PARALLEL ASYNCHRONOUS ALGORITHMS FOR DISCRETE DATA

TECHNICAL REPORT NO. CENG 89-15

BY

AYDIN URESIN AND MICHEL DUBOIS

JUNE 1989

This work was performed under NSF Grant DCCR-8709997 and a grant from AT&T

Parallel Asynchronous Algorithms for Discrete Data

Aydın Üresin Michel Dubois

Abstract

Many problems in the area of symbolic computing can be solved by iterative algorithms. Implementations of these algorithms on multiprocessors can be synchronized or asynchronous. Asynchronous implementations are potentially more efficient, because synchronization is a major source of performance degradation in most multiprocessor systems.

In this paper, sufficient conditions for the convergence of asynchronous iterations to desired solutions are given. The main sufficient condition is shown to be also necessary for the case of finite data domains. The results are applied to prove the convergence of three asynchronous algorithms for the all-pairs shortest path problem, the consistent labeling problem, and a neural net model.

1 Introduction

The class of iterative algorithms is significant and has many applications in the areas of numerical as well as symbolic computing. Most algorithms in this class are characterized by an operator $\mathbf{F}$ successively applied to some data $\mathbf{x}$, starting with the initial value of the data $\mathbf{x}(0)$, and are represented by the following iteration formula :

$$\mathbf{x}(k+1) = \mathbf{F}(\mathbf{x}(k)), \qquad k = 0, 1, 2, \ldots \tag{1}$$

where $\mathbf{x}(k)$ is the value of $\mathbf{x}$ at the k-th iteration. If $\mathbf{x}$ is a vector, the computation of its components at each iteration can be performed in parallel. Shared-memory multiprocessors [7] are particularly well suited for the implementation of the iteration of the form (1).

The vector $\mathbf{x}$ resides in the shared memory, and the components of $\mathbf{F}$ are computed by different processors. Strict application of Equation (1) results in a synchronized iterative algorithm where each processor has to wait for all the other components to be updated before starting the next iteration. Asynchronous iterative algorithms (asynchronous iterations), on the other hand, allow processors to continue computations after an update, without requiring them to wait for the other processors. Since synchronization is a major source of performance degradation, asynchronous algorithms are potentially more efficient than their synchronized counterparts [8]. In the general model of asynchronous iterations [2] [5], the processors do not even have to be assigned to the update of a fixed set of components throughout all computations, and furthermore, more than one processor may be working on the same component at the same time. Therefore, a processor may not be using the most recent value of the component that it is updating.

Of course, the major concern regarding any algorithm is its correctness, and in an asynchronous iterative algorithm issues of correctness and convergence to the desired result coincide. In this paper, we identify the general tools for the study of the convergence of asynchronous iterations, and we demonsrate that these tools can be applied quickly and easily to some practical problems. We first define "asynchronous contraction"; an equivalent version of this definition was previously given in [18] and [19]. Asynchronous contraction is an extremely simple concept; however, under the least restricted form of asynchrony, it is proven to be not only sufficient for the convergence of asynchronous iterations on any data (Theorem 1), but also necessary for convergence in the case of finite data domains (Theorem 2). In other words, as in [2], we make no assumptions on the computational model, except for the weakest fairness assumptions, and, for this general model, asynchronous contraction defines the largest possible class of asynchronous iterative algorithms that are correct, at least for finite data domains. Depending on the application, asynchronous contraction may not always be easily verifiable in practical cases. Hence, we derive simpler tools for the analysis of convergence in Section 4. Propositions 3, 4 and 5 are immediate implications of asynchronous contraction. Proposition 6 is for a more restricted form of asynchrony. It is shown that none of the sets of conditions given by these propositions is necessary for convergence.

The results are applicable to a wide variety of iterative algorithms to solve problems such as dynamic programming, shortest path problems, network flow problems, unsupervised pattern clustering, consistent labeling, and artificial neural networks. Among these, the all-pairs shortest path problem, the consistent labeling, and an artificial neural network model are given as examples in Section 5. The all-pairs shortest path problem is a familiar and fundamental application [1]. Although simple, it is widely used. Consistent labeling is a generic problem that has a broad range of applications in the area of artificial intelligence and computer vision [6].

In the general setting of the consistent labeling problem, a set of objects to be labeled and the set of labels compatible with each individual object are given. Label sets contain a priori information about unary constraints; i.e., which labels can be assigned to which objects. There are also binary constraints, namely, each object pair is compatible with a certain set of label pairs. In general, we may be given N-ary constraints. The problem is to find an assignment of labels to objects that satisfies all of the constraints. *Relaxation* *labeling* is an iterative process to reduce the search space of the problem. At each iteration of this process, for every object and for every constraint relevant to this object, we discard each label from the current label set of the object, if we cannot find an assignment of labels to other objects that satisfies the constraint together with unary constraints. Note that after convergence, label sets may still contain more than one label, necessitating a further search, but in most cases the procedure significantly reduces the search space. Many common problems in artificial intelligence, including scene labeling, edge interpretation, and graph isomorphism can be shown to reduce to the consistent labeling problem.

In the scene labeling problem, we are given a picture of a scene, such as an office, that has objects which need to be identified, or labeled. The label set is also given which may include "chair," "table," "desk," or "telephone." The relative positions of the objects in the picture and a priori world information define the constraints. For example, if object i is on top of object j in the picture, then object i cannot be labeled with "table" if object j is labeled with "telephone," because the world model dictates that a table cannot sit on top of a telephone.

In the graph isomorphism problem the goal is to find, if any, a one-to-one correspondance between the vertices of two graphs, such that the matched vertices have exactly the same connections. In this case, the objects are the vertices of one graph, and the labels are the vertices of the other one. The constraints are derived from the connections between the vertices. Other applications that can be reduced to the consistent labeling problem can be found in [6].

2 Preliminaries

The following notations are adopted throughout the paper: $S = S_1 \times S_2 \times \cdots \times S_n$ is the Cartesian product of n sets from which the shared data take values; $\mathbf{F} : S \to S$ is a mapping function; $\mathbf{N}$ is the set of nonnegative integers; $\mathbf{N}^+$ is the set of positive integers; and $\mathbf{I} = \{1, 2, \ldots, n\}$ is the index set the elements of which correspond to the components of the shared data.

The definition given below formulates the model of asynchronous iterative algorithms that we will assume throughout the paper. We define it in terms of a *schedule* ($\mathcal{S}$), along with an iteration operator and initial data. Informally, a schedule is a timing of accesses to global data. For example, in a shared-memory system these accesses include both the fetching and the storing of iterate components.

Definition 1 An *asynchronous iteration* with respect to the schedule $\mathcal{S} = (\{\alpha(k)\}, \{\beta(k)\})$, corresponding to $\mathbf{F}$ and starting with $\mathbf{x}(0) \in S$, is a sequence $\{\mathbf{x}(k)\}$ such that

$$x_i(k) = \begin{cases} x_i(k-1) & \text{if} \quad i \notin \alpha(k), \\ F_i(x_1(\beta_1(k)), x_2(\beta_2(k)), \ldots, x_n(\beta_n(k))) & \text{if} \quad i \in \alpha(k), \end{cases}$$

for all $k \in \mathbf{N}^+$ and $i \in \mathbf{I}$, and is denoted by $(\mathbf{F}, \mathbf{x}(0), \mathcal{S})$, where $\{\alpha(k)\}$ is a sequence of subsets of $\mathbf{I}$ and $\{\beta(k)\}$ is a sequence of elements of $\mathbf{N}^n$.

$\alpha(k)$ is the *update set at* k, and we say that the i-th component is *updated* at k if $i \in \alpha(k)$. We assume that all of the components are updated at 0; i.e., $\alpha(0) = \mathbf{I}$. The vector $\mathbf{u}(k)$, with components $u_j(k) = x_j(\beta_j(k))$ is called the *input of the k-th update*, and $F_i(\mathbf{u}(k))$ is said to be *generated at k for the i-th component.*

A schedule ($\mathcal{S} = (\{\alpha(k), \beta(k)\})$) is assumed to satisfy the following conditions:

[A1] $\beta_i(k) < k$ for all $k \in \mathbf{N}^+$ and $i \in \mathbf{I}$; i.e., at any time instance, the values of the shared data components can depend only on their past values.

[A2] Each $i \in \mathbf{I}$ occurs in an infinite number of $\alpha(k)$'s; i.e., each component is updated infinitely often (non-starvation condition).

