

A COMPRESSION ALGORITHM FOR PROBABILITY TRANSITION MATRICES*

WILLIAM M. SPEARS[†]

Abstract. This paper describes a compression algorithm for probability transition matrices. The compressed matrix is itself a probability transition matrix. In general the compression is not error-free, but the error appears to be small even for high levels of compression.

Key words. probability transition matrix, transient behavior, compression, lumping, aggregation

AMS subject classifications. 15A51, 15A04

1. Introduction. Many discrete systems can be described by a Markov chain model in which each state of the Markov model is some discrete state of the dynamical system. If there are N states, then the Markov chain model is defined by an $N \times N$ matrix Q called the “1-step probability transition matrix,” where $Q(i, j)$ is the probability of going from state i to state j in one step. The n -step behavior is described by the n th power of Q , Q^n . For many systems, the number of states is enormous and there is a computational advantage in reducing N .

Previous methods for reducing the number of states (referred to as *compression*, *aggregation*, or *lumping* methods) have focused on techniques that provide good estimations of the steady-state behavior of the Markov model. The focus of this paper, however, is on *transient* behavior, and the goal is to produce an algorithm for compressing Q matrices in a way that yields good estimates of the transient behavior of the Markov model. The algorithm described in this paper compresses a Q matrix into a smaller Q matrix with less states. In general, the compression will not be without error, so the goal is to provide an algorithm that compresses the original Q matrix without significant error. Although computing a compressed matrix might take some time, the savings resulting from using this compressed matrix in all subsequent computations can more than offset the compression time.

The organization of this paper is as follows. Section 2 introduces the compression algorithm, which compresses pairs of states by taking a *weighted* average of the row entries for those two states, followed by summing the two columns associated with those two states. Section 2 also introduces the important concepts of row and column *equivalence*, which are important for identifying pairs of states that can be compressed with no error. Section 3 provides mathematical justification for taking the weighted average of row entries and shows that the weights are simply column sums of probability mass. Section 4 proves that pairs of states that are row or column equivalent lead to perfect compression. Section 5 introduces an analysis of error and uses this to define a metric for row and column *similarity* which can be used to find pairs of states that yield almost perfect compression. Later sections illustrate the utility of the compression algorithm through experiments.

2. The Compression Algorithm at a High Level. The entries in the Q matrix, $p_{i,j} \equiv Q(i, j)$, represent the conditional probability that the system will tran-

*This work was supported by ARPA Order # D106/03.

[†]AI Center - Code 5514, Naval Research Laboratory, 4555 Overlook Avenue, Washington DC, 20375. Phone: 202-767-9006, FAX: 202-767-3172 (spears@aic.nrl.navy.mil).

sition to state j in one step, given that it currently is in state i .¹ Now suppose that states i and j have been chosen for compression. The new compressed state is referred to as state $\{i \vee j\}$. Compressing states i and j together means that the combined state represents being in either state i or state j . Since this is a disjunctive situation, the probability of transition from state k into the compressed state is simply the sum $p_{k,\{i \vee j\}} = p_{k,i} + p_{k,j}$. Stated another way, part of the compression algorithm is to sum columns of probability numbers.

However, in general, transitions *from* a compressed state are more complicated to compute. Clearly, the probability of transitioning from the compressed state to some other state $p_{\{i \vee j\},k}$ must lie somewhere between $p_{i,k}$ and $p_{j,k}$, depending on how much time is spent in states i and j . Thus a weighted average of row entries appears to be called for, where the weights reflect the amount of time spent in states i and j . Precisely how to do this weighted average is investigated in Section 3.

The algorithm for compressing two states i and j together is as follows:²

Compress-states(i,j)

- (a) Compute a weighted average of the i th and j th rows.
Place the results in rows i and j .
- (b) Sum the i th and j th columns.
Place the results in column i . Remove row j and column j .

The compression algorithm has two steps. It takes as input a matrix Q_u (an uncompressed Q matrix). Step (a) averages the row entries, producing an intermediate row-averaged matrix Q_r . Step (b) sums column entries to produce the final compressed matrix Q_c . Step (a) is the sole source of error, since in general it is difficult to estimate the amount of time spent in states i and j .

Now that the compression algorithm has been outlined, it is important to define what is meant by “perfect” compression. As mentioned before, analysis of n -step transition probabilities (i.e., *transient* behavior of the Markov chain) can be realized by computing Q^n . For large Q matrices this is computationally expensive. It would be less expensive to compress Q and to then raise it to the n th power. If the compression algorithm has worked well then the n th power of the compressed matrix Q_c should be (nearly) identical to compressing the n th power of the uncompressed matrix Q_u . In other words, perfect compression has occurred if $(Q_u^n)_c = Q_c^n$.

It turns out that there are two situations under which perfect compression can be obtained. The first situation is referred to as “row equivalence”, in which the two states i and j have identical rows (i.e., $\forall k p_{i,k} = p_{j,k}$). In this case the weighted averaging can not produce any error, since the weights will be irrelevant. The second situation is referred to as “column equivalence”, in which state i has column entries that are a real multiple q of the column entries for state j (i.e., $\forall k p_{k,i} = qp_{k,j}$). The intuition here is that when this situation occurs, the ratio of time spent in state i to state j is precisely q . The details of this can be found in Section 4.

However, for arbitrary matrices, compressing an arbitrarily chosen pair of states will not necessarily lead to good results. Thus, the goal is to identify pairs of states i and j upon which the above compression algorithm will work well. It turns out that pairs of states that are row or column *similar* are good candidates for compression. The justification for these measures will be provided in Section 5.

¹The notation $p_{i,j}^{(n)} \equiv Q^n(i,j)$ denotes the entries of the n -step probability transition matrix Q^n .

²The algorithm is written this way because it makes it amenable to mathematical analysis.

At a high level, of course, this simple compression algorithm must be repeated for many pairs of states, if one wants to dramatically reduce the size of a Q matrix. The high level compression algorithm is simply:

- Compress()
 Repeat as long as possible
 (i) Find the pair of states i and j most similar to each other.
 (ii) Compress-states(i,j)

3. The Compression Algorithm in More Detail. In the previous section the compression algorithm was described in two steps. Step (a) is where error can occur and care must be taken to mathematically justify the weighted averaging of rows. This can be done by attempting to force $(Q_u^2)_c$ to be as similar as possible to Q_c^2 (later sections will generalize this to higher powers). This is mathematically difficult, but fortunately it suffices to force Q_u^2 to be as similar as possible to $Q_u Q_r$, which is much simpler and focuses on the row-averaged matrix Q_r explicitly. The intuition behind this is that if compression is done correctly, passage through the new compressed state should affect the 2-step transition probabilities as little as possible.³ This will be shown with a 4×4 Q matrix, and then generalized to an arbitrary $N \times N$ matrix. The result will be the weighted row averaging procedure outlined earlier. This particular presentation has been motivated by a concern for comprehension and hence is not completely formal. A completely formal presentation is in the Appendix.

3.1. Weighted Averaging with a 4×4 Matrix. Consider a general uncompressed 4×4 matrix Q_u for a Markov chain model of 4 states, as well as the general intermediate matrix Q_r :

$$Q_u = \begin{bmatrix} p_{1,1} & p_{1,2} & p_{1,3} & p_{1,4} \\ p_{2,1} & p_{2,2} & p_{2,3} & p_{2,4} \\ p_{3,1} & p_{3,2} & p_{3,3} & p_{3,4} \\ p_{4,1} & p_{4,2} & p_{4,3} & p_{4,4} \end{bmatrix} \quad Q_r = \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & r_{1,4} \\ r_{2,1} & r_{2,2} & r_{2,3} & r_{2,4} \\ r_{3,1} & r_{3,2} & r_{3,3} & r_{3,4} \\ r_{4,1} & r_{4,2} & r_{4,3} & r_{4,4} \end{bmatrix}$$

