Probability in the Engineering and Informational Sciences, 4, 1990, 493–521. Printed in the U.S.A.

QUICKER CONVERGENCE FOR ITERATIVE NUMERICAL SOLUTIONS TO STOCHASTIC PROBLEMS: PROBABILISTIC INTERPRETATIONS, ORDERING HEURISTICS, AND PARALLEL PROCESSING

ALBERT G. GREENBERG AND ROBERT J. VANDERBEI
AT&T Bell Laboratories, Murray Hill, New Jersey

Gauss–Seidel is a general method for solving a system of equations (possibly nonlinear). It makes repeated sweeps through the variables; within a sweep as each new estimate for a variable is computed, the current estimate for that variable is replaced with the new estimate immediately, instead of on completion of the sweep. The idea is to use new data as soon as it is computed. Gauss–Seidel is often efficient for computing the invariant measure of a Markov chain (especially if the transition matrix is sparse), and for computing the value function in optimal control problems. In many applications the computation can be significantly improved by appropriately ordering the variables within each sweep. A simple heuristic is presented here for computing an ordering that quickens convergence. In parallel processing, several variables must be computed simultaneously, which appears to work against Gauss–Seidel. Simple asynchronous parallel Gauss–Seidel methods are presented here. Experiments indicate that the methods retain the benefit of a good ordering, while further speeding up convergence by a factor of P if P processors participate.

In this paper, we focus on the optimal stopping problem. A probabilistic interpretation of the Gauss–Seidel (and the Jacobi) method for computing the value function is given, which motivates our ordering heuristic. However, the ordering heuristic and parallel processing methods apply in a broader context, in particular, to the important problem of computing the invariant measure of a Markov chain.

 0269-9648/90 $5.00 + .00

1. INTRODUCTION

Iterative methods play an important role in solving systems of equations (possibly nonlinear). The simplest iterative method, called *Jacobi*, makes repeated sweeps through the variables. Within a sweep new estimates for each variable are computed using current estimates, and on completion of the sweep, the current estimates for the variables are replaced with new ones. The *Gauss-Seidel* method is similar, except that as soon as the new estimate of a variable is computed, it replaces the current one, and so immediately influences the computation for the next variable. It is well-known that the Gauss-Seidel method often converges more quickly in theory (e.g., the Stein-Rosenberg theorem [13]), and even more often converges more quickly in practice.

However, in many applications of the Gauss-Seidel method to the solution of stochastic problems, the order in which the variables are updated in a sweep matters: One ordering may lead the computation to converge to the correct result substantially quicker than another. In this paper, we present a simple heuristic, called the *flow deficit method*, for generating an ordering that quickens convergence. The heuristic is used just once, before beginning the sweeps, and is about as expensive in time and memory as one sweep. Roughly, the heuristic consistently permutes the rows and columns of a matrix, shifting most of the mass either above or below (whichever is desired) the diagonal. The motivation for the heuristic springs from simple probabilistic interpretations, presented below, of the Gauss-Seidel and Jacobi methods.

The (slower) Jacobi method is extremely well-suited to parallel processing since the updates to the variables during a sweep are independent. In contrast, Gauss-Seidel seems inherently serial. In this paper, we present two easy-to-program asynchronous parallel Gauss-Seidel methods, which retain the advantage of using Gauss-Seidel with a well-chosen sweep ordering. By *asynchronous*, we mean that the processors need not operate in lockstep; some may operate significantly faster than others. In this setting, processor coordination mechanisms, such as critical sections, substantially degrade the computation's efficiency unless used infrequently. Our first parallel method avoids processor coordination almost entirely. However, it is possible for two processors to both work on updating the same variables at the same time—obviously, a waste. Our second method uses infrequent processor coordination to minimize this possibility. Experiments on a shared-memory parallel processor suggest that the methods are efficient, the second more so, and retain the benefit of a good ordering. The second method appears to solve large problems P times faster than the serial Gauss-Seidel method, if P processors are available.

In this paper, we concentrate on calculating the value function in optimal stopping problems (described below). However, the ordering heuristic and parallel methods are applicable in a much broader context. We report below on the effectiveness of the heuristic and parallel methods with the important problem of calculating the invariant measure, or equilibrium distribution, of a Markov

chain. As shown below, Gauss–Seidel methods for optimal stopping always converge. In contrast, depending on the sweep ordering, Gauss–Seidel methods for invariant measure calculation either eventually converge or oscillate. In the appendix, we show how to adapt our ordering heuristic to construct a sweep ordering for invariant measure calculation that is likely to lead to quicker convergence than alternative orderings (though this is not guaranteed) and is guaranteed to lead to eventual convergence. Recently, we applied the ordering heuristic to van der Wal and Schweitzer's [16] beautiful Gauss–Seidel methods for computing lower and upper bounds for the invariant measure. We found that the relative benefit (in reducing the number of iterations needed to meet a given error bound) derived from the ordering heuristic was roughly the same as reported below for the usual Gauss–Seidel method for invariant measure calculation. This benefit largely recovered the efficiency lost to slack in the bounds.*

The *optimal stopping problem* involves a Markov chain X_t on a finite state space E and a nonnegative real-valued pay-off function f defined on E. X_t starts at some state $x \in E$. A controller eventually stops X_t at some time τ and collects the pay-off $f(X_\tau)$. We consider the problem of finding the expected pay-off under an optimal control:

$$v(x) = E_x f(X_{\tau^*}) = \sup_\tau E_x f(X_\tau), \tag{1.1}$$

where τ is a stopping time [5], i.e., the decision to stop at τ depends only on the state of the process at times earlier than τ. The function v is called the *value function*. Several types of stochastic problems can be formulated and efficiently solved as optimal stopping problems. For example, there is the problem of finding the probability that a Markov chain exits a given domain through a given set. An example of such a problem from queueing theory is given below. More generally, there is the Dirichlet problem associated with a Markov chain and a domain in its state space. Several other problems are closely related, such as the problem of finding the expected time a Markov chain needs to exit a given domain.

An ergodic Markov chain corresponding to a stochastic matrix P has a unique invariant measure, which is a row vector π that solves the following system of equations:

$$\pi = \pi P$$

$$\pi e = 1,$$

where e denotes the vector of all ones. A tremendous variety of stochastic problems reduce to calculating the invariant measure of a Markov chain [14].

As noted above, we will present a probabilistic interpretation of the Jacobi and Gauss–Seidel method for the optimal stopping problem. This interpretation

*In our experiments, slack in the bounds led to roughly twice as many iterations being performed as really necessary to meet a given error bound.

(i) makes clear why Gauss-Seidel out-performs Jacobi and why some Gauss-Seidel sweep orderings out-perform others, (ii) motivates the heuristic for cheap, automatic design of a good sweep ordering, and (iii) motivates parallel methods that retain the benefit of a good (serial) sweep ordering. Probabilistic interpretations of iterative numerical methods are not new. Van Nunen and Wessels [17] presented probabilistic interpretations of the Gauss-Seidel and Jacobi methods for solving a class of linear equations. Mitra and Tsoucas [11] gave a probabilistic interpretation of the Gauss-Seidel method for invariant measure calculation. For the sake of comparison and to make this paper self-contained, the results of Ref. 11 are summarized in the appendix. Goodman and Madras [6] gave probabilistic interpretations for classical Dirichlet problems (the discrete Laplacian on a square domain).*

In his pioneering paper [2], Baudet published the result of experiments with simple, parallel Gauss-Seidel methods, in which each processor cyclically updates a disjoint set of consecutive (in the sweep ordering) coordinates. This works well if processors execute their tasks at comparable speeds. Unfortunately, if the speeds are different (because, for example, of an uneven load distribution), then this method is limited to the speed of the slowest processor. Our two parallel methods, like the chaotic relaxation methods of Chazan and Miranker [4] and Lubachevsky and Mitra [6], are not limited in this way. Our experimental results are in keeping with Baudet's in that we found simple methods using very little processor coordination provide nearly ideal performance. In Section 4, we prove that under the assumption of bounded delays and regular visitation of every state, parallel Gauss-Seidel methods for optimal stopping always converge to the correct result. A similar result was proven by Bertsekas [3].

The paper is organized as follows. In Section 2, we review the Jacobi and Gauss-Seidel methods for solving optimal stopping problems, present their probabilistic interpretations, and then consider several examples. One of the examples, involving a series of queues, is carried throughout the paper and provides our numerical test cases. In Section 3, we present our ordering heuristic and report the results of experiments testing the heuristic on a queueing theory example. In Section 4, we present a model of parallel Gauss-Seidel methods for optimal stopping and prove that correct convergence is guaranteed. We then present two simple asynchronous Gauss-Seidel methods. We implemented and tested the two on a Sequent Balance 21000 multiprocessor. We report the results of experiments run on the Sequent, again for the queueing theory example.

2. PROBABILISTIC INTERPRETATIONS

In this section, we first present the well-known recursive equations for the value function in an optimal stopping problem. Next, we review the Jacobi and

*For classical Dirichlet problems, the Gauss-Seidel method converges at twice the speed of the Jacobi method, irrespective of the ordering. In general, there is no limit on how much faster Gauss-Seidel is than Jacobi.

