OPTIMAL PARTITIONS OF DATA IN HIGHER DIMENSIONS

BRADLEY W. JACKSON*, JEFFREY D. SCARGLE**, AND CHRIS CUSANZA, DAVID BARNES, DENNIS KANYGIN, RUSSELL SARMIENTO, SOWMYA SUBRAMANIAM, TZU-WANG CHUANG***

Abstract. Consider piece-wise constant approximations to a function of several parameters, and the problem of finding the best such approximation from measurements at a set of points in the parameter space. We find good approximate solutions to this problem in two steps: (1) partition the parameter space into cells, one for each of the N data points, and (2) collect these cells into blocks, such that within each block the function is constant to within measurement uncertainty. We describe a branch-and-bound algorithm for finding the optimal partition into connected blocks, as well as an $O(N^2)$ dynamic programming algorithm that finds the exact global optimum over this exponentially large search space, in a data space of any dimension. This second solution relaxes the connectivity constraint, and requires additivity and convexity conditions on the block fitness function, but in practice none of these items cause problems. From the wide variety of intelligent data understanding applications (including cluster analysis, classification, and anomaly detection) we demonstrate two: partitioning of the State of California (2D) and the Universe (3D).

1. Introduction

A common problem in science and engineering is the estimation of a multivariate signal – that is, a function, the domain of which is a multidimensional parameter space – from a set of distributed data points. The data are most commonly of two types: (1) measurements of a dependent variable, or (2) locations of points within the data space. In the latter case the signal of interest is the density of the points, per unit volume in the parameter space, as a function of position in the parameter space. Out algorithms can apply to these and many other data modes, but the point, or event, data mode (2) will be used here as the main example.

In one dimensional time series, an example is the measurement of the varying intensity, or light curve, of an astronomical source by determining the arrival times of individual photons. In other applications one might have a set of points in the plane or in a 3-dimenional space, again representing the overall intensity of a signal, say from a collection of different sources. We also have the following problem in astronomical data analysis. A key example described below is data on the positions of galaxies in 3D space, determined in a redshift survey of perhaps a million galaxies. We want to segment the galaxies into regions that are roughly uniform in density. The high-density regions might represent galaxy clusters or other interesting structures. We start with a partition of the data into cells, one for each galaxy, and consider subpartitions of this starting partition into blocks that are unions of cells. The goal is to find the optimal such partition of the data.

In general, suppose we are given a set of N data points in a bounded region X of $\Re_n$ and let C be a set of N corresponding cells that partition the data space X, one cell for each point. A convenient way to construct such a partition is as the Voronoi tessellation of the point. The Voronoi cell corresponding to a point consists of the part of the data space closer to it than to any of the $N-1$ other points. A block B is defined to be any union of cells from C; in a connected block the corresponding cells are connected.

For a given set of data points our goal is to find the best piece-wise constant function that represents the data. Each partition of the data into blocks defines a corresponding piece-wise constant

*Department of Mathematics, San Jose State University, jackson@math.sjsu.edu, **Space Science Division, NASA Ames Research Center, Jeffrey.D.Scargle@nasa.gov, ***San Jose State University, Center for Applied Mathematics and Computer Science.

function that is constant on the blocks. To quantify what we mean by the best partition we assign a numerical value to each partition and then try to solve the resulting combinatorial optimization problem. Such a quantity goes by many names depending on the application: goodness of fit, risk, cost, objective function, fitness, and many others. Here we simply use the generic term "value", and for example refer to the value of a partition or of a block (since we see below that the value of a partition is defined using the values of its blocks). This quantity is meant to measure how well the corresponding block-wise constant model (*i.e.* constant over the blocks making up the partition) fits the data.

