Efficient Representation of State Spaces for Some Dynamic Models[1]

Gautam Gowrisankaran
University of Minnesota

This Version: May 1, 1998
Preliminary; Comments Welcome

JEL Classification Code: C63

Abstract:

Many important economic problems require computation over state spaces that are not hypercubes. Examples include industry models of multi-product differentiated product firms, Bayesian learning problems with noisy signals and real business cycle models with heterogeneous agents. These problems have not been analyzed partly because of the difficulty in efficiently representing their state spaces on a computer. I develop a representation algorithm for the state spaces of the above problems, which potentially allows them to be solved with computational methods such as dynamic programming. I find that using this representation reduces the computation time and space by several orders of magnitude relative to a naïve representation.

[1] I would like to thank Ariel Pakes, Ken Judd, Paul McGuire, Souresh Saha, seminar participants at the 1996 Stanford Institute for Theoretical Economics (SITE) conference and two anonymous referees for helpful discussions. Author's address: Department of Economics, University of Minnesota, 271 - 19th Avenue South, Minneapolis, MN 55455 U.S.A., e-mail: gautam@econ.umn.edu; Phone: (612) 625-8310; Fax: (612) 624-0209.

Section 1: Introduction

The goal of this paper is to present data structures and coding mechanisms that can be used to efficiently represent the complex state spaces necessary for solving some discrete state dynamic programming problems. Why should we care about efficiently representing models with complex state spaces? Many important dynamic economic questions have not been adequately answered in part because they require models where the state spaces are not hypercubes, representable by multi-dimensional arrays, or other structures that are easily representable on the computer, such as trees or lists. While these complex spaces can often be inefficiently represented with multi-dimensional arrays by duplicating elements, this wastes time and space. The methods that I present here reduce the required computational time and space for these problems by several orders of magnitude relative to naïve methods. This can make feasible problems that have been too computationally complex to analyze in the past.

In this paper, I provide representations for two different state spaces: the space of possible densities for discretized probability distributions over a given compact interval and the space of possible ownership structures for a set of N differentiated products.[2]

The space of probability densities is useful in examining Bayesian learning problems where the posterior distributions are not in a parametric family. In addition, a composition of two of these spaces can be used to examine macroeconomic real business cycle models with heterogeneous agents, by letting an element represent the distribution of capital stock holdings.[3]

---

[2] I am implicitly restricting my attention to models that are anonymous in the sense that actions are restricted to be a function of the current values of state variables, so that the identity of the firms does not matter.

[3] Heterogeneous agent models are of use in answering many macroeconomic policy questions on optimal taxation and investment policies, because they capture salient features of the data that cannot be explained in a representative agent framework. (See Rios-Rull (1995, p. 98)).

Lastly, this space can be used to examine differentiated products industries,[4] by letting an element represent the distribution of firms in an industry.

The space of ownership structures can be used to analyze industry models of *multi-product* differentiated product firms, by combining the space with the differentiated products space above. These models are useful in answering antitrust policy questions for differentiated products markets.

The idea behind my representation method is to use a table lookup to quickly encode and decode elements of the state space onto a one-dimensional array, without any repetition of elements. I illustrate my method with the product ownership example using 3 products. In this case, I want to represent the set of possible ownership structures for a list of $N = 3$ differentiated products. A naïve data structure for this space would store the set of possible ownership structures in a 3-dimensional array, by listing the owner of each product as a number from 1 to 3. However, this duplicates certain structures. For instance, the structures 1-1-2 and 1-1-3 represent the same state: in both structures, one firm owns the first two products and another firm owns the third product; by the anonymity assumption, it does not matter whether the third product is owned by a firm called '3' or '2'. There are, in fact, only 5 non-duplicated elements in the state space; namely 1-1-1, 1-1-2, 1-2-1, 1-2-2 and 1-2-3. Thus, for this example, the method transforms the state space into a one-dimensional array of size 5.

How much savings in time and space can one achieve by using my methods? The results of this paper show that the duplication of states from a naïve data structure is no mere quibble. For the 3 product example, the savings from this method versus a naïve data structure are more than 5-fold: my data structure has 5 elements, while the naïve one has 27 elements. As I demonstrate in Tables 2 and 4, with most problems of sufficient scale to be realistic, the ratio is several orders of magnitude. In addition, the computation time for an algorithm is often more than linear in the size of the state space, since dynamic algorithms with more states often take more

[4] Ericson and Pakes (1995) analyze such a model.

steps to converge. Thus, the savings in computation time from using this method are even larger than the savings in size.

How efficient are my methods? In terms of space, my methods are efficient for the spaces of ownership structures and of probability distributions, since they do not duplicate any elements for these spaces. For the composition state space of differentiated products multi-product firms (used to analyze differentiated products mergers), my methods are almost efficient, but they duplicate ownership structures for those product structures with less than N different products. Table 4 shows the extent of the duplication, which is relatively small. In terms of time, since any encoding method has to read every component of the state, the encoding time must be at least linearly proportional to the dimension N of the state vector. My methods are efficient (and in fact take no more than the time required for a few additions), because the computational time to encode or decode an element is linear in N.

The remainder of this paper is structured as follows. In Section 2, I detail the encoding method. In Section 3, I discuss the probability distribution space and its applications. In Section 4, I discuss the product ownership space and its application to the multi-product differentiated products problem. Section 5 concludes.

Section 2: A Method for Efficiently Using Complex State Spaces

The idea of my method is to represent complex state spaces in a one-dimensional array. Let the state space be called Z, where Z may depend on some parameter such as the number of products and let o(Z) be the number of elements in Z.[5] There are 3 steps to my method. First, I develop a rule that transforms the set of states in the space into a subset of some multi-dimensional array; let this subset be called X. Second, I find the number of states in X (or

[5] Throughout this work, I use the symbol o(X) to refer to the cardinality of the set X.

equivalently, Z) by performing mathematical induction on the sizes of subspaces of X. Third, I use these measures of cardinality to define a quickly computable bijective (i.e. both one-to-one and onto) encoding function, enc, that maps from X to the one-dimensional representation array and a decoding function that is its inverse. Because the representation is space efficient, the one-dimensional array will have o(Z) elements and hence enc will map from X to the set of integers $\{0,\ldots,o(Z)-1\}$. I now briefly explain each of these steps using the 3 product ownership example (of Section 4). In Sections 3 and 4, I detail these steps for the two models of this paper.

The first step is to define a multi-dimensional array in which the state space Z can be represented, and then to select a subset of elements, X, within this array that is isomorphic to the state space. Recall that for the 3 product ownership example, the dimension is 3, and X has 5 elements, 1-1-1, 1-1-2, 1-2-1, 1-2-2 and 1-2-3. An element is in X when the first component is one, and when subsequent components jump by no more than one from the previous highest component. As I prove formally in Section 4, X includes a representation of all of the elements in Z, but avoids duplication, as intended. To illustrate, 1-3-2 is not in X because the 2^{nd} component is 2 higher than the 1^{st}; instead, the rule selects 1-2-3 to represent this state. Similarly, 1-1-3 is not in X (because the 3^{rd} component is 2 higher than the 2^{nd}), but 1-1-2 is instead.