Gauss–Seidel methods for calculating the value function. We then present simple probabilistic interpretations of the two methods. Last, we present examples showing that the probabilistic interpretation of Gauss–Seidel gives insight into its rate of convergence, and pointing the way to a heuristic to optimize the rate of convergence.

Recall that the optimal stopping problem involves a pay-off function f and a Markov process X_t with transition matrix P and state space E. X_t starts at some state $x \in E$, and if stopped at some time τ, collects the pay-off $f(X_\tau)$. Any decision procedure for τ is allowed, provided only that τ is a *stopping time* (see Dynkin and Yushkevich [5]), meaning that, for any $t = 0,1,2,\ldots$, the decision to stop at time t, i.e. $\{\tau = t\}$, depends only on $X_0, X_1, \ldots, X_t$. The value function $v(x)$ denotes the expected pay-off if X_t is stopped optimally [see Eq. (1.1)]. When at state x, we could stop X_t immediately and obtain pay-off $f(x)$. On the other hand, we could decide to take one step and then proceed optimally from there, to obtain pay-off $Pv(x)$. By the principle of dynamic programming (see Dynkin and Yushkevich [5]), these two possibilities are sufficient. That is, $v(x)$ is the smallest solution to the following nonlinear system of equations:

$$v(x) = \max\{f(x), Pv(x)\}. \tag{2.1}$$

The optimal strategy is described in terms of the support set of v:

$$\Gamma = \{x : v(x) = f(x)\}.$$

Indeed, τ^* is the first hitting time of Γ:

$$\tau^* = \inf\{t : X_t \in \Gamma\}.$$

In the Jacobi method, a sequence of estimates of the value function $v(x)$ is computed:

$$v_0(x) = f(x) \tag{2.2.a}$$

$$v_t(x) = \max\{v_{t-1}(x), Pv_{t-1}(x)\}, \qquad t = 1,2,3,\ldots. \tag{2.2.b}$$

It follows from the analysis below (or [5]) that $v_t(x)$ increases monotonically to $v(x)$ as t tends to infinity, so recursion (2.2.b) is equivalent to one in which $f(x)$ replaces the first occurrence of $v_{t-1}(x)$ inside the maximum. To implement the Jacobi method, two copies of the vector v must be kept in store, an old and a new copy. The new copy is calculated using the old copy. When the calculation is complete, the new copy becomes the old copy for the next iteration.

In the Gauss–Seidel method, just one vector is held in store. An order relation $<$ is imposed on the state space E. In general, this relation has *no bearing* on the original problem. An iteration consists of a pass over the states x following the ordering, computing new estimates of $v(x)$, and overwriting the single vector with those estimates. More precisely, given an order relation $<$ on E, split the transition matrix P into its strict lower part and the remaining upper part, so that for any function h defined on E,

$$Lh(x) = \sum_{y<x} p(x,y)h(y),$$

$$Uh(x) = \sum_{y\geq x} p(x,y)h(y).$$

The Gauss-Seidel method proceeds as follows:

$$w_0(x) = f(x) \tag{2.3.a}$$

$$w_t(x) = \max\left\{ w_{t-1}(x), \sum_{y<x} p(x,y)w_t(y) + \sum_{y\geq x} p(x,y)w_{t-1}(y) \right\}$$

$$= \max\{ w_{t-1}(x), Lw_t(x) + Uw_{t-1}(x) \}. \tag{2.3.b}$$

It follows from Theorems 1 and 2 below that

$$v_t(x) \leq w_t(x) \leq v(x).$$

Hence, for optimal stopping, the Gauss-Seidel method is always better than the Jacobi method, both in speed of convergence and in storage requirement.

Now, we give the two methods probabilistic interpretations. First, recall (1.1). The value function is given by

$$v(x) = \sup_{\tau} E_x f(X_\tau),$$

where τ denotes a stopping time. As a warm-up, we shall prove the following (well-known) result for the tth Jacobi iterate. This proof is included because it and the proof of Theorem 2 below show graphically (at a glance) why the Jacobi method is slower than the Gauss-Seidel approach.

THEOREM 1:

$$v_t(x) = \sup_{\tau\leq t} E_x f(X_\tau).$$

Let us try to motivate the analogous result for the Gauss-Seidel method. While motivating, we ignore the nonlinearity created by the maximization in (2.3.b) and make believe that we can simply subtract $LW_t(x)$ from both sides to isolate $w_t(x)$ on the left-hand side. Then we multiply by $(I-L)^{-1}$ to solve for $w_t(x)$ in terms of $w_{t-1}(x)$. Still ignoring nonlinearities, we would end up with the matrix $(I-L)^{-1}U$ acting on $w_{t-1}(x)$. This matrix is a stochastic matrix. It is the transition matrix for the Markov chain obtained from X_t by observing it only at times at which it takes a high step: a step from a state x to a state $y \geq x$. This embedded Markov chain will play a key role in our study of the Gauss-Seidel method. A program similar to this one is carried out in Mitra and Tsoucas [11] and Goodman and Madras [6] where there are no nonlinearities with which to contend. Here, however, the problem appears on first blush to be more subtle. The nonlinearities can however be overcome and a simple result obtained.

Let

$$\sigma_t = \inf\{s : s > \sigma_{t-1}, X_s \geq X_{s-1}\} \tag{2.4.a}$$

and

$$\sigma_0 = 0. \tag{2.4.b}$$

In words, σ_t represents the time just after the tth high step. Using σ_t, we prove a result similar to Theorem 1 for the tth Gauss-Seidel iterate.

THEOREM 2:

$$w_t(x) = \sup_{\tau \leq \sigma_t} E_x f(X_\tau).$$

The most important property of σ_t is that

$$\sigma_t \geq t.$$

Theorem 1 says the Jacobi iterate v_t is itself the value function for a finite horizon counterpart to the original optimal stopping problem where the Markov process must stop no later than time t. Similarly, Theorem 2 says that the Gauss-Seidel iterate w_t is the value function for a finite horizon counterpart where the Markov process must stop no later than the random time σ_t. Since $\sigma_t \geq t$, these two theorems immediately establish that v_t and w_t are monotonically increasing with $v_t \leq w_t$ and limit v.

PROOF OF THEOREM 1: Consider the Markov process Y_s, whose state space is $\{0,1,2,\ldots\} \times E$ and whose transitions are described in Figure 1. Let us refer

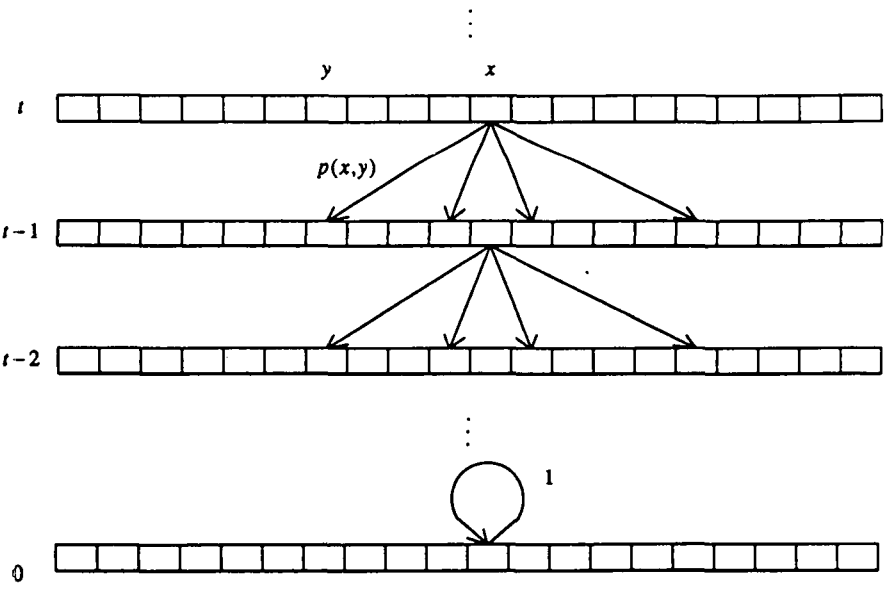

FIGURE 1. Transitions for Y_s.

to the first component t of a state (t,x) as the level and the second component $x \in E$ as the position. As Figure 1 shows, the level of Y_s is simply decremented, by one at each step, until Y_s hits level 0 and is absorbed. The position of Y_s is statistically identical to X_s up until Y_s hits level 0.

