Using Lazy Evaluation to Simulate Realistic-size Repertoires in Models of the Immune System

DEREK J. SMITH, STEPHANIE FORREST, DAVID H. ACKLEY
Department of Computer Science,
University of New Mexico,
Albuquerque, NM 87131, U.S.A.
E-mail: dsmith@cs.unm.edu
E-mail: forrest@cs.unm.edu
E-mail: ackley@cs.unm.edu

ALAN S. PERELSON
Theoretical Division,
Los Alamos National Laboratory
Los Alamos, NM 87545, U.S.A.
E-mail: asp@lanl.gov

We describe a method of implementing efficient computer simulations of immune systems that have a large number of unique B- and/or T-cell clones. The method uses an implementation technique called lazy evaluation to create the illusion that all clones are being simulated, while only actually simulating a much smaller number of clones that can respond to the antigens in the simulation. The method is effective because only 0.001–0.01% of clones can typically be stimulated by an antigen, and because many simulations involve only a small number of distinct antigens. A lazy simulation of a realistic number of clones and 10 distinct antigens is 1000 times faster and 10 000 times smaller than a conventional simulation — making simulations of immune systems with realistic-size repertoires computationally tractable.

© 1998 Society for Mathematical Biology

1. Introduction

The B- and T-cell repertoires of vertebrate immune systems can recognize and respond to almost all foreign antigens, even laboratory-derived ones that almost surely have never been seen in evolutionary history. The repertoire can also distinguish, to a fine level of detail, between foreign antigens and the components of the body it protects. To achieve this broad, yet detailed, coverage, the immune system maintains a large number of highly specific clones, where a clone is a set of cells derived from a single precursor and which almost assuredly has a unique B- or T-cell receptor. In this paper we discuss only the B-cell repertoire; however, the method is also applicable to the T-cell repertoire.
The murine B-cell repertoire maintains $10^7$–$10^8$ distinct clones (Köhler, 1976; Klinman et al., 1976; Klinman et al., 1977), each of which typically can be stimulated by only $10^{-5}$–$10^{-4}$ of all possible antigens (Nossal and Ada, 1971; Edelman, 1974; Jerne, 1974). In order for an antigen to stimulate a B-cell it must bind to antigen-specific receptors on the surface of the B-cell.

The binding affinity between receptors and antigens is based on complementarity at the molecular level. Perelson and Oster (1979) introduced an abstract model of binding in which molecules are considered as points in a ‘shape space’ and affinity is measured as a function of the distance between such points. Modelers have used a variety of methods to represent molecules in shape space. Segel and Perelson (1988) and DeBoer et al. (1992b) examined one- and two-dimensional shape spaces in which the shape of molecules was represented by one or two real numbers, e.g., the depth, or depth and width, of a binding cleft or protrusion on a molecule. Affinity was then measured as a function of the Euclidean distance\(^1\) between the shapes. Seiden and Celada (1992), Forrest and Perelson (1991), and Perelson et al. (1996) [after Farmer et al. (1986)], represented molecules as strings of 8, 32, and 64 bits, respectively, and measured affinity as a function of the Hamming distance\(^2\) (or a variation on it) between them. Weisbuch and Oprea (1994) and Detours et al. (1996) represented molecules as strings of digits chosen from a 4- and 16-letter alphabet, respectively. Smith et al. (1997b) determined that representing molecules as strings of 20 symbols, with each symbol chosen from a four-letter alphabet, and affinity measured as a function of Hamming distance, as well as using a realistic-size repertoire of $10^7$ B-cell clones, gave good fits to immunological data important for a model of cross-reactive memory.

To make simulations of $10^7$ clones computationally tractable, we use a technique called lazy evaluation (Friedman and Wise, 1976; Henderson and Morris, 1976). This technique (as illustrated in the next section) delays calculations, and the building of data structures, until they are needed. When not all calculations and data structures affect the result of a program, and when the relevant ones can be identified efficiently, lazy evaluation can result in significant savings in run time and memory usage. In the case of the immune system, lazy evaluation can be effective because only 0.001–0.01% of all clones are usually stimulated by any particular antigen, and because many simulations involve only a small number of distinct antigens.