This model optimization can be implemented by maximizing some measure of model fitness, such as its posterior probability. As described elsewhere [Scargle(1998)], by marginalizing all the model parameters except those defining the identities of the blocks, we get a value that depends on the number of points in the block, and the size of the block, but not on the locations of the data points. For any block, B, in P, we denote its size (length, area, volume, ...) by $a(B)$, its population by $n(B)$ = the number of data points in block B, and the block's point density by $n(B)/a(B)$. Under suitable assumptions, amounting to modeling the event detection process as a finite Bernoulli lattice, the posterior for a block, marginalized over the event rate, is the β distribution [Scargle(1998)], eq.(23),

$$f(a(B), n(B)) = \beta(a(B) - n(B) + 1, n(B) + 1) \tag{1}$$

$$= \Gamma(n(B) + 1) * \Gamma(a(B) - n(B) + 1)/\Gamma(a(B) + 2). \tag{2}$$

This formula holds for data in any dimension (the definition of $a(B)$ changes to the appropriate measure of volume for the given dimension). The likelihood of a given partition is the product of the likelihoods over all the blocks in that partition since we assume that the probabilities on each region are independent of each other. Thus the best (most likely) subpartition is one which maximizes

$$V = \prod f(a(B_i), n(B_i)) \ , \tag{3}$$

where the product is over the blocks. We refer to a partition which achieves the maximum value as an optimal partition. The algorithm finds the global optimum; in practice this solution is unique, but there is no guarantee that this always holds. Note that a partition which maximizes V also maximizes its logarithm:

$$W = logV = \sum g(a(B_i), n(B_i)), \tag{4}$$

where $g(a(B_i), n(B_i)) = logf(a(B_i), n(B_i))$. This logarithmic expression is introduced because the dynamic programming algorithm only works if the fitness function is additive over the blocks. Thus our final goal is to find a partition P_{max} which maximizes $W = \sum g(a(B_i), n(B_i))$, summed over all the blocks in the partition.

It is not obvious at this point but our algorithm automatically determines the optimal number of blocks. In stark contrast, in most other analysis methods this parameter must be fixed ahead of time.

2. Finding Optimal Partitions in Dimension 1

Suppose that g is the function that assigns a value to any block; using eq. (4) the value $W(P)$ of any partition P is equal to the sum of the values of its blocks, $\sum g(a(B_i), n(B_i))$, thus satisfying the additivity required by the basic dynamic programming algorithm ([Jackson,Scargle,et.al.(2003)]). Let P_{max} be a partition optimal with respect to W, and let P_0 be any subpartition of P_{max} – that is, a subset of the blocks making up P_{max}. It follows from the additive property that P_0 is an optimal partition of the set that it covers. This is known as the principle of optimality [Bellman(1957)]. Using this principle we showed in [Jackson,Scargle,et.al.(2003)] that dynamic programming gives a highly efficient $O(N^2)$ algorithm for finding the optimal partition of N data points on an interval. Once the optimal partitions of the first j cells, $j = 0, 1, 2, \ldots, i$ are found, the optimal partition of the first $i + 1$ cells can be found by determining which of the the following $i + 1$ partitions has the

maximum value. For each $j = 0, 1, 2, \ldots, i$ consider the optimal partition of the first j cells followed by a single block containing the remaining cells $j + 1, \ldots, i + 1$. Using the principle of optimality we see that the partition with the maximum value in this group will be the optimal partition of the first i+1 blocks. This is the key relation that makes the optimization algorithm so simple.

The incremental way that this algorithm operates on the data also allows it to operate nicely in an on-line mode (performing calculations on the first i data points as we are waiting for the next data point to be transmitted). This mode has been found to be very useful in the rapid detection of changepoints in a data stream, for example to detect x-ray or gamma-ray flares from NASA space-borne observatories.