Now, define an optimal stopping problem for Y_s by repeating f on every level: the pay-off $\bar{f}(t,x)$ at each state (t,x) is $f(x)$. The transition matrix Q of Y_s acts on functions $h_t(x)$ of level t and position x according to

$$Qh_t(x) = \begin{cases} Ph_{t-1}(x) & \text{if } t > 0 \\ h_0(x) & \text{if } t = 0. \end{cases}$$

Let $\bar{v}_t(x)$ denote the corresponding value function at state (t,x). Since level 0 is absorbing,

$$\bar{v}_t(x) = \sup_{\tau} E_{(t,x)} \bar{f}(Y_\tau) = \sup_{\tau \le t} E_{(t,x)} \bar{f}(Y_\tau) = \sup_{\tau \le t} E_x f(X_\tau).$$

Applying the principle of dynamic programming to this finite horizon problem, we see that

$$\begin{aligned} \bar{v}_t(x) &= \max\{f(x), P\bar{v}_{t-1}(x)\}, \qquad t = 1,2,3,\ldots \\ \bar{v}_0(x) &= f(x). \end{aligned} \tag{2.5}$$

These two equations uniquely determine $\bar{v}_t(x)$ for all t and x. To finish the proof, we must show that $\bar{v}_t$ coincides with v_t. It is clearly true for $t = 0$ and $t = 1$. Proceeding inductively, suppose it is true for all level indices less than t. Starting from the definition of v_t given in (2.2), we have

$$v_t = \max\{v_{t-1}, Pv_{t-1}\}.$$

From the induction hypothesis, we then get

$$v_t = \max\{\bar{v}_{t-1}, P\bar{v}_{t-1}\},$$

and using (2.5) we can replace $\bar{v}_{t-1}$ to yield

$$v_t = \max\{f, P\bar{v}_{t-2}, P\bar{v}_{t-1}\}.$$

Now, we know that $v_{t-2} \le v_{t-1}$ and so again using our induction hypothesis, we see that $\bar{v}_{t-2} \le \bar{v}_{t-1}$. Hence, the middle quantity in the maximization above can be removed and the right-hand side becomes $\bar{v}_t$. ■

Proof of Theorem 2: Corresponding to the Gauss-Seidel method there is a slightly different stochastic process Z_s on $\{0,1,2,\ldots\} \times E$. Again, the position of Z_s is statistically identical to the original Markov process, but now the level index evolves differently. Let us refer to a jump from position x to position y as a *(strict) low step* if $y < x$ and as a *high step* if $y \ge x$, with respect to the ordering $<$ of the updates in the Gauss-Seidel method. In Figure 2, low steps are

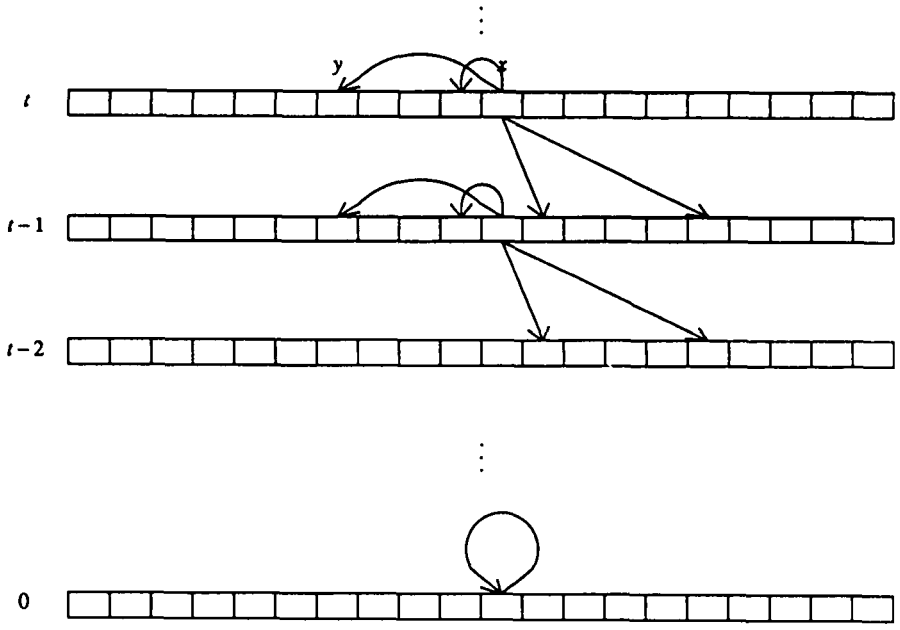

FIGURE 2. Transitions for the Gauss-Seidel process.

to the left and high steps are to the right. Until hitting level 0 where Z_s is absorbed, if Z_s takes a low step, then the level is unchanged, whereas if Z_s takes a high step, then the level is decremented by one.

Again, define an optimal stopping problem by repeating f on every level. Let $\bar{f}$ denote the repeated pay-off function. The transition matrix Q of Z_s acts on functions $h_t(x)$ according to

$$Qh_t(x) = \begin{cases} Lh_t(x) + Uh_{t-1}(x) & \text{if } t > 0 \\ h_0(x) & \text{if } t = 0. \end{cases}$$

Let $\bar{w}_t(x)$ denote the value function at state (t,x). Since level zero is absorbing and is first reached at time σ_t, it follows that

$$\bar{w}_t(x) = \sup_{\tau} E_{(t,x)}\bar{f}(Z_\tau) = \sup_{\tau \le \sigma_t} E_{(t,x)}\bar{f}(Z_\tau) = \sup_{\tau \le \sigma_t} E_x f(X_\tau).$$

Applying the principle of dynamic programming to this finite horizon problem, we see that

$$\begin{aligned} &\bar{w}_t(x) = \max\{f(x), L\bar{w}_t(x) + U\bar{w}_{t-1}(x)\}, \qquad t = 1,2,3,\ldots. \\ &\bar{w}_0(x) = f(x). \end{aligned} \tag{2.6}$$

These two equations uniquely determine $\bar{w}_t(x)$ for all t and x. To finish the proof, we must show that $\bar{w}_t$ coincides with w_t. It is clearly true for $t = 0$ and $t = 1$. Proceeding inductively, suppose it is true for all level indices less than t and, on level t, it is true for all $y < x$. Starting from the definition of $w_t(x)$ given in (2.3), we have

$$w_t(x) = \max\{w_{t-1}(x), Lw_t(x) + Uw_{t-1}(x)\}.$$

From the induction hypothesis, we then get

$$w_t(x) = \max\{\bar{w}_{t-1}(x), L\bar{w}_t(x) + U\bar{w}_{t-1}(x)\},$$

and using (2.6), we can replace $\bar{w}_{t-1}(x)$ to yield

$$w_t(x) = \max\{f(x)L\bar{w}_{t-1}(x) + U\bar{w}_{t-2}(x), L\bar{w}_t(x) + U\bar{w}_{t-1}(x)\}.$$

Now, we know that $w_{t-2} \le w_{t-1}$ and $w_{t-1} \le w_t$. Therefore, again using our induction hypothesis, we see that $\bar{w}_{t-2} \le \bar{w}_{t-1}$ and $\bar{w}_{t-1}(y) \le \bar{w}_t(y)$ for $y < x$. Hence, the middle quantity in the maximization above can be removed and the right-hand side becomes $\bar{w}_t(x)$. ■

Theorem 2 shows that the Gauss-Seidel method for calculating the value function converges fastest if the Markov process X_t is likely to take low steps, so $\sigma_{t+1} - \sigma_t$ is likely to be large. For some problems, all orderings are about the same in that $\sigma_{t+1} - \sigma_t$ is nearly constant, independent of the ordering. On the other hand, there are natural problems, in particular from queueing theory, in which the ordering matters.

Example 1 (*Uniform motion*): Let X_t be uniform motion to the left on the interval $E = \{0, 1, \ldots, n\}$ with absorption at zero. (For $i > 0$, $i \to i - 1$ with probability 1.) Assume that the pay-off function is one at zero and zero elsewhere. Clearly, the optimal strategy is to stop when X_t gets absorbed at zero, so the value function is identically one. The initial estimate of the value function is one at state zero and zero elsewhere. In the Jacobi method, each iteration changes one of the zeros to a one, so the procedure converges in n iterations. In the Gauss-Seidel method under the default ordering, the procedure converges in only one iteration. However, if we reverse the ordering so that n becomes the smallest state and zero becomes the largest, then the Gauss-Seidel method takes n iterations.

The probabilistic interpretation is quite simple. In the default order, σ_1 is the first hitting time of zero, which is the optimal stopping time τ^*. In the reverse order, $\sigma_t = \min\{\tau^*, t\}$, so n iterations are required since τ^* equals n if X_t starts at $x = n$.

Example 2 (*Queues in Series*): An infinite buffer of jobs at station 0 feeds station 1, and station 2 drains into an infinite buffer (see Figure 3). Station i has the capacity to buffer up to N_i jobs, $i = 1,2$. Service times at station i, $i = 0,1,2$, are independent and exponentially distributed with mean $1/\mu_i$. Let n_i denote the

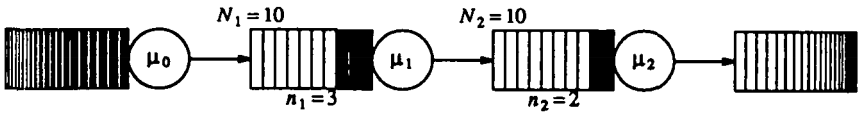

FIGURE 3. Queues in Series.

number of jobs in station i. Station i starts to service a job if and only if a job is present in its buffer ($n_i > 0$) and the next buffer is not full ($n_{i+1} < N_{i+1}$).

