7.7 A MODEL IN WHICH SEVERAL PROCESSORS UPDATE THE SAME VARIABLES

We now present a model of distributed iterative computation in which several processors are allowed to update the same component of the vector being iterated. As the updates of different processors will be generally different, the agreement algorithm of Section 7.3 will be used to reconcile the individual updates. This model will be used in the next section in the context of a stochastic gradient–like algorithm.

We motivate the model of this section by means of a simple example. Let y be a random variable with unknown finite mean m and variance σ^2. Suppose that there are p processors and that the ith processor observes, at times $t = 1, 2, \ldots$, an independent realization (sample value) of y, to be denoted by $y_i(t)$. [For instance, the processors might be generating $y_i(t)$ by Monte Carlo simulation; alternatively, the observations $y_i(t)$ could be sensor data obtained at geographically distributed sensor sites.] The objective of the processors is to estimate the mean of y. At any time $t \geq 1$, each processor can compute the estimate $m_i(t) = \big(\sum_{\tau=1}^{t} y_i(\tau)\big)/t$ based on its own data. Furthermore, the computation of $m_i(t)$ can be carried out recursively by means of the formula

$$m_i(t+1) = m_i(t) + \frac{1}{t+1}\big(y_i(t+1) - m_i(t)\big),$$

initialized with $m_i(0) = 0$. This equation resembles the equation $x(t+1) = x(t) + \gamma s(t)$ of Section 7.5 [cf. Eq. (5.3)]; here $1/(t+1)$ plays the role of a (time–varying and decreasing) stepsize γ and $y_i(t+1) - m_i(t)$ plays the role of $s(t)$.

The variance $E\big[(m_i(t) - m)^2\big]$ of $m_i(t)$ is equal to σ^2/t. A better estimate, with variance $\sigma^2/(tp)$, is obtained if the processors were to exchange their individual estimates at time t and compute their average. This averaging could be performed once and for all at the end of the algorithm. However, it may be desirable that the processors obtain good estimates of m while the generation of new observations $y_i(t)$ is in progress. This necessitates that the averaging of their individual estimates be performed more frequently. One possibility is that at each time t, once the processors obtain the new realizations $y_i(t)$ and compute their new individual estimates, they exchange and average their estimates. In this case, the computation consists of interleaved phases involving computation of $y_i(t)$ and averaging. This is a synchronous algorithm, which may be undesirable if, say, some processors are faster than others or if communication delays are too long.

An alternative approach is to overlap the individual computations and the averaging process. This can be done by letting the processors execute the agreement algorithm of Section 7.3, trying to agree on a common estimate, while they keep obtaining new samples $y_i(t)$, which they immediately incorporate into their estimates. We are faced here with two opposing effects: the agreement algorithm tends to bring the estimates of the processors closer together, whereas the generation of new samples has the potential of increasing the difference of their estimates. The interaction of these two effects substantially complicates the question of asynchronous convergence.

We now proceed to the formal description of the model to be employed. The reader is advised at this point that the notation to be introduced is somewhat different from earlier sections. This is because components will no more be associated with a unique

processor; an additional superscript will be used to specify the processor transmitting a message with the value of some component. The general notational convention to be followed here and in the next section is that superscripts denote processors and subscripts denote components.

We consider an algorithm that iterates on a vector x belonging to the vector space $\Re^n$. Furthermore, we assume that $\Re^n$ is expressed as a Cartesian product of m subspaces of lower dimensions, that is, $\Re^n = \Re^{n_1} \times \cdots \times \Re^{n_m}$, where $n_1 + \cdots + n_m = n$. Accordingly, any vector $x \in \Re^n$ is decomposed as $x = (x_1, \ldots, x_m)$, with x_ℓ belonging to $\Re^{n_\ell}$. We refer to x_ℓ as the ℓth component of x.