Lazy evaluation can be programmed explicitly in traditional programming languages, or implicitly by using languages in which all evaluations are performed lazily (Turner, 1979, 1985; Hudak et al., 1992). Lazy evaluation has been applied

\(^1\)The Euclidean distance is the familiar square root of the sum of the squares of the differences in each dimension. The Euclidean distance between receptors $a_1, a_2 \ldots a_n$ and $b_1, b_2 \ldots b_n$, is
$$\sqrt{\sum_{1 \leq i \leq n} (a_i - b_i)^2}$$

\(^2\)The Hamming distance is a count of the number of locations in which the receptors differ. The Hamming distance between the receptors ABDCDADDA and ABACCDADCA is 2 because they differ in the two underlined locations.
in numerous domains including: animation (Elliott and Hudak, 1997), simulation of integrated circuits (Dunne et al., 1993) [and a production simulator based on Yoshino et al. (1987)], sound synthesis (Dannenberg et al., 1992), and dictionary look-up (Lucas, 1995). In this paper we describe how lazy evaluation can be programmed explicitly in models of the immune system.

2. Algorithm

In a conventional eager approach to immune-system simulation, computation time is taken and memory space explicitly allocated to generate all clones at the start of the simulation [Fig. 1(a)]. When an antigen is introduced, the clones that can be stimulated by it [said to be within its ball of stimulation (Perelson and Oster, 1979)] already exist and the simulation proceeds [Fig. 1(b)]. In the modified lazy simulation, no clones are generated at the start of the simulation (Fig. 2a). Instead, when an antigen is introduced, the simulation is suspended while clones within the ball of stimulation of the antigen are generated [Fig. 2(b)]. In this way, all clones that could be stimulated by the antigen appear and act as in an eager simulation. The absence of the remaining clones has no effect on the simulation other than making it run faster and take less memory. Clones must not be added
When a new antigen is added to the simulation

\[
\text{loop for } i \text{ from } 0 \text{ to } r \text{ do} \\
\quad \text{loop for } j \text{ from } 1 \text{ to } \text{num-clones}(i) \text{ do} \\
\quad \quad \text{let } \text{clone} = \text{mutate}(\text{center}, i) \\
\quad \quad \quad \text{if } \{\text{clone is outside the ball of stimulation} \}
\quad \quad \quad \{\text{of all previously added antigens}\} \\
\quad \quad \quad \text{then } \{\text{add clone to the simulation}\}
\]

where \( r \) is the radius of a ball of stimulation;

where \( \text{num-clones}(i) \) produces a random number from the binomial

distribution \( B(n, p_i) \), where \( n \) is the number of clones in an eager simulation,
and \( p_i \) is the probability that a clone is a radius \( i \) from an antigen;

and where \( \text{mutate}(\text{center}, i) \) mutates \( i \) distinct locations of the
string representing the center of the ball of stimulation.

Figure 3. Pseudo-code describing the lazy algorithm for generating clones.

to regions of a ball of stimulation where they have already been created by the
introduction of previous antigens—this would result in too many clones in the
intersections of balls of stimulation [Fig. 2(c)].

For a lazy simulation to be functionally equivalent to an eager simulation, clones
generated within a ball of stimulation must be added in the same distribution they
would have had in an eager simulation. The correct distribution depends on how
receptors and antigens are represented, and how affinity between them is measured.
Here we describe a lazy algorithm for receptors represented as strings of symbols,
and affinity measured as a function of the Hamming distance between receptors. In
an eager simulation using this representation, receptors are generated by choosing
each symbol from a uniform distribution—loosely mimicking the random genetic
process used by vertebrate immune systems to generate clonal diversity (Leder, 1991).