[A3] For each $l \in \mathbf{N}$ and each $i \in \mathbf{I}$, there can only exist a finite number of k's (possibly zero) such that $\beta_i(k) = l$; i.e., after a coordinate value is generated, this value can be used as inputs of only a finite number of updates. $\square$

The above definition describes the most general form of asynchronous iterative algorithms. The level of asynchrony defined by [A1]-[A2] allows dynamic allocation of components to processors, as well as static allocation. In static allocation, the processors are assigned to fixed components throughout the computation; if, furthermore, each component is assigned to only one processor, then the processor which is assigned to compute the i-th component is always aware of the most recent value of x_i. In terms of the above definition, this corresponds to the case where $\beta_i(k) = k-1$ when the i-th component is updated at k. Since the definition does not impose this restriction, processors may be freely assigned to different components in the course of a computation; i.e., dynamic allocation is allowed.

As an example to the above definition, the schedule given below corresponds to the synchronized implementation of the point-Jacobi iteration [2], where all the components of an iterate are computed in parallel, at the same time:

$$\alpha(k) = \{1, 2, \ldots, n\}$$

$$\beta_i(k) = k - 1,$$

for all $k \in \mathbf{N}^+$ and $i \in \mathbf{I}$. Similarly, the schedule corresponding to the Gauss-Seidel iteration, where the components are updated sequentially and where each update uses the most recent iterate values, is shown in the following:

$$\alpha(k) = \{1 + [(k-1) \bmod n]\},$$

$$\beta_i(k) = k - 1,$$

for all $k \in \mathbf{N}^+$ and $i \in \mathbf{I}$. It can easily be shown that the Jacobi and Gauss-Seidel iterations satisfy conditions [A1]-[A3].

These assumptions are very natural and weak conditions. Practically, they impose no restriction on the environment. It is only required that each component of the global data should be updated sufficiently many times [A2], and that outdated iterate values should not be used forever, although they can be used for a while [A3]. It is not possible to guarantee convergence without [A2], and it is assumed explicitly or implicitly by all the other computational models in the literature. [A1] is even more obvious; clearly it exists in all definitions of asynchronous iterations.

In an early paper on asynchronous iterations [5], [A3] is in a more restricted form: the computation time of a component; i.e., the time between the fetch of the data used in a component update and the actual update of the component in the global store is bounded. Under this model, necessary and sufficient conditions for the convergence of asynchronous iterations corresponding to linear and real functions were given. Baudet [2] relaxed the condition imposed on the computation time by assuming that it was only finite and not necessarily bounded. In this respect, Definition 1 is very similar to the one in [2]. The only restriction of [2] is that the data domain is real and continuous. Under this model, Baudet gave a sufficient condition for the convergence of asynchronous iterations corresponding to the general class of real functions. In the model given in [10], the computation time of each iterate as well as the period in which all the components are updated are assumed to be bounded. It is further assumed that at least one component is always computed using the most recent value of that component; i.e., there exists one i such that $\beta_i(k) = k - 1$, for $i \in \alpha(k)$. This is naturally satisfied if at least one component is assigned to a fixed processor throughout the computation. Under these assumptions, asynchronous algorithms for a class of linear and real functions are shown to be convergent, without satisfying the convergence conditions given in [5].

The set of sufficient conditions for convergence given in [3] is more general than the aforementioned ones and is applicable to non-numerical data. The computational model, however, is not the most general one, because each component is updated using the most recent value of that component; i.e., $\beta_i(k) = k - 1$, for all $i \in \alpha(k)$. Some applications for the same model are presented in [4], [16] and [17]. Weaker conditions can be found in [15] for a model even more restrictive. In this model there are a number of processes and one global data item. One after another, in random order, the processes update the data item, but the computations of different components do not overlap. In other words, $\beta_i(k) = k - 1$ for $i \in \alpha(k)$, and also $\alpha(k)$'s are singletons. Even though this model is fairly restrictive for parallel processing, it has practical applications, one of which is given in [15]. In [14] asynchronous algorithms are given for boundary value problems, and [11] studies asynchronous iterations for some classes of stochastic or deterministic control problems.

Robert extended the classical convergence results of numerical analysis to cover discrete data [12]. He proved some sufficient conditions for the convergence of asynchronous iterations for discrete data, assuming the restricted model, where the computations do not overlap; i.e., $\beta_i(k) = k - 1$, for $i \in \alpha(k)$, and also $\alpha(k)$'s are singletons. These conditions use dependency information and seem to be strong, but they are easy to test in practical applications. They simply require that the transitive closure of the dependency matrix $\mathbf{B}(\mathbf{F})$ is null, where $B_{ij}(\mathbf{F})$ is 0 if the computation of the i-th component does not depend on the value of the j-th component and is 1 otherwise.

In later proofs, we will make use of a different, but equivalent, formulation of [A2] and [A3] which is stated by Proposition 1. Note that [A2] is equivalent to the existence of an increasing integer sequence $\{\varphi'(K)\}$ starting with 0, such that for all K, every component is updated between $\varphi'(K)$ and $\varphi'(K+1)$. On the other hand, [A3] implies that for all $\bar{l}$ there exists $\bar{k} \geq \bar{l}$, such that no k greater than $\bar{k}$ and no l less than $\bar{l}$ satisfy $\beta_i(k) = l$. Consequently, we can construct an increasing sequence of integers $\{\varphi''(K)\}$ starting with 0, such that

$$j \geq \varphi''(K) \;\Rightarrow\; \beta_i(j) \geq \tau_i''(K-1), \quad \forall K \in \mathbf{N}^+,\ \forall i \in \mathbf{I}.$$

Here $\tau_i''(K-1)$ is defined as the first update instance of the i-th component not earlier than $\varphi''(K-1)$. It is also evident that the existence of such $\{\varphi''(K)\}$ implies [A3], and Proposition 1 immediately follows.

Proposition 1 [A2] *and* [A3] *hold if and only if there exists an increasing sequence of integers* $\{\varphi(K)\}$, *satisfying :*

- $\varphi(0) = 0$,
- *each component, $i \in \mathbf{I}$, is updated at some j such that*

$$\varphi(K) \leq j < \varphi(K+1), \qquad \forall K \in \mathbf{N},$$

- $j \geq \varphi(K) \Rightarrow \beta_i(j) \geq \tau_i(K-1) \geq \varphi(K-1), \quad \forall K \in \mathbf{N}^+,\ \forall i \in \mathbf{I}$

  where $\tau_i(K-1)$ is as defined before; i.e., the first instance not earlier than $\varphi(K-1)$ at which the i-th component is updated. $\square$

Following the terminology in [12], we call the set $\Phi(K) = \{j | \varphi(K) \leq j < \varphi(K+1)\}$ the K-th *pseudo-cycle* of S , and the sequence $\{\Phi(K)\}$ is said to be the *pseudo-cycle sequence* associated with S. There are an infinite number of $\{\varphi(K)\}$'s and $\{\Phi(K)\}$'s satisfying the above conditions, given a schedule. In the rest of the paper $\{\varphi(K)\}$ and $\{\Phi(K)\}$ (when there is no ambiguity) will represent an arbitrary one of these, although we will not explicitly mention it every time.

A schedule satisfying [A2] and [A3] is called a *pseudo-periodic schedule*. As we shall show in Theorem 1, at the end of a pseudo-cycle $\Phi(K)$ (from $\varphi(K)$ to $\varphi(K+1)$), it is guaranteed that there is an "improvement" made towards the solution. Therefore, associating $\Phi(K)$'s to actual periods of time suggests an obvious way of estimating the total computation time.

Now we define a class of operators that will later be shown to converge to the unique solution for all pseudo-periodic schedules. In the next section, it is also proven that this is the largest such class for finite data domains. In other words, for data domains that contain only a finite number of elements, no operator outside this class is guaranteed to converge for all pseudo-periodic schedules.

Definition 2 An operator $\mathbf{F}$ is an *asynchronously contracting operator* (ACO) on a subset $D(0)$ of $S = S_1 \times S_2 \times \cdots \times S_n$ iff there exists a sequence of sets $\{D(K)\}$ such that

[C1] for all $K \in \mathbf{N}$, $D(K)$ is the Cartesian product of n sets; i.e.,

$$D(K) = D_1(K) \times D_2(K) \times \cdots \times D_n(K);$$

[C2] there exists $M \in \mathbf{N}$ such that

$$D(K+1) \subset D(K), \qquad 0 \leq K < M,$$

$$D(K) = \{\xi\}, \qquad K \geq M;$$

i.e., the set $D(K)$ gets smaller with increasing K, and after $K = M$, it reduces to a singleton;

[C3] $\mathbf{x} \in D(K) \Rightarrow \mathbf{F}(\mathbf{x}) \in D(K+1), \qquad \forall K \in \mathbf{N}.$

ξ will be called the point of convergence of $\{D(K)\}$. $\square$

As will be shown in Theorem 1, $D(K)$ is the set in which the iterates stay after the K-th pseudo-cycle, and therefore the iterates converge to ξ in at most M pseudo-cycles. Notice that the above definition contains explicit mentions of the integer M and of the point of convergence ξ, which may not be known in a non-trivial real example. Asynchronous contraction, nevertheless, is an important concept, since it establishes the starting point of our study.

