High Performance Algorithms and Software for Nonlinear Optimization pp. 316-335
G. Di Pillo and A. Murli, Editors

Understanding Asynchronous Parallel Pattern Search[1]

Tamara G. Kolda[2] (tgkolda@sandia.gov)
Computational Sciences and Mathematics Research Department, Sandia National Laboratories, Livermore, CA 94551–9217

Virginia J. Torczon[3] (va@cs.wm.edu)
Department of Computer Science, College of William & Mary, P. O. Box 8795, Williamsburg, VA 23187–8795

Abstract

Asynchronous parallel pattern search (APPS) is a nonlinear optimization algorithm that dynamically initiates actions in response to events, rather than cycling through a fixed set of search directions, as is the case for synchronous pattern search. This gives us a versatile concurrent strategy that allows us to effectively balance the computational load across all available processors. However, the semi-autonomous nature of the search complicates the analysis. We concentrate on elucidating the concepts and notation required to track the iterates produced by APPS across all participating processes. To do so, we consider APPS and its synchronous counterpart (PPS) applied to a simple problem. This allows us both to introduce the bookkeeping we found necessary for the analysis and to highlight some of the fundamental differences between APPS and PPS.

Keywords: nonlinear optimization, asynchronous parallel optimization, pattern search, global convergence, distributed computing, cluster computing.

[1]Revision: 2.3 Date: 2002/11/21 00:32:36

[2]This research was sponsored by the Mathematical, Information, and Computational Sciences Division at the United States Department of Energy and by Sandia National Laboratories, a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000.

[3]This research was funded by the Computer Science Research Institute at Sandia National Laboratories and by the National Science Foundation under Grant CCR-9734044.

1 Introduction

In this paper, we consider parallel variants of pattern search to solve nonlinear unconstrained optimization problems of the form

$$\min_{x \in \mathbb{R}^n} f(x), \quad \text{where } f : \mathbb{R}^n \to \mathbb{R}. \tag{1}$$

We assume that the evaluation of f is computationally expensive, hence our interest in using either distributed or parallel computing environments to solve the problem. We concentrate on the parallelization of the search strategy, rather than on the evaluation of f, though the techniques we discuss here can be adapted to handle problems for which the computation of f also can be distributed. While we assume that the function f is continuously differentiable, we further assume that ∇f is unavailable and that approximations to ∇f are not reliable. For such problems, pattern search methods are one possible solution technique since they neither require nor explicitly estimate derivatives. Pattern search methods also have a long history of success when applied to such problems (see [3] for some recent examples).

While software for parallel optimization methods that do not require estimates of derivatives has been available for some time (e.g., see [6, 5]), here we examine the question of parallelizing pattern search. Our goal is to address the fact that for many of the functions that we have considered, the amount of time required to finish a single evaluation of the function can vary appreciably. In [3] we detail factors that can contribute to such variations. The bottom line, though, is that if we rely on synchronized parallel optimization algorithms, then the variations in the function evaluation time can lead to idle time on the processes during the course of the search. Idle time means that computational resources are wasted as one or more processes wait for the rest of the participating processes to catch up.

We investigate the behavior of both an "obvious" parallel variant of pattern search (PPS) as well as asynchronous parallel pattern search (APPS) [3], which was designed to eliminate the idle time that plagues synchronized parallel search techniques such as PPS. Although we designed APPS with an eye toward analysis, it was a secondary consideration during the development of the algorithm and software. We now have finished the analysis of APPS [4]. Here we highlight some of the differences between PPS and APPS from an analytic point of view.

Specifically, we consider both PPS and APPS applied to a simple example. Using this example, we introduce the bookkeeping that we found necessary for the analysis and illustrate the fundamental differences between APPS and PPS. In particular, we concentrate on the question of tracking iterates *across* processes in the asynchronous case since that is what most complicates the analysis.

We emphasize that the system of bookkeeping we introduce here is *not* needed for the actual implementation of APPS. However, we found the bookkeeping necessary to prove that, under the usual assumptions for pattern search analysis, all the processes share a common accumulation point that is also a stationary point of (1).

2 An illustrative example

To illustrate the behavior of APPS we use the problem

$$\min_{x\in\mathbb{R}^2} f(x_1, x_2) = (x_1 - 3)^2 + (x_2 - 2)^2 + (x_1 + x_2 - 4)^2, \tag{2}$$

which is given as an illustrative exercise in [1]. The global minimizer of (2) is $x^* = \left(\frac{8}{3}, \frac{5}{3}\right)^T$, with $f(x^*) = \frac{1}{3}$.

We assume we have $p = 4$ processes, and let $\mathcal{P} = \{1, \ldots, p\}$ denote the set of processes. Each process is in charge of a single search direction. We let $\mathcal{D} = \{d_1, \ldots, d_p\}$ denote the set of search direction. For this example, we choose the positive and negative unit coordinate vectors to form the set of search directions; i.e.,

$$\mathcal{D} = \left\{ \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} -1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ -1 \end{pmatrix} \right\}.$$

We choose $x^0 = (3, 5)^T$ as the initial iterate and $\Delta^0 = 1$ as the initial value of the step-length control parameter.

3 PPS

Here we outline our variant of parallel pattern search (PPS), to which the existing global analysis [7] immediately applies. We then illustrate the potential for idle time when using this approach.

3.1 Preliminary Notation

We start by adopting the usual notion of an infinite sequence of iterations from $k = 1, 2, \ldots$. At the end of iteration $k - 1$ (we treat $k = 0$ as a special case to handle the initialization of the search), we assume every process knows the best point x^{k-1}, where by "best point" we mean that $f(x^{k-1})$ is the best known value of f. We assume that if multiple points produce the best known value of f, ties are broken in a predictable fashion. Associated with x^{k-1} is the step-length control parameter Δ^{k-1}. Each process $i \in \mathcal{P}$ ends iteration $k - 1$ by constructing its trial point $x^{k-1} + \Delta^{k-1} d_i$ and initiating an evaluation of $f(x^{k-1} + \Delta^{k-1} d_i)$. The simultaneous start of the function evaluations at the trial points on each of the p processes signals the start of iteration k.

When all of the participating processes are finished with their evaluation of f, they communicate these values to each other and determine the new values of x^k and Δ^k as follows. If there exists $i \in \mathcal{P}$ such that

$$f(x^{k-1} + \Delta^{k-1} d_i) < f(x^{k-1}),$$

then we say that $k \in \mathcal{S}$ where $\mathcal{S}$ denotes the set of *successful* iterations. Furthermore, we define $\omega(k) = i$ to be the index of the process that produced the decrease. In other words, $\omega : \mathcal{S} \to \mathcal{P}$. Using these definitions, the update rules are then

$$x^k = \begin{cases} x^{k-1} + \Delta^{k-1}\, d_{\omega(k)}, & \text{if } k \in \mathcal{S} \text{ and} \\ x^{k-1}, & \text{otherwise; and} \end{cases}$$

$$\Delta^k = \begin{cases} \lambda^k\, \Delta^{k-1}, & \text{if } k \in \mathcal{S} \text{ and} \\ \theta^k \Delta^{k-1}, & \text{otherwise.} \end{cases}$$

For this illustration, we choose $\theta^k = \frac{1}{2}$ for all $k \notin \mathcal{S}$. We choose $\lambda^k = 2$ if $\omega(k) = \omega(k-1)$, i.e., the same direction produces decrease for two or more successive iterations; otherwise, we choose $\lambda^k = 1$.