For the lazy simulation, clones must be generated only within balls of stimulation.
To do this we develop a method to generate clones at radius \( i \) from the center of a
ball of stimulation, and then repeat the method at radii 0 through \( r \), where \( r \) is the
radius of a ball of stimulation. The probability, \( p_i \), that a randomly selected clone
in an eager simulation is radius \( i \) from the center of a ball is given by

\[
p_i = \binom{d}{i} \left( \frac{k - 1}{k} \right)^i \left( \frac{1}{k} \right)^{(d-i)},
\]

where \( d \) is the number of symbols in the string representation of the receptor, and
\( k \) is the number of possible symbols at each location in the string. Further, in an
eager simulation with \( n \) clones, the probability of \( j \) clones at radius \( i \) is given by
the binomial

\[ B(n, p_i) = \binom{n}{j} (p_i)^j (1 - p_i)^{n-j}. \]

Thus, the number of clones, \( \hat{j} \), to generate at radius \( i \) from the center of a ball should be sampled from this binomial distribution. Each of the \( \hat{j} \) clones is generated by changing \( i \) distinct symbols in the string that represents the receptor at the center of the ball of stimulation.

To avoid multiply generating clones in the intersections of balls of stimulation, each new clone is added to the lazy repertoire only if it is outside the balls of stimulation of all antigens already in the simulation. In the next two sections we verify that the lazy algorithm generates clones in the same distributions as the eager algorithm, and compare the algorithmic costs of the lazy and eager algorithms.

### 3. Verification

We generated a complex test case to check whether the lazy algorithm generates clones in the same distributions as the eager algorithm, especially in the case of multiply overlapping balls of stimulation. Following Smith et al. (1997b), molecules in the test were represented by strings of 20 symbols, each symbol was chosen from a four-letter alphabet, and balls of stimulation had radius 5. A seed antigen was generated by randomly selecting each of its symbols from a uniform distribution, and 10 test antigens were generated in a cluster around the seed. Each test antigen was generated by mutating \( m \) randomly selected unique symbols of the seed, where \( m \) was chosen from a uniform distribution in the range 0–3 so the balls of stimulation of the test antigens would have intersections of various sizes. Clones were generated, according to the lazy algorithm, and the number of clones at radii 0–5 for each antigen were counted. The experiment was replicated 100,000 times. For 10,000 of these experiments, the algorithm was metered to record the balls of stimulation that a newly generated clone fell within. These data were used to determine how much of each ball of stimulation was populated with clones generated by previous antigens.

Table 1 shows that the 10 antigens were at varying Hamming distances from each other and thus had varying overlaps. Table 2 shows that these overlaps resulted in many different proportions of balls of stimulation being populated by clones generated by prior antigens, and were thus a reasonable test of the lazy algorithm. Figure 4 shows that the observed and expected distributions are the same when compared visually, and Table 3 shows they are the same when compared statistically. Thus, the lazy algorithm worked correctly.
Table 1. The pairwise Hamming distances between the 10 test antigens used in the experimental verification of the algorithm.

<table>
<thead>
<tr>
<th>Antigen</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>6</td>
<td>6</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>6</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>6</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>6</td>
<td>4</td>
<td>3</td>
<td>6</td>
<td>0</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

Antigen string: CCBDDDBCABCDCCDADAD

Note: The table is symmetric about the main diagonal because Hamming distance is commutative.

The table shows that the antigens were at various Hamming distances from each other.