Let there be p processors. Each processor i has available at each time t a vector $x^i(t) \in \Re^n$, with components $x^i_\ell(t) \in \Re^{n_\ell}$, $\ell = 1, \ldots, m$. For every component index ℓ, we let $C_\ell \subset \{1, \ldots, p\}$ be the set of processors who are in charge of updating the ℓth component x_ℓ, based on their own measurements or computations. We call C_ℓ the set of *computing processors* for component ℓ and we assume that it is nonempty for each ℓ. If $i \in C_\ell$, we let T^i_ℓ be the set of times at which processor i updates x_ℓ.

For every processor i, any component x_ℓ, and any time t, let $\gamma^i_\ell(t)s^i_\ell(t)$ be the update in x_ℓ due to a computation by processor i. Here $\gamma^i_\ell(t)$ is a positive stepsize and $s^i_\ell(t) \in \Re^{n_\ell}$ is an update direction. We could, without loss of generality, absorb $\gamma^i_\ell(t)$ into $s^i_\ell(t)$; our choice of notation is dictated by the application to be considered in the next section. Naturally, we assume that if $i \notin C_\ell$ or if $t \notin T^i_\ell$, then no computation is performed and $s^i_\ell(t) = 0$.

For every component index ℓ, there is a directed graph $G_\ell = (N, A_\ell)$ used to model the exchange of messages carrying a value of x_ℓ. The node set N is the set $\{1, \ldots, p\}$ of processors and the arc set A_ℓ is the set of all pairs (j, i) such that processor j keeps sending messages x_ℓ to processor i. For $(j, i) \in A_\ell$, we let T^{ij}_ℓ be the set of times that such a message is received by processor i and we assume that this set is infinite. For any $t \in T^{ij}_\ell$, we use $\tau^{ij}_\ell(t)$ to denote the time at which the message x^j_ℓ (received by i at time t) was transmitted by processor j. Thus, the message received by processor i at time t is equal to $x^j_\ell\big(\tau^{ij}_\ell(t)\big)$. We assume that $1 \leq \tau^{ij}_\ell(t) \leq t$. We also use the convention that T^{ii}_ℓ is the set of all nonnegative integers and that $\tau^{ii}_\ell(t) = t$, for all t.

The algorithm is initialized at time 1 with each processor i having a vector $x^i(1) \in \Re^n$ in its memory. Processor i can, at any time t, receive messages $x^j_\ell\big(\tau^{ij}_\ell(t)\big)$ from other processors, incorporate these messages into its memory by forming a convex combination with its own value $x^i_\ell(t)$, and finally incorporate the result $\gamma^i_\ell(t)s^i_\ell(t)$ of its own computations. We thus postulate that for every i and ℓ, the variable $x^i_\ell(t)$ is updated according to the formula

$$x^i_\ell(t+1) = \sum_{\{j \mid t \in T^{ij}_\ell\}} a^{ij}_\ell(t) x^j_\ell\big(\tau^{ij}_\ell(t)\big) + \gamma^i_\ell(t)s^i_\ell(t), \tag{7.1}$$

where the coefficients $a^{ij}_\ell(t)$ are nonnegative scalars satisfying

$$\sum_{\{j \mid t \in T^{ij}_\ell\}} a^{ij}_\ell(t) = 1, \qquad \forall \ell, t, i.$$

Equation (7.1) is the general description of the algorithm but it takes somewhat simpler forms in special cases. For example, if processor i receives no messages x_ℓ^j at time t, then $\{j \mid t \in T_\ell^{ij}\} = \{i\}$ and Eq. (7.1) simplifies to

$$x_\ell^i(t+1) = x_\ell^i(t) + \gamma_\ell^i(t)s_\ell^i(t).$$

Similarly, if $i \notin C_\ell$ or if $t \notin T_\ell^i$, then $s_\ell^i(t) = 0$ and Eq. (7.1) becomes

$$x_\ell^i(t+1) = \sum_{\{j \mid t \in T_\ell^{ij}\}} a_\ell^{ij}(t)x_\ell^j\big(\tau_\ell^{ij}(t)\big). \tag{7.2}$$

Equation (7.2) resembles the equations defining the agreement algorithm of Section 7.3, although it is somewhat more general, due to the dependence of the coefficients $a_\ell^{ij}(t)$ on time. In any case, the iteration (7.2) will be referred to as the *underlying agreement algorithm.* Notice that we essentially have a decoupled set of agreement algorithms, one for each component x_ℓ. Coupling between components can arise, however, in Eq. (7.1) because $s_\ell^i(t)$ can depend on the entire vector $x^i(t)$.