3.2 Applying PPS

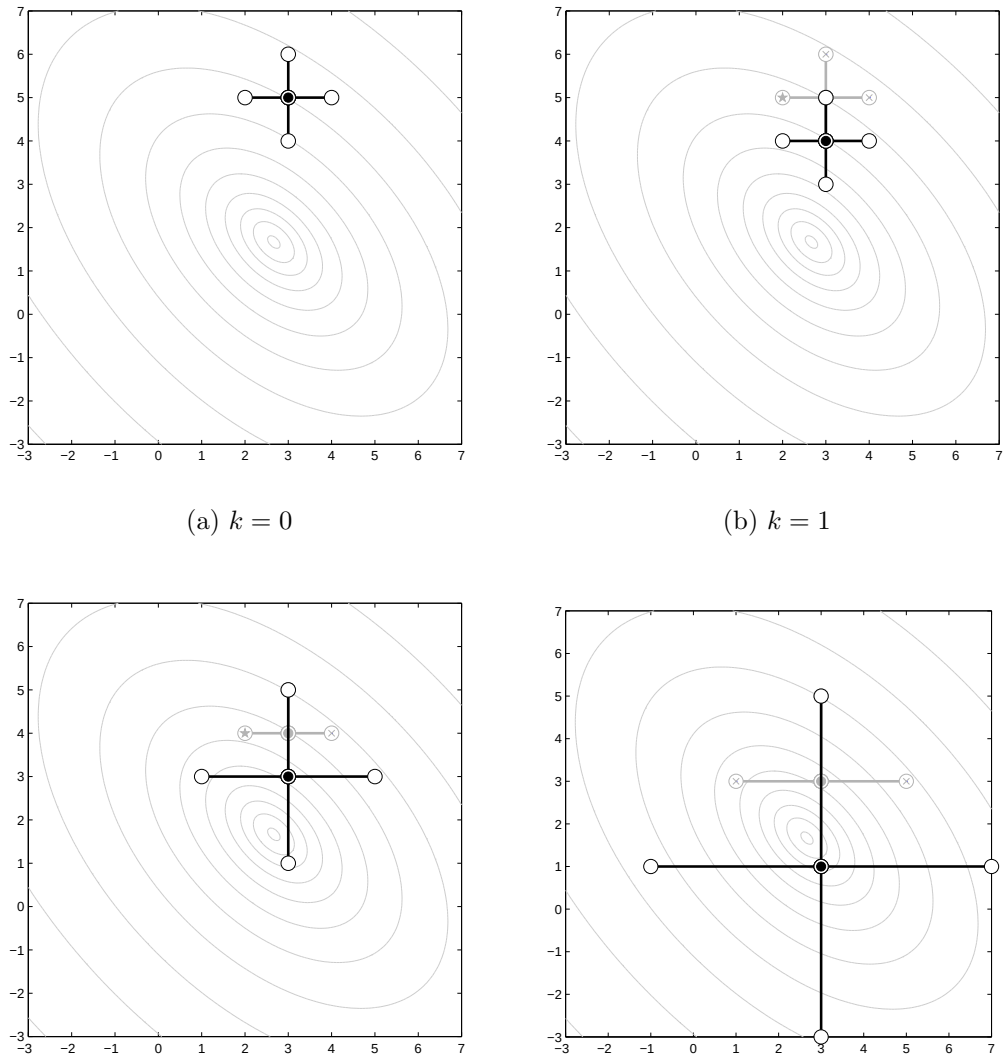
In Figure 1 we illustrate the first three iterations of PPS applied to (2). We show the level sets of f in the background. We use open circles to indicate the points at which the function is being evaluated. We use the symbol '•' inside an open circle to indicate the best known point. The step from the best known point to each of the trial points is also shown.

For $k > 0$, the results from the previous time step are shown in the background in gray. There are three possible outcomes. First, a trial point from the previous iteration may have become a new best point, in which case the search has moved to that point. Another alternative is that the evaluation at a trial point did improve upon the best known value at that iteration, indicated with the symbol '⋆' inside an open circle, but it was superseded by an even better point from another process. Finally, it is possible that the value at the trial point did not improve upon the best known value at that iteration, as indicated with the symbol '×' inside an open circle.

For this example, the first three iterations are successful, so $k \in \{1, 2, 3\} \subseteq \mathcal{S}$. Note that we expand the length of the step for iterations 2 and 3 since we have taken two or more successful steps, successively, along the same direction. Specifically, $\lambda^k = 1$ for $k = 1$ and $\lambda^k = 2$ for $k \in \{2, 3\}$.

3.3 Why PPS may waste computational resources

Implicit in the discussion above is the assumption that all the function evaluations finish at more or less the same time. That assumption, while convenient, is usually not what occurs for the problems that motivated APPS. This shortcoming can be attributed to any number of factors, as described in detail in [3]. The net effect of this discrepancy between what we have assumed and what we actually see in practice is that we may have processes that are sitting idle while waiting for other processes to finish their evaluation of f.



(a) $k = 0$ (b) $k = 1$

(c) $k = 2$ (d) $k = 3$

Figure 1: Four iterations of PPS applied to (2).

To illustrate this effect, we introduce the notion of a global clock, as is done in other proofs for asynchronous algorithms; e.g., see [2]. We let the infinite set $\mathcal{T} = \{1, 2, \ldots\}$ denote the index of time steps. We assume the time steps are of fine enough resolution that at most one event occurs per time step, per process. We assume that one time step is no greater than the minimum amount of time required to finish an evaluation of f at any x. In other words, we can finish at most one function evaluation per process per time step.

We revisit our example from the previous section and assume that the amount of time required for each function evaluation varies. Obviously, for (2) this is a contrivance we introduce for the sake of illustration; nonetheless, we use it to show what we have seen can happen in practice [3].

In Figure 2 we illustrate which time steps on each process are spent computing a function value and which are spent waiting for all the other processes to finish their function evaluations. The solid vertical bars running across all processes indicate the synchronization barrier inherent in PPS and signal both the end of the previous iteration and the the start of the next iteration. The solid horizontal bars represent time steps spent computing a function value; the small vertical bars represent the end of a function evaluation (recall that the previous iteration ends with the initiation of a function evaluation). The dots represent idle time.

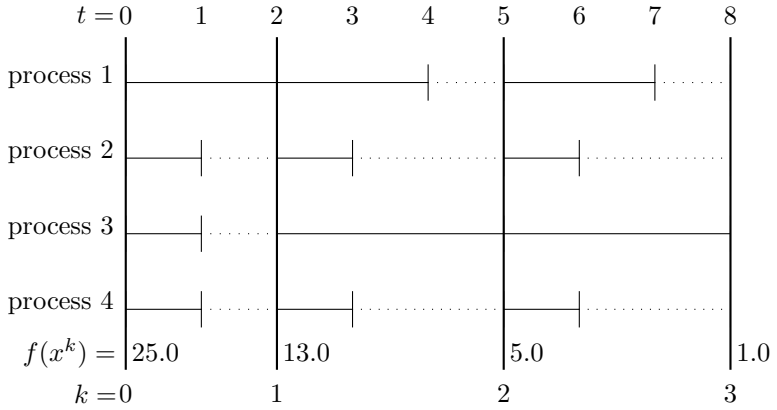


Figure 2: An illustration showing which time steps on each process in our example for PPS might be spent computing a function value versus sitting in an idle state.

Our illustration indicates that each function evaluation takes between 1 and 3 times steps. On average, each process is idle for roughly 3 of the 8 time steps in our illustration (i.e., about 40% of the time steps). If we could somehow use the idle time steps for computation, for this example we could potentially compute as many as 13 additional function evaluations (one per idle time step) through time step $t = 8$. Though, in general, this may be unduly optimistic, it is reasonable to expect that we should be able to compute more function values in the same number of time steps.