The state of the system (n_1, n_2) evolves as a continuous-time Markov chain. There are up to three transitions out of a state:

$$\begin{aligned}
(n_1, n_2) &\to (n_1 + 1, n_2) && \text{at rate } \mu_0 && \text{if } n_1 < N_1 \\
&\quad (n_1 - 1, n_2 + 1) && \text{at rate } \mu_1 && \text{if } n_1 > 0 \quad \text{and} \quad n_2 < N_2 \\
&\quad (n_1, n_2 - 1) && \text{at rate } \mu_2 && \text{if } n_2 > 0.
\end{aligned}$$

The first transition is associated with the arrival of a new job to station 1, the second with a transfer of a job from station 1 to station 2, and the third with a departure of a job from station 2.

If we start the system with both buffers partially loaded, we might want to know the probability that the inventory reaches a given level L, $n_1 + n_2 = L$, before the system reaches another level L', $n_1 + n_2 = L'$. As mentioned in the introduction, this can be formulated as an optimal stopping problem. It suffices to consider the discrete-time Markov chain embedded at the times where the state changes and altered so that states H_L and $H_{L'}$ are absorbing, where $H_K = \{(n_1, n_2) : n_1 + n_2 = K\}$. The pay-off function is one on H_L and zero elsewhere. Hence, the value function at (n_1, n_2) is just the probability that the system first hits H_L rather than $H_{L'}$, given that the system starts at (n_1, n_2).

For the sake of discussion, suppose that all station service rates μ_i are equal, the case studied by Mitra and Tsoucas [11]. Of the large number of choices of orders one might impose on the state space, the sweeping ones are most natural. These are the ones that sweep across successive rows (or columns). We must decide whether to sweep rows then columns or columns, then rows. Once this is decided, we must then specify whether we sweep from low to high or from high to low. Hence, there are a total of eight sweeping orderings. Each of the eight orderings can be put into one of two categories. First there are those for which two of the three transitions from each state are low steps and those for which two of the three are high steps. The *lexicographic ordering*,

$$(0,0) < (0,1) < \cdots < (0,N_2) < (1,0) < (1,1) < \cdots < (1,N_2) < \cdots,$$

is in the first category, and the *antilexicographic ordering*,

$$(0,0) < (1,0) < \cdots < (N_1,0) < (0,1) < (1,1) < \cdots < (N_1,1) < \cdots,$$

is in the second. Our probabilistic interpretation suggests that if the density of transitions corresponding to low steps is d, then the Gauss–Seidel method should converge at rate $1/(1-d)$ times that of the Jacobi method. Thus, under the lexicographic ordering, the Gauss–Seidel computation should need half as many iterations to converge as needed under the antilexicographic ordering, and two-thirds as many iterations as needed under an ordering chosen uniformly at random. This has been verified experimentally. In the two sections to follow, we report the results of several numerical experiments in solving this example.

We know of two classes of problems in which the ordering does not matter: highly symmetric random walks and highly asymmetric random walks where a large deviation (rare event) makes a significant contribution to the optimal stopping strategy.

Example 3 (*Dirichlet Problem*): Let X_t be a simple random walk on the d-dimensional lattice Z^d with absorption as soon as it leaves a fixed domain $D \subset Z^d$. Suppose that the pay-off function f is zero in the domain D and nonnegative on its complement. Clearly, the optimal stopping time τ^* is the first exit time from D:

$$v(x) = E_x f(X_{\tau^*}). \tag{2.6}$$

This is the well-known formula for the solution of the Dirichlet problem. That is, v is the unique function with the following properties:

$$Av(x) = 0, \qquad x \in D,$$

$$v(x) = f(x), \qquad x \in D^c,$$

where A is the discrete Laplacian:

$$Ah(x) = \frac{1}{2d} \sum_{y \in \partial(x)} h(y) - h(x).$$

[The notation $\partial(x)$ represents the $2d$ neighbors of x.] Intuitively, since any linear ordering of the states will have the property that half of the steps will be high steps and the other half will be low steps, it follows again that Gauss–Seidel should be superior to the Jacobi method by about a factor of two. Again, this is experimentally verifiable. In fact, this behavior is well known (see Goodman and Madras [15]). These authors give a simple proof using probabilistic interpretations of the Jacobi and Gauss–Seidel methods similar to those presented above.

Example 4 (*Simple One-Dimensional Random Walk*): Let X_t be a random walk on the interval $E = \{0,1,\ldots,n\}$ with zero a reflecting boundary and n an absorbing one. When at position i $(1 < i < n)$, the walker steps to $i = 1$ with probability $1 - \epsilon$ and to $i + 1$ with probability ϵ. Assume the pay-off is one at position n and zero elsewhere. Clearly, the optimal strategy is to let X_t run until it hits n and collects pay-off one. As ϵ tends to zero, the expected time to hit n from any state $i < n$ tends to e^{cn}, where c is a constant that depends on ϵ. It

follows from our probabilistic interpretation that the number of Gauss-Seidel iterations needed for convergence is of the same order. If started at $i < n$, then with overwhelming probability X_t marches to zero, sticks to the neighborhood of zero for about e^{cn} time, and then marches to n. Thus, irrespective of the ordering, the overwhelming majority of steps are high steps from $X_t = 0$ to $X_{t+1} = 0$. The ordering is essentially irrelevant.

3. A HEURISTIC FOR ORDERING THE STATE SPACE

In this section, we present a heuristic for generating a state space ordering under which the underlying Markov process is likely to take strict low steps. Such orderings quicken the convergence of the Gauss-Seidel method for calculating the value function in optimal stopping.

This method also applies to the Gauss-Seidel method for calculating the invariant measure of a Markov chain. Mitra and Tsoucas [11] showed that orderings under which the chain is likely to take strict high steps quicken convergence. It is trivial to modify our heuristic to generate orderings when strict high steps are likely. Mitra and Tsoucas [11] also gave a sufficient condition (described in the appendix) to guarantee convergence of the method without underrelaxation.* Another simple modification to flow deficit (also described in the appendix) gives orderings that (i) make strict high steps likely and (ii) satisfy the Mitra and Tsoucas condition for convergence without underrelaxation.

Flow deficit is a greedy algorithm (see Aho et al. [1]). Initially, all states belong to a candidate set C. A numerical *merit* is associated with each state in C. At step i of the algorithm, a state x of maximal merit is selected and given position i in the ordering. Before going on to step $i + 1$, state x is removed from C, perhaps changing the merits of other states of C.

Let $p(x,y)$ denote the numerical weight associated with the transition from state x to state y; if x and y are the states of a continuous- (discrete-) time Markov chain, then $p(x,y)$ is the rate (probability) associated with the transition from x to y. We consider two possible assignments of merits to the states x in C:

inflow-outflow:

$$\text{merit}(x) = \sum_{y \in C} p(y,x) - \sum_{y \in C} p(x,y).$$

outflow-inflow:

$$\text{merit}(x) = \sum_{y \in C} p(x,y) - \sum_{y \in C} p(y,x).$$

*Roughly, the underrelaxed computation stores at the next update to a given coordinate a weighted average of the newly calculated value and the value of the last update (see Varga [15]). Underrelaxation guarantees the convergence of the Jacobi and Gauss-Seidel methods for calculating the invariant measure of an irreducible Markov chain. However, in our experiments and those reported by Mitra and Tsoucas [11], underrelaxation uniformly slowed convergence.

To compute the value function in an optimal stopping problem, we want an ordering under which the associated random walk is likely to take strict low steps (jumps to states lower in the ordering). In this case, inflow-outflow is the appropriate measure of merit. Intuitively, for a given state x, if the measure inflow-outflow is large, then state x is locally attracting and so should be low in order. To compute the invariant measure of a Markov chain, we want an ordering under which the associated random walk is likely to take strict high steps (jumps to states higher in the ordering) (see Mitra and Tsoucas [11]). In that case, outflow-inflow is the appropriate measure of merit. Intuitively, this puts locally repelling states low in order.

Let us reconsider Example 1 (uniform motion), an optimal stopping problem in which the state space $E = \{0,1,\ldots,n\}$, the transition from state i is deterministic to state $i-1$ for $i = 1,2,\ldots,n$, and state 0 is absorbing. Initially, inflow-outflow is 1 for state 0, 0 for states 2 through $n-1$, and -1 for state n. Hence, state 0 is selected, assigned the first position in the ordering, and removed, changing inflow-outflow to 1 for state 1, 0 for states 2 through $n-1$, and -1 for state n. State 1 is then selected, assigned the second position in the ordering, and removed, changing inflow-outflow to 1 for state 2, 0 for states 3 through $n-1$, and -1 for state n, and so forth. Thus, the method generates the ordering $0,1,\ldots,n$, which, as we saw in the previous section, is optimal.