[C2] and [C3] in the above definition are intuitive. In fact, any definition of contraction imposes conditions of a similar nature. On the contrary, [C1] might seem counter-intuitive.

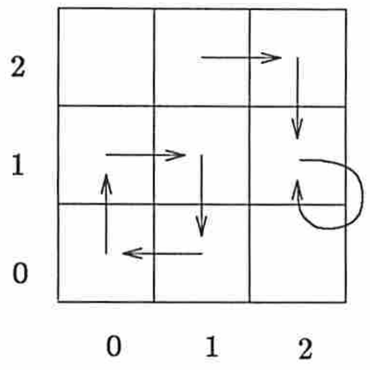

Figure 1: Asynchronous distortion

Yet, it is the requirement of an ACO that is due to the asynchronous nature of the computations. [C1] is needed because of *asynchronous distortions* which occur in asynchronous algorithms. The connection between this behavior and condition [C1] is the main issue in this paper. When asynchronous distortion occurs, an iterate value in $D(K)$ is generated, which is not possible to reach by successive applications of the iteration operator starting from the initial data. It will be shown later that there exists a schedule such that each point of $D(K)$ is visited in the K-th pseudo-cycle, when $D(0)$ contains a finite number of elements. In other words, all points in $D(K)$ are reachable by asynchronous distortion. Since we want convergence for *all* schedules, we need to guarantee the invariance of $D(K)$ under asynchronous distortion. Therefore, [C1] is *necessary* for convergence for finite $D(0)$ when no restriction is enforced on the schedule. Whether it is necessary for domains that contain an infinite number of elements, however, is still an open question. The reason for the uncertainty is that the proof of the necessary condition for the finite case is based on the construction of an asynchronous iteration such that all of the points in $D(K)$ are visited in the K-th pseudo-cycle of this asynchronous iteration. If $D(K)$ contained an infinite number of elements, then the constructed schedule would not be pseudo-periodic.

Example 1 Figure 1 displays an iteration operator in which the arc from each element in the domain points to the value we obtain when we apply the operator to that element. Consider the iteration that starts with $\mathbf{x}(0) = (1, 2)$ and that converges synchronously as well as for many asynchronous schedules to $\mathbf{y} = (2, 1)$, as long as the iterates do not leave the path from $\mathbf{x}(0)$ to $\mathbf{y}$. However, some schedules may cause distortion; i.e., the iterates may be "deflected" to the cycle in the lower left part of the domain. For instance, let there be two processors: P1 and P2. P1 first updates x_1 and then x_2. Right after this, P2 updates x_2, using the initial value of x_1 and the most recent value of x_2. At this point, the iterate value is $\mathbf{z} = (\mathbf{1}, \mathbf{0})$, and starting with $\mathbf{z}$, the algorithm cycles even if the remaining part of the schedule is synchronous. $\square$

In the above example, we observe that the cause of asynchronous distortion is the overlapping of the computations of x_2. If we disallow this so that each update of a component uses the most recent value of that component, then the iterates always converge to $\mathbf{y}$. Therefore, [C1] of the above definition is a necessary condition only when the schedule is not restricted. In fact, Proposition 6 gives a sufficient condition for a class of schedules without enforcing [C1].

To conclude this section, we give the following proposition for the sake of completeness.

Proposition 2 *The point of convergence ξ of $\{D(K)\}$ is also the unique fixed point of* $\mathbf{F}$ *in* $D(0)$.

Proof. ξ is a fixed point, because from Definition 2 , $\mathbf{F}(\xi) \in D(M+1) = \{\xi\}$. Suppose that there exists another fixed point ξ' which is not equal to ξ. From [C3] of Definition

2, $\xi' \in D(K)$ implies $\xi' \in D(K+1)$, and by induction $\xi' \in D(K)$ for all $K \in \mathbf{N}$ which contradicts [C2] of Definition 2. $\square$

3 Necessary and Sufficient Condition

In this paper, the word *convergence* has a slightly restricted meaning. We say that a sequence $\{\mathbf{x}(k)\}$ converges to ξ if it reaches ξ in a finite number of steps; i.e., there exists an integer m such that $\mathbf{x}(k) = \xi$ for all $k \geq m$. This definition, albeit restrictive, is applicable to the implementations of all iterative algorithms in digital computers: because of finite word length, infinite domains can only be represented as finite domains.

Theorem 1 gives a sufficient condition for convergence on infinite or finite domains. This condition is also proven to be necessary in Theorem 2, for the case of finite domains. It should be clear, however, that the condition is still necessary for implementations of algorithms on infinite domains in digital computers because of finite word length and finite memory size.

3.1 Sufficient Condition

From the discussion of Definition 2, it should be clear that if $\mathbf{F}$ is asynchronously contracting, then the corresponding asynchronous iteration converges to the fixed point of $\mathbf{F}$, for all schedules. We, nevertheless, give a formal statement and a proof of it, for completeness.

Theorem 1 (Sufficient Condition) *If $\mathbf{F}$ is an ACO on a set $D(0)$, then any asynchronous iteration $\{\mathbf{x}(k)\} = (\mathbf{F}, \mathbf{x}(0), \mathcal{S})$ with $\mathbf{x}(0) \in D(0)$ converges to the unique fixed point ξ of $\mathbf{F}$ in $D(0)$, for all pseudo-periodic schedules $\mathcal{S}$.*

Proof. We will simply show, by induction, that starting from $\tau_i(K)$, the i-th component of the iterates indefinitely enters $D_i(K)$ for all $i \in \mathbf{I}$; i.e.,

$$k \geq \tau_i(K) \;\Rightarrow\; x_i(k) \;\in\; D_i(K), \qquad \forall i \in \mathbf{I},\; K \in \mathbf{N}. \tag{2}$$

Suppose this is true for $K = L$ (induction hypothesis) and consider the update at $\tau_{i'}(L+1)$. Since the i-th component of the input vector for this update is generated no earlier than $\tau_i(L)$, the input vector is in $D(L)$, by the hypothesis. In other words, for $j = \tau_{i'}(L+1)$

$$\mathbf{u}(j) \;=\; (x_1(\beta_1(j),\; x_2(\beta_2(j)), \ldots) \;\in\; D(L).$$

Note that this would not always be true if $D(L)$ were not a Cartesian product. As a consequence, we have

$$x_{i'}(j) \;=\; F_{i'}(\mathbf{u}(j)) \;\in\; D_{i'}(L+1)$$

for $j = \tau_{i'}(L+1)$, and by induction it can be generalized for $j \geq \tau_{i'}(L+1)$. Since this is valid for all $i' \in \mathbf{I}$, (2) holds for $K = L+1$, and by induction for all $K \in \mathbf{N}$. Here, we are omitting the proof of the basis (that all the iterates are in $D(0)$), but the argument is the same.

(2) implies that after all the components are updated in the K-th pseudo-cycle the iterates indefinitely fall into $D(K)$, and therefore the iteration converges to ξ in at most M pseudo-cycles. $\square$

3.2 Necessary Condition

The following definitions and the subsequent lemmas establish the background required for the proof of Theorem 2.

Definition 3 Let $\{\mathrm{x}(k)\} = (\mathbf{F}, \mathrm{x}(0), \mathcal{S})$ be an asynchronous iteration and $\{\Phi(K)\}$ be a pseudo-cycle sequence associated with it. The *outcome* of $\{\mathrm{x}(k)\}$ in the K-th pseudo-cycle, $P(K) = P_1(K) \times P_2(K) \times \cdots \times P_n(K)$, is the set such that

$$P_i(K) = \{x_i(j) | i \in \alpha(j) \text{ and } j \in \Phi(K)\}, \qquad \forall i \in \mathbf{I}, \;\; K \in \mathbf{N}. \qquad \square$$

In the following discussions, without explicitly mentioning it we assume that $P(K)$, $P'(K)$, $P''(K)$, ... correspond to $\{\mathrm{x}(k)\}$, $\{\mathrm{x}'(k)\}$, $\{\mathrm{x}''(k)\}$, ... respectively.

Lemma 1 is based on the concept of *merging* asynchronous iterations. Merging $\{\mathrm{x}'(k)\}$ and $\{\mathrm{x}''(k)\}$, each starting with the same initial vector $\mathrm{x}(0)$, means the construction of another asynchronous iteration $\{\mathrm{x}(k)\}$ in the following way. In the K-th pseudo-cycle of $\{\mathrm{x}(k)\}$, first the outputs of the K-th pseudo-cycle of $\{\mathrm{x}'(k)\}$ are obtained in the same order, then this procedure is repeated for the K-th pseudo-cycle of $\{\mathrm{x}''(k)\}$, as shown in Example 2.