In the next section we show that APPS allows us to eliminate the idle time caused by the synchronization barrier in PPS. We achieve this by resorting to a family of adaptive, event-driven, peer-to-peer pattern search algorithms.

4 APPS

The general strategy for asynchronous parallel pattern search, from the perspective of a single process $i \in \mathcal{P}$, can be outlined as follows:

- evaluate $f(x_i^{best} + \Delta_i^{best}\, d_i)$;
- if $f(x_i^{best} + \Delta_i^{best}\, d_i) < f(x_i^{best})$, then broadcast (non-blocking) the result to all other processes;
- update local values x_i^{best} and Δ_i^{best} based on current local information and any messages that may have arrived from other processes;
- repeat.

By design, APPS removes the synchronization barrier in PPS, which is what introduces the idle time seen in Figure 2. The price we pay—both to the specification and to the analysis of APPS—is that each process has its own notion of the best known point seen so far, as well as its own value for Δ. Any success on one process is communicated to all other processes participating in the search, but the successful process carries on from its new best point without waiting for a response from the other processes. As it turns out, by adding a few mild conditions, we can still ensure the global convergence of the search. And, in practice, we see better performance [3].

Before proceeding with an illustration of our asynchronous parallel strategy, we introduce some useful notation.

4.1 Preliminary notation

First, we formalize the indexing by time steps. Instead of indexing based on a notion of iterations, we switch to indexing based on time steps, letting the set $\mathcal{T} = \{1, 2, \ldots\}$ denote the index of time steps. Thus, x_i^t is used for the best point known to process i at time step t. Similarly, Δ_i^t is the value of the step-length control parameter on process i at time step t. So if process i starts a function evaluation at time step t, the trial point at which the function evaluation will be made is $x_i^t + \Delta_i^t\, d_i$. As we mentioned earlier, the time steps are assumed to be of fine enough resolution so that at most one function evaluation finishes per process per time step.

4.1.1 Partitioning the time steps

We partition the time steps in $\mathcal{T}$. We define two sets that satisfy $\mathcal{T} = \mathcal{S}_i \cup \mathcal{U}_i$. The set $\mathcal{S}_i$ is the set of all time steps that are *successful* on process i, where by successful we mean that x_i^t is updated; i.e., a new best point has been found. The set $\mathcal{U}_i$ is the complementary set of all time steps that are *unsuccessful* on process i.

We further partition $\mathcal{S}_i$ as follows: $\mathcal{S}_i = \mathcal{I}_i \cup \mathcal{E}_i$, where $\mathcal{I}_i$ is the set of time steps that are *internal* successes on process i and $\mathcal{E}_i$ is the set of *external* successes on process i. An internal success occurs when a new x_i^t is found by process i. An external success on process i occurs when x_i^t is updated as a result of a message from some process $j \in \mathcal{P}$, $j \neq i$.

We also define the set $\mathcal{C}_i \subseteq \mathcal{U}_i$, where $\mathcal{C}_i$ is the set of time steps that are *contractions* on process i. We define contractions to be those time steps at which Δ_i^t is reduced. Finally, the set $\mathcal{U}_i \setminus \mathcal{C}_i$ is the subset of time steps at which no interesting events occur, i.e., $x_i^t = x_i^{t-1}$, and $\Delta_i^t = \Delta_i^{t-1}$.

We then define $\mathcal{T}_i$ as the subset of time steps at which an interesting event occurs on process i (i.e., either x_i^t and/or Δ_i^t is/are changed) so that $\mathcal{T}_i = \mathcal{S}_i \cup \mathcal{C}_i$.

4.1.2 Identifying the source of a change

For the purposes of the analysis, we need a way to identify the source of a change for any given update. To this end, we introduce the following three generating functions:

$$\begin{aligned}
\omega_i(t) &= \text{the generating process index for the update at time step } t \text{ on process } i, \\
\nu_i(t) &= \text{the time index for the completion of the function evaluation that produced the update at time step } t \text{ on process } i, \text{ and} \\
\tau_i(t) &= \text{the time index for the initiation of the function evaluation that produced the update at time step } t \text{ on process } i.
\end{aligned}$$

Here $\omega_i(\cdot) : \mathcal{S}_i \to \mathcal{P}$, $\nu_i(\cdot) : \mathcal{S}_i \to \mathcal{T}$, $\tau_i(\cdot) : \mathcal{S}_i \to \mathcal{T}$, and $0 \leq \tau_i(t) < \nu_i(t) \leq t$.

4.1.3 Defining x_i^t and Δ_i^t

The generating functions allow us to give the following general definitions for x_i^t and Δ_i^t. For every $t \in \mathcal{T}$, $t > 0$, the best point x_i^t for process $i \in \mathcal{P}$ is defined to be:

$$x_i^t = \begin{cases} x_{\omega_i(t)}^{\tau_i(t)} + \Delta_{\omega_i(t)}^{\tau_i(t)} \, d_{\omega_i(t)}, & \text{if } t \in \mathcal{S}_i, \text{ and} \\ x_i^{t-1}, & \text{otherwise.} \end{cases} \tag{3}$$

We initialize the procedure with $x_i^0 = x^0$, for all $i \in \mathcal{P}$.

For every $t \in \mathcal{T}$, $t > 0$, the step-length control parameter Δ_i^t for process $i \in \mathcal{P}$ is defined to be:

$$\Delta_i^t = \begin{cases} \lambda_{\omega_i(t)}^{\nu_i(t)} \, \Delta_{\omega_i(t)}^{\tau_i(t)}, & \text{if } t \in \mathcal{S}_i, \\ \theta_i^t \Delta_i^{t-1}, & \text{if } t \in \mathcal{C}_i, \text{ and} \\ \Delta_i^{t-1}, & \text{otherwise.} \end{cases} \tag{4}$$

Again, the initialization is assumed to be $\Delta_i^0 = \Delta^0$ for all $i \in \mathcal{P}$. For the purposes of our example, $\theta_i^t = \frac{1}{2}$ and $\lambda_i^t \in \{1, 2\}$ for all $i \in \mathcal{P}$ and for all $t \in \mathcal{T}$. These choices are consistent with the full specifications for these parameters given in [4].

4.2 Applying APPS

We illustrate the first eight time steps of applying APPS to (2). Because each process now acts as a semi-autonomous agent, we find it helpful to present the results in three different formats.