Given our motivation of the model (7.1), we want to ensure that the underlying agreement algorithm (7.2) works properly. That is, we would like the values of the variables x_ℓ^i, possessed by different processors, to converge to a common value in the absence of further computations. This turns out to be the case under some reasonable assumptions.

Example 7.1.

We show that the agreement algorithm of Section 7.3 is a special case of Eq. (7.2). Suppose that for each processor i and for every j who communicates x_ℓ to i [i.e., if $(j,i) \in A_\ell$], the set T_ℓ^{ij} does not depend on j. We are thus assuming that $T_\ell^{ij} = R_\ell^i$ for all j such that $(j,i) \in A_\ell$, where R_ℓ^i is some set. Accordingly, at any time t, either $t \notin R_\ell^i$ and processor i receives no message x_ℓ^j or $t \in R_\ell^i$ and processor i receives simultaneously messages x_ℓ^j from all processors j that are supposed to send such messages. This is not as unrealistic as it sounds. For example, processor i might physically receive these messages at different times, store them in a buffer, and then read them all at the same time t. As far as the mathematical description of the algorithm is concerned, this situation is identical to the case of simultaneous receptions.

We now assume for each $(j,i) \in A_\ell$ and $t \in R_\ell^i$, that the value of the coefficient $a_\ell^{ij}(t)$ is positive and independent of t, and will be denoted by a_ℓ^{ij}. Let us define for convenience $a_\ell^{ij} = 0$ if $j \neq i$ and $(j,i) \notin A_\ell$. Then Eq. (7.2) can be rewritten as

$$x_\ell^i(t+1) = \sum_{j=1}^{p} a_\ell^{ij}x_\ell^j\big(\tau_\ell^{ij}(t)\big), \qquad \text{if } t \in R_\ell^i,$$

$$x_\ell^i(t+1) = x_\ell^i(t), \qquad \text{if } t \notin R_\ell^i.$$

These two equations are identical to those describing the agreement algorithm of Section 7.3 [cf. Eqs. (3.2) and (3.3)], except for somewhat different notation. If we now introduce the partial asynchronism assumption and assume that there exists some i such that $a_\ell^{ii} > 0$ and

such that there exists a positive path (in the graph G_ℓ) from i to all other processors, then the sequence generated by Eq. (7.2) is guaranteed to converge to agreement, geometrically.

Example 7.2.

We now consider an alternative to Example 7.1, in which received messages are immediately incorporated in the memory of the receiving processor. For any i and ℓ and any $j \neq i$ such that $(j,i) \in A_\ell$, we are given a constant $a_\ell^{ij} \in (0,1)$. Furthermore, we assume that each processor i receives at most one message x_ℓ^j at each time unit; equivalently, the sets T_ℓ^{ij}, with $j \neq i$, are disjoint for any fixed i. In practice, physical message receptions could be simultaneous, but a processor can always place incoming messages in a buffer and read them one at a time. Thus, our assumption is not entirely unrealistic. Suppose that for every $t \in T_\ell^{ij}$, the incoming message $x_\ell^j\big(\tau_\ell^{ij}(t)\big)$ is taken into account by processor i by letting

$$x_\ell^i(t+1) = a_\ell^{ij} x_\ell^j\big(\tau_\ell^{ij}(t)\big) + (1 - a_\ell^{ij}) x_\ell^i(t) + \gamma_\ell^i(t) s_\ell^i(t), \qquad t \in T_\ell^{ij}.$$

We also let $x_\ell^i(t+1) = x_\ell^i(t) + \gamma_\ell^i(t) s_\ell^i(t)$ if t does not belong to any T_ℓ^{ij}. It is easily seen that we are dealing with a special case of Eq. (7.1). The underlying agreement algorithm is identical to the one considered in Exercise 3.2 of Section 7.3. According to the result of that exercise, such an agreement algorithm is guaranteed to converge geometrically if the partial asynchronism assumption holds and provided that for some i, there exists a positive path (in the graph G_ℓ) from i to every other processor j.