Example 2

$$\begin{aligned}
\{\mathrm{x}'(k)\} &= \underbrace{(1,1)}_{\Phi'(0)} \; \underbrace{(2,1)\;(2,2)}_{\Phi'(1)} \; \underbrace{(3,2)\;(3,3)}_{\Phi'(2)} \cdots \\
\{\mathrm{x}''(k)\} &= \underbrace{(1,1)}_{\Phi''(0)} \; \underbrace{(1,5)\;(5,5)}_{\Phi''(1)} \; \underbrace{(6,5)\;(6,6)}_{\Phi''(2)} \cdots \\
\{\mathrm{x}(k)\} &= \underbrace{(1,1)}_{\Phi(0)} \; \underbrace{(2,1)\;(2,2)\;(2,5)\;(5,5)}_{\Phi(1)} \; \underbrace{(3,5)\;(3,3)\;(6,3)\;(6,6)}_{\Phi(2)} \cdots
\end{aligned}$$

Each of these sequences starts with the same initial vector $\mathrm{x}(0) = (1,1)$, and for $K > 0$ $\Phi(K)$ has two phases. For example, in $\Phi(1)$ first x_1 then x_2 are updated, each time yielding the value 2, because this phase corresponds to $\Phi'(1)$. In the second phase of $\{\Phi(K)\}$, x_2 and x_1 are updated respectively, each time yielding 5, as in $\Phi''(K)$. $\square$

Lemma 1 *Let* $\{\mathrm{x}'(k)\} = (\mathbf{F}, \mathrm{x}(0), \mathcal{S}')$ *and* $\{\mathrm{x}''(k)\} = (\mathbf{F}, \mathrm{x}(0), \mathcal{S}'')$ *be asynchronous iterations w.r.t.* $\mathcal{S}'$ *and* $\mathcal{S}''$*, respectively. There exists an asynchronous iteration* $\{\mathrm{x}(k)\} = (\mathbf{F}, \mathrm{x}(0), \mathcal{S})$ *such that* $P(K) = P'(K) \cup P''(K)$ *for all* $K \in \mathbf{N}$.

Proof. The desired $\{\mathrm{x}(k)\}$ is obtained by merging $\{\mathrm{x}'(k)\}$ and $\{\mathrm{x}''(k)\}$. The formal definition of the schedule for $\{\mathrm{x}(k)\}$ is given by the series of formulas below.

$$\varphi(K) = \begin{cases} 0 & \text{if } K = 0 \\ \varphi'(K) + \varphi''(K) - 1 & \text{otherwise} \end{cases}$$

Literally, the length of the K-th pseudo-cycle of $\{\mathrm{x}(k)\}$ is the sum of the lengths of the K-th pseudo-cycles of $\{\mathrm{x}'(k)\}$ and $\{\mathrm{x}''(k)\}$, for $K > 0$.

$$\delta'(k) = \begin{cases} 0 & \text{if } k = 0 \\ k + \varphi''(K) - 1 & \text{if } k > 0 \text{ and } k \in \Phi'(K) \end{cases}$$

$$\delta''(k) = \begin{cases} 0 & \text{if } k = 0 \\ k + \varphi'(K+1) - 1 & \text{if } k > 0 \text{ and } k \in \Phi''(K) \end{cases}$$

δ' (δ'') simply maps the indices of the sequence $\{\mathrm{x}'(k)\}$ ($\{\mathrm{x}''(k)\}$) to the corresponding indices of $\{\mathrm{x}(k)\}$. The inverse mapping is defined by

$$\delta^{-1}(k) = \begin{cases} 0 & \text{if } k = 0 \\ k - \varphi''(K) + 1 & \text{if } k > 0 \text{ and } 0 \le k - \varphi(K) < |\Phi'(K)| \\ k - \varphi'(K+1) + 1 & \text{if } |\Phi'(K)| \le k - \varphi(K) < |\Phi'(K)| + |\Phi''(K)| \end{cases}$$

δ^{-1} maps the indices of $\{\mathrm{x}(k)\}$ to the corresponding indices of $\{\mathrm{x}'(k)\}$ or $\{\mathrm{x}''(k)\}$. In this definition, the second (third) line corresponds to the elements of the first (second) phase of $\Phi(K)$. If k is in the first (second) phase of a pseudo-cycle, the value of $\alpha(k)$ is the same as the value of α' (α'') for the corresponding element of $\{\mathrm{x}'(k)\}$ ($\{\mathrm{x}''(k)\}$); i.e.

$$\alpha(k) = \begin{cases} \alpha'(\delta^{-1}(k))) & 0 \le k - \varphi(K) < |\Phi'(K)| \\ \alpha''(\delta^{-1}(k))) & \text{if } |\Phi'(K)| \le k - \varphi(K) < |\Phi'(K)| + |\Phi''(K)| \end{cases}$$

β can be defined in a similar manner, as follows.

$$\beta_i(k) \quad = \quad \begin{cases} \delta'(\beta_i'(\delta^{-1}(k))) & 0 \le k - \varphi(K) < |\Phi'(K)| \\ \delta''(\beta_i''(\delta^{-1}(k))) & \text{if } |\Phi'(K)| \le k - \varphi(K) < |\Phi'(K)| + |\Phi''(K)| \end{cases}$$

It is obvious that the resulting sequence indeed satisfies the assumptions of asynchronous iteration and the requirement of the lemma. $\square$

Lemma 2 *Let* $\{\mathbf{x}^{(1)}(k)\} = (\mathbf{F}, \mathbf{x}(0), \mathcal{S}^{(1)})$, $\{\mathbf{x}^{(2)}(k)\} = (\mathbf{F}, \mathbf{x}(0), \mathcal{S}^{(2)}), \cdots$ and $\{\mathbf{x}^{(m)}(k)\} = (\mathbf{F}, \mathbf{x}(0), \mathcal{S}^{(m)})$ *be asynchronous iterations corresponding to* $\mathbf{F}$ *and starting with* $\mathbf{x}(0)$. *Then, there exists* $\{\mathbf{x}(k)\} = (\mathbf{F}, \mathbf{x}(0), \mathcal{S})$, *such that*

$$P(K) = P^{(1)}(K) \cup P^{(2)}(K) \cup \cdots \cup P^{(m)}(K), \qquad \forall K \in \mathbf{N}.$$

Proof. Obvious from the previous lemma. $\square$

Lemma 2 states that the pseudo-cycles of different asynchronous iterations on the same $\mathbf{F}$ and $\mathbf{x}(0)$ can be merged to form of a new asynchronous iteration.

Lemma 3 *Let* $\{\mathbf{x}'(k)\} = (\mathbf{F}, \mathbf{x}(0), \mathcal{S}')$ *be an asynchronous iteration. If* $a_{i_1} \in P_{i_1}'(K)$ *and* $a_{i_2} \in P_{i_2}'(K)$, *then there exists an asynchronous iteration* $\{\mathbf{x}(k)\} = (\mathbf{F}, \mathbf{x}(0), \mathcal{S})$ *such that for some* $p \in \Phi(K)$, $x_{i_1}(p) = a_{i_1}$, and $x_{i_2}(p) = a_{i_2}$ *for all* $K \in \mathbf{N}^+$, *and* $i_1 \neq i_2$.

Proof. Let a_{i_1} and a_{i_2} be the outputs of the updates at j_1 and $j_2 \in \Phi'(K)$, respectively; i.e.,

$$i_1 \in \alpha'(j_1) \ , \ x_{i_1}'(j_1) = a_{i_1},$$

and

$$i_2 \in \alpha'(j_2) \ , \ x_{i_2}'(j_2) = a_{i_2} \ .$$

Now, construct the first K pseudo-cycles of $\{\mathrm{x}(k)\}$ by inserting an iterate at the end of the K-th pseudo-cycle of $\{\mathrm{x}'(k)\}$ such that a_{i_1} and a_{i_2} are the outputs of this update. More formally, define

- $\alpha(k) = \alpha'(k),\ \beta(k) = \beta'(k)$; therefore, $\mathrm{x}(k) = \mathrm{x}'(k)$, for $k < \varphi'(K+1)$;
- $\alpha(p) = \{i_1, i_2\}$, $\beta_{i_1}(p) = \beta'_{i_1}(j_1)$, $\beta_{i_2}(p) = \beta'_{i_2}(j_2)$;
- $\varphi(L) = \varphi'(L)$ for $L \leq K$;
- $\varphi(K+1) = \varphi'(K+1) + 1$,

where $p = \varphi'(K+1)$. Also, for $k > p+1$, $\alpha(k)$ and $\beta(k)$ are chosen freely to satisfy the conditions of Proposition 1. Furthermore, it can easily be seen that the first K pseudo-cycles of $\{\mathrm{x}(k)\}$ satisfy these conditions. $\square$

The generalization of the above proof gives the following lemma.