Dynamic programming has also been shown to be an efficient technique for finding the optimal solution for a variety of other 1-dimensional data analysis problems [Hubert(1997), Kay(1998), Kehagias,Nicolau,Fragkou,Petridis(2004), Quintana,Iglesias(2003), Vidal(1993)]. In most cases one seeks the optimal partition into K blocks, for some fixed K. However, our algorithm is able to compare partitions with different numbers of blocks, so the number of blocks is automatically determined by the data. This feature requires the specification of a prior distribution for K; as described in [Scargle(1998)] a convenient choice is a geometric prior, which acts as an effective penalty against large numbers of blocks. The parameter in this prior in principle is an undetermined parameter, the value of which affects the number of blocks in the optimum solution. In practice it is relatively easy to choose a good value, for example based on simulations using pure noise data sets, and calibrating the prior parameter based on the desired false positive rate. In addition, the solution is relatively insensitive to the value of the parameter, over a rather large range of its values.

There is relatively little literature dealing with finding the optimal partition of a set of data points in higher dimensions. Indeed, for many standard problems in higher dimensions it is known that the problem of finding the optimal partition is NP-complete. Unlike the situation in dimension 1, dynamic programming does not work nearly as well in higher dimensions. One limitation on the efficiency of a dynamic programming algorithm is that one must, at some point, compute the value of each possible connected block. In all dimensions the size of the search space, the set of all possible partitions of the N data cells, is exponential in N. As remarked above, in 1D the dynamic programming algorithm allows an implicitly complete search of this space in $O(N^2)$; but in dimension 2 or higher this trick does not apply directly, and the worst-case complexity of even dynamic programming will be exponential. In these dimensions, one can have a cell adjacent to each of the other $N - 1$ cells and it will be contained in 2^{N-1} different connected blocks and any straightforward dynamic programming algorithm will have to compute the value of each of these blocks.

3. A Branch-and-Bound Algorithm for Data in Higher Dimensions

In higher dimensions we also wanted to find an efficient algorithm for determining the optimal partition of a given set of cells into blocks. In applications there are two related but distinct problems: Find the optimal partition of the data space into (1) arbitrary blocks (that is, with no constraints) and (2) connected blocks. In the former, the cells making up a block can lie anywhere in the data space, whereas in the latter, they must form a connected region. We say that a block B is connected, if and only if for any two cells c, d in B there is a sequence of cells $c = c_0, c_1, c_2, \ldots, c_m = d$ in B such that any two consecutive cells c_i, c_{i+1} are adjacent, $i = 0, 1, \ldots, m - 1$. Contour maps provide an analogy. In the analog of (1) the levels may contain any number of separate contours that correspond to the same value. In the analog of (2), contour curves for the same level that do not intersect each other are considered distinct.

In principle, the two problems can be quite different. In practice, the main difference is that in (1) regions of the data space widely separated from each other can combine their statistical weight to make structures that in (2) would have a smaller value, since the components of a disconnected block would be treated as separate smaller blocks and thus given less overall weight.

Figures 1 and 2 exhibit these concepts for a simple example, treating California counties as data cells and the population density as the dependent variable of interest. In the first figure the blocks, indicated by colors, are constrained to be connected, and in the second this constraint is relaxed. Note that in Figure 2 the block containing counties with densities in the range 12 to 32 persons per square mile form a fragmented (purple) figure – with 5 parts under the edge-based definition of connected, and 4 under the vertex-based one. In many applications one would consider these fragments as effectively different blocks, for example if geographic differences were important. With this re-interpretation of Figure 2 these two optimal partitions are rather similar.