t	0	1	2	3	4	5	6	7	8
finish $f(trial_1)$	–	–	35.0	–	21.0	33.0	–	9.0	3.0
$f(\cdot)$ from 2	–	–	–	–	–	–	–	–	–
$f(\cdot)$ from 3	–	–	–	–	17.0	–	19.0	–	–
$f(\cdot)$ from 4	–	–	13.0	5.0	1.0	–	–	–	–
$\omega_1(t)$	–	–	4	4	4	–	–	–	–
$\tau_1(t)$	–	–	0	1	2	–	–	–	–
$\nu_1(t)$	–	–	1	2	3	–	–	–	–
λ_1^t	–	–	–	–	–	–	–	–	–
x_1^t	3.0	3.0	3.0	3.0	3.0	3.0	3.0	3.0	3.0
	5.0	5.0	4.0	3.0	1.0	1.0	1.0	1.0	1.0
$f(x_1^t)$	25.0	25.0	13.0	5.0	1.0	1.0	1.0	1.0	1.0
Δ_1^t	1.0	1.0	1.0	2.0	4.0	2.0	2.0	1.0	0.5
$trial_1 =$	4.0	–	4.0	–	7.0	5.0	–	4.0	3.5
$x_1^t + \Delta_1^t\, d_1$	5.0	–	4.0	–	1.0	1.0	–	1.0	1.5
finish $f(trial_2)$	–	41.0	–	–	32.5	25.0	5.0	1.0	0.5
$f(\cdot)$ from 1	–	–	–	–	–	–	–	–	–
$f(\cdot)$ from 3	–	–	17.0	19.0	–	–	–	–	–
$f(\cdot)$ from 4	–	–	–	1.0	13.0	5.0	–	–	–
$\omega_2(t)$	–	–	3	4	–	–	–	–	2
$\tau_2(t)$	–	–	1	2	–	–	–	–	7
$\nu_2(t)$	–	–	2	3	–	–	–	–	8
λ_2^t	–	–	–	–	–	–	–	–	1
x_2^t	3.0	3.0	1.0	3.0	3.0	3.0	3.0	3.0	3.0
	5.0	5.0	5.0	1.0	1.0	1.0	1.0	1.0	1.5
$f(x_2^t)$	25.0	25.0	17.0	1.0	1.0	1.0	1.0	1.0	0.5
Δ_2^t	1.0	0.5	2.0	4.0	4.0	2.0	1.0	0.5	0.5
$trial_2 =$	3.0	3.0	–	–	3.0	3.0	3.0	3.0	3.0
$x_2^t + \Delta_2^t\, d_2$	6.0	5.5	–	–	5.0	3.0	2.0	1.5	2.0
finish $f(trial_3)$	–	19.0	17.0	–	–	25.0	–	33.0	9.0
$f(\cdot)$ from 1	–	–	–	–	–	–	–	–	–
$f(\cdot)$ from 2	–	–	–	–	–	–	–	–	0.5
$f(\cdot)$ from 4	–	–	–	13.0	–	1.0	5.0	–	–
$\omega_3(t)$	–	3	3	4	–	4	–	–	2
$\tau_3(t)$	–	0	1	0	–	2	–	–	7
$\nu_3(t)$	–	1	2	1	–	3	–	–	8
λ_1^t	–	1	2	–	–	–	–	–	–
x_3^t	3.0	2.0	1.0	3.0	3.0	3.0	3.0	3.0	3.0
	5.0	5.0	5.0	4.0	4.0	1.0	1.0	1.0	1.5
$f(x_3^t)$	25.0	19.0	17.0	13.0	13.0	1.0	1.0	1.0	0.5
Δ_3^t	1.0	1.0	2.0	1.0	1.0	4.0	4.0	2.0	0.5
$trial_3 =$	2.0	1.0	-1.0	–	–	-1.0	–	1.0	2.5
$x_3^t + \Delta_3^t\, d_3$	5.0	5.0	5.0	–	–	1.0	–	1.0	1.5
finish $f(trial_4)$	–	13.0	5.0	1.0	–	41.0	–	13.0	5.0
$f(\cdot)$ from 1	–	–	–	–	–	–	–	–	–
$f(\cdot)$ from 2	–	–	–	–	–	–	–	–	–
$f(\cdot)$ from 3	–	–	19.0	17.0	–	–	–	–	–
$\omega_4(t)$	–	4	4	4	–	–	–	–	–
$\tau_4(t)$	–	0	1	2	–	–	–	–	–
$\nu_4(t)$	–	1	2	3	–	–	–	–	–
λ_4^t	–	1	2	2	–	–	–	–	–
x_4^t	3.0	3.0	3.0	3.0	3.0	3.0	3.0	3.0	3.0
	5.0	4.0	3.0	1.0	1.0	1.0	1.0	1.0	1.0
$f(x_4^t)$	25.0	13.0	5.0	1.0	1.0	1.0	1.0	1.0	1.0
Δ_4^t	1.0	1.0	2.0	4.0	4.0	2.0	2.0	1.0	0.5
$trial_4 =$	3.0	3.0	3.0	3.0	–	3.0	–	3.0	3.0
$x_4^t + \Delta_4^t\, d_4$	4.0	3.0	1.0	-3.0	–	-1.0	–	0.0	0.5

Table 1: Results from the first eight time steps of APPS applied to (2).

- Table 1 provides a complete accounting of all events across all four processes up through time step $t = 8$. At every time step, we specify x_i^t, $f(x_i^t)$, and Δ_i^t. Where appropriate, we also specify the values for the generating functions, expansion parameter, trial point, and trial and incoming function values. We do not indicate the contraction parameter since we always choose $\theta_i^t = \frac{1}{2}$ for all $t \in \mathcal{C}_i$.

- Figure 3 represents some of the data from Table 1 using timelines. In particular, we can see the start of each function evaluation (and the conclusion of the previous function evaluation), represented using a vertical bar. We represent internal successes, external successes, and contractions by circles, squares, and $\times$'s, respectively. Changes in $f(x_i^t)$ are shown underneath each timeline. This figure is particularly useful for tracing messages. Solid arcs indicate messages that result in an external success on the receiving process. Dotted arcs indicate messages that have no effect on the receiving process (i.e., they simply are discarded).

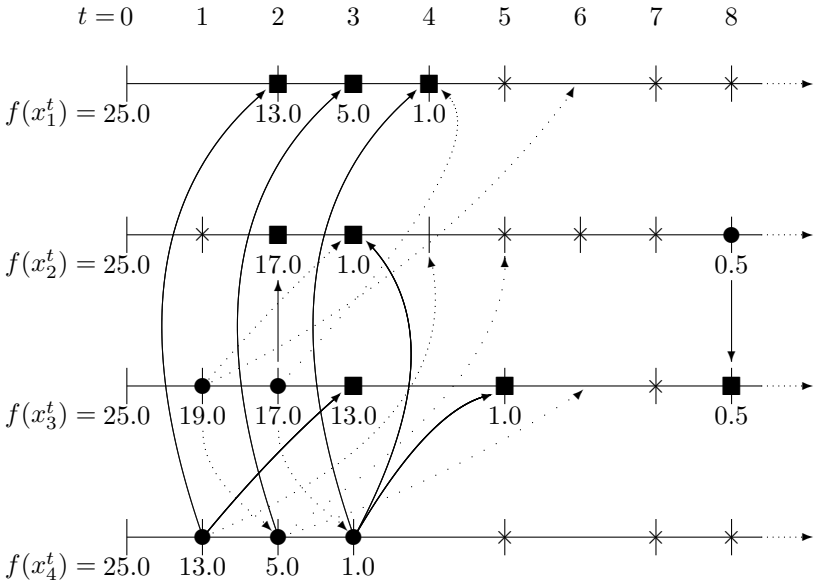
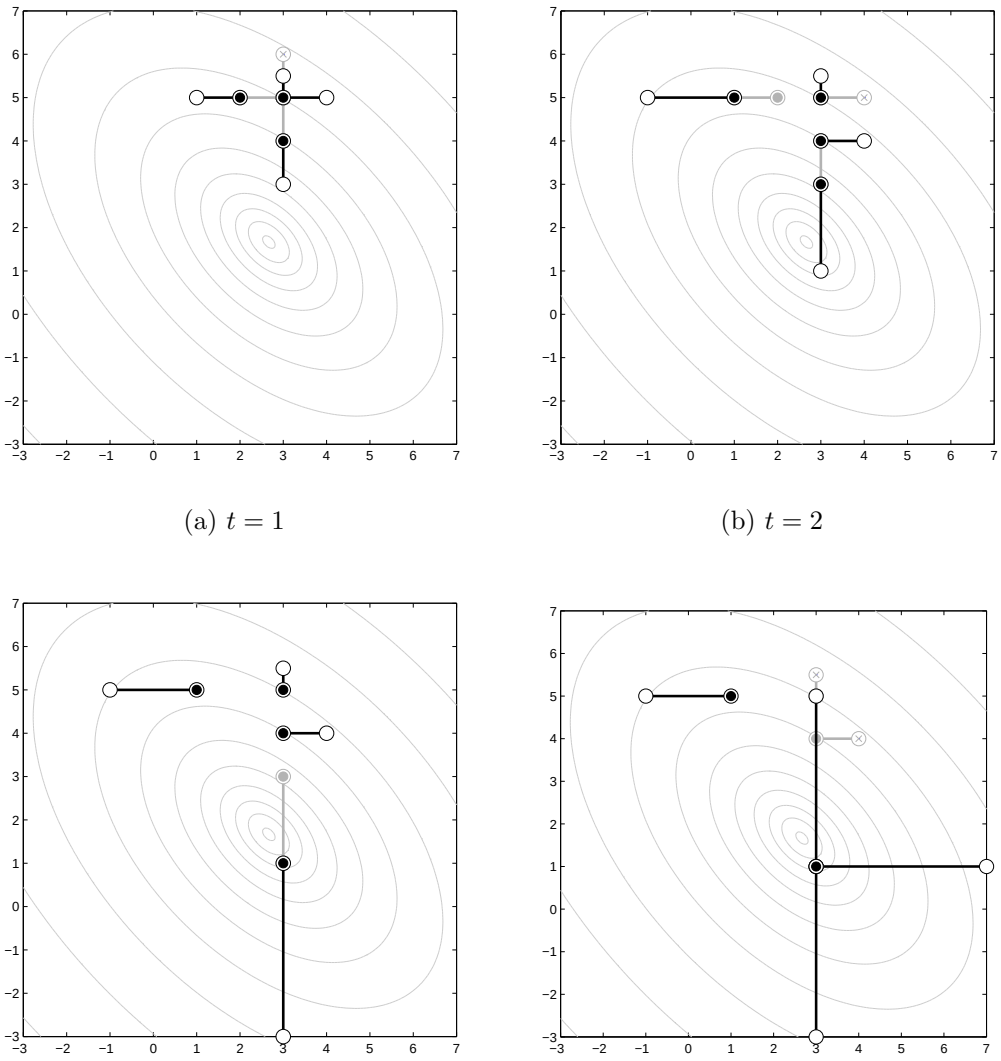