Lemma 4 *Let $\{\mathrm{x}'(k)\} = (\mathbf{F}, \mathrm{x}(0), S')$ be an asynchronous iteration. There exists $\{\mathrm{x}(k)\} = (\mathbf{F}, \mathrm{x}(0), S)$ such that for any n-dimensional vector $\mathrm{a} \in P'(K)$, there exists a j in the K-th cycle of $\{\mathrm{x}(k)\}$ such that $\mathrm{x}(j) = \mathrm{a}$.* $\square$

This lemma simply states that given a vector a, each component of which is the output of an update in the K-th pseudo-cycle of a particular asynchronous iteration, we can construct another asynchronous iteration such that the value of the global data is a, at some time instance in the K-th pseudo-cycle.

Now, we can prove Theorem 2, which says that the sufficient condition of convergence given in Theorem 1 (i.e., $\mathbf{F}$ being asynchronously contracting) is also necessary for convergence for finite data domains.

Theorem 2 (Necessary Condition) *Let* $\mathbf{F}$ *be an operator defined in a domain* S *that contains a finite number of elements. If for all pseudo-periodic schedules, all asynchronous iterations* $\{\mathbf{x}(k)\} = (\mathbf{F}, \mathbf{x}(0), \mathcal{S})$, *starting with* $\mathbf{x}(0) \in S$ *and corresponding to* $\mathbf{F}$, *converge to the same fixed point* ξ, *then* $\mathbf{F}$ *is asynchronously contracting on a set* $D(0) \subseteq S$, *where* $\mathbf{x}(0) \in D(0)$.

Proof. Define $D(K) = D_1(K) \times D_2(K) \times \cdots \times D_n(K)$ such that

$$D_i(K) = \{a_i | a_i \in P_i(K) \text{ for some } \mathcal{S}\}, \qquad \forall i \in \mathbf{I},\ \forall K \in \mathbf{N}.$$

We shall show by mathematical induction that there exists an asynchronous iteration $\{\mathbf{x}'(k)\} = (\mathbf{F}, \mathbf{x}(0), \mathcal{S}')$ such that

$$P'(K) = D(K), \qquad \forall K \in \mathbf{N}.$$

We will only show the induction part, since the argument for $K = 0$ is very similar. Assume that the above statement is true for $K < L$. Since the domain of $\mathbf{F}$ is finite, the iterates of all asynchronous iterations can take a finite number of values; therefore, $D(L)$ contains a finite number of elements. Consequently, there are $m - 1$ schedules that satisfy

$$D(L) = P^{(1)}(L) \cup P^{(2)}(L) \cup \cdots \cup P^{(m-1)}(L),$$

and from the hypothesis, there exists $\mathcal{S}^{(m)}$, such that,

$$P^{(m)}(K) = D(K), \qquad K < L.$$

For all $j = 1, 2, \ldots, m$ and for all $K \in \mathbf{N}$, $P^{(j)}(K) \subseteq D(K)$. From Lemma 2 there exists an asynchronous iteration, $\{\mathbf{x}'(k)\} = (\mathbf{F}, \mathbf{x}(0), \mathcal{S}')$ such that

$$P'(K) = D(K), \qquad \forall K \leq L,$$

and by induction, $P'(K) = D(K)$ for all $K \in \mathbf{N}$. Then, from Lemma 4 there exists an asynchronous iteration $\{\mathbf{x}(k)\} = (\mathbf{F}, \mathbf{x}(0), \mathcal{S})$ such that

$$D(K) \subseteq \{\mathbf{x}(j) | j \in \Phi(K)\}, \quad \forall K \in \mathbf{N}.$$

Now, we can show that the sequence $\{D(K)\}$ satisfies the conditions of Definition 2 :

[C1]: This condition is satisfied by definition.

[C2]: Definition 2 permits us to coalesce two consecutive pseudo-cycles into one; i.e., there exists a $\overline{\mathcal{S}}$, corresponding to $\{\mathbf{x}(k)\}$, such that $\overline{\Phi}(K) = \Phi(K) \cup \Phi(K+1)$. Therefore,

$$D(K+1) \subseteq D(K) \cup D(K+1) \subseteq \overline{P}(K) \subseteq D(K).$$

On the other hand, for $D(K) \neq \{\xi\}$, $D(K+1) \neq D(K)$. To see this, let us assume that $D(K) = D(K+1)$; i.e., each component value in $D(K)$ can be generated by using input values also in $D(K)$. Since all of the elements of $D(K)$ are generated in the $(K+1)$-th pseudo-cycle, there exists a schedule such that in pseudo-cycle $K+2$ all the points of $D(K)$ are produced. Applying the same argument to pseudo-cycles $K+3, K+4, \ldots$, we can construct an asynchronous iteration that indefinitely repeats the same elements in $D(K)$. This contradicts the hypothesis of the theorem, except when $D(K) = \{\xi\}$. Thus,

$$D(K+1) \subset D(K), \qquad \forall\, D(K) \neq \{\xi\}.$$

Since $\{\mathbf{x}(k)\}$ converges to ξ, there exists a $M' \in \mathbf{N}$ such that

$$\mathbf{x}(j) = \xi, \qquad \forall j \geq \varphi(M'-1) \ .$$

Therefore,

$$D(K) = \{\xi\}, \qquad \forall\, K \geq M' \ ,$$

and [C2] follows.

[C3]: There exists $\{\mathbf{x}'(k)\} = (\mathbf{F}, \mathbf{x}(0), \mathcal{S}')$ such that $P'(K) = D(K)$ for $K \in \mathbf{N}$. Consequently, for all $\mathbf{x} \in P'(K)$ and $i \in \mathbf{I}$, there exists $\{\mathbf{x}''(k)\} = (\mathbf{F}, \mathbf{x}(0), \mathcal{S}'')$ such that

- $P''(K) = P'(K)$, and
- $x_i(j_i) = F_i(\mathbf{x})$ for some $j_i \in \Phi''(K+1)$.

Therefore, $F_i(\mathbf{x}) \in P''_i(K+1)$ for all $i \in \mathbf{I}$, and hence $\mathbf{F}(\mathbf{x}) \in D(K+1)$, which completes the proof. $\square$

Finally, for the sake of completeness, we give the following theorem, which is the restatement of Theorem 1 and Theorem 2 combined.

Theorem 3 *Let* $\mathbf{F}$ *be an operator defined in a domain* S *that contains a finite number of elements. For all pseudo-periodic schedules, any asynchronous iteration* $(\mathbf{F}, \mathbf{x}(0), \mathcal{S})$ *corresponding to* $\mathbf{F}$ *and starting with any initial guess* $\mathbf{x}(0)$ *converges to a fixed point of* $\mathbf{F}$ *if and only if* $\mathbf{F}$ *is asynchronously contracting in a set* $D(0)$ *that contains* $\mathbf{x}(0)$. $\square$

4 Other Sufficient Conditions

In this section, four sets of sufficient conditions are given, based on an ordering relation defined on the domain of the iterated data. These conditions are efficiently verifiable, as demonstrated in Section 5. They do not explicitly mention M, and they imply the existence of a fixed point in the considered domain. Except for Proposition 6, they also imply the unicity of the fixed point.

Definition 4 A relation $\preceq$ in a set A is called an ordering relation on A iff, for every $\mathbf{a}, \mathbf{b}, \mathbf{c} \in A$,

- $\mathbf{a} \preceq \mathbf{a}$,
- $\mathbf{a} \preceq \mathbf{b}, \mathbf{b} \preceq \mathbf{a} \;\Rightarrow\; \mathbf{a} = \mathbf{b}$,
- $\mathbf{a} \preceq \mathbf{b}, \mathbf{b} \preceq \mathbf{c} \;\Rightarrow\; \mathbf{a} \preceq \mathbf{c}$. $\square$

We write $\mathbf{a} \prec \mathbf{b}$ if $\mathbf{a} \preceq \mathbf{b}$, but $\mathbf{a} \neq \mathbf{b}$. Also, in the following propositions $\{\mathbf{y}(K)\}$ refers to the sequence such that

- $\mathbf{y}(0) = \mathbf{x}(0)$,
- $\mathbf{y}(K) = \mathbf{F}(\mathbf{y}(K-1)), \qquad \forall K \in \mathbf{N}^+$.

Throughout this paper, $\preceq$ is assumed to be an ordering relation on the set $D(0) = D_1(0) \times \cdots \times D_n(0)$ such that $\mathbf{a} \preceq \mathbf{b}$ iff $a_i \preceq_i b_i$ for all $i \in \mathbf{I}$, and $\mathbf{a}, \mathbf{b} \in D(0)$, where $\preceq_i$ is an ordering relation on $D_i(0)$. We will not repeat that $D(0)$ is a Cartesian product every time.

The following proposition will be applied to the all-pairs shortest path problem in Section 5. It does not need $D(0)$ to be an infinite set, whereas Propositions 4-6 do require this restriction.