The notation $r_{i,j} \equiv Q_r(i,j)$ is used to prevent confusion with the $p_{i,j}$ in Q_u . Without loss of generality the goal will be to compress the 3rd and 4th states (rows and columns) of this matrix. Since the 3rd and 4th states are being compressed, rows 1 and 2 of Q_r must be the same as Q_u (i.e., averaging rows 3 and 4 will not affect rows 1 and 2). Denoting $\{3 \vee 4\}$ to be the compressed state, the intermediate matrix is:

$$Q_r = \begin{bmatrix} p_{1,1} & p_{1,2} & p_{1,3} & p_{1,4} \\ p_{2,1} & p_{2,2} & p_{2,3} & p_{2,4} \\ r_{\{3 \vee 4\},1} & r_{\{3 \vee 4\},2} & r_{\{3 \vee 4\},3} & r_{\{3 \vee 4\},4} \\ r_{\{3 \vee 4\},1} & r_{\{3 \vee 4\},2} & r_{\{3 \vee 4\},3} & r_{\{3 \vee 4\},4} \end{bmatrix}$$

The $r_{\{3 \vee 4\},k}$ represent the weighted average of rows 3 and 4 of Q_u . Recall that step (a) of Compress-states(3,4) will place that average in both rows 3 and 4, which is why rows 3 and 4 of Q_r are the same. The trick now is to determine what $r_{\{3 \vee 4\},1}$, $r_{\{3 \vee 4\},2}$, $r_{\{3 \vee 4\},3}$, and $r_{\{3 \vee 4\},4}$ should be in order to produce a reasonable compression. This is done by considering Q_u^2 and $Q_u Q_r$.

³More formally, it can be shown that if $Q_u^2 = Q_u Q_r$ then $(Q_u^2)_c = Q_c^2$ for row or column equivalent situations. See Section 4.

$$Q_u^2 = \begin{bmatrix} p_{1,1}^{(2)} & p_{1,2}^{(2)} & p_{1,3}^{(2)} & p_{1,4}^{(2)} \\ p_{2,1}^{(2)} & p_{2,2}^{(2)} & p_{2,3}^{(2)} & p_{2,4}^{(2)} \\ p_{3,1}^{(2)} & p_{3,2}^{(2)} & p_{3,3}^{(2)} & p_{3,4}^{(2)} \\ p_{4,1}^{(2)} & p_{4,2}^{(2)} & p_{4,3}^{(2)} & p_{4,4}^{(2)} \end{bmatrix} \quad Q_u Q_r = \begin{bmatrix} a_{1,1}^{(2)} & a_{1,2}^{(2)} & a_{1,3}^{(2)} & a_{1,4}^{(2)} \\ a_{2,1}^{(2)} & a_{2,2}^{(2)} & a_{2,3}^{(2)} & a_{2,4}^{(2)} \\ a_{3,1}^{(2)} & a_{3,2}^{(2)} & a_{3,3}^{(2)} & a_{3,4}^{(2)} \\ a_{4,1}^{(2)} & a_{4,2}^{(2)} & a_{4,3}^{(2)} & a_{4,4}^{(2)} \end{bmatrix}$$

The notation $a_{i,j}^{(2)}$ is used to prevent confusion with the $p_{i,j}^{(2)}$ in Q_u^2 . Since the goal is to have $Q_u^2 = Q_u Q_r$, it is necessary to have $p_{i,j}^{(2)}$ be as similar as possible to $a_{i,j}^{(2)}$. The $p_{i,j}^{(2)}$ values can be computed using $p_{i,j}$ values, while the $a_{i,j}^{(2)}$ values require the unknowns $r_{\{3\vee 4\},1}$, $r_{\{3\vee 4\},2}$, $r_{\{3\vee 4\},3}$, and $r_{\{3\vee 4\},4}$.

For example, $p_{1,1}^{(2)}$ can be computed by multiplying Q_u by itself:

$$p_{1,1}^{(2)} = p_{1,1}p_{1,1} + p_{1,2}p_{2,1} + p_{1,3}p_{3,1} + p_{1,4}p_{4,1}$$

However, $a_{1,1}^{(2)}$ is computed by multiplying Q_u and Q_r :

$$a_{1,1}^{(2)} = p_{1,1}p_{1,1} + p_{1,2}p_{2,1} + (p_{1,3} + p_{1,4})r_{\{3\vee 4\},1}$$

In the ideal situation we would like both of these to be equal. This implies that:

$$r_{\{3\vee 4\},1} = \frac{p_{1,3}p_{3,1} + p_{1,4}p_{4,1}}{p_{1,3} + p_{1,4}}$$

But we can write another formula for $r_{\{3\vee 4\},1}$ by considering $p_{2,1}^{(2)}$ and $a_{2,1}^{(2)}$:

$$p_{2,1}^{(2)} = p_{2,1}p_{1,1} + p_{2,2}p_{2,1} + p_{2,3}p_{3,1} + p_{2,4}p_{4,1}$$

$$a_{2,1}^{(2)} = p_{2,1}p_{1,1} + p_{2,2}p_{2,1} + (p_{2,3} + p_{2,4})r_{\{3\vee 4\},1}$$

Again, we would like both of these to be equal. This implies that:

$$r_{\{3\vee 4\},1} = \frac{p_{2,3}p_{3,1} + p_{2,4}p_{4,1}}{p_{2,3} + p_{2,4}}$$

Similarly, consideration of $p_{3,1}^{(2)}$ and $a_{3,1}^{(2)}$ yields:

$$r_{\{3\vee 4\},1} = \frac{p_{3,3}p_{3,1} + p_{3,4}p_{4,1}}{p_{3,3} + p_{3,4}}$$

while consideration of $p_{4,1}^{(2)}$ and $a_{4,1}^{(2)}$ yields:

$$r_{\{3\vee 4\},1} = \frac{p_{4,3}p_{3,1} + p_{4,4}p_{4,1}}{p_{4,3} + p_{4,4}}$$

What has happened here is that the four elements in the first column of $Q_u Q_r$ lead to four expressions for $r_{\{3\vee 4\},1}$. In general, all four expressions for $r_{\{3\vee 4\},1}$ can not hold simultaneously (although we will investigate conditions under which they will hold later). The best estimate is to take a weighted average of the four expressions for $r_{\{3\vee 4\},1}$ (this is related to the concept of ‘‘averaging’’ probabilities – see Appendix for more details). This yields:

$$r_{\{3\vee 4\},1} = \frac{(p_{1,3} + p_{2,3} + p_{3,3} + p_{4,3})p_{3,1} + (p_{1,4} + p_{2,4} + p_{3,4} + p_{4,4})p_{4,1}}{(p_{1,3} + p_{2,3} + p_{3,3} + p_{4,3}) + (p_{1,4} + p_{2,4} + p_{3,4} + p_{4,4})}$$

Note how the final expression for $r_{\{3\vee 4\},1}$ is a weighted average of the row entries $p_{3,1}$ and $p_{4,1}$, where the weights are column sums for columns 3 and 4. In general the elements of $Q_u Q_r$ in the k th column will constrain $r_{\{3\vee 4\},k}$:

$$r_{\{3\vee 4\},k} = \frac{(p_{1,3} + p_{2,3} + p_{3,3} + p_{4,3})p_{3,k} + (p_{1,4} + p_{2,4} + p_{3,4} + p_{4,4})p_{4,k}}{(p_{1,3} + p_{2,3} + p_{3,3} + p_{4,3}) + (p_{1,4} + p_{2,4} + p_{3,4} + p_{4,4})}$$

Once again, note how the expression for $r_{\{3\vee 4\},k}$ is a weighted average of the row entries $p_{3,k}$ and $p_{4,k}$, where the weights are column sums for columns 3 and 4.

3.2. Weighted Averaging with an $N \times N$ Matrix. The previous results for a 4×4 matrix can be extended to an $N \times N$ matrix. Without loss of generality compress states $N-1$ and N . Then the N elements of column k yield N expressions for each $r_{\{N-1\vee N\},k}$. The best estimate is (see Appendix for details):

$$r_{\{N-1\vee N\},k} = \frac{(p_{1,N-1} + \dots + p_{N,N-1})p_{N-1,k} + (p_{1,N} + \dots + p_{N,N})p_{N,k}}{(p_{1,N-1} + \dots + p_{N,N-1}) + (p_{1,N} + \dots + p_{N,N})}$$

Note again how the weights are column sums for columns $N-1$ and N . Generalizing this to compressing two arbitrary states i and j yields:

$$r_{\{i\vee j\},k} = \frac{(\sum_l p_{l,i})p_{i,k} + (\sum_l p_{l,j})p_{j,k}}{\sum_l p_{l,i} + \sum_l p_{l,j}}$$

or:

$$(3.1) \quad r_{\{i\vee j\},k} = \frac{m_i p_{i,k} + m_j p_{j,k}}{m_i + m_j}$$

where m_i and m_j are the sums of the probability mass in columns i and j of Q_u .