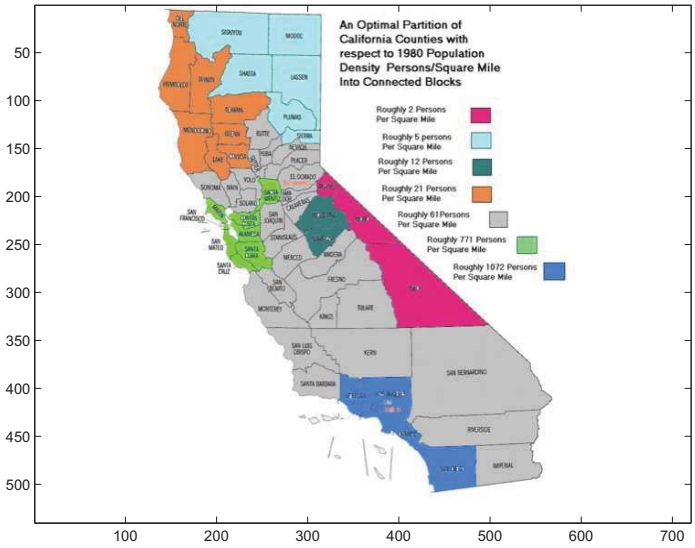

FIGURE 1. Partition of the state of California. The data cells are the counties, and the blocks are connected sets of counties with similar population densities.

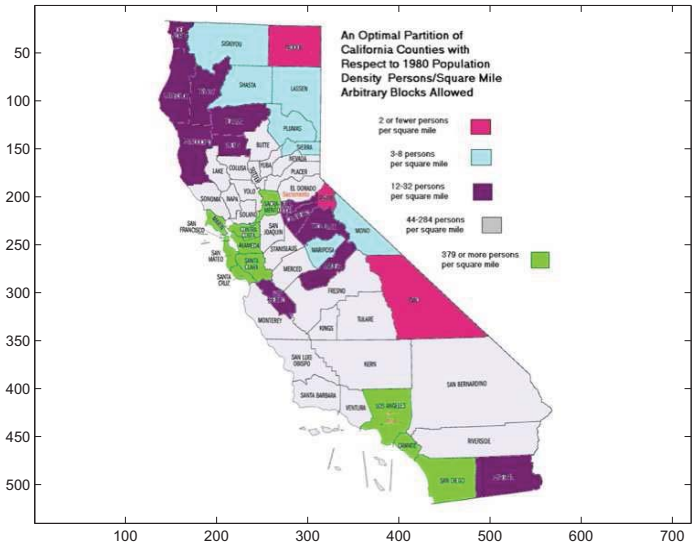

FIGURE 2. Partition of the state of California.

For most applications the unconstrained case, (1) above, seems appropriate: allow all possible partitions into blocks, connected or not. We will exhibit an efficient dynamic programming algorithm for finding the optimal partition in this case, as well as a branch-and-bound algorithm for case (2).

In comparing our partitioning techniques with some of the standard data clustering techniques there are two main issues to consider. Note first that our method compares all partitions of the data, regardless of the number of blocks. The standard techniques for clustering a set of data points [Alpert,Kahng(1997)] into K clusters, so that the maximum cluster diameter (or the sum of the cluster diameters) is minimized, require the number of clusters to be fixed ahead of time. It is somewhat ironic that in similar contexts some authors have not noticed this feature of the dynamic programming approach.

The second comparison issue has to do with computational complexity. For dimension 2 and higher it is known that these standard problems are NP-complete [Garey,Johnson(1979)]. Our $O(N^2)$ algorithm for finding the optimal partition of N data points into arbitrary blocks, in any dimension, solves this exponentially complex problem by searching the solution space of all possible partitions in a way that is exhaustive implicitly rather than explicitly. That our branch-and-bound algorithm is implicit in this sense does not prevent the worst-case complexity from being exponential; hence without special assumptions it is probably not practical for very large problems. We don't yet know if there is an efficient algorithm for finding an optimal partition into block constrained to be connected.