Proposition 3 $\mathbf{F}$ *has a fixed point* ξ *to which every asynchronous iteration* $\{\mathbf{x}(k)\} = (\mathbf{F}, \mathbf{x}(0), S)$ *with* $\mathbf{x}(0) \in D(0)$ *converges if:*

[C4] $\mathbf{F}$ *is closed on* $D(0)$*; i.e., for all* $\mathbf{a} \in D(0)$, $\quad \mathbf{F}(\mathbf{a}) \in D(0)$,

[C5] $\{\mathbf{y}(K)\}$ *converges, and also for all* $K \in \mathbf{N}$, $\mathbf{y}(K) \succeq \mathbf{y}(K+1)$.

[C6] $\mathbf{F}$ *is monotone on* $D(0)$*; i.e., for all* $\mathbf{a}, \mathbf{b} \in D(0)$, $\quad \mathbf{a} \preceq \mathbf{b} \Rightarrow \mathbf{F}(\mathbf{a}) \preceq \mathbf{F}(\mathbf{b})$.

Proof. From [C4] the elements of $\{\mathbf{y}(K)\}$ take values from the set $D(0)$, and from [C5] there exists $M \in \mathbf{N}$ such that

- $\mathbf{y}(K) \succ \mathbf{y}(K+1), \qquad 0 \leq K < M$, and
- $\mathbf{y}(K) = \xi, \qquad K \geq M$,

where ξ is a fixed point. Define $\{D(K)\}$ such that

$$D(K) = \{\mathbf{a} | \xi \preceq \mathbf{a} \preceq \mathbf{y}(K) \text{ and } \mathbf{a} \in D(0)\}.$$

Obviously, $D(K)$ satisfies the first two conditions of Definition 2. On the other hand, from [C6] for all $\mathbf{a} \in D(K)$, $\xi \preceq \mathbf{F}(\mathbf{a})$, and

$$\mathbf{F}(\mathbf{a}) \preceq \mathbf{F}(\mathbf{y}(K)) = \mathbf{y}(K+1).$$

Also, since $\mathbf{F}(\mathbf{a}) \in D(0)$ (from [C4]) , $\mathbf{F}(\mathbf{a}) \in D(K+1)$. Hence, all the conditions of Definition 2 and therefore the condition of Theorem 1 are satisfied, and the claim follows. $\square$

The following proposition is the obvious consequence of Proposition 3.

Proposition 4 *Let* $D(0)$ *be a finite set. Then* $\mathbf{F}$ *has a fixed point* ξ *to which every asynchronous iteration* $\{\mathbf{x}(k)\} = (\mathbf{F}, \mathbf{x}(0), \mathcal{S})$ *with* $\mathbf{x}(0) \in D(0)$ *converges if:*

[C7] $\mathbf{F}$ *is closed on* $D(0)$*; i.e., for all* $\mathbf{a} \in D(0)$, $\quad \mathbf{F}(\mathbf{a}) \in D(0)$*;*

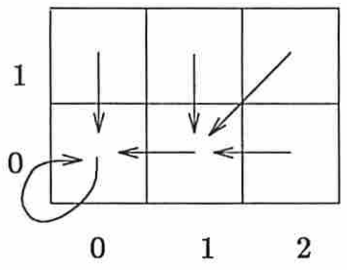

Figure 2: An operator satisfying the conditions of Propositions 3 and 4

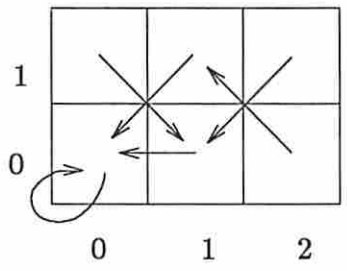

Figure 3: An ACO not satisfying the conditions of Propositions 3 and 4

[C8] **F** *is non-expansive on* $D(0)$*; i.e., for all* $\mathrm{a} \in D(0)$, $\quad \mathbf{F}(\mathrm{a}) \preceq \mathrm{a}$;

[C9] **F** *is monotone on* $D(0)$*; i.e., for all* $\mathrm{a}, \mathrm{b} \in D(0)$, $\quad \mathrm{a} \preceq \mathrm{b} \Rightarrow \mathbf{F}(\mathrm{a}) \preceq \mathbf{F}(\mathrm{b})$.

Proof. Conditions [C7] and [C9] are the same as [C4] and [C6] in Proposition 3, and since $D(0)$ is finite, [C5] of Proposition 3 is also satisfied. □

Example 3 The operator given by Figure 2 satisfies the conditions of the above propositions when $\preceq$ is selected as the regular ordering relation defined on numbers. The notation of the figure is the same as the one given in Example 1. □

The question may arise of whether the conditions of Proposition 3 and 4 are necessary for convergence. The following counter-example shows that they are not.

Example 4 $\mathbf{F}$ is an operator defined on the elements of $D(0) = \{0,1,2\} \times \{0,1\}$, as shown in Figure 3. $\mathbf{F}$ is asynchronously contracting on $D(0)$, because if we take

$$D(1) = \{0,1\} \times \{0,1\}, \qquad D(2) = \{0,1\} \times \{0\}, \qquad D(3) = D(4) = \cdots = \{(0,0)\},$$

then for all K,

$$\mathbf{x} \in D(K) \Rightarrow \mathbf{F}(\mathbf{x}) \in D(K+1).$$

Suppose that $\mathbf{F}$ also satisfies the conditions of the above propositions. Since $\mathbf{F}$ is non-expansive,

$$\mathbf{F}(0,1) \preceq (0,1) \Rightarrow 1 \preceq 0,$$

and

$$\mathbf{F}(1,0) \preceq (1,0) \Rightarrow 0 \preceq 1,$$

which is a contradiction, and therefore, $\mathbf{F}$ is *not* non-expansive but still asynchronously contracting. $\square$

Another set of sufficient conditions is given below.

Proposition 5 *Let $D(0)$ be a finite set. Then, every asynchronous iteration $\{\mathbf{x}(k)\} = (\mathbf{F}, \mathbf{x}(0), S)$ with $\mathbf{x}(0) \in D(0)$ converges if*

[C10] *$\mathbf{F}$ is closed on $D(0)$; i.e., for all $\mathbf{a} \in D(0)$, $\mathbf{F}(\mathbf{a}) \in D(0)$;*

[C11] *there exists a fixed point $\xi \in D(0)$ such that for all $\mathbf{a} \in D(0)$ and $i \in \mathbf{I}$, $F_i(\mathbf{a}) \prec_i a_i$ if $a_i \neq \xi_i$, and $F_i(\mathbf{a}) = \xi_i$ otherwise.*

Proof. ξ is the unique fixed point in $D(0)$. This can easily be seen by contradiction; i.e., by assuming the existence of another fixed point $\xi' \neq \xi$. Then, [C11] is not satisfied for $\mathbf{a} = \xi'$. Now, define $\{D(K)\}$ such that

$$D_i(K) = D_i(K-1) - R_i(K-1), \qquad \forall K \in \mathbf{N}^+,\ \forall i \in \mathbf{I},$$

where $R_i(K)$ is the set of maximal elements of $D_i(K)$, except ξ_i; i.e.,

$$R_i(K) = \{a_i | (a_i \in D_i(K)) \text{ and } (\forall b_i \in D_i(K),\ a_i \preceq_i b_i \Rightarrow a_i = b_i)\} - \{\xi_i\}, \quad \forall K \in \mathbf{N} \text{ and } \forall i \in \mathbf{I}.$$

It is obvious that every $D_i(K)$ has a maximal element, because of the fact that $D_i(K)$ is finite. Therefore, $R_i(K)$'s are non-empty, except when $D_i(K) = \{\xi_i\}$. Thus, $D_i(K+1) \subset D_i(K)$ except when $D_i(K) = \{\xi_i\}$, from which the first two conditions of Definition 2 are satisfied.

To prove the last condition, we first note that $\xi \in D(0)$, and since for all $i \in \mathbf{I}$ $\xi_i \notin R_i(K)$, then $\xi \in D(K)$, for all $K \in \mathbf{N}$. Now, for a given $\mathbf{a} \in D(K)$, there are two possibilities, for all $K \in \mathbf{N}$,

- if $\mathbf{a} = \xi$, then $\mathbf{F}(\mathbf{a}) = \xi \in D(K+1)$;
- if $\mathbf{a} \neq \xi$, then $F_i(\mathbf{a}) \prec_i a_i$ or $a_i = \xi_i$, and therefore, $F_i(\mathbf{a}) \notin R_i(K)$, which implies that $F_i(\mathbf{a}) \in D_i(K+1)$, for all $i \in \mathbf{I}$.

In both cases, the conditions of the ACO are satisfied. From Theorem 1 the claim of the proposition follows. $\square$

As in Proposition 3 the conditions given in Proposition 5 are not necessary ones. Example 4 can be used to support this. The major difference between Proposition 4 and Proposition 5 is that the latter one does not impose monotonicity.