Equation 3.1 indicates how to compute the $r_{\{i\vee j\},k}$ entries in Q_r . Note how they are computed using the weighted average of the row entries in rows i and j . The weights are simply the column sums. This justifies the row averaging component of the compression algorithm described in the previous section. Intuitively stated, the column mass for columns i and j provide good estimates of the relative amount of time spent in states i and j . The estimates are used as weights to average the transitions from i to state k and from j to k , producing the probability of transition from the combined state $\{i \vee j\}$ to k .

3.3. Mathematical Restatement of the Compression Algorithm. Now that the weighted averaging of rows i and j has been explained, it is only necessary to sum columns i and j in order to complete the compression algorithm. The whole algorithm can be expressed simply as follows. Assume that two states have been chosen for compression. Let S denote the set of all N states, and let the non-empty sets S_1, \dots, S_{N-1} partition S such that one S_i contains the two chosen states, while each other S_i is composed of exactly one state. Let m_i denote the column mass of state i . Then the compressed matrix Q_c is:

$$(3.2) \quad Q_c(x, y) = \frac{1}{\sum_{i \in S_x} m_i} \sum_{i \in S_x} \left[m_i \sum_{j \in S_y} p_{i,j} \right]$$

This corresponds to taking a weighted average of the two rows corresponding to the two chosen states, while summing the two corresponding columns. The other entries in the Q matrix remain unchanged. Consider an example in which states 2 and 3 are compressed. In that case $S_1 = \{1\}$ and $S_2 = \{2, 3\}$. Q_c is described by:

$$\begin{aligned} Q_c(1, 1) &= p_{1,1} \\ Q_c(1, 2) &= p_{1,2} + p_{1,3} \\ Q_c(2, 1) &= \frac{1}{m_2 + m_3} [m_2 p_{2,1} + m_3 p_{3,1}] \\ Q_c(2, 2) &= \frac{1}{m_2 + m_3} [m_2 (p_{2,2} + p_{2,3}) + m_3 (p_{3,2} + p_{3,3})] \end{aligned}$$

Applying this to the following column equivalent matrix Q_u produces perfect results ($(Q_u^2)_c = Q_c^2$):

$$Q_u = \begin{bmatrix} .7 & .1 & .2 \\ .4 & .2 & .4 \\ .1 & .3 & .6 \end{bmatrix} \Rightarrow Q_u^2 = \begin{bmatrix} .55 & .15 & .30 \\ .40 & .20 & .40 \\ .25 & .25 & .50 \end{bmatrix} \Rightarrow (Q_u^2)_c = \begin{bmatrix} .55 & .45 \\ .30 & .70 \end{bmatrix}$$

$$Q_c = \begin{bmatrix} .7 & .3 \\ .2 & .8 \end{bmatrix} \Rightarrow Q_c^2 = \begin{bmatrix} .55 & .45 \\ .30 & .70 \end{bmatrix}$$

In summary, this section has justified the use of column mass as weights in the row averaging portion of the compression algorithm. The whole compression algorithm is stated succinctly as a mathematical function, which can compress any arbitrary pair of states. However, as stated earlier, compression of arbitrary pairs of states need not lead to good compression. The goal, then, is to identify such states. This is investigated in the next section, and relies upon the concepts of row and column equivalence.

4. Special Cases in Which Compression is Perfect. If compression is working well, then the compressed version of Q_u^n should be (nearly) identical to Q_c^n . As suggested in Section 2, there are two situations under which perfect compression will occur. The first situation is when two states are row equivalent. The intuition here is that the row average of two identical rows will not involve any error, and thus the compression will be perfect. The second situation is when two states are column equivalent. The intuition for this situation is that if the column \mathbf{c}_i is equal to $q\mathbf{c}_j$, then the ratio of time spent in state i to state j is exactly q . Under these circumstances the weighted row average will also produce no error.

This section will prove that $(Q_u^n)_c = Q_c^n$ when the two states being compressed are either row equivalent or column equivalent. This will hold for any n and for any Q_u matrix of size $N \times N$. The method of proof will be to treat the compression algorithm as a linear transformation f , and then to show that $f(Q_u^n) = (f(Q_u))^n$, where $f(Q_u) = Q_c$.

4.1. Row Equivalence and the Compression Algorithm. This subsection will prove that when two states are row equivalent, compression of those states can be described by a linear transformation (matrix multiplication). The compression algorithm compresses an $N \times N$ matrix Q_u to an $(N - 1) \times (N - 1)$ matrix Q_c . However, for the sake of mathematical convenience all of the matrix transformations will be with $N \times N$ matrices. Without loss of generality it is assumed that states $N - 1$ and N are being compressed. When it comes time to expressing the final compression, the N th row and column will simply be ignored, producing the $(N - 1) \times (N - 1)$ compressed matrix. The “•” notation is used to denote entries that are not important for the derivation.

Assume that states $N - 1$ and N are row equivalent. Thus $\forall k$ $p_{N-1,k} = p_{N,k}$. Using Equation 3.1 to compute the row averages yields:

$$r_{\{N-1 \vee N\},k} = \frac{m_{N-1}p_{N-1,k} + m_N p_{N,k}}{m_{N-1} + m_N} = \frac{p_{N-1,k}(m_{N-1} + m_N)}{m_{N-1} + m_N} = p_{N-1,k}$$

and the compressed matrix should have the form:

$$Q_c = \begin{bmatrix} p_{1,1} & \cdots & p_{1,N-2} & p_{1,N-1} + p_{1,N} \\ p_{2,1} & \cdots & p_{2,N-2} & p_{2,N-1} + p_{2,N} \\ \vdots & & \vdots & \vdots \\ p_{N-1,1} & \cdots & p_{N-1,N-2} & p_{N-1,N-1} + p_{N-1,N} \end{bmatrix}$$

THEOREM 4.1. *If states N and $N - 1$ in Q_u are row equivalent then $Q_c = TQ_uT$ and $TT = I$, where*

$$T = \left[\begin{array}{c|cc} I & & 0 \\ \hline 0 & 1 & 0 \\ & 1 & -1 \end{array} \right]$$

Proof: $Q_c = TQ_uT$ can be expressed as follows:

$$Q_c = T \begin{bmatrix} p_{1,1} & \cdots & p_{1,N-2} & p_{1,N-1} & p_{1,N} \\ p_{2,1} & \cdots & p_{2,N-2} & p_{2,N-1} & p_{2,N} \\ \vdots & & \vdots & \vdots & \vdots \\ p_{N-1,1} & \cdots & p_{N-1,N-2} & p_{N-1,N-1} & p_{N-1,N} \\ p_{N,1} & \cdots & p_{N,N-2} & p_{N,N-1} & p_{N,N} \end{bmatrix} \left[\begin{array}{c|cc} I & & 0 \\ \hline 0 & 1 & 0 \\ & 1 & -1 \end{array} \right] =$$

$$\left[\begin{array}{c|cc} I & & 0 \\ \hline 0 & 1 & 0 \\ & 1 & -1 \end{array} \right] \begin{bmatrix} p_{1,1} & \cdots & p_{1,N-2} & p_{1,N-1} + p_{1,N} & \bullet \\ p_{2,1} & \cdots & p_{2,N-2} & p_{2,N-1} + p_{2,N} & \bullet \\ \vdots & & \vdots & \vdots & \vdots \\ p_{N-1,1} & \cdots & p_{N-1,N-2} & p_{N-1,N-1} + p_{N-1,N} & \bullet \\ p_{N-1,1} & \cdots & p_{N-1,N-2} & p_{N-1,N-1} + p_{N-1,N} & \bullet \end{bmatrix} =$$

$$\begin{bmatrix} p_{1,1} & \cdots & p_{1,N-2} & p_{1,N-1} + p_{1,N} & \bullet \\ p_{2,1} & \cdots & p_{2,N-2} & p_{2,N-1} + p_{2,N} & \bullet \\ \vdots & & \vdots & \vdots & \vdots \\ p_{N-1,1} & \cdots & p_{N-1,N-2} & p_{N-1,N-1} + p_{N-1,N} & \bullet \\ \bullet & \cdots & \bullet & \bullet & \bullet \end{bmatrix}$$