Examples 7.1 and 7.2 demonstrate that we can realistically assume that the underlying agreement algorithm (7.2) converges geometrically and such an assumption will be introduced shortly. Before doing so, we shall develop an alternative representation that is equivalent to Eq. (7.1) but easier to work with.

Suppose that we have fixed the sets T_ℓ^{ij} and the values of the coefficients $a_\ell^{ij}(t)$ and $\tau_\ell^{ij}(t)$ for all $\ell, i, j, t \in T_\ell^{ij}$. (We refer to any fixed choice of these sets and variables as a *realization* of the underlying agreement algorithm.) Let us now fix some component index ℓ. It is seen from Eq. (7.1) that for any fixed t and i, the value of $x_\ell^i(t)$ is a linear function of the variables $\{x^j(1) \mid j = 1, \ldots, p\}$ and $\{\gamma_\ell^j(\tau) s_\ell^j(\tau) \mid j = 1, \ldots, p,\ \tau = 1, \ldots, t-1\}$. (This is easily proved by induction, using the fact that the composition of two linear functions is linear.) It follows that there exist scalar coefficients $\Phi_\ell^{ij}(t,\tau)$, $t > \tau \geq 0$, such that

$$x_\ell^i(t) = \sum_{j=1}^{p} \Phi_\ell^{ij}(t,0) x_\ell^j(1) + \sum_{\tau=1}^{t-1} \sum_{j=1}^{p} \Phi_\ell^{ij}(t,\tau) \gamma_\ell^j(\tau) s_\ell^j(\tau). \tag{7.3}$$

Equation (7.3) is more explicit than the original Eq. (7.1). On the other hand, the coefficients $\Phi_\ell^{ij}(t,\tau)$ depend on the particular realization of the underlying agreement algorithm and are therefore usually unknown.

Example 7.3.

The best way of visualizing the coefficient $\Phi_\ell^{ij}(t,\tau)$ is by considering the following sequence of events. Let us fix j, ℓ, and τ. For simplicity, suppose that each component is one-dimensional ($n_\ell = 1$) and that $x_\ell^k(1) = 0$ for all k. Suppose that processor j performs a

computation at time τ and obtains $\gamma_\ell^j(\tau)s_\ell^j(\tau) = 1$. Furthermore, suppose that this is the only computation ever to be performed by any processor, that is $s_\ell^k(\sigma) = 0$, unless $k = j$ and $\sigma = \tau$. For such a sequence of events, Eq. (7.3) shows that for $t > \tau$, $\Phi_\ell^{ij}(t,\tau) = x_\ell^i(t)$. In other words, $\Phi_\ell^{ij}(t,\tau)$ is the value (at time t and at processor i) generated by the underlying agreement algorithm, initialized at time $\tau+1$ with $x^j(\tau+1) = 1$, $x^k(\tau+1) = 0$ for $k \neq j$, and with all messages in transit at time $\tau+1$ equal to zero.

There are several conclusions that can be drawn from Example 7.3. First, we must have

$$0 \leq \Phi_\ell^{ij}(t,\tau) \leq 1, \qquad \forall i,j,\ell,t > \tau.$$

Furthermore, if the underlying agreement algorithm is geometrically convergent (as is the case in Examples 7.1 and 7.2), then the coefficients $\Phi_\ell^{ij}(t,\tau)$ converge geometrically, as t tends to infinity, to a limit independent of i. We introduce the notation $\Phi_\ell^j(\tau)$ to denote this common limit. The variable $\Phi_\ell^j(\tau)$ is interpreted as the value of the component x_ℓ on which all processors would agree under the sequence of events specified in Example 7.3.