Table 2. Overlaps in the balls of stimulation of the 10 test antigens

<table>
<thead>
<tr>
<th>Antigen</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.00</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>2</td>
<td>0.05</td>
<td>0.95</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>3</td>
<td>0.21</td>
<td>0.03</td>
<td>0.77</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>4</td>
<td>0.33</td>
<td>0.06</td>
<td>0.17</td>
<td>0.45</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>5</td>
<td>0.05</td>
<td>0.01</td>
<td>0.06</td>
<td>0.02</td>
<td>0.86</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>6</td>
<td>0.05</td>
<td>0.01</td>
<td>0.03</td>
<td>0.05</td>
<td>0.00</td>
<td>0.86</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>7</td>
<td>0.21</td>
<td>0.03</td>
<td>0.09</td>
<td>0.13</td>
<td>0.01</td>
<td>0.02</td>
<td>0.51</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>8</td>
<td>0.10</td>
<td>0.03</td>
<td>0.05</td>
<td>0.09</td>
<td>0.01</td>
<td>0.02</td>
<td>0.14</td>
<td>0.57</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>9</td>
<td>0.01</td>
<td>0.01</td>
<td>0.05</td>
<td>0.10</td>
<td>0.01</td>
<td>0.01</td>
<td>0.02</td>
<td>0.01</td>
<td>0.71</td>
<td>—</td>
</tr>
<tr>
<td>10</td>
<td>0.21</td>
<td>0.03</td>
<td>0.09</td>
<td>0.13</td>
<td>0.01</td>
<td>0.02</td>
<td>0.05</td>
<td>0.01</td>
<td>0.02</td>
<td>0.44</td>
</tr>
</tbody>
</table>

*This shows that the balls of stimulation of the 10 test antigens all overlapped each other; thus, many of the clones within a ball of stimulation were generated by the lazy algorithm operating on prior overlapping antigens. The proportions generated by each antigen are shown in this table. For example, for the fourth antigen, on average, 0.33 of the clones in its ball of stimulation were already generated by the first antigen, 0.06 by the second antigen, 0.17 by the third antigen, and 0.45 were generated de novo by the lazy algorithm on injection of the fourth antigen. The data were calculated by metering the lazy algorithm to record which balls of stimulation a newly generated clone fell within. The varying proportions suggest that the 10 antigens were a reasonable test of the lazy algorithm.*

Table 3. Results of the $\chi^2$ goodness-of-fit tests on the observed and expected distributions of clones.

<table>
<thead>
<tr>
<th>Radius</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.00</td>
<td>0.11</td>
<td>0.09</td>
<td>1.34</td>
<td>0.12</td>
<td>0.11</td>
<td>0.31</td>
<td>0.01</td>
<td>0.37</td>
<td>0.72</td>
</tr>
<tr>
<td>2</td>
<td>4.56</td>
<td>1.50</td>
<td>0.89</td>
<td>3.08</td>
<td>1.19</td>
<td>2.61</td>
<td>1.71</td>
<td>4.55</td>
<td>0.48</td>
<td>1.04</td>
</tr>
<tr>
<td>3</td>
<td>5.15</td>
<td>3.06</td>
<td>5.40</td>
<td>0.43</td>
<td>0.60</td>
<td>2.16</td>
<td>4.13</td>
<td>4.87</td>
<td>3.20</td>
<td>1.69</td>
</tr>
<tr>
<td>4</td>
<td>12.57</td>
<td>10.68</td>
<td>22.56</td>
<td>17.21</td>
<td>6.44</td>
<td>11.24</td>
<td>12.10</td>
<td>7.33</td>
<td>17.46</td>
<td>10.08</td>
</tr>
<tr>
<td>5</td>
<td>39.47</td>
<td>44.45</td>
<td>31.72</td>
<td>31.74</td>
<td>45.25</td>
<td>39.95</td>
<td>35.96</td>
<td>48.35</td>
<td>28.22</td>
<td>54.58</td>
</tr>
</tbody>
</table>