This is precisely what Q_c should be. Thus the compression of two row equivalent states can be expressed simply as TQ_uT . The first T performs row averaging (which is trivial) and the second T performs column summing. The reader will also note that some elements of T do not appear to be important for the derivation that $Q_c = TQ_uT$. This is true, however, the purpose of these elements is to ensure that $TT = I$, since this fact will also be used to help prove that $(Q_u^n)_c = Q_c^n$.

$$TT = \left[\begin{array}{c|cc} I & & 0 \\ \hline 0 & 1 & 0 \\ & 1 & -1 \end{array} \right] \left[\begin{array}{c|cc} I & & 0 \\ \hline 0 & 1 & 0 \\ & 1 & -1 \end{array} \right] = I$$

4.2. Column Equivalence and the Compression Algorithm. This subsection will prove that when two states are column equivalent, compression of those states can be described by a linear transformation. Assume without loss of generality that states $N-1$ and N are column equivalent. Thus $\forall k$ $p_{k,N-1} = qp_{k,N}$, and $m_{N-1} = qm_N$. Using Equation 3.1 to compute the row averages yields:

$$r_{\{N-1 \vee N\},k} = \frac{m_{N-1}p_{N-1,k} + m_N p_{N,k}}{m_{N-1} + m_N} = \frac{qp_{N-1,k} + p_{N,k}}{q+1}$$

and the compressed matrix should have the form:

$$Q_c = \left[\begin{array}{cccc} p_{1,1} & \cdots & p_{1,N-2} & p_{1,N-1} + p_{1,N} \\ p_{2,1} & \cdots & p_{2,N-2} & p_{2,N-1} + p_{2,N} \\ \vdots & & \vdots & \vdots \\ \frac{qp_{N-1,1} + p_{N,1}}{q+1} & \cdots & \frac{qp_{N-1,N-2} + p_{N,N-2}}{q+1} & \frac{qp_{N-1,N-1} + p_{N,N-1} + qp_{N-1,N} + p_{N,N}}{q+1} \end{array} \right]$$

THEOREM 4.2. *If states N and $N-1$ in Q_u are column equivalent then $Q_c = XQ_uY$ and $YX = I$ where*

$$Y = \left[\begin{array}{c|cc} I & & 0 \\ \hline 0 & 1 & \frac{1}{q} \\ & 1 & -1 \end{array} \right] \quad X = \left[\begin{array}{c|cc} I & & 0 \\ \hline 0 & \frac{q}{q+1} & \frac{1}{q+1} \\ & \frac{q}{q+1} & -\frac{q}{q+1} \end{array} \right]$$

Proof: $Q_c = XQ_uY$ can be expressed as follows:

$$Q_c = X \left[\begin{array}{ccccc} p_{1,1} & \cdots & p_{1,N-2} & p_{1,N-1} & p_{1,N} \\ p_{2,1} & \cdots & p_{2,N-2} & p_{2,N-1} & p_{2,N} \\ \vdots & & \vdots & \vdots & \vdots \\ p_{N-1,1} & \cdots & p_{N-1,N-2} & p_{N-1,N-1} & p_{N-1,N} \\ p_{N,1} & \cdots & p_{N,N-2} & p_{N,N-1} & p_{N,N} \end{array} \right] \left[\begin{array}{c|cc} I & & 0 \\ \hline 0 & 1 & \frac{1}{q} \\ & 1 & -1 \end{array} \right] =$$

$$\left[\begin{array}{c|cc} I & & 0 \\ \hline 0 & \frac{q}{q+1} & \frac{1}{q+1} \\ & \frac{q}{q+1} & -\frac{q}{q+1} \end{array} \right] \left[\begin{array}{cccc} p_{1,1} & \cdots & p_{1,N-2} & p_{1,N-1} + p_{1,N} & \bullet \\ p_{2,1} & \cdots & p_{2,N-2} & p_{2,N-1} + p_{2,N} & \bullet \\ \vdots & & \vdots & \vdots & \vdots \\ p_{N-1,1} & \cdots & p_{N-1,N-2} & p_{N-1,N-1} + p_{N-1,N} & \bullet \\ p_{N,1} & \cdots & p_{N,N-2} & p_{N,N-1} + p_{N,N} & \bullet \end{array} \right] =$$

$$\left[\begin{array}{cccc} p_{1,1} & \cdots & p_{1,N-2} & p_{1,N-1} + p_{1,N} & \bullet \\ p_{2,1} & \cdots & p_{2,N-2} & p_{2,N-1} + p_{2,N} & \bullet \\ \vdots & & \vdots & \vdots & \vdots \\ \frac{qp_{N-1,1} + p_{N,1}}{q+1} & \cdots & \frac{qp_{N-1,N-2} + p_{N,N-2}}{q+1} & \frac{qp_{N-1,N-1} + p_{N,N-1} + qp_{N-1,N} + p_{N,N}}{q+1} & \bullet \\ \bullet & \cdots & \bullet & \bullet & \bullet \end{array} \right]$$

This is precisely what Q_c should be. Thus the compression of two column equivalent states can be expressed simply as XQ_uY . X performs row averaging and Y performs column summing. The reader will note that some elements of X and Y are not important for the derivation that $Q_c = XQ_uY$ (e.g., T could be used instead of Y). This is true, however, the purpose of these elements is to ensure that $YX = I$, since this fact will be used to help prove that $(Q_u^n)_c = Q_c^n$ at the end of this section.

$$YX = \left[\begin{array}{c|cc} I & & 0 \\ \hline 0 & 1 & \frac{1}{q} \\ & 1 & -1 \end{array} \right] \left[\begin{array}{c|cc} I & & 0 \\ \hline 0 & \frac{q}{q+1} & \frac{1}{q+1} \\ & \frac{q}{q+1} & -\frac{q}{q+1} \end{array} \right] = I$$

4.3. Some Necessary Lemmas. Before proving that $(Q_u^n)_c = Q_c^n$ for row or column equivalent states, it is necessary to prove some simple lemmas. The idea is to show that if Q_u is row or column equivalent, so is Q_u^n . This will allow the previous linear transformations to be applied to Q_u^n as well as Q_u .

Let square matrices A and B be defined as matrices of row and column vectors respectively:

$$A = \begin{bmatrix} a_{1,1} & \cdots & a_{1,N} \\ \vdots & & \vdots \\ a_{N,1} & \cdots & a_{N,N} \end{bmatrix} = \begin{bmatrix} \mathbf{a}_1 \\ \vdots \\ \mathbf{a}_N \end{bmatrix}$$

$$B = \begin{bmatrix} b_{1,1} & \cdots & b_{1,N} \\ \vdots & & \vdots \\ b_{N,1} & \cdots & b_{N,N} \end{bmatrix} = [\mathbf{b}_1 \quad \cdots \quad \mathbf{b}_N]$$

Then the matrix product AB can be represented using dot product notation:

$$AB = \begin{bmatrix} \mathbf{a}_1 \cdot \mathbf{b}_1 & \cdots & \mathbf{a}_1 \cdot \mathbf{b}_N \\ \vdots & & \vdots \\ \mathbf{a}_N \cdot \mathbf{b}_1 & \cdots & \mathbf{a}_N \cdot \mathbf{b}_N \end{bmatrix}$$

LEMMA 4.3. *Row equivalence is invariant under post-multiplication.*

Proof: Suppose states i and j of A are row equivalent ($\mathbf{a}_i = \mathbf{a}_j$). Then $\forall k \mathbf{a}_i \cdot \mathbf{b}_k = \mathbf{a}_j \cdot \mathbf{b}_k$. So, states i and j in AB must be row equivalent.