Finally, let us notice that if i is a computing processor for component ℓ (that is, if $i \in C_\ell$), then it is desirable that each update $\gamma_\ell^i(t)s_\ell^i(t)$ of processor i can influence the value of x_ℓ^j for all other processors j by a nonnegligible amount in the long run. This can be expressed mathematically by requiring that there exists a positive constant η such that $\Phi_\ell^i(\tau) \geq \eta$ for all $i \in C_\ell$ and all $\tau \geq 0$.

Example 7.1. (*cont.*)

Suppose that for every computing processor i for component ℓ (i.e., $i \in C_\ell$), we have $a_\ell^{ii} > 0$ and that there exists a positive path in the graph G_ℓ from processor i to every other processor j. Then processor i is a "distinguished" processor in the terminology of Section 7.3 and part (c) of Prop. 3.1 applies. In particular, if $x_\ell^k(1) = 0$ for every k and ℓ, and if $\gamma_\ell^i(\tau)s_\ell^i(\tau) = 1$ is the only nonzero update in the course of the algorithm (as in Example 7.3), then the value on which the processors will eventually agree is no smaller than the positive constant η of Prop. 3.1(c). We have already shown that the agreed upon value is equal to $\Phi_\ell^i(\tau)$ (Example 7.3). We conclude that the inequality $\Phi_\ell^i(\tau) \geq \eta > 0$ holds for all $\tau \geq 0$. A similar argument can be made for the agreement algorithm of Example 7.2.

We summarize the above discussion in Assumption 7.1 to follow. In particular, we have shown that Assumption 7.1 is satisfied when the underlying agreement algorithm is as in Example 7.1 or Example 7.2, provided that for each component ℓ, there is a communication path from every computing processor to every other processor. This assumption can be also shown to hold for more general choices of the underlying agreement algorithm [Tsi84].

Assumption 7.1. The sets T_ℓ^{ij} and the variables $a_\ell^{ij}(t)$, $\tau_\ell^{ij}(t)$, defining a realization of the underlying agreement algorithm [cf. Eqs. (7.1) and (7.2)], are such that the following hold:

(a) For all i, j, and $t > \tau \geq 0$, we have $0 \leq \Phi_\ell^{ij}(t,\tau) \leq 1$.

(b) For any i, j, and $\tau \geq 0$, the limit of $\Phi_\ell^{ij}(t,\tau)$, as t tends to infinity, exists, is the same for all i, and is denoted by $\Phi_\ell^j(\tau)$.

(c) There exists some $\eta > 0$ (independent of the particular realization) such that $\Phi_\ell^j(\tau) \geq \eta$ for all $j \in C_\ell$ and $\tau \geq 0$.

(d) There exist constants $A > 0$ and $\rho \in (0,1)$ (independent of the particular realization) such that

$$|\Phi_\ell^{ij}(t,\tau) - \Phi_\ell^j(\tau)| \leq A\rho^{t-\tau}, \qquad \forall t > \tau \geq 0.$$

Equation (7.3) has a simple structure, but is still difficult to manipulate when it comes to the analysis of specific algorithms. The reason is that we have one such equation for each ℓ and i, and all of these equations are, in general, coupled. Thus, we need to keep track of all the vectors $x^1(t), \ldots, x^p(t)$, simultaneously. Analysis would be easier if we could associate with the algorithm a single vector $y(t)$ that summarizes the information contained in the vectors $x^i(t)$. It turns out that this is possible and the following choice of $y(t)$ is particularly useful. We define the ℓth component of $y(t)$ by

$$y_\ell(t) = \sum_{i=1}^{p} \Phi_\ell^i(0) x_\ell^i(1) + \sum_{\tau=1}^{t-1} \sum_{i=1}^{p} \Phi_\ell^i(\tau)\gamma_\ell^i(\tau)s_\ell^i(\tau), \qquad \ell = 1, \ldots, m. \tag{7.4}$$

The interpretation of $y(t)$ is quite interesting. Let us fix some time $\bar{t}$ and suppose that $\gamma_\ell^i(\tau)s_\ell^i(\tau) = 0$ for all $\tau \geq \bar{t}$. Consider Eq. (7.3) and take the limit as $t \to \infty$. We only need to keep the summands for $\tau < \bar{t}$ and we are left with the defining expression for $y_\ell(\bar{t})$. In other words, if the processors stop performing any updates at time $\bar{t}$, but keep communicating and forming convex combinations of their states, as in the underlying agreement algorithm, they will asymptotically agree and $y(\bar{t})$ is precisely the vector on which they will agree.