Consider Example 2 (queues in series). For concreteness, let the two buffer sizes $N_1 = N_2 = 2$ and the rates $\mu_0 = \mu_1 = \mu_2 = 1$. Take inflow-outflow as the measure of merit. Figure 4 illustrates the first four steps of the flow deficit

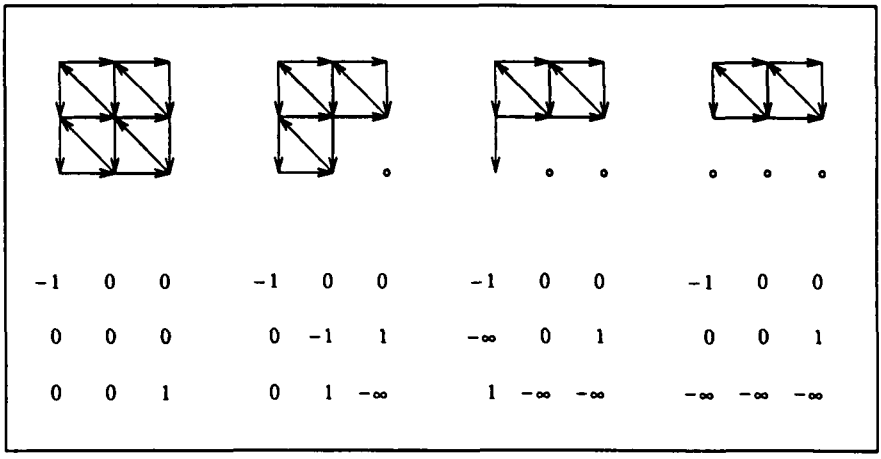

FIGURE 4. Example: Two-queue problem with both buffer capacities 2 and all rates equal to 1. This illustrates the first four steps of the flow deficit method, with inflow-outflow as the measure of merit. The top row shows the state space and the bottom the merits.

method. The top row shows the state space (n_1 varies from 0 to 2 horizontally, and n_2 varies from 0 to 2 vertically) and the bottom the merits. The inflow-outflow of state (n_1, n_2) is just the number of transitions into that state, minus the number of transitions out of that state. Initially, state (2,0) has the greatest merit, 1, as shown in the first column of Figure 4. Hence, (2,0) is selected and removed, changing the merits of states (1,0), (2,1), and (1,1) to 1, 1, and -1, respectively, as shown in the second column of Figure 4. As a result, at the second step of the algorithm, there is a tie: Both (1,0) and (1,1) have maximal merit. Here we choose (1,0), remove it, and update the merits of its neighbors as shown in the third column of Figure 4. At the next step, (0,0) is chosen, leading to the situation depicted in the fourth column. If we continue to break ties in a similar fashion, we are led to a simple scan, which like the lexicographic ordering, is such that two out of the three transitions from an interior state are low steps:

$$(N_1,0) < (N_1 - 1,0) < \cdots < (0,0) < (N_1,1) < (N_1 - 1,1) < \cdots < (0,1) < \cdots.$$

As noted above, orderings of this type lead to fast convergence.

Some remarks on computational costs are in order. Let M denote the total number of states and Γ the number of nonzero transitions between states ($M \leq \Gamma \leq M^2$). The cost of one iteration of the Gauss–Seidel method is $O(\Gamma)$. In our implementation, the key data structure is a *heap* (see Aho et al. [1]) which along with a small number of auxiliary data structures keeps the members of the candidate set C partially ordered by their merits. At each step of the algorithm, we remove a state x from C at cost $O(\log M)$ and then update the merits of all states in C with nonzero transitions either into or out of x, at cost $O(\log M)$ for each such transition. The total cost of the algorithm is $O(\Gamma \log M)$. This cost is paid just once, as a preprocessing step before starting the Gauss–Seidel iterations, and is typically negligible compared to the total cost of the Gauss–Seidel iterations.

To conclude this section, we report the results of some experiments testing the flow deficit method on Example 2, the queues in series. The tests involved the hitting probability calculation (a special case of optimal stopping) described in the previous section and invariant measure calculation.*

Of course, it is necessary to measure the distance between successive iterates u and v in the computation. We took

$$d(u,v) = \max_i \left| \frac{(u_i - v_i)}{v_i} \right|,$$

*In the invariant measure calculations, all orderings tested converged without underrelaxation. Indeed, we found underrelaxation always slowed convergence. Similar observations were reported by Mitra and Tsoucas [11]. We did not try the modification to the flow deficit heuristic that gives a theoretical guarantee of convergence without underrelaxation. That modification would alter the cost of generating the ordering only slightly and, we believe, alter the number of iterations to achieve convergence only slightly.

as the measure of distance. The iterations were stopped at the first i where the distance between the $(i-1)$st and ith iterates was found to be less than 10^{-6}.

Table 1 shows the results for the problem of queues in series with buffer sizes $N_1 = N_2 = 100$, hitting set levels $L = 100$ and $L' = 0$, and

rates $\mu_0 = 1$, $\mu_1 = 1$, $\mu_2 = 1$, and

rates $\mu_0 = 4$, $\mu_1 = 1$, $\mu_2 = 4$.

Table 1 shows the number of iterations needed under

an ordering generated automatically by our code for the flow deficit method (using inflow-outflow for hitting probability calculation and outflow-inflow for invariant measure calculation),

either lexicographic ordering (hitting probability calculation) or antilexicographic ordering (invariant measure calculation), and

an ordering chosen uniformly at random.

For hitting probability calculation, there is little to choose between the efficiency of the flow deficit ordering and the lexicographic ordering for the balanced case in which all $\mu_i = 1$. As expected, about 3/2 times as many iterations are required under random ordering. The corresponding results for invariant measure calculation are similar. The lexicographic and antilexicographic order-

Table 1. Number of Gauss-Seidel iterations needed for the problem of two queues in series with both rates 1 and both buffer sizes N

Hitting Probability Calculations

Parameters						Number of Iterations		
μ_0	μ_1	μ_2	N	L	L'	Flow Deficit	Lex.	Random
1	1	1	100	100	0	10,907	10,941	15,264
4	1	4	100	100	0	85	184	224

Invariant Measure Calculations

Parameters				Number of Iterations		
μ_0	μ_1	μ_2	N	Flow Deficit	Antilex.	Random
1	1	1	100	5716	5858	8229
4	1	4	100	322	408	521

ings were designed for the balanced case. These orderings to not do well in the unbalanced case in which $\mu_0 = \mu_2 = 4$ and $\mu_1 = 1$. This illustrates one difficulty with designing orderings by hand: An ordering good for one set of parameters is not necessarily good for another.

These results show that for some natural examples, the flow deficit method generates good orderings. We admit that examples exist in which the method generates orderings worse than those designed by hand.

4. PARALLEL GAUSS-SEIDEL

In Section 4.1 we present a model for parallel Gauss-Seidel methods for optimal stopping. Although our notation differs, our model is essentially the same as that first studied by Chazan and Miranker [4] to solve systems of equations. We show that under a mild no-starvation condition all such methods converge to the correct result. Lubachevsky and Mitra [10] established a similar result for parallel methods to compute the invariant measure of a Markov chain. In Section 4.2 we present two simple asynchronous parallel methods. The experiments in optimal stopping and invariant measure calculation described below suggest both methods are efficient, one especially so. Those readers willing to accept the result that parallel Gauss-Seidel works are invited to skip to Section 4.2.

For parallel Gauss-Seidel methods, we do not know of a simple probabilistic interpretation of the computation like that given in Section 2 for sequential Gauss-Seidel methods. Finding such an interpretation is a challenging open problem.

4.1. A Model of Parallel Methods

In the sequential (one-processor) Gauss-Seidel method, one processor cycles through the state space E, at each state $x \in E$ updating the estimate of the value function $v(x)$. If several processors are available, then several updates may be performed concurrently. Since processors may experience variable lags in accessing memory, the data used in an update for state x may be stale, that is, the data may be the results of much earlier updates. It is crucial to model these lag times.

Given any pattern of reads and writes to memory, some of which may be simultaneous, it is always possible to find a serialization of those reads and writes that leaves the computation unchanged. For theoretical purposes, the serialization is easier to deal with than the original pattern of accesses. Thus, we shall define a parallel Gauss-Seidel method in terms of a sequence of updates and corresponding sequences of lag times.

More precisely, the following sequences determine a parallel method: an update sequence

$$a_1, a_2, a_3, \ldots,$$

where each a_i is a member of the state space E, and for each $x \in E$, a sequence of lag times

$$s_1(x), s_2(x), s_3(x), \ldots,$$

where each $s_k(x)$ satisfies $0 \le s_k(x) \le k-1$. At update time k, the next estimate of $v(a_k)$ is computed using, for each state x, the estimate of $v(x)$ available at an earlier time $s_k(x)$. To streamline the notation, we define the selection operator S_k on functions $h_t(x)$ by

$$S_k h(x) = h_{S_k(x)}(x),$$

that is, S_k picks out the values determined by the lag times. The parallel method proceeds as follows:

$$w_0(x) = f(x).$$

$$w_k(x) = \begin{cases} \max\{S_k w(x), PS_k w(x)\}, & x = a_k, \\ w_{k-1}(x), & x \ne a_k, \end{cases} \qquad k = 1,2,\ldots. \tag{4.1}$$

The operator S_k in (4.1) accounts for the lags in accessing the data used in the kth update (see Figure 5).