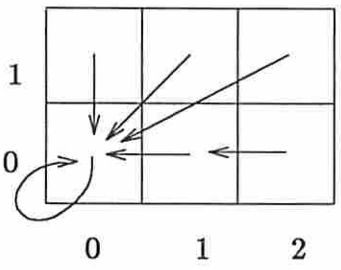

Figure 4: An operator which is not monotone

Example 5 Figure 4 describes an operator satisfying Proposition 5 but not Propositions 3 and 4. In this example, the only ordering relation that would satisfy [C5] of Proposition 3 (or [C8] of Proposition 4) is the one defined for numbers in the usual sense, but for this ordering relation, monotonicity does not hold: although $(2,0) \preceq (2,1)$, $\mathbf{F}(2,0) \succ \mathbf{F}(2,1)$. $\square$

In the following proposition, a condition is imposed on the schedule; namely, a processor assigned to compute a component always uses the most recent value of that component as the input of an update. The conditions imposed on $\mathbf{F}$, on the other hand, are weakened.

Proposition 6 *Let $D(0)$ be a finite set. Then, an asynchronous iteration $(\mathbf{F}, \mathbf{x}(0), \mathcal{S})$ converges to a fixed point of $\mathbf{F}$ in $D(0)$ if the following conditions hold :*

[C12] *For all $k \in \mathbf{N}^+$, S satisfies*

$$i \in \alpha(k) \;\Rightarrow\; \beta_i(k) = k - 1;$$

[C13] *$\mathbf{F}$ is closed in $D(0)$;*

[C14] *$\mathbf{F}$ is non-expansive in $D(0)$; i.e., for all $\mathrm{a} \in D(0)$, $\mathbf{F}(\mathrm{a}) \preceq \mathrm{a}$.*

Proof. For all $k \in \mathbf{N}^+$, $i \in \mathbf{I}$ there are two cases possible :

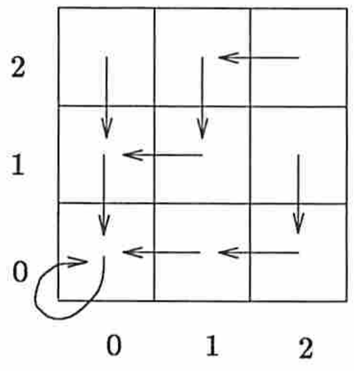

Figure 5: An operator which satisfies the conditions of Proposition 6 but is not ACO

- $i \in \alpha(k)$. In this case, $\beta_i(k) = k - 1$ and

  $$x_i(k) = \mathbf{F}(u(k)),$$

  where,

  $$u_i(k) = x_i(\beta_i(k)) = x_i(k-1).$$

  Since $\mathbf{F}$ is non-expansive,

  $$x_i(k) \preceq_i x_i(k-1).$$

- $i \notin \alpha(k)$. By definition, $x_i(k) = x_i(k-1)$.

Consequently, $\{\mathbf{x}(k)\}$ is monotonically decreasing, and since $D(0)$ is finite, it converges to some ξ. Therefore, there exists an M such that $\mathbf{x}(k) = \xi$ for all $k \geq \varphi(M)$. Also, for all $i \in \mathbf{I}$ there exists $l \in \Phi(M+1)$, such that $i \in \alpha(l)$. Hence,

$$\exists l \in \Phi(K+1), \text{ such that } x_i(l) = F_i(u(l)),$$

where $u_{i'}(l) = x_{i'}(\beta_{i'}(l))$ for all $i' \in \mathbf{I}$. Since $\beta_{i'}(l) \geq \varphi(M)$, $u(l) = \xi$ and $x_i(l) = F_i(\xi) = \xi_i$, which proves that ξ is a fixed point. $\square$

Example 6 The conditions of Proposition 6 do not imply asynchronous contraction. The operator given in Figure 5 supports this statement: it satisfies the conditions of the proposition, although it is not asynchronously contracting. □

5 Applications

5.1 All-pairs Shortest Path Problem

Given a directed graph and the cost of each edge, the all-pairs shortest path problem consists of finding the lowest cost of any path from node i to node j, for each ordered pair (i, j). An iterative algorithm to solve this problem for the case where all the costs are non-negative can be given by :

$$C_{i,j}(k+1) = \min_{p}\{C_{i,p}(k) + C_{p,j}(k)\}, \qquad k = 0, 1, 2, \ldots$$

and the initial value $C_{i,j}(0)$ is the cost of the edge from i to j. If $i = j$, then $C_{i,j}(0) = 0$, and if there is no edge from i to j, then $C_{i,j}(0)$ is represented by the symbol ∞ and is larger than any number [1]. In matrix form the above can be written as

$$\mathbf{C}(k+1) = \Gamma(\mathbf{C}(k)), \qquad k = 0, 1, 2, \ldots$$

which has the same format as (1).

Let us define $D(0)$ as a Cartesian product of $D_{i,j}(0)$'s where $D_{i,j}(0)$ is the union of $\{\infty\}$ and the set of the costs of all paths from i to j. Notice that although $D(0)$ may be an infinite set, this does not prevent the application of Proposition 3, which does not require finite domains. $\mathbf{C} \in D(0)$ implies that $C_{i,p}$ and $C_{p,j}$ are the costs of some paths from i to p and p to j, respectively. Consequently, $C_{i,p} + C_{p,j}$, and therefore, $\Gamma_{i,j}(\mathbf{C})$ is the cost of some path from i to j. As a result, $\Gamma(C) \in D(0)$, which satisfies [C4] of Proposition 3. The other conditions can also easily be satisfied, which means that the iterative algorithm given above can be implemented in a multiprocessor environment with no synchronization.

5.2 Consistent Labeling Problem

The consistent labeling problem has already been described informally in the introduction. Here, we give the formal definition of the consistent labeling problem as in [13]. Let $A = \{a_1, \ldots, a_n\}$ be the set of objects to be labeled and $\Lambda = \{\lambda_1, \ldots, \lambda_m\}$ be the set of possible labels. Also, let Λ_i, be the set of labels compatible with a_i and Λ_{ij} be the set of label pairs compatible with a_i and a_j ; thus $(\lambda, \lambda') \in \Lambda_{ij}$ means that it is possible to label a_i with label λ and a_j with label λ'. By a *labeling* $L = (L_1, \ldots, L_n)$ of A , we mean an assignment of a set of labels $L_i \subseteq \Lambda_i$ to each $a_i \in A$. The labeling is *consistent* if for all i, j and for all $\lambda \in L_i$, there exists a $\lambda' \in L_j$ that is compatible with λ, i.e. $(\lambda, \lambda') \in \Lambda_{ij}$. We say that a labeling $L = \{L_1, \ldots, L_n\}$ *contains* another labeling L' if $L'_i \subseteq L_i$ for $i = 1, 2, \ldots, n$. The *greatest consistent labeling* L^∞ is a consistent labeling such that any other consistent labeling is contained in L^∞.

According to this model, the discrete relaxation procedure operates as follows. It starts with the initial labeling $L(0) = \{\Lambda_1, \ldots, \Lambda_n\}$. During each step, we eliminate from each L_i all labels λ, such that λ violates the condition of consistent labeling. We shall refer to the operation executed at each iteration as Δ. Therefore, for each iteration k $L(k+1) = \Delta(L(k))$ and Δ_i is the operator that discards from L_i all labels λ such that for at least one a_j there is no label $\lambda' \in L_j$ which is compatible with λ. It has been shown in [13] that this iterative procedure, implemented synchronously, converges to L^∞, and the following proposition states that it can also be implemented without any synchronization.

Proposition 7 *Any asynchronous iteration* $(\Delta, L(0), S)$ *converges to* L^∞.

Proof. Define $D(0) = D_1(0) \times \ldots \times D_n(0)$ such that

$$D_i(0) = \{X_i | L_i^\infty \subseteq X_i \subseteq L_i(0)\}, \qquad i \in \mathbf{I}.$$

Since the set of all labels is finite, $D(0)$ is a finite set, and it is readily shown in [13] that Δ is closed and non-expansive and [C8] of Proposition 4 are satisfied, w.r.t. the ordering relation $\subseteq$.

Now, let $L, \bar{L} \in D(0)$ be labelings such that $L \subseteq \bar{L}$ and λ be a label discarded from $\bar{L}_i$ by Δ_i when applied to $\bar{L}$. This implies that for at least one a_j, there is no label $\lambda' \in \bar{L}_j$ which is compatible with λ. Consequently, there is no label $\lambda' \in L_j$ which is compatible with λ, and therefore, if λ is in L_i, it is also discarded from L_i, by Δ_i, when applied to L. Thus, $\Delta(L) \subseteq \Delta(\bar{L})$; i.e., [C9] of Proposition 4 is satisfied, and the claim follows. $\square$

5.3 A Neural Net Model

Artificial neural net models (for short, neural nets) refer to computational paradigms that involve massive processing by large numbers of simple processing elements that are densely interconnected [9]. The simplest one of the computational elements, or nodes, used in neural nets sums its weighted inputs and passes the result through a nonlinearity, as shown in Figure 6. The node is characterized by an internal threshold or offset θ and by the type of the nonlinearity. More complex nodes may include temporal integration or other types of time dependencies and more complex mathematical operations than summation.