The second step is to determine the number of elements in X using mathematical induction on the sizes of subspaces of X, which I call Y for the product ownership case. The subsets Y specify the ownership of the first few products and allow for any possible ownership structures over the remaining products. A simple induction argument is able to link the sizes of all of the subsets together by expressing the size of a particular subset as a combination of sizes of subsets where more of the ownerships are specified. Then, the size of X (and hence of Z) can be determined simply by working the induction argument backwards to find the number of elements when none of the ownerships are specified, which is the same as the whole set. Using induction, I also construct an 'encoding table' $o\left(Y\left(h,\hat{N}\right)\right)$, which lists the number of elements in each subspace; in this table, h indicates the number of different firms among the products for which ownership is already specified and $\hat{N}$ the number of products with ownership left to specify.

The third step is to construct a bijective encoding function from the state space onto the set $\{0,\ldots,o(Z)-1\}$, and a decoding function that is the inverse of this function. The logic behind the encoding is to order the states with a dictionary or lexicographic ordering. Then, if some component of a state A is one higher than the same component of a different state B, the function would encode A onto a number that is higher than the encoding of B by exactly the number of states in between A and B. While the details are cumbersome, the idea is exactly the same as what one would use to put integers in numerical order, where for instance, all 3 digit numbers starting with 1-1 would go before all 3 digit numbers starting with 1-2. The difference is that for the multi-product problem there are not always the same number of elements that start with a given number of digits: in the integers there are always ten 3-digit numbers that start with 1-1, 1-2, or any other combination of the first two digits, while here there are 2 elements that start with 1-1 (namely 1-1-1 and 1-1-2), but 3 elements that start with 1-2 (namely 1-2-1, 1-2-2 and 1-2-3). I find the number of elements between two elements using the encoding table. For example, to encode 1-2-1, I must find out how many elements start with 1-1, as 1-1-1 is the first element numerically and these are the elements that numerically lie between 1-1-1 and 1-2-1. To find this number, I ask how many 1-tuples I can find where the previous digits are 1-1. The answer, which is that there are 2 (namely 1-1-1 and 1-1-2), can be found by evaluating $o(Y(1,1))$.[6] Thus, as 1-1-1 is encoded as 0 (as it is the first element), and 1-2-1 is the second element after 1-1-1, 1-2-1 should be encoded as 2.

Two things about this process are of note. First, the second and third steps are necessary only for time efficiency, in the sense that one can construct a space efficient encoding function using only the first step. This can be done by storing a numbered list of the elements in X, and then defining an encoding function that cycles through the list each time to find the number of the element to encode. The problem with this method is that it is very time inefficient: since dynamic programming algorithms do not access elements in any specified order, the access time is proportional to the size of the state space times the dimension of the state vector.

Second, a desirable aspect of this representation is that it provides an easy way of checking whether the encoding and decoding functions have been accurately coded. For a particular element of the state space, one can determine accuracy by encoding the element, decoding the encoded element, and examining whether the computer returns the original value. By doing this for all elements in the space, one can test whether the encoding and decoding programs are inverses of each other, and hence correct.

Section 3: The Probability Density Space

In this section, I analyze the representation of the space of all probability densities on the interval [0,1]. In order to represent this space on the computer, I discretize it: I divide the interval $[0,1]$ into N-1 regions, and allow mass to fall on any of the N endpoints, $\frac{0}{N-1}, \frac{1}{N-1}, \ldots, \frac{N-1}{N-1}$, of the regions. I allow the mass that falls on any of these endpoints to be one of M discrete values, $\frac{0}{M-1}, \frac{1}{M-1}, \ldots, \frac{M-1}{M-1}$, from 0 to 1. Call this discretized space $Z(N, M)$ (or Z for short).

The space Z is of use in modeling Bayesian learning problems. Aghion *et al* (1991) analyze Bayesian learning with a model of a seller who sets a reservation price for her good each period and a potential buyer who then decides whether to accept or reject the good in that period. The buyer has some mean reservation value and a noisy realization of this value each period; the seller has a prior on this mean reservation value; this prior has mass on a compact interval that can be normalized to [0,1]. The state space is the set of possible posterior distributions of the

---

[6] The first '1' is because there is only one firm specified in 1-1, and the second '1' is because there is one

unknown mean reservation price. Because of the noisy signal, the posterior will not belong to any parametric family (except for a few special cases). The state space is thus the set of arbitrary probability distributions on $[0,1]$, which can be discretized into $Z(N, M)$.

This space is also of use in analyzing industry models of differentiated firms. One such example is the Ericson-Pakes (1995) paper. In this paper, firms each have some characteristic, which we can think of as quality of the product. Ericson-Pakes show that in equilibrium, quality will take on values from the set $\{0,1,\ldots,\overline{w}-1\}$ and that there will be a maximum of $\overline{N}$ firms. If there are less than $\overline{N}$ firms active, then the non-existent firms can be represented by a firm with quality 0. The Ericson-Pakes state space (which I call $E(\overline{w},\overline{N})$) is a count of the number of firms that are of each quality level. Since the total number of firms is essentially fixed at $\overline{N}$, one can represent $E(\overline{w},\overline{N})$ as $Z(\overline{w},\overline{N}+1)$, by letting each component of a vector in Z represents the *proportion* of firms with that quality level. For example, if $\overline{w}=4$ and $\overline{N}=5$, then the state $\left(\frac{1}{5},\frac{2}{5},\frac{2}{5},\frac{0}{5}\right)\in Z(\overline{w},\overline{N}+1)$ indicates that one firm has quality 0, two firms have quality 1, two firms have quality 2 and none has quality 3.[7]

Finally, $Z(N, M)$ is of use in analyzing real business cycle dynamic general equilibrium models with heterogeneity of capital stock holdings. In real business cycle models, allowing for heterogeneity of capital stock holdings can add significant realism to the analysis of such important policy questions as optimal taxation or investment policies. With heterogeneous agents,

---

product for which ownership must still be specified.

[7] In previous work, Pakes and McGuire (1994) and Pakes, Gowrisankaran and McGuire (1993) propose and implement an efficient representation for the Ericson-Pakes state space that uses the fact that the state is exchangeable in the order of each firm's competitors. Their representation lists each firm's quality in descending order of quality. In contrast, my representation here (and the Ericson-Pakes notation) lists the number of each firms at each quality level. Thus, the above example would be represented by Pakes McGuire as $(2,2,1,1,0)$. It is easy to show that the two representations are equivalent and hence will have exactly the same number of states. However, the representation here will have a lower dimension (and hence be less cumbersome) if $\overline{N}$ is large relative to $\overline{w}$, while the Pakes-McGuire representation will have a lower dimension if $\overline{w}$ is large.

there will be usually be three kinds of distributional variables.[8] First, there is the aggregate productivity level. Second, each agent is subject to some exogenous idiosyncratic shock. Third, each agent responds to her shock by choosing some value of consumption and a corresponding value of savings and hence of the endogenous state variable (capital) that is unique to her. Both the aggregate productivity level and the idiosyncratic shock are assumed to have two levels and to follow some first-order Markov transition process between the levels. The state space for this type of model is thus the set of all possible joint distributions for these three variables. Agents are heterogeneous in equilibrium because each agent's capital stock is a function of her history of savings decisions which in turn depends on her history of productivity shocks.