*All observed $\chi^2$ values (except one) were below their respective critical $\chi^2$ value. Thus, there is no evidence ($p = 0.05$) for rejecting the hypothesis that the observed data were in their expected distributions, and we conclude the lazy algorithm worked correctly. The one exception (antigen 3, radius 4) appears to be a Type I error, due to statistical variation, because its $\chi^2$ value was less than the critical value when the experiment was repeated. One such error in 20 tests is to be expected at $p = 0.05$. The data used were from the same 100,000 simulations used to make the plots of Fig. 4. Not enough clones were generated at radius 0 (8 in 100,000 simulations) to perform the test.*
Figure 4. The expected (dashed curve) and observed (solid impulses) distributions of the number of clones at radii 4 and 5 from antigens 1 and 10. These data were collected from the application of the lazy algorithm to the sequential introduction of the 10 test antigens in 100,000 independent simulations and counting the number of clones at each radius within the ball of stimulation of each antigen. Antigens 1 and 10 are shown because they had the least and most number of clones, respectively, generated by prior antigens. Plots showing the distributions for the other antigens, and other radii, showed similar visual correspondence between the observed and expected distributions.

4. ALGORITHMIC COST

In this section we compare the algorithmic cost of the lazy and eager algorithms. The number of clones generated in a lazy simulation is $g \times p \times n$, where $g$ is the number of distinct antigens in the simulation, $p$ is the proportion of the repertoire that can be stimulated by an antigen, and $n$ is the total number of clones in the eager simulation. Each of these clones needs to be checked to see if it falls within the ball of stimulation of any previously added antigen. Thus, the total number of checks after adding $g$ antigens is

$$pn \sum_{0 \leq j < g} j = \frac{g(g - 1)}{2} pn.$$  

If we assume that the cost of generating a clone is approximately the same as the cost of checking if a clone is in the ball of stimulation of an antigen, then the total cost of generating the lazy repertoire for $g$ antigens is

$$gpn + \frac{g(g - 1)}{2} pn = \frac{g^2 + g}{2} pn.$$
Figure 5. A comparison of the algorithmic cost of creating the B-cell repertoire by the lazy and eager algorithms. The algorithmic cost was measured as the sum of the number of clones generated plus the number of checks that a clone was within the ball of stimulation of an antigen. Calculations were done for a realistic-size repertoire with $n = 10^7$ and $p = 10^{-5}$. The lazy algorithm costs less than the eager algorithm when there are less than 447 distinct antigens in a simulation, and costs 1000 times less when there are less than 10 distinct antigens in the simulation.

The cost of the eager method is $n$ because it generates $n$ clones and does not have to do any checks. Comparing the cost of the lazy and eager algorithms, the lazy method is more efficient than the eager method when $g$ is less than approximately $\sqrt{\frac{2}{p}}$.

For a realistic-size repertoire with $n = 10^7$ and $p = 10^{-5}$, in a simulation with 10 distinct antigens, the lazy algorithm will create less than 0.01% of the clone repertoire at 0.1% of the cost of the eager algorithm. For less than 447 distinct antigens, and the same realistic-size repertoire, the lazy algorithm is lower cost than the eager algorithm (Fig. 5). Simulations of more than 447 antigens, which would probably include simulations of immune networks, would be more efficient using an eager algorithm.

When there are more than 10 distinct antigens in a realistic-size simulation, most of the algorithmic cost of the lazy method is in the $g^2$ comparisons of new clones with previously added antigens. In simulations involving hundreds of antigens, this $g^2$ cost could be reduced by various methods. One method is to compare clones only against antigens that are within distance $2r$ of the antigen for which clones are being generated. The algorithm still works correctly because antigens at greater than $2r$ distance cannot have intersecting balls of stimulation. In this case, the $g^2$ comparisons become a worst case, and the actual number of comparisons depends on the distances between the antigens in the simulation. Another method to reduce the $g^2$ comparisons is to generate clones in a ball larger than a ball of stimulation.
and thus avoid lazy generation for any future antigens whose balls of stimulation fall completely within the previously generated larger balls. This second method is effective when the antigens are tightly clustered.