LEMMA 4.4. *Column equivalence is invariant under pre-multiplication.*

Proof: Suppose states i and j of B are column equivalent ($\mathbf{b}_i = q\mathbf{b}_j$). Then $\forall k \mathbf{a}_k \cdot \mathbf{b}_i = q\mathbf{a}_k \cdot \mathbf{b}_j$. So, states i and j in AB must be column equivalent.

LEMMA 4.5. *Row and column equivalence are invariant under raising to a power.*

Proof: $Q^n = QQ^{n-1}$. Thus, if states i and j are row equivalent in Q , they are row equivalent in Q^n by Lemma 4.3. Similarly, $Q^n = Q^{n-1}Q$. Thus, if states i and j are column equivalent in Q , they are column equivalent in Q^n by Lemma 4.4.

Lemma 4.5 indicates that the previous linear transformations can be applied to Q_u^n to produce $(Q_u^n)_c$ when two states in Q_u are row or column equivalent.

4.4. Theorems for Perfect Compression. Given the previous theorems concerning the linear transformations and Lemma 4.5, it is now possible to state and prove the theorems for perfect compression. The Q matrix can be considered to be Q_u in these theorems.

THEOREM 4.6. *If Q is row equivalent, then $Q^n = QQ_r^{n-1}$ implies $(Q^n)_r = Q_r^n$, and $(Q^n)_c = Q_c^n$*

Proof: If Q is row equivalent, then so is Q^n by Lemma 4.5. If $Q^n = QQ_r^{n-1}$ then $(Q^n)_r = (QQ_r^{n-1})_r = TQQ_r^{n-1} = Q_r^n$, and $(Q^n)_c = (QQ_r^{n-1})_c = T(QQ_r^{n-1})T = TQQ_c^{n-1} = TQTTQ_c^{n-1} = Q_cTQ_c^{n-1} = Q_c^n$.

THEOREM 4.7. *If Q is column equivalent, then $Q^n = QQ_r^{n-1}$ implies $(Q^n)_r = Q_r^n$, and $(Q^n)_c = Q_c^n$*

Proof: If Q is column equivalent, then so is Q^n by Lemma 4.5. If $Q^n = QQ_r^{n-1}$ then $(Q^n)_r = (QQ_r^{n-1})_r = XQQ_r^{n-1} = Q_r^n$, and $(Q^n)_c = (QQ_r^{n-1})_c = X(QQ_r^{n-1})Y = XQQ_c^{n-1} = XQTTQ_c^{n-1} = Q_cTQ_c^{n-1} = Q_c^n$.

These two theorems illustrate the validity of trying to force Q_u^2 to be as similar as possible to Q_uQ_r in Section 3.

THEOREM 4.8. *If Q is row equivalent, then $(Q^n)_c = Q_c^n$.*

Proof: If Q is row equivalent, then so is Q^n by Lemma 4.5. Then $(Q^n)_c = TQ^nT = TQ \cdots QT$. Since $TT = I$, then $(Q^n)_c = TQTTQ \cdots QTTQT = Q_c^n$.

THEOREM 4.9. *If Q is column equivalent, then $(Q^n)_c = Q_c^n$.*

Proof: If Q is column equivalent, then so is Q^n by Lemma 4.5. Then $(Q^n)_c = XQ^nY = XQ \cdots QY$. Since $YX = I$, then $(Q^n)_c = XQYXQ \cdots QYXQY = Q_c^n$.

These theorems hold for all n and for all row or column equivalent $N \times N$ Q matrices, and highlight the importance of row and column equivalence. If two states are row or column equivalent, then compression of those two states is perfect (i.e., $(Q^n)_c = Q_c^n$).

5. Error Analysis and a Similarity Metric. The previous sections have explained how to merge pairs of states and have explained that row or column equivalent pairs will yield perfect compression. Of course, it is highly unlikely that pairs of states will be found that are perfectly row equivalent or column equivalent. The goal then is to find a similarity metric that measures the row and column similarity (i.e., how close pairs of states are to being row or column equivalent). If the metric is formed correctly, those pairs of states that are more similar should yield less error when compressed. This section will derive an expression for error and then use this as a similarity metric for pairs of states.

We will use $Q_u Q_r$ and Q_u^2 to estimate error. As mentioned before, it is desirable to have the entries in those two matrices be as similar as possible. Consider compressing two states i and j . Then the entries in Q_u^2 are:

$$p_{x,y}^{(2)} = p_{x,i}p_{i,y} + p_{x,j}p_{j,y} + \sum_{k \neq i,j} p_{x,k}p_{k,y}$$

The entries in $Q_u Q_r$ are:

$$a_{x,y}^{(2)} = (p_{x,i} + p_{x,j})r_{\{i \vee j\},y} + \sum_{k \neq i,j} p_{x,k}p_{k,y}$$

Then the error associated with the (x, y) th element of $Q_u Q_r$ is:

$$Error_{i,j}(x, y) = a_{x,y}^{(2)} - p_{x,y}^{(2)} = (p_{x,i} + p_{x,j})r_{\{i \vee j\},y} - p_{x,i}p_{i,y} - p_{x,j}p_{j,y}$$

Using Equation 3.1 for $r_{\{i \vee j\},k}$ (and substituting y for k) yields:

$$Error_{i,j}(x, y) = (p_{x,i} + p_{x,j}) \left[\frac{m_i p_{i,y} + m_j p_{j,y}}{m_i + m_j} \right] - p_{x,i}p_{i,y} - p_{x,j}p_{j,y}$$

Now denote $\alpha_{i,j}(y) = p_{i,y} - p_{j,y}$. This is a measure of the row similarity for rows i and j at column y (and will be explained further below). Then:

$$Error_{i,j}(x, y) = (p_{x,i} + p_{x,j}) \left[\frac{m_i(p_{j,y} + \alpha_{i,j}(y)) + m_j p_{j,y}}{m_i + m_j} \right] - p_{x,i}(p_{j,y} + \alpha_{i,j}(y)) - p_{x,j}p_{j,y}$$

This simplifies to:

$$Error_{i,j}(x, y) = \frac{(m_i p_{x,j} - m_j p_{x,i}) \alpha_{i,j}(y)}{m_i + m_j}$$

Denote $\beta_{i,j}(x) = (m_i p_{x,j} - m_j p_{x,i}) / (m_i + m_j)$. Then:

$$Error_{i,j}(x, y) = \beta_{i,j}(x) \alpha_{i,j}(y)$$

Now $\beta_{i,j}(x)$ can be considered to be a measure of column similarity for columns i and j at row x (this will be shown more explicitly further down). Since only the magnitude of the error is important, and not the sign, the absolute value of the error should be considered:

$$|Error_{i,j}(x,y)| = |\beta_{i,j}(x)\alpha_{i,j}(y)|$$

Recall that $Error_{i,j}(x,y)$ is the error associated with the (x,y) th element of $Q_u Q_r$, if states i and j are compressed. The total error of the whole matrix is:

$$Error_{i,j} = \sum_x \sum_y |Error_{i,j}(x,y)| = \sum_x \sum_y |\beta_{i,j}(x)\alpha_{i,j}(y)|$$

But this can be simplified to:

$$Error_{i,j} = \left(\sum_x |\beta_{i,j}(x)|\right) \left(\sum_y |\alpha_{i,j}(y)|\right)$$

To understand this equation consider the situation where states i and j are row equivalent. Then $\forall y p_{i,y} = p_{j,y}$. This indicates that $\forall y \alpha_{i,j}(y) = 0$ and $Error_{i,j} = 0$. Thus there is no error associated with compressing row equivalent states i and j , as has been shown in earlier sections.

Consider the situation where states i and j are column equivalent. Then $\forall x p_{x,i} = qp_{x,j}$ and $m_i = qm_j$. It is trivial to show that $\forall x \beta_{i,j}(x) = 0$ and as a consequence $Error_{i,j} = 0$. Thus there is no error associated with compressing column equivalent states i and j , as has been shown in earlier sections.