For tractability, previous models with heterogeneous agents, such as Krusell and Smith (1995), have approximated that agents' actions depend only on the mean of the distribution of capital stock holdings. Here, I am allowing actions to depend on the entire distribution, and not just on the mean. Because there are a continuum of agents in this model, each with his own independent idiosyncratic shock, the fraction of individuals with low (or high) productivity shocks will be constant in equilibrium. Hence, to keep track of the joint distribution of capital stock holdings and productivity shocks, I need only find the distribution of capital stock holdings for high productivity agents and the distribution of capital stock holdings for low productivity agents. The joint distribution will be a function of these two distributions multiplied by the (constant) equilibrium fractions of each type of shock. To find the entire state space, I can then take the composition of three things: these two densities and the aggregate productivity shock. Generally, the distribution cannot be parametrized as belonging to any particular family of distributions. However, because capital stock holdings can be shown to be bounded for most models, the support of the distribution of the distribution can be normalized to the interval $[0,1]$.

[8] Kydland and Prescott (1982) originally proposed this real business cycle framework.

Thus, the state space for this model will be the composition of two probability density spaces and a 0-1 indicator for the aggregate productivity level, i.e. $Z \times Z \times \{0,1\}$.

In the remainder of this section, I focus solely on the efficient representation of $Z(N, M)$. One can easily derive the efficient representation of a composition of spaces from this to find the state space representation for the real business cycle models.[9] I proceed using the three steps from Section 2.

Step 1:

The first step in the process is to define a multi-dimensional array and a subset of the array that contains a representation of all of the elements in the state space. As the elements of $Z(N, M)$ are multiples of $1/(M-1)$, a natural starting point is to blow up each element of $Z(N, M)$ by $M-1$, resulting in an N-dimensional array indexed from 0 to $M-1$. I now formally define the subset of this array, $X(N, M)$ (or X for short) that will be isomorphic to $Z(N, M)$:

Definition: Let $X(N, M) = \left\{ (x_1, \ldots, x_N) \in \mathbb{Z}^N \;\middle|\; 0 \le x_j \le M-1, \forall j = 1, \ldots, N, \sum_{j=1}^{N} x_j = M-1 \right\}$.

As $X(N, M)$ is just the discretized N-dimensional simplex multiplied by a factor of $M-1$, a map from Z to X that multiplies each element by $M-1$ is easily shown to be bijective.

[9] One can find the efficient encoding of a composition of two spaces as follows: if enc is the efficient encoding for $Z(N, M)$, then a composite encoding function $\text{enc}_C : Z(N, M) \times Z(N, M) \to \mathbb{Z}$ for $(z_1, z_2) \in Z(N, M) \times Z(N, M)$ is $\text{enc}_C(z_1, z_2) = \text{enc}(z_1) \cdot o(Z(N, M)) + \text{enc}(z_2)$. It is easy to verify that enc_C will first map all of the elements with the first z_1, then all of the elements with the second z_1, etc. Thus, provided that enc is efficient, enc_C will be as well.

Thus, unlike the product ownership case, it is apparent that the state space $Z(N, M)$ is isomorphic to the representation $X(N, M)$, and I can proceed directly to the second step.

Step 2:

The next step in the process is to split X into relevant subspaces, and evaluate how many elements are in X, by induction on these subspaces. I do not need a separate set of subspaces Y here, as the induction can be done by examining the cardinality of $X(N, M)$ through induction in N and M, as can be seen below.

Theorem 1: Using induction, the number of elements in $X(N, M)$ can be described as follows:

Base case: $\hat{N} = 1$ or $\hat{M} = 1$. The number of elements is:

$$o\left(X\left(1, \hat{M}\right)\right) = o\left(X\left(\hat{N}, 1\right)\right) = 1.$$

Inductive case: $1 < \hat{N} \le N$ and $1 < \hat{M} \le M$. The number of elements is:

$$o\left(X\left(\hat{N}, \hat{M}\right)\right) = o\left(X\left(\hat{N} - 1, \hat{M}\right)\right) + o\left(X\left(\hat{N}, \hat{M} - 1\right)\right).$$

Proof: I split the proof into assertions of the base case and of the inductive hypothesis.

Base case:

As $X\left(1, \hat{M}\right)$ is the set of all 1-dimensional vectors of integers that sum to $\hat{M} - 1$, $X\left(1, \hat{M}\right)$ has exactly one element, namely $x = \left(\hat{M} - 1\right)$. Thus, $o\left(X\left(1, \hat{M}\right)\right) = 1$. Similarly, $X\left(\hat{N}, 1\right)$ is the set of all $\hat{N}$-dimensional vectors that sum to 0, so it also has one element, namely $x = (0, \ldots, 0)$. Thus, $o\left(X\left(\hat{N}, 1\right)\right) = 1$, as well.

Inductive case:

Let $\hat{N}$ and $\hat{M}$ with $1 < \hat{N} \le N$ and $1 < \hat{M} \le M$ be given. Assume that the theorem holds for all cases $X(\overline{N}, \overline{M})$ (with $1 \le \overline{N} \le N$ and $1 \le \overline{M} \le M$) where either $\overline{N} < \hat{N}$ or $\overline{M} < \hat{M}$. I want to show that it holds for the $X(\hat{N}, \hat{M})$ case.[10] Take any $x \in X(\hat{N}, \hat{M})$, and consider all of the possible choices for the first component of x, x_1. If $x_1 = 0$, then for x to be in $X(\hat{N}, \hat{M})$ the remaining $\hat{N} - 1$ components that must add up to $\hat{M} - 1$. Thus, there are $o(X(\hat{N} - 1, \hat{M}))$ elements of $X(\hat{N}, \hat{M})$ where $x_1 = 0$. Now suppose that $x_1 = 1$. Then there are $\hat{N} - 1$ components that must add up to $\hat{M} - 2$, for a total of $o(X(\hat{N} - 1, \hat{M} - 1))$ elements. Similarly, if $x_1 = 2$, there are a total of $o(X(\hat{N} - 1, \hat{M} - 2))$ elements, etc., for a grand total of $\sum_{m=1}^{\hat{M}} o(X(\hat{N} - 1, m))$ elements. Using the base case hypothesis that $o(X(\hat{N} - 1, 1)) = o(X(\hat{N}, 1))$, then the inductive hypotheses for the $(\hat{N}, 2)$ case, the $(\hat{N}, 3)$ case, etc., up to the $(\hat{N}, \hat{M} - 1)$ case, I obtain that:

$$
\begin{aligned}
o(X(\hat{N}, \hat{M})) = \sum_{m=1}^{\hat{M}} o(X(\hat{N} - 1, m)) &= o(X(\hat{N} - 1, \hat{M})) + \sum_{m=1}^{\hat{M}-1} o(X(\hat{N} - 1, m)) \\
&= o(X(\hat{N} - 1, \hat{M})) + o(X(\hat{N}, \hat{M} - 1)),
\end{aligned}
$$

which completes the proof.■

Table 1, which I call the encoding table, uses Theorem 1 to compute a rectangular grid of $o(X(N, M))$ for several values of N and M.[11] Note that the number of elements in the space is

[10] Given this inductive structure, the induction can be performed in the following order: first, evaluate the first row and column in outwards order, then evaluate the remaining elements of the second row and column in outwards order, then the third, etc.

[11] Using induction, it can easily be shown that Table 3 can be evaluated via a simple combinatorial formula, namely $o(X(N, M)) = {}^{(M+N-2)}C_{M-1} = (M + N - 2)! / ((M - 1)!(N - 1)!)$.

symmetric in N and M. To facilitate the exposition of Step 3, I have added a row of zeros for the $M = 0$ case.