5. DISCUSSION

We have described an algorithm that uses lazy evaluation to generate only the subset of clones that can be stimulated by the antigens introduced into a simulation. Because this subset is typically a tiny proportion of the clone repertoire, the algorithm permits the efficient simulation of realistic-size repertoires. Correctness of the algorithm was checked by showing that, in a test case of 10 overlapping antigens, the algorithm produced clones in the same distributions as an eager algorithm. Analysis of the algorithm showed, in simulations of realistic-size repertoires involving less than 10 distinct antigens, that less than 0.01% of the expressed repertoire was created at less than 0.1% of the cost of creating a complete repertoire. We have implemented a lazy simulation of the humoral immune response that uses a realistic-size repertoire with a steady-state size of \(10^7\) B-cell clones and a turnover of \(5 \times 10^5\) B-cell clones every 6 h (Smith et al., 1997a). Simulations of the sequential infection by three antigens that have overlapping balls of stimulation, over a simulated period of 200 days, takes less than 2 min of CPU time, running in Lisp, on a Sun Ultra 2/2300.

The algorithm we described is specific for models in which receptors are represented as strings of symbols and affinity is calculated as a function of the Hamming distance between receptors. The method could also be applied to other representations of receptors and other methods of calculating affinity, by changing the calculation of the probability distribution of clones within a ball of stimulation and the method of generating clones within a ball of stimulation. The method is applicable to both agent-based models in which each cell is represented individually, and to differential equation-based models in which each clone is represented by a differential equation.

Some neural network and associative memory models have similar structure and function to the immune system models discussed above (Smith et al., 1996). Thus, the lazy method can also be applied to implementation of such models. In particular, the method can be applied without modification to the Sparse Distributed Memory (SDM) model (Kanerva, 1988). The method can also be applied, with modifications similar to those described for other representations of receptors, to the Cerebellar Model Arithmetic Computer (Albus, 1981), the Theory of Cerebellar Cortex (Marr, 1969), WISARD (Aleksander et al., 1984), and variations on SDM (Jaekel, 1989a, b). Danforth (1997) used a lazy-like method, and a modified SDM learning rule (Danforth, 1991), to improve the performance of SDM. Danforth’s method adds at most one hard location (the SDM equivalent of a clone) on each write to the memory, at the exact location of the write. This is in contrast to the cluster of hard
locations that would be added by our method. Both methods significantly reduce the number of hard locations in a simulation (compared with an equivalent eager simulation), and distribute hard locations in accordance with the distribution of addresses used to write to the memory. Danforth’s method modifies the behavior of the SDM; our method leaves the behavior unchanged, and only modifies the implementation.

General immune system models that include clones with receptors have been simulated with the order of \(10^3\) clones (DeBoer and Perelson, 1991; Celada and Seiden, 1996; Detours et al., 1996). Lattice-based cellular models that use only one or two bits to represent the concentration of highly simplified clones, and measure affinity by neighborhood on the lattice, have simulated \(10^4\) clones (DeBoer et al., 1992a), and \(10^8\) clones (Stauffer and Sahimi, 1994) (the latter on a Cray–YMP). The lazy evaluation method presented here is the first to permit realistic-size repertoires of \(10^7–10^8\) clones for general immune-system models.

ACKNOWLEDGMENTS

The authors gratefully acknowledge the ongoing support of the Santa Fe Institute and its Joseph P. and Jeanne M. Sullivan program in theoretical immunology. This work was also supported by Grants ONR (N00014-95-1-0364), NSF (IRI9157644), and NIH (RR06555). Portions of this work were performed under the auspices of the U.S. Department of Energy. DJS also acknowledges the University of New Mexico Computer Science Department AI fellowship and Digital Equipment fellowship, Santa Fe Institute Graduate Fellowship, and the help of Doug Danforth, Ron Hightower, Peter Hraber, Terry Jones, Pentti Kanerva, Catherine Macken, Ronald Moore, Francesca Shready, Suzanne Sluizer, Paul Stanford, and Carla Wofsky. This paper was written while SF was visiting the MIT Artificial Intelligence Laboratory.

REFERENCES


Received 31 August 1997 and accepted 11 November 1997