Figure 3: Timelines illustrating the first eight time steps of APPS applied to (2).

- Figures 4–5 illustrate the first eight time steps of APPS applied to (2). The illustrations are similar to those for PPS given in Figure 1. However, unlike PPS, up to p different best points may have been used to construct the trial points being evaluated at any time step; see, e.g., Figure 4(b). Also, up to p different values of the step-length control parameter may have been used to construct the trial points being evaluated at any time step; see, e.g., Figure 4(c).



(a) $t = 1$ (b) $t = 2$

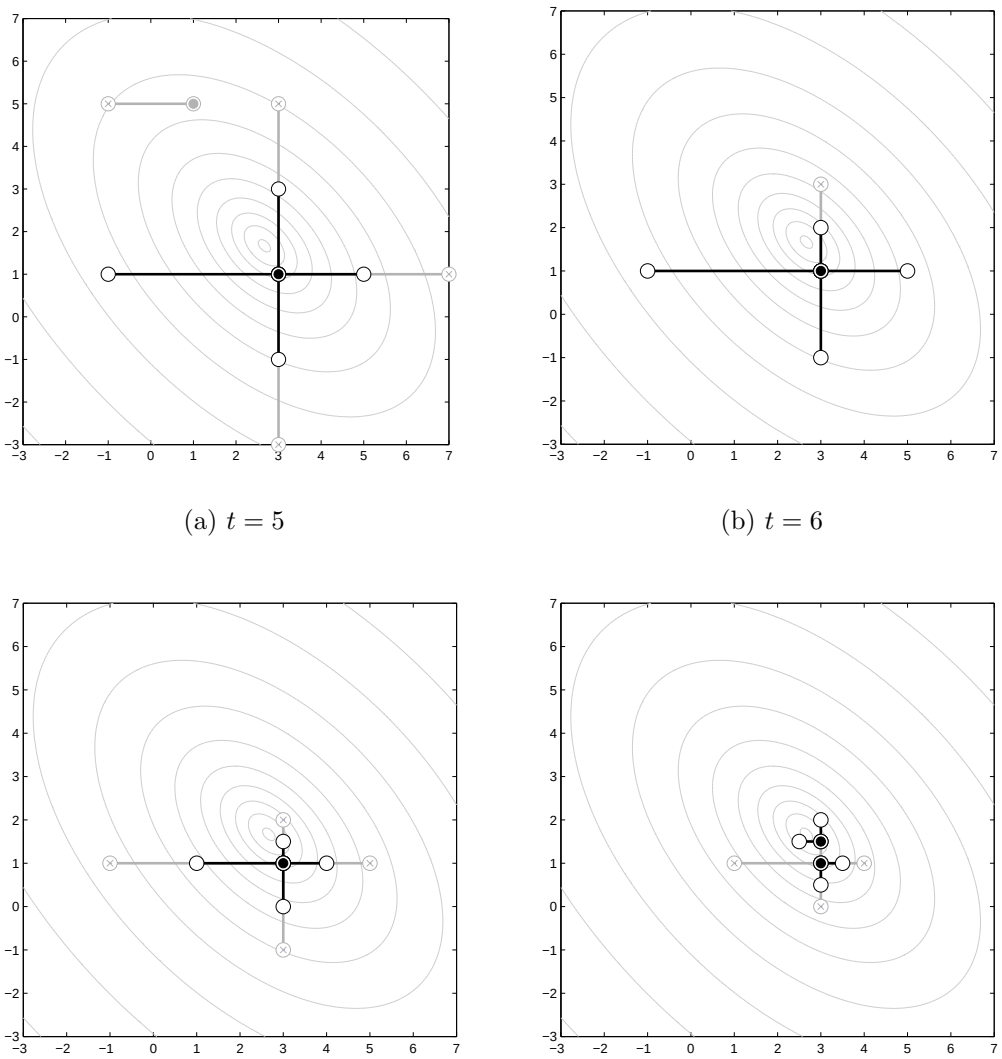
(c) $t = 3$ (d) $t = 4$

Figure 4: Asynchronous parallel pattern search applied to (2)

To give a sense of how to interpret these various representations, we highlight a few situations.

4.2.1 An internal success

At time step $t = 0$, process 4 starts a function evaluation at the trial point $x_4^0 + \Delta_4^0 d_4 = (3.0, 5.0)^T + (1.0) \cdot (0, -1)^T = (3.0, 4.0)^T$. The function evaluation at this trial point finishes at time step $t = 1$ with a function value of 13.0. This is an internal success,



(a) $t = 5$ (b) $t = 6$

(c) $t = 7$ (d) $t = 8$

Figure 5: Asynchronous parallel pattern search applied to (2)

i.e., time step $t = 1 \in \mathcal{I}_4$. Since the generating functions map from the set of successful time steps, in Table 1 we show their values whenever a time step is a success. In this case, $\omega_4(1) = 4$, $\tau_4(1) = 0$, and $\nu_4(1) = 1$. Since the time step is an internal success, we also specify the expansion parameter. In this case, $\lambda_4^1 = 1$. The fact that the time step is an internal success means that we update both the best point and the step-length control parameter. In this case, $x_4^1 = (3.0, 4.0)^T$ and $\Delta_4^1 = \lambda_4^1 \Delta_4^0 = 1 \cdot (1.0) = 1.0$. Finally, process 4 ends the time step by constructing

the new trial point $x_4^1 + \Delta_4^1 d_4 = (3.0, 4.0)^T + (1.0) \cdot (0, -1)^T = (3.0, 3.0)^T$.