Remark: Table 2 compares the number of states for this space for some values of N and M that might be relevant. The second column of the table indicates the number of states using the efficient data structure and coding method, while the third column indicates the number of states using a naïve data structure which allows for all the elements of the discretized hypercube, instead of only those that lie on the simplex. The results show substantial savings relative to the naïve method. With the efficient method, a problem with 6 probability elements and 20 grid points has 2002 states; hence the overall state space for the heterogeneous agent problem would have $2 \cdot 2002^2$ or roughly 8 million states. Thus, this problem is likely to be feasible using the efficient coding method and state-of-the-art computers.

<u>**Step 3:**</u>

Now that I have shown how to compute the number of elements in Y, and listed these values in Table 1, the next step is to provide an easily computable encoding function that maps from X onto the set $\{0,\ldots,o(X)-1\}$. As I discussed in Section 2, the idea of the encoding process is to use Table 1 to count how many elements of X start with some particular subsequence, and to use this information to add to the encoding value. I choose to encode elements in a dictionary or lexicographic order; i.e. the order is increasing, with the last digit increasing first. Thus, for the $X(6,5)$ case, the first few elements are 0-0-0-0-0-4, 0-0-0-0-1-3, 0-0-0-0-2-2, 0-0-0-0-3-1, 0-0-0-0-4-0, 0-0-0-1-0-3, 0-0-0-1-1-2, 0-0-0-1-2-1, 0-0-0-1-3-0, 0-0-0-2-0-2, 0-0-0-2-1-1, etc.

Before defining the encoding function, I illustrate the logic with an example The idea is to determine how many elements lie in between any two given values for a component, and increment the encoding value by exactly this amount. To illustrate, consider the question of where to encode 3-1-0-0-0-0, for the $N = 6, M = 5$ case. I must first ask which elements lie between 0-0-0-0-0-4 (the first of the 0's) and 3-0-0-0-0-1 (the first of the 3's). The answer is all of the elements that start with 0 (which is all of the $N-1$ tuples that add up to $M-1$) and all of the elements that start with 1 (which is all of the $N-1$ tuples that add up to $M-2$) and all of the elements that start with 2 (which is all of the $N-1$ tuples that add up to $M-3$). By definition, there are $o\left(X\left(N-1, M\right)\right)$ (or 70) of the first, $o\left(X\left(N-1, M-1\right)\right)$ (or 35) of the second, and $o\left(X\left(N-1, M-2\right)\right)$ (or 15) of the last. However, given the induction argument in Theorem 1, I can simplify this expression and obtain that there are $o\left(X\left(N, M\right)\right)-o\left(X\left(N, M-3\right)\right)$ elements. Then, I must ask which elements are between 3-0-0-0-0-1 (the first of the 3-0's) and 3-1-0-0-0-0. The answer is all of the elements that start with 3-0 (which is all of the $N-2$ tuples that add up to $M-4$). By definition, there are $o\left(X\left(N-2, M-3\right)\right)$ (or 4) of these, which can also be modified via Theorem 1 to write that there are $o\left(X\left(N-1, M-3\right)\right)-o\left(X\left(N-1, M-4\right)\right)$ elements.[12] Thus, 3-1-0-0-0-0 is encoded onto element $124\left(=70+35+15+4\right)$.

I now formally define the encoding function, and prove that it is a bijection.

Definition: For $x=\left(x_1, \ldots, x_N\right)$, let the encoding function $\text{enc}: X\left(N, M\right) \rightarrow \left\{0, \ldots, o\left(X\left(N, M\right)\right)-1\right\}$ be defined by:

$$\text{enc}\left(x\right)=\sum_{n=1}^{N}\left[o\left(X\left(N-n+1, M-\left(x_1+\cdots+x_{n-1}\right)\right)\right)-o\left(X\left(N-n+1, M-\left(x_1+\cdots+x_n\right)\right)\right)\right].$$

[12] While the substitution actually adds terms in this case, it reduces the overall number of summation elements in the encoding function from order N^2 to order N, and thus saves time.

Theorem 2: $\text{enc}(x)$ is a bijection from $X(N,M)$ to $\{0,\ldots,o(X(N,M))-1\}$.

Proof: The strategy of the proof is exactly the same as for the multi-product firm case. I first prove by induction that the range of the function for appropriate subsets occupies consecutive blocks, and then apply this to the whole set to show that the function is a bijection. See the appendix for details.■

Given this encoding function, the question remains as to whether the function and its inverse can be easily computed. To encode an element of the state space, 2N elements of the encoding table are accessed, where the table elements to access are determined from the sum of the previous components of the state space element. Thus, an element can be encoded in time that is proportional to N (the dimension of the state vector) by starting with component 1 and ending with component N, and storing the sum of the previous elements at each of the N steps. It is slightly more complicated to decode an element. One has to start with the first component, and find the highest possible value of this component that would yield an encoded value no greater than the actual encoded value. The remainder of the encoded value is then used for the second component, and so on. Because one has to search within each component for the correct value, this version of the decoding function takes time proportional to N^2 to complete. However, using this version, one can construct an array of size $o(X(N,M))$ that lists the (decoded) state space element for each encoded value $0,\ldots,o(X(N,M))-1$. With this array, the decoding time is also linear in the dimension of the state space, after the initial setup cost.[13]

[13] Note that this representation is much more time or space efficient than other representations, such as tree structures or hash tables. While a tree structure would be space efficient, encoding or decoding an element would require a comparison of order $\log(o(X(N)))$ state space elements (see Aho, Hopcraft and Ullman (1983)). Since each state space element has dimension N, each comparison takes time of order N. Thus, the total computation time with a tree structure is of order $N\cdot\log(o(X(N)))$, which is a factor $\log(o(X(N)))$

Section 4: The Product Ownership Space

In this section, I analyze the representation of the space of all possible ownership structures for a set of N differentiated products. Because the products are differentiated, the identity of the products that are owned by each firm (and not just the number of products owned) is relevant. Call the space $Z(N)$ (or Z for short, again).

This space Z is of use in modeling industry models of differentiated products with *multi-product* firms. One can use such multi-product firm models to analyze mergers and antitrust policies in a differentiated products industry and to examine economies of scope from producing multiple products. For tractability, previous analyses of mergers, even those with differentiated firms, have had to assume that the products are homogeneous.[14]

One can view the state space for differentiated product multi-product firms as the composition of two spaces: the space of distinct sets of N differentiated products and the space of possible ownership structures over these products. As discussed in Section 3, Ericson and Pakes (1995) and Pakes and McGuire (1994) use the differentiated products state space, which I call $E(\overline{w}, N)$. Accordingly, I need only focus on efficiently representing $Z(N)$. As in the real business cycle models of Section 3, one can then easily combine the encoding of $Z(N)$ with an existing efficient encoding of $E(\overline{w}, N)$ to form an encoding for the entire space.

There is one important caveat to note. The set of possible ownership structures will only be $Z(N)$ for those elements of $E(\overline{w}, N)$ where there are N distinct products. As an example, if $N = 3$ but there are only two products, there are only two distinct potential ownership structures,

higher than the time for my method. With a hash table, one could approach a time efficient encoding, but only if the size of hash table were much larger than the size of the state space. Thus, a hash table would be less space efficient and no more time efficient.