The vector $y(t)$ is a very convenient tool for analysis, primarily because it is generated by the recursion

$$y_\ell(t+1) = y_\ell(t) + \sum_{i=1}^{p} \Phi_\ell^i(t)\gamma_\ell^i(t)s_\ell^i(t), \qquad \ell = 1, \ldots, m, \tag{7.5}$$

which is an immediate consequence of Eq. (7.4)

Example 7.4. *Specialized Computation*

We now show that our earlier model of asynchronous computation in which each processor is associated with a different component (e.g., as in Section 7.5) is a special case of the present model, and we evaluate the coefficients $\Phi_\ell^i(t)$ for this particular context. Let the number m of components be the same as the number p of processors, and let processor j be the only computing processor for component j, that is, $C_j = \{j\}$. Each processor j broadcasts its value of x_j from time to time to all other processors and these values are received after some bounded delay. Processor i, upon receiving at time t a value of x_j that

has been sent at some time $\tau_j^{ij}(t) \leq t$ by some processor $j \neq i$ [thus, processor i receives the message $x_j^j\big(\tau_j^{ij}(t)\big)$], lets

$$x_j^i(t+1) = x_j^j\big(\tau_j^{ij}(t)\big). \tag{7.6}$$

Also, processor i lets

$$x_j^i(t+1) = x_j^i(t), \tag{7.7}$$

if no message is received from processor $j \neq i$ at time t. Equations (7.6) and (7.7) are a trivial special case of an agreement algorithm. Comparing with Eq. (7.1), we have $a_j^{ij}(t) = 1$, if processor i receives a message x_j^j from j at time t. In this example, if processor j stops performing any computations, the values $x_j^i(t)$ possessed by different processors all agree, within finite time. (Finite time convergence can be viewed as a special case of geometric convergence.) Furthermore, the value on which they agree is the value possessed by the computing processor j. Using the interpretation of the coefficients $\Phi_i^j(t)$ provided by Example 7.3, we see that $\Phi_j^j(t) = 1$ for all j and t, and $\Phi_i^j(t) = 0$ if $i \neq j$.

The model we have developed will be used in the next section, in the context of a stochastic gradient algorithm.

7.8 STOCHASTIC GRADIENT ALGORITHMS

Let $F : \Re^n \mapsto \Re$ be a differentiable cost function to be minimized, subject to no constraints. Suppose, however, that we only have access to noisy measurements of the gradient. That is, for any $x \in \Re^n$, we cannot compute $\nabla F(x)$, but we have access to $\nabla F(x) + w$, where w is a random variable representing measurement or observation noise. Such a situation often arises in statistics or in system identification ([RoM51], [Lju77], [LjS83], and [PoT73]).

One could still try to use the gradient algorithm, as if no noise was present, which gives rise to the equation

$$x(t+1) = x(t) - \gamma\Big(\nabla F\big(x(t)\big) + w(t)\Big), \tag{8.1}$$

where $w(t)$ is the noise in the measurement of $\nabla F\big(x(t)\big)$. This algorithm does not converge, in general, to the minimum of F. For example, suppose that for each t, the random variable $w(t)$ is independent of $x(t)$ and has variance $\sigma^2 > 0$. It follows from Eq. (8.1) that the variance of $x(t)$ is at least $\gamma^2\sigma^2$ and, therefore, $x(t)$ fails to converge (in the mean square sense). What may happen, at best, is that $x(t)$ reaches a neighborhood of a (local) minimum x^* of F and moves randomly around x^*. The mean square distance of $x(t)$ from x^* increases with the variance of $x(t)$ and can be made small only if γ is chosen small. On the other hand, if γ is excessively small, it can take a very large number of steps to reach a neighborhood of a local minimum. We can strike a balance