To analyze the behavior of the algorithm, we need to identify special times that mark the completion of a sweep of the state space. Let $\partial(x) = \{y : p(x,y) \ne 0\}$, the members of E reachable in one transition from state x. Let

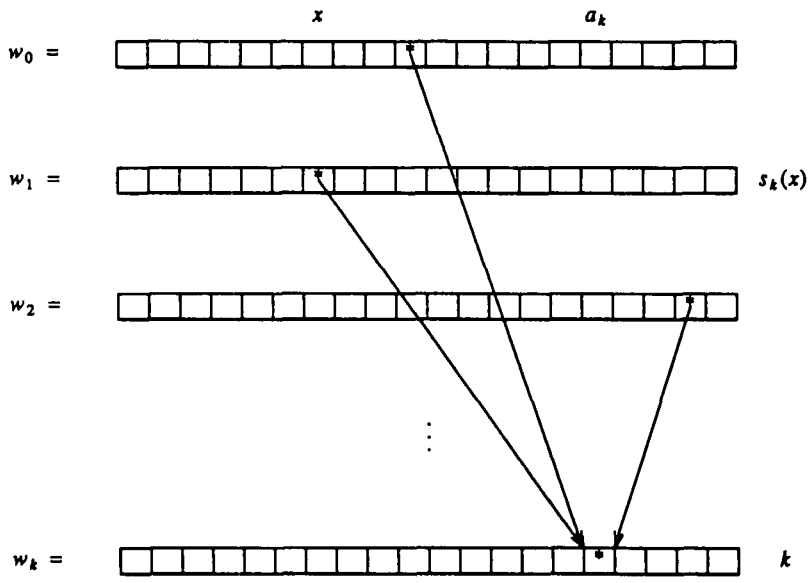

FIGURE 5. Lag times.

$$t_0(x) = 0 \qquad \text{for all } x,$$

$$t_k(x) = \begin{cases} \min\{S_k t(y) : y \in \partial(x) \cup x\} + 1 & x = a_k, \\ t_{k-1}(x) & x \neq a_k, \end{cases} \qquad k = 1,2,\ldots,$$

$$t_k = \min_x t_k(x), \qquad k = 1,2,\ldots.$$

Hence, t_k represents the total number of complete sweeps that have taken place up to the serialized instant k.

Our main analytical result is the following theorem.

THEOREM 3: $v_{t_k}(x) \leq w_k(x) \leq v(x)$.

PROOF: We prove by induction on k that

$$v_{t_k(x)}(x) \leq w_k(x) \leq v(x),$$

which suffices since the Jacobi iterate v_k is monotonically increasing and $t_k \leq t_k(x)$. For $k = 0$, we have $t_0(x) = 0$ and so

$$v_0(x) = f(x) = w_0(x) \leq v(x).$$

Now suppose that

$$v_{t_j(x)}(x) \leq w_j(x) \leq v(x) \qquad \text{for all } j < k,\ x \in E. \tag{4.2}$$

Then, for $x \neq a_k$, we have $t_k(x) = t_{k-1}(x)$ and $w_k(x) = w_{k-1}(x)$. Hence,

$$v_{t_k(x)}(x) = v_{t_{k-1}(x)}(x) \leq w_{k-1}(x) = w_k(x) = w_{k-1}(x) \leq v(x),$$

where the inequalities follow from the induction hypothesis (4.2). For $x = a_k$, note that

$$t_k(x) \leq S_k t(y) + 1 \qquad \text{for all } y \in \partial(x) \cup x. \tag{4.3}$$

Hence,

$$\begin{aligned} v_{t_k(x)}(x) &= \max\left[v_{t_k(x)-1}(x), \sum_{y\in\partial(x)} p(x,y) v_{t_k(x)-1}(y)\right] \\ &\leq \max\left[v_{S_k t(x)}(x), \sum_{y\in\partial(x)} p(x,y) v_{S_k t(y)}(y)\right] \\ &\leq \max\left[w_{S_k(x)}(x), \sum_{y\in\partial(x)} p(x,y) w_{S_k(y)}(y)\right] \\ &= w_k(x) \\ &= \max[S_k w(x), P S_k w(x)] \\ &\leq \max[v(x), P v(x)] \\ &= v(x), \end{aligned}$$

where the first inequality follows from (4.3) and the other two inequalities follow from (4.2). This completes the proof. ■

To guarantee convergence, we require only that $\lim_{k\to\infty} t_k = \infty$. In the next theorem, we give a sufficient condition for this to occur. First, we need some definitions.

DEFINITION: *We say that the* delays are uniformly bounded *if there exists an integer d such that*

$$S_k(x) \geq k - d.$$

We say that the states are updated regularly *if there exists an integer M such that, for every integer* $k_0 \geq 1$,

$$\bigcup_{k=k_0}^{k_0+M-1} \{a_k\} = E.$$

THEOREM 4: *If the delays are uniformly bounded and the states updated regularly, then*

$$\lim_{k\to\infty} t_k = \infty.$$

More precisely, if delays are uniformly bounded by d and all the states updated in every interval of length M, then

$$t_k \geq \left\lfloor \frac{k+d}{M+d} \right\rfloor.$$

PROOF: Since every state is visited by time M, we see that $t_k \geq 1$ for $k \geq M$. During the interval $k = M + d + 1, M + d + 2, \ldots, 2M + d$, all states are visited again, and since the delay is at most d, it follows that $t_k \geq 2$ for $k \geq 2M + d$. Continuing in this way, we see that $t_k \geq j$ for $k \geq j(M + d) - d$. This completes the proof. ■

4.2. Simple, Efficient Asynchronous Gauss-Seidel

In this section, we present two simple parallel Gauss-Seidel algorithms. The algorithms are asynchronous, meaning the processors need not run in lockstep. In one of the algorithms, *Async 1*, there is essentially no coordination between processors. In the other, *Async 2*, a mild type of coordination is used, but infrequently, and the attendant overhead is negligible.

Both algorithms are easy to implement on a variety of parallel computers. The number of processors dedicated to the computation may vary, even while the computation is in progress. The processors may run at different speeds and experience variable delays, say, in accessing memory. A few slow processors will not slow down the whole computation, as they would in a synchronous algorithm.

We implemented the algorithms on a Sequent Balance 21000 multiprocessor. All processors share a single, large memory and other resources. For each of our experiments, we reserved a fixed number P of processors, $1 \le P \le 16$. While our experiments ran, other jobs that we had no control over ran on different processors.

In the sequential Gauss-Seidel method, one processor cycles through the state space, following the fixed ordering. Suppose the total number of states is $M = |E|$ and the states are ordered $0,1,\ldots,M-1$. When at the ith state, the processor updates the associated value estimate and then goes on to state $(i+1) \pmod M$.

Our first algorithm, *Async 1*, is the natural generalization for parallel processing. Suppose P processors are available, numbered $0,1,\ldots,P-1$. The basic task of processor p is the same as in the sequential algorithm: sweeping through the state space following the ordering and updating the associated values. However, instead of starting its sweep at state 0, processor p starts at state g_p, where g_p is a parameter, and updates states in the order $g_p, g_p+1,\ldots, M-1,0,1,\ldots,g_p-1$.

How should the g_p be chosen? A snapshot of the computation would reveal a subset of *active* states $A \subset \{0,1,\ldots,M\}$ of size $|A| \le P$, the number of processors. ($|A| < P$ happens if two or more processors simultaneously update the same state.) If the gaps between the active states are small, then the processors will be bunched together during their sweeps. As a result,

> Not infrequently, two processors will attempt to update the same state simultaneously, an obvious waste.
>
> Since typically the ordering is such that the value computed at state p depends strongly on the values of nearby states, simultaneous updates will be made on the basis of similar data. Hence, the computation will behave like the inferior Jacobi method.

Thus, it appears that the gaps between the g_p should be large. In our experiments, we took

$$g_p = p\left\lfloor \frac{M}{P} \right\rfloor,$$

for $p = 0,1,\ldots,P-1$, so the gaps are as large as possible. Although this choice initially separates the active states, it gives no guarantee that the states remain separated throughout the computation.

Our second algorithm, *Async 2*, uses a variant of the *fetch_and_add* instruction (see Gottlieb et al. [7,8] and [12]) to regulate the separation. An execution of

$$a \leftarrow \textit{fetch\_and\_add}(x,b)$$

is equivalent to an indivisible execution of the two instructions

$$1.\ a \leftarrow x$$

$$2.\ x \leftarrow x + b,$$

where a is a private variable (accessible just to the processor executing the *fetch_and_add*) and x a shared variable (accessible to all processors). We need

$$a \leftarrow mod\_fetch\_and\_add(x, b, c),$$

defined to be equivalent to an indivisible execution of the two instructions

$$1.\ a \leftarrow x$$

$$2.\ x \leftarrow x + b \ (mod\ c),$$

where a is a private variable and x a shared variable. We implemented *mod_fetch_and_add* on the Sequent multiprocessor using a hardware locking mechanism.

