SAND2003–8550, Sandia National Laboratories, Livermore, California, October 2003.

[12] T. G. Kolda and V. J. Torczon, *On the convergence of asynchronous parallel pattern search*, Tech. Rep. SAND2001–8696, Sandia National Laboratories, Livermore, California, February 2002.

[13] ———, *Understanding asynchronous parallel pattern search*, in High Performance Algorithms and Software for Nonlinear Optimization, G. Di Pillo and A. Murli, eds., vol. 82 of Applied Optimization, Kluwer Academic Publishers, Boston, 2003, pp. 316–335.

[14] R. M. Lewis and V. Torczon, *Rank ordering and positive bases in pattern search algorithms*, Tech. Rep. 96–71, Institute for Computer Applications in Science and Engineering, Mail Stop 132C, NASA Langley Research Center, Hampton, Virginia 23681–2199, 1996.

[15] ———, *Pattern search algorithms for bound constrained minimization*, SIAM Journal on Optimization, 9 (1999), pp. 1082–1099.

[16] ———, *Pattern search methods for linearly constrained minimization*, SIAM Journal on Optimization, 10 (2000), pp. 917–941.

[17] R. M. Lewis, V. Torczon, and M. W. Trosset, *Why pattern search works*, Optima, 59 (1998), pp. 1–7. Also available as ICASE Technical Report 98–57, Institute for Computer Applications in Science and Engineering, NASA Langley Research Center, Hampton, Virginia.

[18] S. Lucidi and M. Sciandrone, *A derivative-free algorithm for bound constrained optimization*, Computational Optimization and Applications, 21 (2002), pp. 119–142.

[19] ———, *On the global convergence of derivative-free methods for unconstrained optimization*, SIAM Journal on Optimization, 13 (2002), pp. 97–116.

[20] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra, *MPI: The Complete Reference, Volume 1, The MPI Core, 2nd Edition*, MIT Press, Cambridge, Massachusetts, 1998.

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