1-1 and 1-2, and not five. If $N = 3$ and there are three products but the products all have the same quality, then there are only three distinct potential ownership structures, 1-1-1, 1-1-2, and 1-2-3. Because some of the elements in this space are repetitious, the true efficient state space has less elements in it than $E(\overline{w}, N) \times Z(N)$ does. While one can use $E(\overline{w}, N) \times Z(N)$ as the state space, this will result in a larger storage space and time than if one used a completely efficient state space representation. As I detail in Table 4, the size of this redundancy is not very large.

Remark: There is an alternate method of representing this state space, which uses recursion. This method works as follows: first let each of up to N firms own up to M products, and encode each firm separately using the Pakes and McGuire (1994) method. The encoded value of the products for each firm can be thought of as one 'composite product' (that includes up to M separate products) with a characteristic indexed from 0 to $o(E(\overline{w}, M)) - 1$.[15] To complete the representation, encode each firm by recursively using the Pakes and McGuire (1994) method on the previously encoded 'composite products'; this yields a single encoded value for the industry which is indexed from 0 to $o(E(o(W(\overline{w}, M)), N)) - 1$.

If mergers to monopoly are permitted, then one must set $M = N$. In this case, the alternate representation will have far more states. As an indication, if one were looking at the $N = 4$ case with the original representation, then the state space with the alternate representation would include all of the many states where there are 16 products, 4 owned by each of 4 firms. If for some exogenous reason one can limit M to be much less than N (i.e. one can limit the number of products per firm independently

[14] See Perry and Porter (1985) and Gowrisankaran (1996).
[15] Indeed, this same method can be also be used with single-product firms where products have more than one characteristic.

of the number of products), then the alternate representation will likely yield fewer states than the original representation.

I now focus on the efficient representation of $Z(N)$ (which I abbreviate by Z), which is the space of possible ownership structures for a set of N differentiated products. I again proceed using the three steps from Section 2.

Step 1:

I first define a multi-dimensional array and a subset of the array which will contain the elements in $Z(N)$. I use an N dimensional hypercube indexed from 1 to N. An element of this array lists the owner of each product; thus, if $N = 4$, the element 3-1-1-2 indicates that firm 1 owns products 2 and 3, firm 2 owns product 4 and firm 3 owns product 1. Given this array, I now define a subset $X(N)$ (or X for short) of this array that I claim has exactly one element for each element in the set $Z(N)$ of potential ownership structures.

Definition: Let $X(N) = \left\{ (x_1, \ldots, x_N) \in \mathbb{Z}^N \;\middle|\; x_1 = 1, 1 \le x_n \le \max\{x_1, \ldots, x_{n-1}\} + 1, \forall n = 2, \ldots, N \right\}$.

In order to show that the sets Z and X are isomorphic, I define a mapping f from Z to X, and show that f is bijective. Let $x = f(z)$ be defined as follows: find the firm that owns product 1, and put a 1 for x beside all the product numbers that this firm owns, including product 1. Now, look through the products until the first product that does not have a number beside it is reached. Put a 2 beside all the products that this firm owns. Repeat for the next unmarked product, etc., until all the products are marked. By construction, the function f maps into X, as I am only choosing a number that is one higher than any previous number for the markings. Additionally, I prove below that f is bijective.

Theorem 3: f is bijective (i.e. both one-to-one and onto). Therefore, there is exactly one element of X for every element of the state space Z.

Proof: Let me first prove that f is one-to-one, and then that f is onto.

To show that f is one-to-one, take any two distinct elements of Z, z and z' and let $x = f(z)$ and $x' = f(z')$. I want to show that $x \neq x'$. As z and z' are distinct elements, there is some set of products that are all owned by the same firm under z, but not under z'. By construction, these products must have different markings under x and under x'. Thus, $x \neq x'$.

Now to show that f is onto, consider any element $x \in X$. I want to show that $\exists z \in Z$ s.t. $f(z) = x$. Consider the following choice of z: let two goods i and j be owned by the same firm if and only if $x_i = x_j$. Now let $x' = f(z)$. If I can show that $x = x'$ then I will be done, as I will have found an element of Z that maps onto x. Suppose then, by contradiction, that $x \neq x'$. Now consider the first element at which they differ, say the n^{th}, so that $x_n \neq x'_n$. Then, because of the definition of X, x_n and x'_n must be no higher than one higher than the highest of the previous elements, which are the same across x and x'. Thus, it cannot be the case that both x_n and x'_n are higher than any previous element in their sequences. This means that at least one of x_n and x'_n must be a number that appeared before in the sequences; assume without loss of generality that it is x_n. Then, by its construction from x, z specifies that the firm that owns the n^{th} product also owns some other product m (with $m < n$) so that $x_m = x_n$. Now, from the definition of f and x', $x'_m = x'_n$. As x and x' do not differ before the n^{th} element, $x_m = x'_m$. Together, these imply that $x_n = x'_n$, which contradicts our earlier assumption.■

Step 2:

Now that I have defined a set X that is a representation of the state space Z in an N-dimensional array, the next step is to define subspaces of X and evaluate how many elements are in X by induction on these subspaces. Because X and Z are isomorphic, the number of elements in Z is the same as in X. I now define the set of subspaces as a collection of sets $Y(h, \hat{N})$ or Y for short.

Definition: For $h \geq 0$ and $1 \leq \hat{N} \leq N$, let

$$Y(h, \hat{N}) = \left\{ (y_1, \ldots, y_{\hat{N}}) \in \mathbb{Z}^{\hat{N}} \;\middle|\; 1 \leq y_n \leq \max\{h, y_1, \ldots, y_{n-1}\} + 1, \forall n = 1, \ldots, \hat{N} \right\}.$$

The idea behind this definition of Y is that elements of Y form subsequences of elements of X, where the subsequences start in the middle of the sequence and go until the end of the sequence. A subsequence indexed by $(h, \hat{N})$ has dimension $\hat{N}$, and is chosen under the assumption that the previous highest number in the whole sequence (and hence the number of previous different firms) was h. It is easy to verify that $Y(0, \hat{N}) = X(\hat{N})$, and thus I can determine the cardinality of $X(N)$ simply by examining the cardinality of $Y(0, N)$.

To complete step 2 of the process, I now give an inductive formula for the cardinality of $Y(h, \hat{N})$.

Theorem 4: Using induction in $\hat{N}$, the number of elements in $Y(h, \hat{N})$ can be described as follows:

Base case: $\hat{N} = 1$. The number of elements is:

$$o(Y(h,1)) = h + 1.$$

Inductive case: $1 < \hat{N} \le N$. The number of elements is:

$$o(Y(h, \hat{N})) = h \cdot o(Y(h, \hat{N} - 1)) + o(Y(h + 1, \hat{N} - 1)).$$

Proof: I split the proof into assertions of the base case and of the inductive hypothesis.

Base case:

If $\hat{N} = 1$, then there is only one element to choose. By the definition of $Y(h, \hat{N})$, this element must be in the set $\{1, \ldots, h + 1\}$. Thus, there are $h + 1$ possibilities.