Given this, a natural similarity metric is the expression for error:

$$(5.1) \quad Similarity_{i,j} = \left(\sum_x |\beta_{i,j}(x)|\right) \left(\sum_y |\alpha_{i,j}(y)|\right)$$

If the similarity is close to zero then error is close to zero, and pairs of states can be judged as to the amount of error that will ensue if they are compressed.⁴ The compression algorithm can now be written as follows:

- ```

Compress()
 Repeat as long as possible
 (i) Find pair of states i and j such that $Similarity_{i,j} < \epsilon$.
 (ii) Compress-states(i,j)

```

The role of  $\epsilon$  is as a threshold. Pairs of states that are more similar than this threshold can be compressed. By raising  $\epsilon$  one can compress more states, but with a commensurate increase in error.

The paper thus far has fully outlined the compression algorithm for pairs of states, and identified situations under which compression is perfect – namely, when the pairs of states are row or column equivalent. By performing an error analysis, a natural measure of similarity was derived, in which pairs of states that are row or column similar yield small amounts of error in the compression algorithm. The following section outlines some experiments showing the degree of compression that can be achieved in practice.

---

<sup>4</sup>It is useful to think of this as a “Dissimilarity” metric.

**6. Some Experiments.** In order to evaluate the practicality of the compression algorithm, it was tested on some Markov chains derived from the field of genetic algorithms (GAs). In a GA a population of individuals evolves generation by generation via Darwinian selection and perturbation operators such as recombination and mutation. Each individual in the population can be considered to be a point in a search space (see [8] for an overview of GAs).

Each different population of the GA is a state in the Markov chain, and  $p_{i,j}$  is the probability that the GA will evolve from one population  $i$  to another  $j$ , in one generation (time step). The number of states grows extremely fast as the size of the population increases and as the size of individuals increase. The details of the mapping of GAs to Markov chains can be found in [6]. Their use in examining transient behavior can be found in [2].<sup>5</sup>

**6.1. Accuracy Experiments.** The first set of experiments examine the accuracy of the compressed Markov chains by using both  $Q_u^n$  and  $Q_c^n$  to compute the probability distribution  $p^{(n)}$  over the states at time  $n$ . To answer such questions,  $Q_u^n$  must be combined with a set of initial conditions concerning the GA at generation 0. Thus, the *a priori* probability of the GA being in state  $i$  at time 0 is  $p_i^{(0)}$ .<sup>6</sup> Given this, the probability that the GA will be in a particular state  $j$  at time  $n$  is:

$$p_j^{(n)} = \sum_i p_i^{(0)} p_{i,j}^{(n)}$$

It is also possible to compute probabilities over a set of states. Define a predicate  $Pred_J$  and the set  $J$  of states that make  $Pred_J$  true. Then the probability that the GA will be in one of the states of  $J$  at time  $n$  is:

$$p_J^{(n)} = \sum_{j \in J} p_j^{(n)}$$

In this paper,  $J$  represents the set of all states which contain at least one copy of the optimum (i.e., the set of all populations which have at least one individual with the optimum function value). The Markov model is used to compute  $p_J^{(n)}$ , the probability of having at least one copy of the optimum in the population at time  $n$ .

The compression algorithm can thus be evaluated by using both  $Q_u^n$  (ground truth) and  $Q_c^n$  (the estimate) to compute  $p_J^{(n)}$  for different values of  $n$ . The closer the estimate is to ground truth, the better the compression algorithm is working.

Since the goal is to compute probabilities involving states containing the optimum (the  $J$  set),  $J$  states should not be compressed with non- $J$  states. Consequently, the compression algorithm is run separately for both sets of states. The algorithm is:

- Repeat until no new compressed states are created
  - (a) For each state  $i$  in the  $J$  set of the current compressed model
    - (i) Find the most similar state  $j$  in the  $J$  set.
    - (ii) If  $Similarity_{i,j} < \epsilon$ , Compress-states( $i,j$ ).
  - (b) For each state  $i$  in the non- $J$  set of the current compressed model
    - (i) Find the most similar state  $j$  in the non- $J$  set.
    - (ii) If  $Similarity_{i,j} < \epsilon$ , Compress-states( $i,j$ ).

<sup>5</sup>For the GA,  $Q$  has no zero entries and is thus ergodic.

<sup>6</sup>If states  $i$  and  $j$  have been compressed then  $p_{\{i \vee j\}}^{(0)} = p_i^{(0)} + p_j^{(0)}$ .

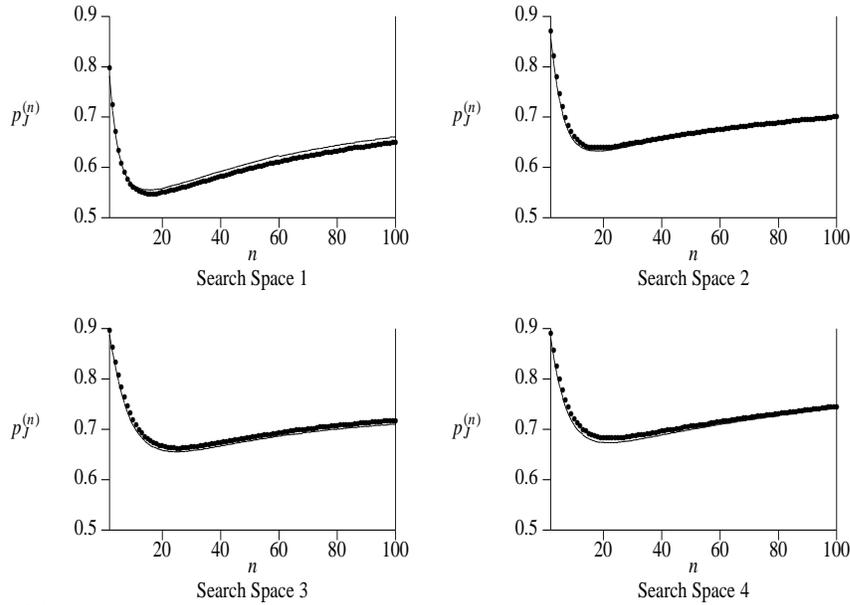


FIG. 6.1.  $p_J^{(n)}$  where  $\epsilon$  is 0.0 and 0.15 for  $N = 455$ . The bold curves represent the exact values, while the non-bold curves represent the values computed from the compressed matrix.

In theory this compression algorithm could result in a two state model involving just  $J$  and non- $J$ . In practice this would require large values of  $\epsilon$  and unacceptable error in  $p_J^{(n)}$  computations.

Four different search spaces were chosen for the GA. This particular set of four search spaces was chosen because experience has shown that it is hard to get a single compression algorithm to perform well on all. Also, in order to see how well the compression algorithm scales to larger Markov chains, four population sizes were chosen for the GA (10, 12, 14, and 16). These four choices of population size produced Markov chains of 286, 455, 680, and 969 states, respectively. Thus, the compression algorithm was tested on sixteen different Markov chains.<sup>7</sup>

Naturally, the setting of  $\epsilon$  is crucial to the success of the experiments. Experiments indicated that a value of 0.15 yielded good compression with minimal error, for all sixteen Markov chains. The results for  $N = 455$  are shown in Figure 6.1. The results for the other experiments are omitted for the sake of brevity, but they are almost identical. The values  $p_J^{(n)}$  are computed for  $n$  ranging from 2 to 100, for both the compressed and uncompressed Markov chains, and graphed as curves. The bold curves represent the exact  $p_J^{(n)}$  values, while the non-bold curves represent the values computed from the compressed matrix.

The figures clearly indicate that the compressed matrix is yielding negligible error. To see how the amount of compression is affected by the size of the Markov chain, consider Table 6.1, which gives the percentage of states removed for each of the sixteen chains. What is interesting is that, for these particular search spaces, the amount of compression is increasing as  $N$  increases (while still yielding negligible error). For  $N = 969$ , over 80% of the states have been removed, yielding  $Q_c$  matrices roughly 3% the size (in terms of memory requirements) of the original  $Q_u$  matrix. It is also interesting to note that different search spaces are consistently compressed to different

<sup>7</sup>See [2] for a definition of these search spaces.

TABLE 6.1  
*The percentage of states removed when  $\epsilon = 0.15$ .*

|                | $N = 286$ | $N = 455$ | $N = 680$ | $N = 969$ |
|----------------|-----------|-----------|-----------|-----------|
| Search Space 1 | 85%       | 88%       | 90%       | 92%       |
| Search Space 2 | 71%       | 76%       | 81%       | 84%       |
| Search Space 3 | 65%       | 73%       | 79%       | 82%       |
| Search Space 4 | 64%       | 73%       | 79%       | 82%       |

degrees. For example, the third and fourth search spaces are consistently compressed less than the first search space. Further investigation into the nature of these search spaces may help characterize when arbitrary Markov chains are hard/easy to compress with this algorithm.