Now turn to a more formal description of the problem and its solution. Let C be any connected set of N cells partitioning a data space X in $\Re_n$. Let P be any partition of X into blocks $B_1, B_2, \ldots, B_M, M \leq N$, consisting of connected unions of cells. Define P^* to be the set of all such partitions of X. Similarly, we define P^{**} to be the set of partitions of X into arbitrary blocks (not necessarily connected). Since the number of cells is finite the number of partitions in P^* or P^{**} is also finite. According to the intermediate density property (see below) the problem of finding an optimal partition of C into arbitrary blocks can be reduced to the 1-dimensional problem of finding an optimal partition of the sorted cells $C_1, C_2, \ldots, C_N$ (in order of monotone density) into blocks assuming that cells C_i and C_{i+1} are adjacent for $i = 1, 2, \ldots, N-1$; the optimal solution for this problem can be found in $O(N^2)$ time using the 1D dynamic programming algorithm described above. In order to apply a branch-and-bound algorithm to finding an optimal partition of P^* we need to be able to find ways of obtaining bounds on the value of a partition without actually computing it. We are searching for the optimal partition in P^*, the set of partitions of the initial cells into connected blocks. To employ the branch-and-bound technique we expand our search to a larger class of problems. We will search for the optimal partition P in P^{**}, the set of partitions of the initial set C of N cells into arbitrary blocks, using the dynamic programming algorithm described above.

Below we list the steps of our branch-and-bound algorithm for finding the optimal partition of C in P^*. The set S is a set of open subproblems that starts with a single problem, that of finding the optimal partition of C in P^{**}. Initially the algorithm's running tally of the current optimum value of the fitness function, called *bestvalue*, is set to negative infinity. As the algorithm progresses, *bestvalue* stores the largest value of a partition in P^* that has been discovered so far. The further steps are as follows:

(1) For some problem T in S, we find the optimal partition P in P^{**}.

(2) If the blocks of the optimal partition are connected, we say that P is a possible optimal solution (POS). Even if the optimal partition P has disconnected blocks then the value of P is an upper bound on the value of an optimal partition in P^*, since P^* is contained in P^{**}. This is the "bounding" part of the branch-and-bound algorithm. If the value of P, $g(P)$, is less than or equal to *bestvalue* then T is removed from S since it cannot lead to a POS with a higher value. If $g(P)$ is greater than *bestvalue*, we define *bestvalue* $= g(P)$. Again T is removed from S and any other subproblem whose upper bound is less than or equal to $g(P)$ is also removed from S. If S is empty, then *bestvalue* is the optimal value of

a partition in P^* and the corresponding partition is an optimal partition, so we stop. If S is nonempty, we continue by returning to step 1 to look at another open problem in S.

(3) If P has disconnected blocks we branch about a pair of adjacent cells i and j. Usually we let i be some cell in a disconnected block and let j be an adjacent cell outside of this block. We consider two subproblems, $T1$, where cells i and j are merged (to form a single cell), and $T2$, where cells i and j are separated (the adjacency between cells i and j is removed). Note that the optimal solution of $T1$ will be the optimal partition in P^* with i and j in the same block. In the optimal solution of $T2$, cells i and j will not be merged directly. To avoid redundancy in the branch-and-bound algorithm one should not consider any future branches which involve merging a pair of cells that result in a cell that contains both i and j since this possibility has already been considered when i and j were merged. We remove T from S and add the two new problems $T1$ and $T2$. We continue by returning to step 1 to look at another open problem in S. This is the "branching" part of the branch-and-bound algorithm.

Eventually every subproblem in S will end up with an associated optimal partition in P^* since we can only branch on an adjacency between two cells once and after branching on every pair of adjacent cells we end up with a partition consisting of nonadjacent connected blocks. The corresponding optimal partition is this partition, which is in P^*. Thus the branch-and-bound algorithm terminates when every subproblem is closed and the best POS discovered so far up to that point is now shown to be optimal. The worst-case complexity of this algorithm is at most $O(2^M)$, where M is the number of adjacencies between the cells in the starting partition. In fact, if we are careful to avoid redundancy as described in the third step above we see that this algorithm is $O(2^N)$, where N is the number of cells in the starting partition. Obviously if the branch-and-bound algorithm is implemented properly we hope that the average complexity is much better than this worst-case complexity.