Inductive case:

Let $\hat{N} > 1$ and suppose this property holds for sequences with $\hat{N} - 1$ elements and arbitrary values of the previous highest element h. Now consider any element $y \in Y(h, \hat{N})$. By the definition of $Y(h, \hat{N})$, it must be the case that y_1 is in the range $1, \ldots, h + 1$. Now suppose $y_1 = h + 1$. Then, y_2 must be no higher than $h + 2$, and the rest of the sequence must similarly follow the rules set out for $Y(\cdot,\cdot)$, with $\hat{N} - 1$ elements and a highest previous element of $h + 1$. Thus, if $y_1 = h + 1$, there are $o(Y(h + 1, \hat{N} - 1))$ possible elements of the set $Y(h, \hat{N})$. Now, if $y_1 \ne h + 1$, then y_1 must be less than or equal to h. For any of these values, the first remaining element of the sequence must be no higher than $h + 1$, and the rest of the sequence must similarly follow the rules set out for $Y(h, \hat{N})$, with $\hat{N} - 1$ elements and the same highest previous element.

Thus, there are $o\left(Y\left(h, \hat{N}-1\right)\right)$ elements for each of these h values of y_1, or a total of $h \cdot o\left(Y\left(h, \hat{N}-1\right)\right)$ elements if $y_1 \neq h+1$. Adding the number of elements if $y_1 = h+1$ and if $y_1 \neq h+1$ gives the desired result.■

Table 3, which I call the encoding table, uses Theorem 4 to derive the values for $o\left(Y\left(h, \hat{N}\right)\right)$ for the first several elements.[16] To facilitate the exposition of Step 3, I added an $\hat{N} = 0$ column to Table 3 as a column of ones, which is the only set of values that is consistent with the other elements. I have illustrated a triangular grid of the matrix around the origin, because this is the order needed to compute the table with the induction argument. Recall that as $Y\left(0, \hat{N}\right) = X\left(\hat{N}\right)$, the cardinality of the state space with different number of products N can be found by inspecting the first row.

Remark: It is useful to compare the size of the state space using a naïve representation that examines every sequence of N elements with the size using the efficient method. Table 4 indicates these statistics, in columns 2 and 3, respectively. For $N \geq 4$, the results show a savings of one or more orders of magnitude from using the efficient representation.

Table 4 also indicates the size of the overall state space for multi-product differentiated products models, with the assumption that the number of quality levels, $\overline{w}$, is 20. Column 4 gives the number of product quality combinations for $\overline{w} = 20$ for different N; these values are derived from Pakes, Gowrisankaran and McGuire (1993). Column 5 provides the overall number of states using my method, which is the number of ownership structures (column 3) times the number of product quality combinations (column 4).

From column 5, one can see that using my method, the five product problem has 2.2 million states. This is roughly at the bounds of feasibility for current microcomputers using a conventional dynamic programming type algorithm. In order to compute models with more products, one would have to combine these methods with a stochastic approximation algorithm, as suggested by Pakes and McGuire (1996).

As my method does not give the completely efficient state space, I have presented the size of the completely efficient space in column 6.[17] The results show that the savings from the completely efficient state space (column 6) versus my representation (column 5) are approximately 35 percent for the five product case and 50 percent for the six product case.

<u>Step 3:</u>

The next step is to provide an easily computable encoding function that maps from X onto the set $\{0,\ldots,o(X)-1\}$. Just as in Section 3, the ordering of states that I use is a dictionary or lexicographic ordering. Thus, for the six product case, the first few elements are 1-1-1-1-1-1, 1-1-1-1-1-2, 1-1-1-1-2-1, 1-1-1-1-2-2, 1-1-1-1-2-3, 1-1-1-2-1-1, 1-1-1-2-1-2, 1-1-1-2-1-3, 1-1-1-2-2-1, 1-1-1-2-2-2, 1-1-1-2-2-3, etc. . Because the ordering of elements is similar to that used in Section 3, the encoding function presented here is also similar. I illustrate the logic with two examples, again for the six product case.

For the first example, where do I encode 1-2-1-1-1-1? Note from the previous paragraph that 1-1-1-1-1-1 is encoded onto the value 0. Because of the fact that I am encoding in numerical

[16] Note that the space of ownership structures for multi-product firms is equivalent to the space of possible partitions of a set. As shown in Stanley (1997, p. 33-4), there is a complex combinatorics formula for the number of elements in this space that makes use of what are called Stirling numbers.
[17] This was computed using the alternate representation method from the previous Remark.

order, this element has to be encoded immediately after all of the elements that start with 1-1. How many elements are there that start with 1-1? By the definition of Y, there are $o(Y(1,4))$ such elements, or 52 total. Thus, the element 1-2-1-1-1-1 should be encoded as 52.

For the second example, where do I encode 1-1-2-3-1-1? I split this up into two questions. First, where do I encode 1-1-2-1-1-1? This answer is similar to the previous example, and so I encode it as $o(Y(1,3))$, which is 15. Second, where do I encode 1-1-2-3-1-1, relative to 1-1-2-1-1-1? To answer this question, I need to know by how many elements 1-1-2-3-1-1 is in front of 1-1-2-1-1-1. Between them are all the elements that start with 1-1-2-1 and all the elements which start with 1-1-2-2. As there are $o(Y(2,2))$ elements that start with 1-1-2-1 and another $o(Y(2,2))$ elements that start with 1-1-2-2, there are a total of 10+10 elements. Thus, the element 1-1-2-3-1-1 should be encoded as $35\left(= o(Y(1,3)) + 2 \cdot o(Y(2,2))\right)$.

I now formally define the encoding function, and then prove that it is a bijection.

Definition: For $x = (x_1, \ldots, x_N)$, let the encoding function $\text{enc}: X(N) \to \{0, \ldots, o(X(N)) - 1\}$ be defined by:

$$\text{enc}(x) = \sum_{n=2}^{N} (x_n - 1) \cdot o\left(Y\left(\max\{x_1, \ldots, x_{n-1}\}, N - n\right)\right).$$

Theorem 5: $\text{enc}(x)$ is a bijection from $X(N)$ onto $\{0, \ldots, o(X(N)) - 1\}$.

Proof: To prove the theorem, I first prove by induction that the range of the function for appropriate subsets occupies consecutive blocks in their entirety, and then apply this to the whole set to show that the function is a bijection. See the appendix for details.■

I have now provided an encoding function and shown that it is a bijection. Just as in Section 3, values can easily be encoded by using the encoding table. (For this problem, the elements to access are determined from the previous highest element.) Thus, the time it takes to encode a state will again be proportional to N, the dimension of the state vector. Again as in Section 3, decoding must be done by reversing the process, and examining what is the largest first component that is no higher than the encoded value, subtracting the coding for this component, and continuing to the second element, etc. Thus, the decoding process can again be completed directly in time proportional to N^2, or in time proportional to N by using a table lookup.

Section 5: Conclusions

In this paper, I provide efficient representations for two state spaces. I show that these representations can help us analyze industry models of multi-product differentiated products firms, macroeconomic real business cycle models with heterogeneous agents and Bayesian learning models with noisy signals. While these models can answer important economic questions, they have not been computed to date in part because their complex state spaces makes solving them computationally very difficult. My representation method reduces the time and space necessary to compute these problems by several orders of magnitude relative to naïve methods.

The results presented in this paper are of interest primarily because they can be used to compute models that have the potential to answer relevant economic questions. They also provide a general framework that I hope will be useful for analyzing many different dynamic problems with complex state spaces. More generally, the results show that the choice of data structure can make a large difference in the computational feasibility of many dynamic economic problems.

References

Aghion, Phillipe, Patrick Bolton, Christopher Harris and Bruno Julien, 1991, Learning through price experimentation by a monopolist facing uncertain demand, *Review of Economic Studies* 58, 621-654.

Aho, Alfred V., John E. Hopcraft, and Jeffrey D. Ullman, 1983, *Data Structures and Algorithms* (Addison Wesley, Reading, Mass.).