**6.2. Timing Experiments.** It is now necessary to examine the computational cost of the compression algorithm. Our prior work, [2] and [9], focused heavily on the insights gained by actually examining  $Q_u^n$ , which involved computations on the order of  $N^3$  (to multiply  $Q_u$  repeatedly). Thus, the primary motivation for producing the compression algorithm was to gain the same insights more efficiently by dramatically reducing  $N$ . Since the second search space is quite representative in terms of the performance of the compression algorithm, we draw our timing results from the experiments with that particular search space. Table 6.2 gives the amount of CPU time (in minutes) needed to compute  $Q_u^n$  as  $n$  ranges from 2 to 100. Table 6.3 gives the amount of time needed to compress  $Q_u$  to  $Q_c$  as well as the time needed to compute  $Q_c^n$  as  $n$  ranges from 2 to 100.<sup>8</sup> Clearly, the compression algorithm achieves enormous savings in time when it is actually necessary to compute powers of  $Q_u$ .

TABLE 6.2  
*The time (in minutes) to compute  $Q_u^n$  for  $n = 2$  to  $n = 100$ .*

|                  | $N = 286$ | $N = 455$ | $N = 680$ | $N = 969$ |
|------------------|-----------|-----------|-----------|-----------|
| Computation Time | 27        | 125       | 447       | 1289      |

TABLE 6.3  
*The time (in minutes) to compress  $Q_u$  and to compute  $Q_c^n$  for  $n = 2$  to  $n = 100$ .*

|                  | $N = 286$ | $N = 455$ | $N = 680$ | $N = 969$ |
|------------------|-----------|-----------|-----------|-----------|
| Compression Time | 0.2       | 0.9       | 3.0       | 9.5       |
| Computation Time | 2.4       | 7.6       | 17.9      | 38.1      |

Another common use of  $Q_u^n$  is to compute the probability distribution  $p^{(n)}$  over the states at time  $n$  (as we did in the previous subsection). If the prior distribution  $p^{(0)}$  is known in advance, however, this is more efficiently done by multiplying  $p^{(0)}$  by  $Q_u$  repeatedly (i.e., this is repeated  $n$  times to produce  $p^{(n)}$ ). The computation is of order  $N^2$  instead of  $N^3$ .

Table 6.4 and Table 6.5 give the amount of time needed to compute  $p^{(n)}$  (from  $Q_u$  and  $Q_c$  respectively). Despite the obvious benefits of computing  $p^{(n)}$  from  $Q_c$ , the compression algorithm is not advantageous in this case since the time needed

<sup>8</sup>All timing results are on a Sun Sparc 20. The code is written in C and is available from the author.

TABLE 6.4  
*The time (in minutes) to compute  $p^{(n)}$  for  $n = 2$  to  $n = 100$ .*

|                  | $N = 286$ | $N = 455$ | $N = 680$ | $N = 969$ |
|------------------|-----------|-----------|-----------|-----------|
| Computation Time | 0.1       | 0.3       | 0.7       | 1.4       |

TABLE 6.5  
*The time (in minutes) to compress  $Q_u$  and to compute  $p^{(n)}$  for  $n = 2$  to  $n = 100$ .*

|                  | $N = 286$ | $N = 455$ | $N = 680$ | $N = 969$ |
|------------------|-----------|-----------|-----------|-----------|
| Compression Time | 0.2       | 0.9       | 3.0       | 9.5       |
| Computation Time | 0.02      | 0.02      | 0.03      | 0.05      |

to compress  $Q_u$  exceeds the time to produce  $p^{(n)}$  from  $Q_u$ . However, there are still occasions when compressing  $Q_u$  and then using  $Q_c$  to compute  $p^{(n)}$  will in fact be more efficient. The first is when it is necessary to compute  $p^{(n)}$  for a large number of different prior distributions (recall that  $Q_c$  does not depend on the prior information and hence need not be recomputed). The second occasion is when it is necessary to compute  $p^{(n)}$  for large  $n$  (e.g., [10] indicates that times on the order of  $10^8$  are sometimes required). In both of these situations the cost of the compression algorithm is amortized. Finally, compression is also advantageous when the prior distribution is not known in advance.<sup>9</sup>

In summary, the compression algorithm is most advantageous when it is necessary to actually examine the powers of  $Q_u$  directly. For computing probability distributions over the states, the compression algorithm will be advantageous if the prior distribution is initially unknown, if a large number of prior distributions will be considered, or if the transient behavior over a long period of time is required.

**7. Related Work.** The goal of this paper has been to provide a technique for compressing (or *aggregating*) discrete-time Markov chains (DTMCs) in a way that yields good estimates of the transient behavior of the Markov model. This section summarizes the work that is most closely related.

There is a considerable body of literature concerning the approximation of transient behavior in Markov chains. Techniques include the computation of matrix exponentials, the use of ordinary differential equations, and Krylov subspace methods [10]. However, all of these techniques are for continuous-time Markov chains (CTMCs), which use an infinitesimal generator matrix instead of a probability transition matrix. It is possible to discretize a CTMC to obtain a DTMC such that the stationary probability vector of the CTMC is identical to that of the DTMC. However, [10] notes that the transient solutions of DTMCs are not the same as those of the corresponding CTMCs, indicating that these techniques will be problematic for computing the transient behavior of DTMCs.

There is also considerable work in aggregation of DTMCs. Almost all theoretical analyses of aggregation (e.g., “block aggregation” [5]) utilize the same functional form:

$$f(Q_u) = Q_c = A Q_u B \quad (AB = I)$$

where  $A$  and  $B$  are matrices that determine the partitioning and the aggregation of the states [3] [4]. This functional form must satisfy two axioms: “linearity” and “state

---

<sup>9</sup>It is also important to emphasize that it is very likely that the compression algorithm can be extensively optimized, producing much better timing results.

partitioning”. Linearity implies that  $A$  and  $B$  do not depend explicitly on the entries in  $Q_u$ . State partitioning implies that the “aggregated” transition probabilities should depend only upon the probabilities associated with the aggregated states (e.g., the aggregation of states  $i$  and  $j$  should only depend on  $p_{i,i}$ ,  $p_{i,j}$ ,  $p_{j,i}$ , and  $p_{j,j}$ ).

Neither axiom is true for compression of column equivalent states in this paper. This is reflected in the fact that in general  $AB = XY \neq I$ . Instead, in this paper  $BA = I$  for both row and column equivalence, yielding desirable properties with respect to the powers of  $Q_u$ . The current results indicate that the relevance of both axioms should be re-examined.

The aggregation technique most closely related to the work in this paper is described by [10], [11] and [12]. This aggregation technique partitions the set of states  $S$  into  $s$  non-empty sets  $S_1, \dots, S_s$ . Denoting the steady state probability of state  $i$  as  $\pi_i$ , then  $\pi_y = \sum_{i \in S_y} \pi_i$  if:

$$(7.1) \quad Q_c(x, y) = \frac{1}{\sum_{i \in S_x} \pi_i} \sum_{i \in S_x} \left[ \pi_i \sum_{j \in S_y} p_{i,j} \right]$$

If compression is performed in this manner, the steady state behavior of the compressed system is the same as the original system. The aggregated matrix can be computed via the method of “stochastic complementation” or via “iterative aggregation/ disaggregation” methods. The former will work on arbitrary matrices but is generally computationally expensive. The latter is most efficient for “nearly completely decomposable” (NCD) matrices (e.g., see [1]). However, the emphasis is always on steady-state behavior, and not on transient behavior. This difference in emphasis can be seen by noting the difference in the choice of weights – the focus in this paper has been on column mass instead of steady state values.