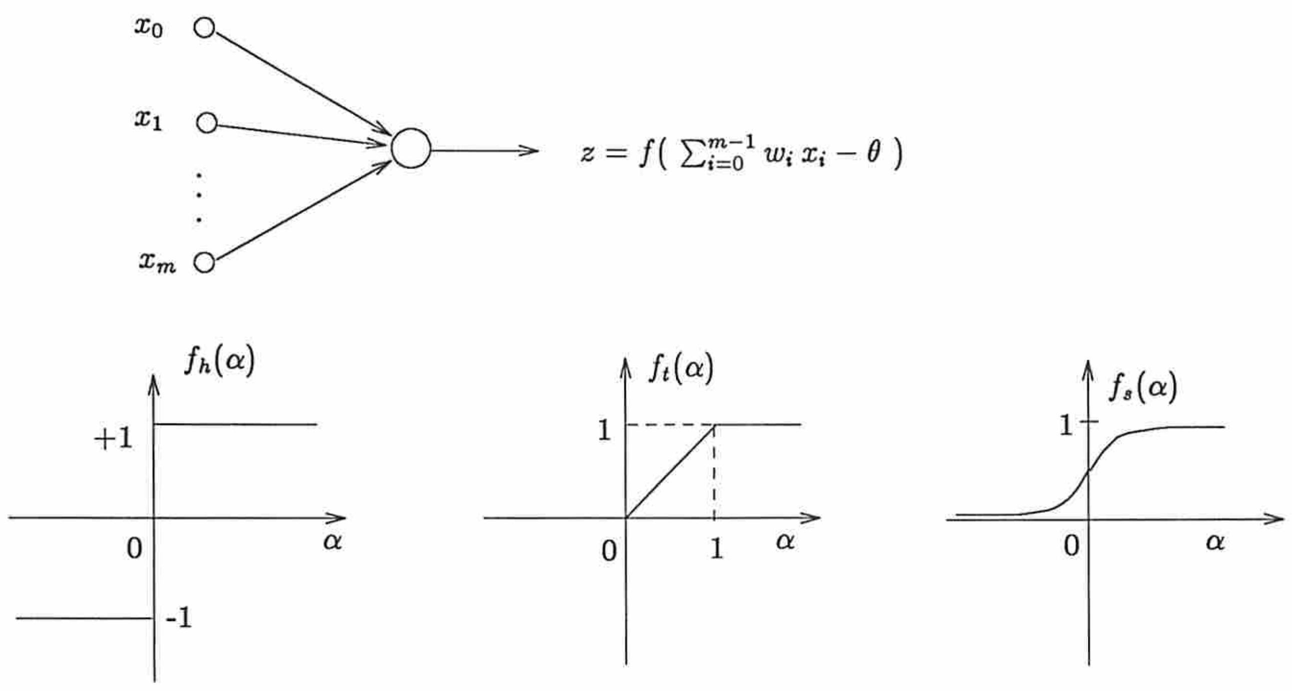

Figure 6: A computational element for a neural net

Consider the neural net model where the i-th node is assigned to an operation in the following form:

$$\Psi_i(\mathbf{x}) = f_t(x_i + s_i \sum_{i \neq j} w_{ij} x_j),$$

where s_i is either $+1$ or -1 and w_{ij}'s are positive integers. Assume that $\mathbf{x}$ is digital; therefore, it takes finite values in the domain D of the form

$$D = \{\frac{p}{q} \mid p \in \mathrm{N} \text{ and } 0 \leq p \leq q\}^n, \tag{3}$$

where $q \in \mathrm{N}^+$. We can easily see that the operator Ψ is not ACO. For example, let $D(0) = \{0.0, 0.5, 1.0\} \times \{0.0, 0.5, 1.0\}$ and

$$\begin{aligned} \Psi_1(\mathbf{x}) &= f_t(x_1 - x_2) \\ \Psi_2(\mathbf{x}) &= f_t(x_1 + x_2). \end{aligned}$$

Then, $D(1) = D(0) = \{0.0, 0.5, 1.0\} \times \{0.0, 0.5, 1.0\}$. Nevertheless, we can apply Proposition 6, because

- there is one-to-one correspondance between the computational elements and the components; therefore, the computation of each component uses the most recent value of that component;
- Ψ is closed in a proper selected domain D of the form (3);
- Ψ is non-expansive in D provided that $\preceq_i$ is equivalent to $\leq$ if $s_i = +1$ and $\geq$ otherwise.

Proposition 8 immediately follows.

Proposition 8 *All asynchronous iterations* $(\Psi, \mathrm{x}(0), S)$ *with* $\mathrm{x}(0) \in D$ *such that* D *is in the form of Equation 3 converge for all schedules satisfying* [C12] *of Proposition 6.* □

6 Conclusion

In this paper we have addressed the correctness issue for asynchronous iterative algorithms in the case of discrete shared data. The conditions are easy to apply to practical problems. We have applied them to three practical problems.

Future work could focus on designing new asynchronous algorithms based on the theory developed in this paper. These algorithms should also be tested on realistic multiprocessors, so that we can compare the convergence rates of asynchronous and synchronized algorithms.

ACKNOWLEDGEMENT. We wish to thank the anonymous referees whose fruitful comments helped to shape the final version of this paper.

References

[1] AHO, A. V., HOPCROFT, J. E., AND ULLMAN, J. D. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1974.

[2] BAUDET, G. M. Asynchronous iterative methods for multiprocessors. *J. ACM 25*, 2 (April 1978), 226–244.

[3] BERTSEKAS, D. P. Distributed asynchronous computation of fixed points. *Mathematical Programming 27* (1983), 107–120.

[4] BERTSEKAS, D. P. Distributed dynamic programming. *IEEE Trans. on Automatic Control 27* (1982), 610–616.

[5] CHAZAN, D., AND MIRANKER, W. Chaotic relaxation. *Linear Algebra and Its Applications 2* (1969), 199–222.

[6] HARALICK, R. M., AND SHAPIRO, L. G. The consistent labeling problem: Part I. *IEEE Trans. on PAMI 1*, 2 (April 1979), 173–184.

[7] HWANG, K., AND BRIGGS, F. A. *Computer Architecture and Parallel Processing.* McGraw-Hill, 1984.

[8] KUNG, H. T. Synchronized and asynchronous parallel algorithms for multiprocessors. In *Algorithms and Complexity : New Directions and Recent Results*, J. F. Traub, Ed., Academic Press, New-York, 1976.

[9] LIPPMANN, R. P. Introduction to computing with neural nets. *IEEE ASSP Magazine* (April 1987), 4–22.

[10] LUBACHEVSKY, B. D., AND MITRA, D. A chaotic asynchronous algorithm for computing the fixed point of a nonnegative matrix of unit spectral radius. *J. ACM 33*, 1 (January 1986), 130–150.

[11] MIELLOU, J. Asynchronous iterations and order intervals. In *Parallel Algorithms & Architectures*, M. C. et al., Ed., North-Holland, 1986.

[12] ROBERT, F. *Discrete Iterations.* Springer-Verlag, Berlin, 1986.

[13] ROSENFELD, A., HUMMEL, R. A., AND ZUCKER, S. W. Scene labeling by relaxation operations. *IEEE Trans. Systems, Man, and Cybernetics 6*, 6 (June 1976), 420–433.

[14] SPITERI, P. Parallel asynchronous algorithms for solving boundary value problems. In *Parallel Algorithms & Architectures*, M. C. et al., Ed., North-Holland, 1986.

[15] TSITSIKLIS, J. N. On the stability of asynchronous iterative processes. *Mathematical Systems Theory 20* (1987), 137–153.

[16] TSITSIKLIS, J. N., AND BERTSEKAS, D. P. Distributed asynchronous optimal routing in data networks. *IEEE Trans. on Automatic Control 31* (1986), 325–332.

[17] TSITSIKLIS, J. N., BERTSEKAS, D. P., AND ATHANS, M. Distributed asynchronous deterministic and stochastic gradient optimization algorithms. *IEEE Trans. on Automatic Control 31* (1986), 803–812.

[18] ÜRESİN, A., AND DUBOIS, M. Generalized asynchronous iterations. In *CONPAR 86* (September 1986), Springer-Verlag, pp. 272–278.

[19] ÜRESİN, A., AND DUBOIS, M. Sufficient conditions for the convergence of asynchronous iterations. *Parallel Computing 10* (1989), 83–92.