Ericson, Richard and Ariel Pakes, 1995, An Alternative Theory of Firm and Industry Dynamics, *Review of Economic Studies* 62, 53-82.

Gowrisankaran, Gautam, 1996, A Dynamic Model of Endogenous Horizontal Mergers, Mimeo, University of Minnesota.

Krusell, Per and Anthony Smith, 1995, Income and Wealth Heterogeneity in the Macroeconomy, *Rochester Center for Economic Research Working Paper No. 399*.

Kydland, Finn E. and Edward C. Prescott, 1982, Time to Build and Aggregate Fluctuations, *Econometrica.*50, 1345-70.

Pakes, Ariel and Paul McGuire, 1994, Computing Markov Perfect Nash Equilibria: Numerical Implications of a Dynamic Differentiated Product Model, *RAND Journal of Economics* 25, 555-589.

Pakes, Ariel and Paul McGuire, 1996, Stochastic Approximations Methods for Computing Markov Perfect Nash Equilibria, Mimeo, Yale University.

Pakes, Ariel, Gautam Gowrisankaran, and Paul McGuire, 1993, Implementing the Pakes-McGuire Algorithm for Computing Markov Perfect Equilibria in Gauss, Mimeo, Yale University.

Perry, Martin K. and Robert H. Porter, 1985, Oligopoly and the Incentive for Horizontal Merger, *American Economic Review* 75, 219-227.

Rios-Rull, José-Victor, 1995, "Models with Heterogeneous Agents," in *Frontiers of Business Cycle Research*, 98-125, ed. Thomas F. Cooley (Princeton Univ. Press, Princeton, NJ).

Stanley, Richard P., 1997, *Enumerative Combinatorics Volume 1* (Cambridge Univ. Press, Cambridge, UK).

Appendix

Theorem 2: $\mathrm{enc}(x)$ is a bijection from $X(N,M)$ to $\{0,\ldots,o(X(N,M))-1\}$.

Proof: The strategy of the proof is exactly the same as for the multi-product firm case. I first prove by induction that the range of the function for appropriate subsets occupies consecutive blocks, and then apply this to the whole set to show that the function is a bijection.

Lemma: Let $W(\bar{x},N') = \{x \in X(N) | x_1 = \bar{x}_1, x_2 = \bar{x}_2, \ldots, x_{N'} = \bar{x}_{N'}\}$, i.e. $W(\bar{x},N')$ is the set of elements of $X(N,M)$ that have $\bar{x}_1,\ldots,\bar{x}_{N'}$ as their first N' components. Then, $\forall \bar{x} \in X(N,M)$ and $0 \le N' \le N-1$, the encoding function maps the elements of $W(\bar{x},N')$ onto $o(X(N-N', M-\bar{x}_1-\ldots-\bar{x}_{N'}))$ consecutive elements.

Proof of Lemma: I prove the lemma by backward induction in N', starting with $N' = N-1$ and ending with $N' = 0$.

Base case:

Let $N' = N-1$ and consider any $\bar{x} \in X(N,M)$. Then, $W(\bar{x}, N-1)$ is the set of elements which are the same up to the second to last component. In addition to the restriction on the first $N-1$ components, for x to be in $W(\bar{x}, N-1)$, x_N must be equal to $M-1-\bar{x}_1-\ldots-\bar{x}_{N-1}$. Hence the unique element of $W(\bar{x}, N-1)$ is vacuously consecutive and the elements of $W(\bar{x}, N-1)$ map onto $o(X(N-N', M-\bar{x}_1-\ldots-\bar{x}_{N'}))$ $(=1)$ consecutive elements.

Inductive case:

Let $N' < N-1$ and $\bar{x} \in X(N, M)$ be given, and suppose the lemma holds for $W(x, N'+1), \forall x \in X(N)$. I want to show that the lemma holds for $W(\bar{x}, N')$. Take $x \in W(\bar{x}, N')$. Recall that $x_1 = \bar{x}_1, \ldots, x_{N'} = \bar{x}_{N'}$ and consider all of the possibilities for $x_{N'+1}$. If $x_{N'+1} = 0$, then, the $o(X(N-N'-1, M-\bar{x}_1-\ldots-\bar{x}_{N'}))$ elements that start with $\bar{x}_1, \ldots, \bar{x}_{N'}, 0$ will all be encoded adjacent to each other by the inductive assumption. Now consider the elements $x \in W(\bar{x}, N')$ with $x_{N'+1} = 1$. These elements will also be consecutive by the inductive assumption, and the first one will have $x_{N'+2} = 0, \ldots, x_{N-1} = 0$. Examining the encoding function, this element will differ from the first $x \in W(\bar{x}, N')$ with $x_{N'+1} = 0$ only in their $N'+1^{\text{th}}$ summation elements, as the N^{th} element does not affect the encoding. Subtracting the encoding of the first $x_{N'+1} = 0$ element from that of the first $x_{N'+1} = 1$ element, I obtain that they are $o(X(N-N', M-\bar{x}_1-\ldots-\bar{x}_{N'})) - o(X(N-N', M-\bar{x}_1-\ldots-\bar{x}_{N'}-1))$ apart. Applying the reverse of the inductive case of Theorem 1 to the above formula, the $x_{N'+1} = 1$ elements are encoded ahead of the $x_{N'+1} = 0$ elements by $o(X(N-N'-1, M-\bar{x}_1-\ldots-\bar{x}_{N'}))$ elements. As this is exactly the number of elements $x \in W(\bar{x}, N')$ such that $x_{N'+1} = 0$, the first element that starts with $\bar{x}_1, \ldots, \bar{x}_{N'}, 1$ is encoded immediately after the last element that starts with $\overline{x}_1,\ldots,\overline{x}_{N'},0$ and thus the elements for $x_{N'+1}=0$ and $x_{N'+1}=1$ will be consecutive.

I can apply exactly the same logic for the $x_{N'+1}=2$ case: the inductive hypothesis shows that the $x_{N'+1}=2$ elements will be consecutive and the encoding function shows that the first $x_{N'+1}=2$ element will be ahead of the first $x_{N'+1}=1$ element by $o\left(X\left(N-N',M-\overline{x}_1-\ldots-\overline{x}_{N'}-1\right)\right)-o\left(X\left(N-N',M-\overline{x}_1-\ldots-\overline{x}_{N'}-2\right)\right)$. Applying Theorem 1 again, this works out to an increment of $o\left(X\left(N-N'-1,M-\overline{x}_1-\ldots-\overline{x}_{N'}-1\right)\right)$, which is exactly the number of elements $x\in W\left(\overline{x},N'\right)$ such that $x_{N'+1}=1$. Thus, the elements of $x\in W\left(\overline{x},N'\right)$ with $x_{N'+1}=2$ will be immediately after those with $x_{N'+1}=1$. Similarly, the same logic holds also for all values of $x_{N'+1}$, up to $x_{N'+1}=M-1-\overline{x}_1-\ldots-\overline{x}_{N'}$. Thus, the elements of $W\left(\overline{x},N'\right)$ will all be adjacent. Finally, by the definition of X, there are $o\left(X\left(N-N',M-\overline{x}_1-\ldots-\overline{x}_{N'}\right)\right)$ of these elements. Therefore, the lemma will hold for the inductive step.

(End of Proof of Lemma.)