In Figure 4(a) we show the new best point and the new trial point for process 4, as well as the activities associated with the other three processes. The information from the previous time step (equivalent to the situation illustrated in Figure 1(a)) is presented in gray. Note that by time step $t = 1$ the progress of the search conducted by APPS already differs markedly from the progress of the search conducted by PPS. In particular, note that for APPS there are now three different best points and two different values of the step-length control parameter in use across the four processes.

Turning to Figure 3 we get a clear picture of how the message traffic influences decisions local to each process. In this case, the internal success on process 4 at time step $t = 1$ is indicated by a black circle on the timeline for process 4. Further, we can see that the messages regarding this point are received by processes 1, 2, and 3 at time steps $t = 2$, $t = 4$, and $t = 3$, respectively. The timelines for the remaining three processes illustrate that when the messages sent by process 4 announcing its internal success at time step $t = 1$ arrive on processes 1 and 3 they trigger external successes on those two processes. However, when it finally arrives, the same information has no direct effect on process 2.

4.2.2 An external success

At time step $t = 3$, process 2 receives messages from both process 3 and process 4. We can see this on the timeline in Figure 3 as well as in Table 1. The best of the messages is from process 4 with a function value of 1.0, so the message from process 3 is discarded. Since the function value of 1.0 from process 4 improves upon $f(x_2^2) = 17.0$, we have an external success on process 2 (i.e., time step $t = 3 \in \mathcal{E}_2$). In Figure 3, this is denoted by a square box. Once again we have a success so, as shown in Table 1, $\omega_2(3) = 4$, $\tau_2(3) = 2$, and $\nu_2(3) = 3$ (i.e., the new best point was constructed by process 4 at time step $t = 2$ and its function evaluation finished at time step $t = 3$).

We update both the best point and the step-length control parameter using information contained in the message from process 4. So $x_2^3 = (3.0, 1.0)^T$, which is equivalent to $x_4^2 + \Delta_4^2 d_4$ or, consistent with the definition given in (3), $x_{\omega_2(3)}^{\tau_2(3)} + \Delta_{\omega_2(3)}^{\tau_2(3)} \, d_{\omega_2(3)}$. A perusal of the complete accounting given in Table 1 verifies this equivalence. Also, $\Delta_2^3 = 4.0$, which is equivalent to $\lambda_4^3 \Delta_4^2 = 2 \cdot (2.0)$ or, consistent with the definition given in (4), $\lambda_{\omega_2(3)}^{\nu_2(3)} \Delta_{\omega_2(3)}^{\tau_2(3)}$. This example illustrates how the generating functions, crafted solely for the analysis, allow us to track an update that arrives in a message from another process back both to the originating process and to the time steps at which critical values were established on the originating process.

Finally, observe that even though the values of x_2^3 and Δ_2^3 have been updated, APPS does not construct a new trial point and begin its evaluation at this time step since process 2 is still in the midst of a function evaluation. We can see this, for example, in Figure 3, where on the timeline for process 2 there is no vertical bar

at time step $t = 3$. We can also see this in Figure 4, where process 2 continues its evaluation of the trial point $(3.0, 5.5)^T$, which it constructed at time step $t = 1$. A new function evaluation will not commence on process 2 until the current evaluation finishes; in this case, at time step $t = 4$. These actions are fully documented in Table 1.

4.2.3 A contraction

At time step $t = 7$ on process 1, the function evaluation that started at time step $t = 5$ finishes. The function value of 9.0 at the trial point $(5.0, 1.0)^T$ does not improve upon $f(x_1^6) = 1.0$, so this time step is not an internal success. Furthermore, no external success has occurred since the evaluation of $(5.0, 1.0)^T$ was started at time step $t = 5$. Therefore, this time step is a contraction on process 1, i.e., time step $t = 7 \in \mathcal{C}_1$.

We update the step-length control parameter following our definition in (4): $\Delta_1^7 = \theta_1^7 \Delta_1^6 = \frac{1}{2} \cdot (2.0) = 1.0$. (As noted earlier, we do not specify θ_i^t in Table 1 since, for this example, we always use $\theta_i^t = \frac{1}{2}$.) Observe that we do not need, and thus do not define, values for the generating functions since the decisions regarding a contraction are local to the process.

The new trial point is then constructed using this reduced value of the step-length control parameter: $x_1^7 + \Delta_1^7 \, d_1 = (3.0, 1.0)^T + (1.0) \cdot (1, 0)^T = (4.0, 1.0)^T$. A new function evaluation is initiated at this trial point, as seen in Figure 5(c).

4.2.4 Revisiting the partitioning of the time steps

Now that we have some familiarity with the different representations of APPS when applied to (2), as well as with what constitutes "interesting events," we can revisit the definitions from §4.1.1 in terms of our example. Let $\hat{\mathcal{T}} = \{1, \ldots, 8\}$. We then summarize the categorization of each time step in $\hat{\mathcal{T}}$ on every process in $\mathcal{P}$:

$$
\begin{array}{lcllcl}
\mathcal{I}_1 \cap \hat{\mathcal{T}} & = & \emptyset & \mathcal{I}_2 \cap \hat{\mathcal{T}} & = & \{8\} \\
\mathcal{E}_1 \cap \hat{\mathcal{T}} & = & \{2,3,4\} & \mathcal{E}_2 \cap \hat{\mathcal{T}} & = & \{2,3\} \\
\mathcal{S}_1 \cap \hat{\mathcal{T}} & = & \{2,3,4\} & \mathcal{S}_2 \cap \hat{\mathcal{T}} & = & \{2,3,8\} \\
\mathcal{C}_1 \cap \hat{\mathcal{T}} & = & \{5,7,8\} & \mathcal{C}_2 \cap \hat{\mathcal{T}} & = & \{1,5,6,7\} \\
\mathcal{U}_1 \cap \hat{\mathcal{T}} & = & \{1,5,6,7,8\} & \mathcal{U}_2 \cap \hat{\mathcal{T}} & = & \{1,4,5,6,7\} \\
\mathcal{T}_1 \cap \hat{\mathcal{T}} & = & \{2,3,4,5,7,8\} & \mathcal{T}_2 \cap \hat{\mathcal{T}} & = & \{1,2,3,5,6,7,8\} \\
\mathcal{I}_3 \cap \hat{\mathcal{T}} & = & \{1,2\} & \mathcal{I}_4 \cap \hat{\mathcal{T}} & = & \{1,2,3\} \\
\mathcal{E}_3 \cap \hat{\mathcal{T}} & = & \{3,5,8\} & \mathcal{E}_4 \cap \hat{\mathcal{T}} & = & \emptyset \\
\mathcal{S}_3 \cap \hat{\mathcal{T}} & = & \{1,2,3,5,8\} & \mathcal{S}_4 \cap \hat{\mathcal{T}} & = & \{1,2,3\} \\
\mathcal{C}_3 \cap \hat{\mathcal{T}} & = & \{7\} & \mathcal{C}_4 \cap \hat{\mathcal{T}} & = & \{5,7,8\} \\
\mathcal{U}_3 \cap \hat{\mathcal{T}} & = & \{4,6,7\} & \mathcal{U}_4 \cap \hat{\mathcal{T}} & = & \{4,5,6,7,8\} \\
\mathcal{T}_3 \cap \hat{\mathcal{T}} & = & \{1,2,3,5,7,8\} & \mathcal{T}_4 \cap \hat{\mathcal{T}} & = & \{1,2,3,5,7,8\}
\end{array}
\tag{5}
$$

Note that these sets are easily constructed by looking at the timelines in Figure 3. Alternatively, they can be constructed by reviewing the entries in Table 1.