4. Intermediate Density Property

To implement the branch-and-bound algorithm described above efficiently, we use something that we call the *intermediate density property*. This property allows the one-dimensional dynamic programming algorithm to be used to find the optimal partition of the data into arbitrary blocks (not necessarily connected), even when the data comes from a higher dimension. This property says that if P_{max} is an optimal partition of a collection of cells into arbitrary blocks, with cells c and d in block B, and if e is a cell with density intermediate to the densities of cells c and d, then e must also be in block B. The proof of the intermediate density property uses the strict convexity of the function g that assigns a value (likelihood) to each of the blocks of a partition. If cell e is not in block B as described above, then the convexity allows us to find a better partition, contradicting the fact that P_{max} is optimal. By the way, note that if the event density in a given application, it may be possible to find some other surrogate of model fitness that has this property, and then the cells can be ordered with respect to this quantity.

Definition: We say that a function $g(x, y)$ is strictly convex on a region X if and only if for any $0 < \lambda < 1$, and every pair of points (x_1, y_1), (x_2, y_2) in X,

$$\lambda g(x_1, y_1) + (1 - \lambda)g(x_2, y_2) \geq g(\lambda x_1 + (1 - \lambda)x_2, \lambda y_1 + (1 - \lambda)y_2) \tag{5}$$

with strict inequality holding unless $x_1 = x_2$ and $y_1 = y_2$.

Let $C = C_1, C_2, \ldots, C_N$ be a set of cells partitioning the data space X, and let P represent a partition of the cells into M blocks, $B_1, B_2, \ldots, B_M$. We usually assume that each cell has 1 data point and thus the population of a block is equal to the number of cells that it contains. Suppose we want to find the optimal partition P_{max} in P^{**} where blocks are allowed to be an arbitrary union

of cells (not necessarily connected). We use the logarithmic form of the objective function

$$g(x,y) = log[f(x,y)] \tag{6}$$
$$= log[\beta(x-y+1,y+1)] \tag{7}$$
$$= log[\int_0^1 p^{x-y}(1-p)^y dp] \tag{8}$$

to compute the value of a partition P. Thus the value of P, $W(P)$ is $\sum g(a(B), n(B))$ where the sum is taken over all the blocks B in P. The density of block B is defined to be its population divided by its area, $d(B) = n(B)/a(B)$. The following result is what we call the intermediate density property.

The Intermediate Density Property: Let P_{max} be a partition in P^{**} that maximizes W. Let B be any block in P_{max} and let C_1, C_2, C_3 be cells in C with C_1 and C_3 in B. If $d(C_1) < d(C_2) < d(C_3)$ then C_2 is also in B.

Let $C = C_1, C_2, \ldots, C_N$ be the starting partition of the data space X in $\Re_n$ into cells, sorted by their densities so that

$$d(C_1) \leq d(C_2) \leq \cdots \leq d(C_N). \tag{9}$$

The intermediate density property implies that for some optimal partition P_{max}, every block B in P_{max} is the union of consecutive cells from C. Thus to find an optimal partition in P^{**} we only need sort the cells by their densities and then assuming that C_i is adjacent to C_{i+1}, for $i = 1, 2, \ldots, N-1$, we apply the 1-d dynamic programming algorithm to these cells in order to efficiently find an optimal partition into arbitrary blocks. Since the same function g is used to assign values for a block no matter what dimension the data comes from, then this algorithm can be applied to find the optimal partition into arbitrary blocks regardless of the dimension of the data. If the blocks of a partition are required to be connected then the branch-and-bound algorithm will have to be used to find the optimal partition.

To prove the intermediate density property, we use several lemmas. First we prove (Lemma 1) that the function g which assigns a value to each of the blocks in a partition is strictly convex, using Holder's inequality. Then we use several properties of a strictly convex function to complete the proof of the intermediate density property.