By inspection, one can see that that the first element with $x_1=0$ is mapped onto the number 0. Note also that $\forall\overline{x}\in X\left(N,M\right)$, $W\left(\overline{x},0\right)=X\left(N,M\right)$. Thus, applying the lemma to $W\left(\overline{x},0\right)$, the elements of $X\left(N,M\right)$ are mapped onto $o\left(X\left(N,M\right)\right)$ consecutive elements starting with 0; i.e. they map onto $\left\{0,\ldots,o\left(X\left(N\right)\right)-1\right\}$. As the cardinality of $\left\{0,\ldots,o\left(X\left(N\right)\right)-1\right\}$ is the same as of $X\left(N\right)$, the encoding mapping is a bijection.■

Theorem 5: $\text{enc}(x)$ is a bijection from $X(N)$ to $\{0, \ldots, o(X(N)) - 1\}$.

Proof: To prove the theorem, I first prove by induction that the range of the function for appropriate subsets occupies consecutive blocks in their entirety, and then apply this to the whole set to show that the function is a bijection.

Lemma: Let $W(\bar{x}, N') = \{x \in X(N) | x_1 = \bar{x}_1, x_2 = \bar{x}_2, \ldots, x_{N'} = \bar{x}_{N'}\}$, i.e. $W(\bar{x}, N')$ is the set of elements of $X(N)$ that have $\bar{x}_1, \ldots, \bar{x}_{N'}$ as their first N' components. Then, $\forall \bar{x} \in X(N)$ and $1 \le N' \le N - 1$, the encoding function maps the elements of $W(\bar{x}, N')$ onto $o(Y(\max\{\bar{x}_1, \ldots, \bar{x}_{N'-1}\}, N - N'))$ consecutive elements.

Proof of Lemma: I prove the lemma by backward induction in N', starting with $N' = N - 1$ and ending with $N' = 1$.

Base case:

Let $N' = N - 1$ and consider any $\bar{x} \in X(N)$. Then, $W(\bar{x}, N')$ is the set of elements which are the same up to the second to last product. The definition of the encoding function shows that elements $x \in W(\bar{x}, N')$ differ only in the last summation element. This summation element is equal to a value from the first column of the Table 3 (which are uniformly 1) multiplied by $x_N - 1$. Thus, all elements of $W(\bar{x}, N')$ will be adjacent to each other, with higher last components having a higher encoded value. Finally, by the definition of Y, there are $o(Y(\max\{\bar{x}_1, \ldots, \bar{x}_{N'-1}\}, N - N'))$ elements of $W(\bar{x}, N')$. Thus, the lemma will hold for this case.

Inductive case:

Let $N' < N - 1$ and $\overline{x} \in X(N)$ be given, and suppose the lemma holds for $W(x, N' + 1), \forall x \in X(N)$. I want to show that the lemma holds for $W(\overline{x}, N')$. Take $x \in W(\overline{x}, N')$. Recall that $x_1 = \overline{x}_1, \ldots, x_{N'} = \overline{x}_{N'}$ and consider all of the possibilities for $x_{N'+1}$. If $x_{N'+1} = 1$, then the $o\left(Y\left(\max\left\{\overline{x}_1, \ldots, \overline{x}_{N'}\right\}, N - N' - 1\right)\right)$ elements that start with $\overline{x}_1, \ldots, \overline{x}_{N'}, 1$ will all be encoded adjacent to each other by the inductive assumption. Now consider $x_{N'+1} = 2$. An examination of the $N' + 1^{th}$ summation element of the encoding function shows that the first element that starts with $\overline{x}_1, \ldots, \overline{x}_{N'}, 2$ is encoded $o\left(Y\left(\max\left\{\overline{x}_1, \ldots, \overline{x}_{N'}\right\}, N - N' - 1\right)\right)$ spaces after the first element that starts with $\overline{x}_1, \ldots, \overline{x}_{N'}, 1$. Thus, the first element that starts with $\overline{x}_1, \ldots, \overline{x}_{N'}, 2$ will be encoded immediately after the last element that starts with $\overline{x}_1, \ldots, \overline{x}_{N'}, 1$. The same logic holds for the $x_{N'+1} = 3$ case up to the $x_{N'+1} = \max\left\{\overline{x}_1, \ldots, \overline{x}_{N'}\right\} + 1$ case. Thus, the elements of $W(\overline{x}, N')$ will all be adjacent. Finally, by the definition of Y, there are $o\left(Y\left(\max\left\{\overline{x}_1, \ldots, \overline{x}_{N'-1}\right\}, N - N'\right)\right)$ of these elements. Therefore, the lemma will hold for the inductive step.

(End of Proof of Lemma.)

By inspection, one can see that that the first element with $x_1 = 1$ (namely 1-1-…-1) is mapped onto the number 0. Note also that $\forall \overline{x} \in X(N)$, $W(\overline{x}, 1) = X(N)$. Applying the lemma to $W(\overline{x}, 1)$, the elements of $X(N)$ are mapped onto $o(Y(1, N - 1))$ consecutive elements starting with 0. As $o(Y(1, N - 1)) = o(Y(0, N)) = o(X(N))$, it follows that the elements are mapped onto

$\{0,\ldots,o(X(N))-1\}$. As the cardinality of $\{0,\ldots,o(X(N))-1\}$ is the same as of $X(N)$, the encoding mapping is a bijection.■

Table 1

Encoding table for the space of all probability distributions, $o(X(N, M))$.

	Number of elements in distribution, N								
Number of grid points, M	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1
2	1	2	3	4	5	6	7	8	9
3	1	3	6	10	15	21	28	36	45
4	1	4	10	20	35	56	84	120	165
5	1	5	15	35	70	126	210	330	495
6	1	6	21	56	126	252	462	792	1287
7	1	7	28	84	210	462	924	1716	3003
8	1	8	36	120	330	792	1716	3432	6435
9	1	9	45	165	495	1287	3003	6435	12870

Table 2
Number of states for the space of all probability distributions.

Size of state space		Number of states	
Number of probability regions, N	Number of grid points, M	Efficient coding method	Naïve coding method
4	10	220	10,000
6	10	2002	1 million
8	10	11440	100 million
10	10	48,620	10 billion
4	20	1540	160,000
6	20	42,504	64 million
8	20	657,800	26 billion
10	20	7 million	10.2 trillion

Table 3

Encoding table for the product ownership state space, $o(Y(h, N))$.

Maximum previous element, h	Number of products, N									
	0	1	2	3	4	5	6	7	8	9
0	1	1	2	5	15	52	203	877	4140	21147
1	1	2	5	15	52	203	877	4140	21147	
2	1	3	10	37	151	674	3263	17007		
3	1	4	17	77	372	1915	10481			
4	1	5	26	141	799	4736				
5	1	6	37	235	1540					
6	1	7	50	365						
7	1	8	65							
8	1	9								
9	1									

Table 4

Number of states for models with multi-product firms, with 20 different quality levels.

	Number of states				
Number of products, N	Number of ownership structures, naïve coding method	Number of ownership structures, efficient coding method	Number of product quality combinations	Total number of states for multi-product firms, my coding method	Total number of states for multi-product firms, completely efficient coding method
1	1	1	20	20	20
2	4	2	210	420	400
3	27	5	1540	7700	6670
4	256	15	8855	132,825	100,815
5	3125	52	42,504	2,210,208	1,409,953
6	46,656	203	177,100	35,951,300	18,552,171