5 Contrasting APPS and PPS

As we can see from our example, APPS has no idle time between function evaluations. Each process constructs a new trial point and initiates a new evaluation of the objective function at that trial point as soon as it finishes its previous function evaluation and whatever updates are necessary. This difference between APPS and PPS is apparent at the end of time step $t = 1$. As Figure 2 makes clear, PPS requires processes 2–4 to wait (i.e., idle) until the function evaluation on process 1 finishes. Thus the four new function evaluations initiated at iteration $k = 1$, shown in Figure 1(b), do not actually begin until time step $t = 2$.

In contrast, APPS immediately begins new function evaluations on processes 2–4 at time step $t = 1$, *despite* the fact that process 1 has not yet finished its function evaluation. Furthermore, there is no synchronization of information across processes. As a consequence, process 2 has a contraction, and processes 3 and 4 each have internal successes on which to base their next trial point. This is illustrated in Figure 4(a).

By time step $t = 2$ we can already see the value of eliminating the idle time. Under APPS, process 3 already has discovered the function value $f(x_4^2) = 5.0$ at the point $x_4^2 = (3.0, 3.0)^T$ (see Table 1). The search using PPS does not discover this point until time step $t = 3$ (see Figure 1 and Figure 2). Under APPS, process 4 can immediately make use of this discovery to construct a new trial point at $(3.0, 1.0)^T$ and initiate a function evaluation at this trial point. Under PPS, process 4 must wait until time step $t = 5$ to construct the trial point $(3.0, 1.0)^T$ and initiate a function evaluation at this trial point.

The cumulative effect of eliminating idle time becomes even more apparent by the end of eight time steps. At time step $t = 8$, the best known function value for APPS is $f(x_3^8) = 0.5$. For PPS, the best known function value is $f(x^4) = 1.0$. Of equal importance is the fact that at time step $t = 8$ the step-length control parameter is 0.5 on all four APPS processes, though they have arrived at this common value by different means. In contrast, the synchronization ensures that all four PPS processes share the value of 4.0 for the step-length control parameter. The consequence of this significant difference in the scaling of the steps is that for APPS, the search is on a scale that is much more appropriate, given the current neighborhood of the search. It takes only a modest extrapolation to future time steps to see that APPS will realize further decreases in the value of the objective more quickly than PPS. Thus we can see that, when the number of time steps required to finish a function evaluation varies, the asynchronous nature of APPS makes it possible to get to lower objective function values more quickly than PPS.

Before we close this discussion we note that although in theory APPS has no idle time, the termination criterion introduces some idle time once we have reason to believe we are reasonably close to a solution. In [3] we discuss the stopping criterion for APPS in detail. The difference between APPS and PPS in practice, though, is that for APPS this idle time only occurs when some subset of the processes believe

that they have identified a possible solution. This subset of processes must wait until either a sufficient number of processes agree that a solution has been identified *or* some other process produces a new best point. This is a situation we expect to see only infrequently and then only toward the end of the search. In contrast, in PPS idle time can occur at each iteration. The results reported in [3] suggest that our expectation appears to be realized in practice.

6 Tracking iterates across processes

The real challenge to analyzing APPS lies in determining the origin of a given iterate. Addressing this question precipitated the seemingly complicated notation we have introduced.

To appreciate the timing effects our analysis must accommodate, we revisit our example from §4 and consider what happens on process 3 (the third timeline from the top in Figure 3). If we look at Table 1, it is clear that up through time step $t = 8$ the subsequence of (distinct) best points on process 3 is:

$$\{x_3^0, x_3^1, x_3^2, x_3^3, x_3^5, x_3^8\} = \left\{ \begin{pmatrix} 3.0 \\ 5.0 \end{pmatrix}, \begin{pmatrix} 2.0 \\ 5.0 \end{pmatrix}, \begin{pmatrix} 1.0 \\ 5.0 \end{pmatrix}, \begin{pmatrix} 3.0 \\ 4.0 \end{pmatrix}, \begin{pmatrix} 3.0 \\ 1.0 \end{pmatrix}, \begin{pmatrix} 3.0 \\ 1.5 \end{pmatrix} \right\}. \quad (6)$$

These correspond exactly to the best points at the time steps $\{0\} \cup \{\mathcal{S}_3 \cap \hat{\mathcal{T}}\}$ (recall the partitioning of the time steps given in (5)). But for the purposes of the analysis, the sequence in (6) is irrelevant. What we really need to know is what sequence of moves brought the search to the current best point $x_3^8 = (3.0, 1.5)^T$. To obtain this information we need to be able to track what happened on *other* processes. This is where the generating functions play a crucial role. Using our generating functions, we can recover the sequence

$$\{x_4^0, x_4^1, x_4^2, x_2^3 = x_4^3, x_3^8 = x_2^8\} = \left\{ \begin{pmatrix} 3.0 \\ 5.0 \end{pmatrix}, \begin{pmatrix} 3.0 \\ 4.0 \end{pmatrix}, \begin{pmatrix} 3.0 \\ 3.0 \end{pmatrix}, \begin{pmatrix} 3.0 \\ 1.0 \end{pmatrix}, \begin{pmatrix} 3.0 \\ 1.5 \end{pmatrix} \right\}, \quad (7)$$

which explains how we actually arrived at x_3^8. It is critical to note that the sequence in (7) could not be recovered simply by looking at the sequence in (6).

In summary, to analyze APPS, we must be able to identify which process produced the update. Further, the update rules for the iterate, given in (3), mean that if $t \in \mathcal{S}_i$ we need to be able to identify the time step at which the function evaluation started so that we can recover both the iterate and the value of the step-length control parameter used to construct the successful point. In addition, the update rules for the step-length control parameter, given in (4), mean that if $t \in \mathcal{S}_i$, we also need to be able to identify the time step at which the function evaluation finished so that we can recover the value of λ_i^t used in the update.

The reason we need this information is that we use it, in part, to verify the algebraic structure of the iterates. In particular, in [4], we derive a result equivalent to Theorem 3.2 in [7]. Verifying the algebraic structure of the iterates assures us that

we have a step-length control mechanism which prevents premature convergence. We obtain this assurance even though the APPS processes act as semi-autonomous agents that can either contract or expand Δ_i^t independently, subject to only a mild modification of the basic rules first outlined in [7].

To see this, we once again return to our example from §4. If we consider x_3^8, we see that our ability to track the sequence of iterates across processes allows us to write x_3^8 as the initial guess, x^0, plus a linear combination of the search directions in $\mathcal{D}$, as follows:

$$\begin{aligned} x_3^8 &= x_2^8 \\ &= x_2^7 + \Delta_2^7 d_2 \\ &= x_4^3 + \Delta_2^7 d_2 \\ &= \left(x_4^2 + \Delta_4^2 d_4\right) + \Delta_2^7 d_2 \\ &= \left(x_4^1 + \Delta_4^1 d_4\right) + \Delta_4^2 d_4 + \Delta_2^7 d_2 \\ &= \left(x_4^0 + \Delta_4^0 d_4\right) + \Delta_4^1 d_4 + \Delta_4^2 d_4 + \Delta_2^7 d_2 \\ &= x^0 + \left(\Delta_2^7\right) d_2 + \left(\Delta_4^0 + \Delta_4^1 + \Delta_4^2\right) d_4 \end{aligned}$$