Under *Async 2*, each processor repeatedly uses *mod_fetch_and_add* to obtain a pointer to a large interval of states and then performs a Gauss-Seidel sweep of the interval. Although two processors may approach one another, nearly updating the same states at the same time, the two separate after executing the *fetch_and_add* to obtain pointers to their next intervals. The details are as follows. A shared variable *next_interval* is initialized to 0. Each processor repeatedly:

1. Executes

$$my\_interval \leftarrow mod\_fetch\_and\_add(next\_interval, K, M),$$

where *my_interval* is a private variable and K a parameter;

2. Performs a Gauss-Seidel sweep of the K states

$$my\_interval,\ (my\_interval + 1)(\text{mod } M), \ldots,$$
$$(my\_interval + K - 1)(\text{mod } M).$$

Choosing, for example, $K = \max\{1, \lfloor M/(2P) \rfloor\}$ on $K = \max\{1, \lfloor M/P \rfloor\}$ should enforce substantial gaps between the active states if $M \gg P$ (the typical case). We took $K = \max\{1, \lfloor M/P \rfloor\}$ in our experiments.

We used a procedure similar to one suggested by Lubachevsky [9] to detect distributed termination. For each sequence of M individual updates, a processor computes an aggregate error estimate

$$\max_i \left| \frac{(u_i - v_i)}{v_i} \right|, \tag{4.4}$$

where v_i represents the new value computed at the processors ith update and u_i the old value. Informally, when a processor finds the aggregate error less than

a given ϵ, it raises its hand (increments a counter using *fetch_and_add*), and when it finds the aggregate error greater than or equal to ϵ it lowers its hand (decrements the counter). After computing M updates, a processor goes on to compute another M unless all processors have their hands raised (the counter equals P, the total number of processors).

For both *Async 1* and *Async 2*, this rule ensures that in a parallel pass over the whole state space, the most recent update at each state resulted in a uniformly small change. In other words, the error computed over a Gauss–Seidel sweep of all states, possibly visiting some states more than once, is uniformly small. We mention a small variation on this stopping rule, which we also tried. After detecting termination as just described, one processor makes an additional sweep over the state space computing the error as in (4.4) but not storing the updates. If this error, which we call the Jacobi residual, is less than ϵ, then the computation is simply restarted. This continues until the Jacobi residual is found to be less than ϵ. In our experiments, in all cases just one pass was required: On the first check, the Jacobi residual was always found to be less than ϵ.

To test the efficiency of *Async 1* and *Async 2*, we ran experiments similar to those described in the previous section for calculating invariant measures and hitting probabilities for Example 2, the queues in series. We shall report just the results for the hitting probability calculation; the results for invariant measure calculation were analogous. We set buffer sizes $N_1 = N_2 = 50$, hitting set levels $L = 50$ and $L' = 0$, and rates $\mu_0 = \mu_1 = \mu_2 = 1$. The flow deficit method was used to generate the state-space ordering. The parameter ϵ of the stopping rule was taken to be 10^{-6}. This is a moderate size problem. There are 2500 states. On one processor, 3303 iterations are required, where an iteration is a sweep of the state space, consisting of M individual updates.

We assessed the efficiency of the parallel computation as follows. For $p = 0, 1, \ldots, P-1$, let $T_P(p)$ denote the number of iterations that processor p performs, where an iteration consists of M individual updates. (Thus, in *Async 2*, an iteration is M/K sweeps of intervals of size K.) Let

$$\max\bigl(T_P(0), T_P(1), \ldots, T_P(P-1)\bigr).$$

As a measure of efficiency, we took the *speedup*, defined as

$$S(P) = \frac{T_1}{T_P},$$

the ratio of the time to solve the problem with one processor to the time to solve the problem with P processors.

As mentioned above, the experiment was carried out on a Sequent Balance 21000 multiprocessor. Figure 6 shows the results. For each algorithm and each value of P ($P = 2, 4, 8, 16$), we made 10 runs. Other computations ran concurrently with ours (but on different processors). Since those computations per-

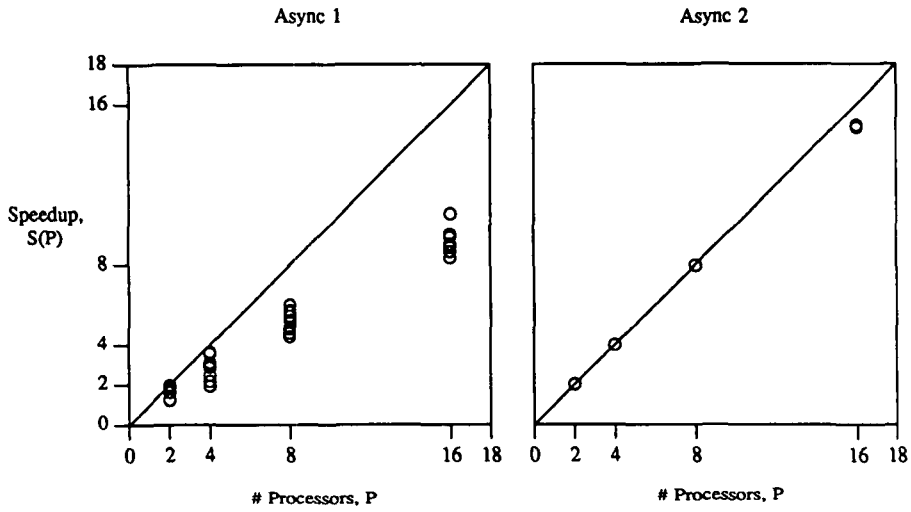

FIGURE 6. Speedups measured on a sequent multiprocessor.

turbed our computation's pattern of accesses to memory, the measure $S(P)$ sometimes varied from run to run. The ten data points (the circles) for each value of P are plotted in Figure 6.

Under *Async 1*, the spread in the results from run to run is significant. For example, for $P = 16$, there is a value of $S(P)$ about 50% larger than the smallest value. Under *Async 2*, the spread in the results is imperceptible, and $S(P)$ is about P. (In all cases, $S(P) \leq P$ although this is hard to discern in Figure 6.) It appears hard to do better. (However, it might be possible to do better, since serialization of the updates performed in the parallel cases could give an ordering that in the sequential case leads to faster convergence than the ordering generated by flow deficit method.)

Async 1 sometimes performs as well as *Async 2*, but the penalty for its higher degree of asynchrony is that it sometimes performs significantly worse. As P, the number of processors, approaches M, the number of states, the algorithms behave more and more like the Jacobi method, and the slope of the speedup decreases. We note that on a synchronous parallel processor dedicated to the Gauss-Seidel computation, there would be little difference between the two algorithms.

In addition to their simplicity and efficiency, the two algorithms have the advantage of being hardly affected by a few slow processors. For example, we ran the experiment described above with $P = 16$, but with two processors artificially slowed to half their normal speed. A dummy job ran on the two processors time-shared with the Gauss-Seidel job). The difference between the resulting speedup data and that plotted in Figure 6 was quite small and consis-

tent with the view that the effective number of processors was 15 rather than 16. In contrast, simpler methods [2], in which the state space and associated computational tasks are predistributed evenly across the processors, run at the speed of the slowest processor.

Acknowledgment

We thank Boris Lubachevsky and Debasis Mitra for very helpful discussions about their work. Debasis kindly discussed with us an unpublished asynchronous parallel algorithm for solving differential equations, which is somewhat similar to the two parallel methods presented here. We thank Dan Heyman for his questions about provably convergent orderings for the invariant measure calculations.

References

1. Aho, A., Hopcroft, J., & Ullman, J. (1974). *The design and analysis of computer algorithms.* New York: Addison Wesley.
2. Baudet, G.M. (1978). Asynchronous iterative methods for multiprocessors. *Journal of the ACM* 25, 2: 226–244.
3. Bertsekas, D. (1982). Distributed dynamic programming. *IEEE Transactions on Automatic Control* AC-27, 3: 610–616.
4. Chazan, D. & Miranker, W. (1969). Chaotic relaxation. *Linear Algebra and Its Applications* 2: 199–222.
5. Dynkin, E.B. & Yushkevich, A.A. (1969). *Markov processes, theorems and problems.* New York: Plenum Press.
6. Goodman, J. & Madras, N. (1987). Random walk interpretations of classical iteration methods. Preprint. New York: Mathematics Department, New York University.
7. Gottlieb, A., Lubachevsky, B.D., & Rudolph, L. (1983). Basic techniques for the efficient coordination of very large numbers of cooperating sequential processors. *ACM Transactions on Programming Languages and Systems* 5, 2: 164–189.
8. Gottlieb, A., Grishman, R., Kruskal, C.P., McAuliffe, K.P., Rudolph, L., & Snir, M. (1983). The NYU ultracomputer—designing a MIMD shared memory parallel machine. *IEEE Transactions on Computers* C-32, 2.
9. Lubachevsky, B.D. (1984). An approach to automating the verification of compact parallel coordination programs I. *Acta Informatica*, 21: 125–169.
10. Lubachevsky, B.D. & Mitra, D. (1987). A chaotic asynchronous algorithm for computing the fixed point of a nonnegative matrix of unit spectral radius. *Journal of the ACM* 33, 1: 130–150.
11. Mitra, D. & Tsoucas, P. (1988). Relaxations for the numerical solutions of some stochastic problems. *Communications in Statistics: Stochastic Models* 4(3): 387–419.
12. Rudolph, L. (1981). Software structures for ultraparallel computing. Ph.D. Thesis, Courant Institute, New York University.
13. Seneta, E. (1981). *Non-negative matrices and Markov chains*, 2nd ed. New York: Springer-Verlag.
14. Tijms, H.C. (1986). *Stochastic modelling and analysis, a computational approach.* New York: Wiley.
15. Varga, R. (1962). *Matrix iterative analysis.* Englewood Cliffs, N.J.: Prentice Hall.
16. van der Wal, J. & Schweitzer, P.J. (1987). Iterative bounds on the equilibrium distribution of a finite Markov chain. *Probability in the Engineering and Information Sciences* 1: 117–131.
17. van Nunen, J. & Wessels, J. (1981). Solving linear systems by methods based on a probabilistic interpretation. *Computing* 26: 209–225.