In a sense the compression algorithm presented in this paper is a generalization of steady state aggregation. The steady state matrix is column equivalent for every pair of states, and the column masses, when renormalized, are the same as the steady state probabilities. Thus the compression algorithm is a generalization of the aggregation formula to transient behavior.<sup>10</sup> This leads to the intriguing hypothesis that this new compression algorithm will be more accurate when describing transient behavior, and less accurate for describing steady state behavior. Preliminary results appear to confirm this hypothesis.

**8. Summary and Discussion.** This paper has introduced a novel compression algorithm for probability transition matrices. The output from the algorithm is a smaller probability transition matrix with less states. The algorithm is designed to aggregate arbitrary (not necessarily NCD) probability transition matrices of DTMCs in order to obtain accurate estimations of transient behavior. Thus it appears to fill the gap between existing transient techniques (which focus on CTMCs) and existing aggregation techniques for DTMCs (which focus on steady-state behavior).

There are a number of potential avenues for further expansion of this research. The first possibility is to compress more than two states at once. Multiple-state compression may yield better results, by allowing for a more accurate estimation of error. Another avenue is to derive estimates of how error propagates to higher powers of  $Q_c$ . The current similarity metric is not necessarily a good indicator of the error at

<sup>10</sup>Note that Lemma 4.5 implies that if  $\mathbf{b}_i = q\mathbf{b}_j$  for states  $i$  and  $j$  in  $Q_u$ , then  $\pi_i = q\pi_j$ .

higher powers of  $Q_c$ , although empirically the results are quite good. However, both of these avenues greatly increase the computational complexity of the algorithm.

The comparison with the related work indicates that this new compression algorithm can be considered to be a generalization of the more traditional aggregation formulas. This indicates yet a third avenue for research. If in fact column mass turns out to yield better weights for the weighted average during transient behavior, then it may be possible to smoothly interpolate between column mass and steady state probabilities as the transient behavior approaches steady state. Of course, this presupposes the existence of the steady state distribution, but efficient algorithms do exist to compute these distributions.

The current algorithm also quite deliberately ignores the roles of the priors  $p_i^{(0)}$ , in order to have as general an algorithm as possible. However, if priors are known, then it may be possible to use this information to improve the weighted averaging procedure (see Appendix), thus once again reducing the error in some situations.

Finally, the amount of compression that can be achieved with negligible error is a useful indicator of whether the system is being modeled at the correct level of granularity. If the probability transition matrix is hard to compress, then the system is probably modeled at a reasonable level of granularity. However, ease of compression indicates that the system is being modeled in too much detail. In these cases monitoring the states that are chosen for compression by the similarity metric can yield important information about the characteristics of the system. This approach could be used to characterize systems that are defined by a probability transition matrix but are still not well understood at a higher level.

**Acknowledgements.** I thank Diana Gordon for pointing out that a method for evaluating the compression algorithm was to show that  $(Q_u^n)_c = Q_c^n$ . Diana also pointed out sections that needed mathematical refinement. I also thank the anonymous reviewers for their very constructive comments.

## REFERENCES

- [1] T. DAYAR AND W. J. STEWART, Quasi-lumpability, lower bounding coupling matrices, and nearly completely decomposable Markov chains. *SIAM J. Matrix Anal. Appl.* v18, #2, (1997).
- [2] K. DE JONG, W. SPEARS, AND D. F. GORDON, Using Markov chains to analyze GAFOs. *Foundations of GAs Workshop*. Morgan Kaufmann, (1994), pp. 115 – 137.
- [3] E. C. HOWE AND C. R. JOHNSON, Aggregation of Markov processes: axiomatization. *Journal of Theoretical Probability* v2, #2, (1989), pp. 201 – 208.
- [4] E. C. HOWE AND C. R. JOHNSON, Linear aggregation of input-output models. *SIAM J. Matrix Anal. Appl.* v10, #1, (1989), pp. 65 – 79.
- [5] J. KEMENY AND J. SNELL, *Finite Markov chains*. D. Van Nostrand, New York, 1960.
- [6] A. E. NIX AND M. D. VOSE, Modelling genetic algorithms with Markov chains. *Annals of Mathematics and Artificial Intelligence* #5, (1992), pp. 79 – 88.
- [7] R. SIDJE AND W. J. STEWART, A survey of methods for computing large sparse matrix exponentials arising in Markov chains. *Submitted for publication*, (1996).
- [8] W. SPEARS, K. DE JONG, T. BAECK, D. FOGEL, AND H. DE GARIS, An overview of evolutionary computation. *Euro. Conf. on ML*, Springer Verlag, v667, (1993), pp. 442 – 459.
- [9] W. SPEARS AND K. DE JONG, Analyzing GAs using Markov models with semantically ordered and lumped states. *Foundations of GAs Workshop*. Morgan Kaufmann, (1996), pp. 85 – 100.
- [10] W. J. STEWART, *Introduction to the numerical solution of Markov chains*. Princeton University Press, New Jersey, 1994.
- [11] W. J. STEWART AND W. WU, Numerical experiments with iteration and aggregation for Markov chains. *ORSA Journal on Computing*, v4, #3, (1992), pp. 336 – 350.
- [12] M. VOSE, Modeling simple genetic algorithms. *Evol. Comp.*, v3, #4, (1995), 453 – 472.

**Appendix.** This appendix formally computes  $r_{\{i \vee j\},k}$ . Let  $S_t$  be the random variable for the Markov chain, which can take on any of the  $N$  state values at time  $t$ . Then the short-hand notation  $p_{i,j}$  is really  $P[S_t = j | S_{t-1} = i]$  and  $p_i^{(t)}$  is really  $P[S_t = i]$ . Recall the definition of conditional probability:  $P[A|B] = P[A \wedge B]/P[B]$ . Recall also the definition for “averaging” probabilities:  $P[A] = \sum_l P[A \wedge B_l]$  where the  $B_l$ ’s are mutually exclusive and exhaust the space. The computation of  $r_{\{i \vee j\},k}$  is straightforward. By definition:

$$r_{\{i \vee j\},k} = P[S_t = k | S_{t-1} = (i \vee j)]$$

By definition of conditional probability and by expanding the disjunctions:

$$\begin{aligned} &= \frac{P[S_t = k \wedge S_{t-1} = (i \vee j)]}{P[S_{t-1} = (i \vee j)]} \\ &= \frac{P[S_t = k \wedge S_{t-1} = i] + P[S_t = k \wedge S_{t-1} = j]}{P[S_{t-1} = i] + P[S_{t-1} = j]} \end{aligned}$$

Expanding via the “averaging” of probabilities yields:

$$= \frac{\sum_l P[S_t = k \wedge S_{t-1} = i \wedge S_{t-2} = l] + \sum_l P[S_t = k \wedge S_{t-1} = j \wedge S_{t-2} = l]}{\sum_l P[S_{t-1} = i \wedge S_{t-2} = l] + \sum_l P[S_{t-1} = j \wedge S_{t-2} = l]}$$

Using the definition of conditional probability several times, and the fact that the process is Markovian yields (in short-hand notation):

$$= \frac{p_{i,k} \sum_l p_{l,i} p_l^{(t-2)} + p_{j,k} \sum_l p_{l,j} p_l^{(t-2)}}{\sum_l p_{l,i} p_l^{(t-2)} + \sum_l p_{l,j} p_l^{(t-2)}}$$

What is interesting to note is the time-dependence of this expression. Since the  $p_l^{(t-2)}$  values are not known in advance, one can only make an assumption of “uniformity” (i.e., that the  $p_l^{(t-2)}$  values are the same for all  $l$ ). If this is done the time-independent expression obtained is:

$$r_{\{i \vee j\},k} = \frac{m_i p_{i,k} + m_j p_{j,k}}{m_i + m_j}$$

where  $m_i$  and  $m_j$  are the sums of the probability mass in columns  $i$  and  $j$ . This is what was obtained more intuitively in Section 3.

Now clearly the uniformity assumption will be wrong in general, which explains why the averaging procedure can lead to errors in numerical computations. However, under conditions of row or column equivalence it is trivial to show that both the time-dependent and time-independent forms lead to the same time-independent answers. Thus, under row or column equivalence the uniformity assumption is irrelevant, and the averaging procedure yields no error. Under row and column similarity the uniformity assumption is nearly irrelevant and the time-independent expression is a good approximation for the time-dependent expression. The error of this approximation is computed in Section 5.