This claim for x_3^8 in fact holds for any x_i^t produced by APPS, a fact which we prove in Lemma 4.1 in [4]. More precisely, we say that

$$x_i^t = x^0 + \sum_{j \in \mathcal{P}} \delta_j(i,t)\, d_j \quad \text{with} \quad \delta_j(i,t) = \sum_{\hat{t} \in \hat{\mathcal{I}}_j(i,t)} \Delta_j^{\tau_j(\hat{t})},$$

where $\delta_j(i,t) = 0$ if $\hat{\mathcal{I}}_j(i,t) = \emptyset$ and $\hat{\mathcal{I}}_j(i,t)$ is constructed recursively using the formula

$$\hat{\mathcal{I}}_j(i,t) = \begin{cases} \emptyset & \text{if } t = 0, \\ \hat{\mathcal{I}}_j(\omega_i(t), \tau_i(t)) \cup \{\nu_i(t)\} & \text{if } t \in \mathcal{S}_i \text{ and } j = \omega_i(t), \\ \hat{\mathcal{I}}_j(\omega_i(t), \tau_i(t)) & \text{if } t \in \mathcal{S}_i \text{ and } j \neq \omega_i(t), \text{ and} \\ \hat{\mathcal{I}}_j(i, t-1) & \text{otherwise.} \end{cases}$$

Returning to x_3^8, as an example we have the following:

$$\hat{\mathcal{I}}_1(3,8) = \emptyset, \quad \hat{\mathcal{I}}_2(3,8) = \{8\}, \quad \hat{\mathcal{I}}_3(3,8) = \emptyset, \quad \text{and} \quad \hat{\mathcal{I}}_4(3,8) = \{1,2,3\}.$$

In other words, the sets $\hat{\mathcal{I}}_j(3,8)$, $j = \{1,2,3,4\}$, indicate which internal successes, on which processes, contributed to the discovery of the best point on process 3 at time step $t = 8$. Using this information, we can determine the multiplier for each search direction

$$\begin{aligned} &\delta_1(3,8) = 0, \qquad \delta_2(3,8) = \Delta_2^{\tau_2(8)} = \Delta_2^7, \\ &\delta_3(3,8) = 0, \qquad \delta_4(3,8) = \left(\Delta_4^{\tau_4(1)} + \Delta_4^{\tau_4(2)} + \Delta_4^{\tau_4(3)}\right) = \left(\Delta_4^0 + \Delta_4^1 + \Delta_4^2\right). \end{aligned}$$

Thus we have a rigorous way to obtain conclusions equivalent to

$$x_3^8 = x^0 + \left(\Delta_2^7\right) d_2 + \left(\Delta_4^0 + \Delta_4^1 + \Delta_4^2\right) d_4$$

for any choice of x_i^t.

Next, we examine the non-zero values of $\delta_j(i,t)$ more carefully. For $\delta_2(3,8)$, we have

$$\begin{aligned}
\delta_2(3,8) &= \Delta_2^7 \\
&= 2^{-1} \cdot \Delta_2^6 \\
&\vdots \\
&= 2^{-1} \cdot 2^{-1} \cdot 2^{-1} \cdot 2^1 \cdot 2^1 \cdot 2^0 \cdot \Delta_4^0 \\
&= 2^{-1}\Delta^0
\end{aligned}$$

Similarly, for $\delta_4(3,8)$, we get

$$\begin{aligned}
\delta_4(3,8) &= \Delta_4^0 + \left(2^0 \cdot \Delta_4^0\right) + \left(2^1 \cdot 2^0 \cdot \Delta_4^0\right) \\
&= \left(2^0 + 2^0 + 2^1\right) \Delta^0.
\end{aligned}$$

By factoring out $2^{-1}\Delta^0$, the shortest successful search step used to reach the current iterate, in both cases we are left with an integer:

$$\delta_2(3,8) = \underbrace{2^0}_{\text{integer}} \cdot (2^{-1}\Delta^0) \qquad \delta_4(3,8) = \underbrace{\left(2^1 + 2^1 + 2^2\right)}_{\text{integer}} \cdot (2^{-1}\Delta^0)\,.$$

If we were to take the lattice of all possible integer combinations of the search directions in $\mathcal{D}$, scale it by $2^{-1}\Delta^0$, and translate it by x^0, we can convince ourselves that the point x_3^8 actually lies on that lattice. So, for x_3^8 we have

$$x_3^8 = x^0 + 1 \cdot \left(2^{-1}\Delta^0\right) d_2 + 8 \cdot \left(2^{-1}\Delta^0\right) d_4.$$

We now are back on familiar territory for pattern search analysis. Theorem 5.2 in [4] guarantees that every successful point produced by the search up until time step t will lie on the lattice formed by the search directions in $\mathcal{D}$, translated by x^0, and scaled by $2^\Gamma\Delta^0$, where $2^\Gamma\Delta^0$ is a lower bound on the shortest successful search step taken up until time step t. What is important to make this result hold is that we can extract a rational set of search directions from $\mathcal{D}$ (this may require a change of basis) and that we choose the θ's and λ's so that they are integer powers of a positive rational constant, with 2 being the usual choice and the one we used for our example.

We hasten to repeat that all this bookkeeping is *not* necessary to implement APPS, only to analyze it. One of the features of pattern search methods is that they are, in some sense, "memory-less". All any process $i \in \mathcal{P}$ really needs to know to proceed with the search is x_i^t, Δ_i^t, d_i, and $f(x_i^t)$ since the next trial point $x_i^t + \Delta_i^t\, d_i$ then can be constructed and $f(x_i^t)$ gives the target value that $f(x_i^t + \Delta_i^t\, d_i)$ must beat for $x_i^t + \Delta_i^t\, d_i$ to become the new best point.

7 Conclusions

We have concentrated here on elucidating the concepts and notation required to track the iterates produced by APPS across processes so that we are able to show that there is at least one subsequence of iterates that is common to all processes. The global convergence argument given in [4] has four basic parts. First, we verify the algebraic structure of the iterates. Second, we show that the subset of time steps at which changes occur either to the best point or to the step-length control parameter is infinite. Third, we prove that a subsequence of the step-length control parameters goes to zero. Finally, we ascertain that there exists a common accumulation point for all processes and that this accumulation point has a zero gradient.

To handle the asynchronous nature of the search, we rely on the assumption that the number of time steps required either for a function evaluation to finish or for a message to arrive from another process are bounded above by finite constants. This allows us to tie together the relative timing of events as the time steps go to infinity and thus guarantee the existence of an accumulation point that is common to all the processes.

Once we identify an "interesting" subsequence of iterates (with their associated values for the step-length control parameter) that is common to all the processes, we use by now standard arguments to finish the analysis.

Acknowledgments

We thank the organizers of the International School of Mathematics "G. Stampacchia" 33rd Workshop: High Performance Algorithms and Software for Nonlinear Optimization for the opportunity to present a summary of our work on asynchronous parallel pattern search.

References

[1] M. Avriel (1976), *Nonlinear Programming: Analysis and Methods*, Prentice-Hall, Englewood Cliffs, New Jersey.

[2] D. Bertsekas and J. Tsitsiklis (1989), *Parallel and Distributed Computation: Numerical Methods*, Prentice-Hall, Englewood Cliffs, New Jersey.

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

[4] T. G. Kolda and V. J. Torczon (2001), "On the convergence of asynchronous parallel pattern search," Tech. Rep. SAND2001–8696, Sandia National Laboratories, Livermore, California.

[5] D. Levine (1995), "Users guide to the PGAPack parallel genetic algorithm library," Tech. Rep. ANL–95/18, Argonne National Laboratory, Argonne, Illinois.

[6] V. Torczon (1992), "PDS: Direct search methods for unconstrained optimization on either sequential or parallel machines," Tech. Rep. 92–09, Rice University, Department of Computational and Applied Mathematics, Houston, Texas.

[7] V. Torczon (1997), "On the convergence of pattern search algorithms," *SIAM J. Optim.*, 7, pp. 1–25.