APPENDIX

In the first part of this section, we briefly summarize Mitra and Tsoucas' analysis of the Jacobi and Gauss-Seidel methods for calculating the invariant measure of a Markov chain (see Mitra and Tsoucas [11]). The Gauss-Seidel method for optimal stopping always converges, no matter what the state-space ordering is. Unfortunately, that is not true of the Gauss-Seidel method for calculating the invariant measure. Mitra and Tsoucas gave a simple test that can be applied to any state-space ordering which, if passed, guarantees that the Gauss-Seidel method for invariant measure converges under that ordering. In the second part of this section, we apply that result to show that convergent orderings always exist. Last, we describe a simple modification to our flow deficit heuristic to generate an ordering that both (1) makes strict high steps likely (and thereby accelerates convergence), and (2) guarantees convergence.

An invariant probability measure corresponding to a stochastic matrix P is a row vector that solves the following system of equations:

$$\pi = \pi P \tag{5.1.a}$$

$$\pi e = 1, \tag{5.1.b}$$

where e denotes the vector of all ones. If the corresponding Markov chain is ergodic, then there is one and only one invariant probability measure. Hence, we assume this to be the case.

The Jacobi method for computing π is to start with an arbitrary probability measure ν^0 (the uniform measure, for example) and to iterate as follows:

$$\nu^{t+1} = \nu^t P.$$

If the Markov chain is ergodic, then ν^t converges to π. Also, ν^t has a simple probabilistic interpretation:

$$\nu^t(x) = P_{\nu^0}\{X_t = x\},$$

where P_ν represents the probability measure on the space of trajectories corresponding to the Markov process starting with the initial distribution ν.

The associated Gauss-Seidel method consists of the following iteration scheme:

$$\mu^{t+1} = \mu^t Q,$$

where $Q = L(I - U)^{-1}$ and

$$Lf(x) = \sum_{y \le x} p(x,y) f(y),$$

$$Uf(x) = \sum_{y > x} p(x,y) f(y).$$

Note that these definitions for L and U are slightly different from those given by (1.6) in that now the diagonal is attached to L instead of U.

Suppose for the moment that μ^t converges, say, to μ^∞. Then μ^∞ satisfies the invar-

iance property (5.1.a), but it is not necessarily a probability. Hence, the solution to (5.1.a) and (5.1.b) is given by

$$\pi = \frac{\mu^{\infty}}{\mu^{\infty} e}.$$

The difficulty is, of course, to establish convergence. Following Mitra and Tsoucas, we present a probabilistic interpretation of the Gauss-Seidel method to aid us in the convergence analysis.

First note that the inverse of $I - U$ exists since U is strictly upper triangular. Therefore, we can write Q in the following form:

$$Q = \sum_{n=0}^{\infty} LU^n. \tag{5.2}$$

It follows from this formula that Q is a nonnegative matrix.

Put

$$\lambda = Le = (I - U)e.$$

Then we see that $Q\lambda = \lambda$. Hence, Q is almost a stochastic matrix. That is, it is a nonnegative matrix that has an eigenvector (λ) whose eigenvalue is one. This is the crucial observation that guides our analysis. In fact, if we can scale the rows and columns of Q so that $q(x,y)$ is replaced by $q(x,y)\lambda(y)/\lambda(x)$, then the resulting matrix would be a stochastic matrix. Unfortunately, some components of λ might be zero and so these states must be treated separately.

It follows immediately from the definition of λ that the two conditions below are equivalent:

$$\lambda(x) = 0 \tag{5.3.a}$$

$$p(x,y) = 0 \qquad \text{for all} \qquad y \le x. \tag{5.3.b}$$

Put

$$A = \{x : \lambda(x) > 0\}$$

$$B = \{x : \lambda(x) = 0\}$$

and partition the matrix Q according to this partition of the states:

$$Q = \begin{bmatrix} Q_{AA} & Q_{AB} \\ Q_{BA} & Q_{BB} \end{bmatrix}.$$

It is clear from equivalence (5.3) that

$$Q_{BA} = Q_{BB} = 0.$$

We also partition our Gauss-Seidel iterates into A and B parts:

$$\mu^t = [\mu_A^t \quad \mu_B^t].$$

In terms of these partitioned subvectors, the Gauss-Seidel method can be written as:

$$\mu_A^{t+1} = \mu_A^t Q_{AA}$$

$$\mu_B^{t+1} = \mu_A^t Q_{AB}.$$

It is now clear that μ^t converges if and only if μ_A^t converges.

We are now prepared to introduce a derived stochastic matrix to help us study the convergence of μ_A^t. Recall that

$$Q_{AA}\lambda_A = \lambda_A$$

and

$$\lambda_A > 0.$$

Put

$$\Lambda = \operatorname{diag}(\lambda_A),$$

and

$$G = \Lambda^{-1} Q_{AA} \Lambda.$$

It is easy to see that G is a stochastic matrix:

$$Ge = \Lambda^{-1} Q_{AA} \Lambda e = \Lambda^{-1} Q_{AA} \lambda_A = \Lambda^{-1} \lambda_A = e.$$

We make one final defintion:

$$\bar{\mu}^t = \mu_A^t \Lambda.$$

Clearly, $\bar{\mu}^t$ converges if and only if μ_A^t converges. Furthermore,

$$\bar{\mu}^{t+1} = \bar{\mu}^t G.$$

Hence, $\bar{\mu}^t$ converges if and only if G is ergodic.

Now we can arrive at a probabilistic interpretation of the Gauss-Seidel iterates. Let $g(x,y)$ denote an element of the matrix G and, similarly, let $q(x,y)$ denote an element of the matrix Q. We see from (5.2) that

$$g(x,y) = \frac{q(x,y)\lambda(y)}{\lambda(x)} = \sum_n P_x\{X_1 < X_2 < \cdots < X_{n+1} = y \geq X_{n+2} | X_0 \geq X_1\}$$

$$= P_x\{X_{\sigma_2 - 1} = y | \sigma_1 = 1\},$$

where

$$\sigma_t = \inf\{s > \sigma_{t-1} : X_s \leq X_{s-1}\},$$

$$\sigma_0 = 0.$$

Note that this definition of σ_t is similar to but not the same as the definition given by formula (2.4). It now follows that the Gauss-Seidel iterates have the probabilistic interpretation:

$$\bar{\mu}^t(y) = P_{\bar{\mu}^0}\{X_{\sigma_{t+1}-1} = y | \sigma_1 = 1\}.$$

How can convergence by guaranteed? The analysis above shows that the iterations converge if and only if the derived stochastic matrix G is ergodic. G depends on the ordering of the state space. Suppose that the original Markov chain is irreducible (every state is reachable from every other state). Following Mitra and Tsoucas (see [11]), we say a transition is high stepping if it goes from state x to state $y > x$. The analysis of Mitra and Tsoucas implies that G is ergodic if the state-space ordering satisfies the following property.

Property R: From every state there is a sequence of high stepping transitions to the largest state in the ordering.

Our first remark is that orderings satisfying Property R always exist. Consider the graph associated with the Markov chain. There is a node for each state and a directed edge for each possible transition. Construct a spanning tree in the graph, a node x is a child of a node y if there is an edge in the tree from x to y. (Tarjan's depth-first search [1] could be used to construct the tree.) Assign an ordering to the nodes such that each node is larger than all its descendents. As a result, the root of the tree is the largest, and the ordering satisfies Property R.

The following modification to the flow deficit heuristic gives a *provably* convergent ordering that favors strict high steps. We reverse the approach of Section 3 and generate the ordering in descending fashion from largest to smallest. Similarly, we reverse the measure of merit and take the measure to be inflow–outflow.

Suppose that there are n states. Choose a state f of greatest merit and assign it position n in the ordering. We now proceed iteratively, where each iteration removes one state from a *candidate set* and adds that state to a *selected set.* At the first iteration the selected set is just $\{f\}$. At each iteration, the candidate set is the set of states not in the selected set that have at least one transition to some state in the selected set. At the ith iteration, a state of greatest merit among those in the candidate set is added to the selected set and assigned position $n - i$ in the ordering.

It is not hard to see that this new algorithm generates an ordering satisfying Property R. The computational cost of the new algorithm is of the same order as that of the original. The only significant difference between the new algorithm and the original is that the candidate sets in the new algorithm are restricted to those that are neighbors of nodes already assigned positions in the ordering. It seems that this restriction would affect the efficacy of the resulting ordering only slightly.