Lemma 1: The function $g(x,y) = log[f(x,y)] = log[\beta(x-y+1,y+1)] = log[\int_0^1 p^{x-y}(1-p)^y dp]$ is strictly convex on the region $X = \{(x,y)|x > 0, y > 0\}$.

Proof of Lemma 1: To show that g is strictly convex we need to show that for any $0 < \lambda < 1$, and every pair of points $(x_1, y_1), (x_2, y_2)$ in X, $\lambda g(x_1, y_1) + (1-\lambda)g(x_2, y_2) \geq g(\lambda x_1 + (1-\lambda)x_2, \lambda y_1 + (1-\lambda)y_2)$, with strict inequality holding unless $x_1 = x_2$ and $y_1 = y_2$. Note that

$$
\begin{aligned}
(10) \quad & g(\lambda x_1 + (1-\lambda)x_2, \lambda y_1 + (1-\lambda)y_2) \\
(11) \quad = \; & log(f(\lambda x_1 + (1-\lambda)x_2, \lambda y_1 + (1-\lambda)y_2)) \\
(12) \quad = \; & log(\int_0^1 p^{\lambda(x_1-y_1)+(1-\lambda)(x_2-y_2)}(1-p)^{\lambda y_1+(1-\lambda)y_2}dp) \\
(13) \quad = \; & log(\int_0^1 [p^{\lambda(x_1-y_1)}(1-p)^{\lambda y_1}][p^{(1-\lambda)(x_2-y_2)}(1-p)^{(1-\lambda)y_2}]dp) \\
(14) \quad = \; & log(\int_0^1 [p^{(x_1-y_1)}(1-p)^{y_1}]^{\lambda}[p^{(x_2-y_2)}(1-p)^{y_2}]^{1-\lambda}dp) \\
(15) \quad \leq \; & log([\int_0^1 p^{x_1-y_1}(1-p)^{y_1}dp]^{\lambda}[\int_0^1 p^{x_2-y_2}(1-p)^{y_2}dp]^{1-\lambda}) \\
(16) \quad = \; & \lambda log(f(x_1,y_1)) + (1-\lambda)log(f(x_2,y_2)) \\
(17) \quad = \; & \lambda g(x_1,y_1) + (1-\lambda)g(x_2,y_2).
\end{aligned}
$$

The inequality in Lemma 1 follows from Holder's Inequality.

Holder's Inequality: For any nonnegative functions $A(x), B(x)$ and real numbers p, q such that for some $0 < \lambda < 1$, $p = 1/\lambda$ and $q = 1/(1-\lambda)$ (equivalently $1/p + 1/q = 1$), we have the following inequality:

$$
(18) \quad \int_0^1 A(x)B(x)dx \leq [\int_0^1 A(x)^p dx]^{\lambda}[\int_0^1 B(x)^q dx]^{1-\lambda},
$$

with equality holding if and only if $A(x)^p/B(x)^q$ is constant almost everywhere on $[0,1]$.

To prove the inequality in Lemma 1 note that if $A(x) = F(x)^{\lambda}$ and $B(x) = G(x)^{1-\lambda}$, then

$$
\begin{aligned}
(19) \quad & \int_0^1 F(x)^{\lambda}G(x)^{1-\lambda}dx \\
(20) \quad \leq \; & [\int_0^1 [F(x)^{\lambda}]^p dx]^{\lambda} \cdot [\int_0^1 [G(x)^{1-\lambda}]^q dx]^{1-\lambda} \\
(21) \quad = \; & [\int_0^1 F(x)dx]^{\lambda} \cdot [\int_0^1 G(x)dx]^{1-\lambda},
\end{aligned}
$$

with equality holding if and only if $F(x)/G(x)$ is constant almost everywhere on $[0,1]$.

Lemma 2: For any positive reals m, n_1, n_2, the function $h(x) = g(x, n_1) + g(m-x, n_2)$ is strictly convex on $I = (n_1, m - n_2 + 1)$.

Proof of Lemma 2: First we note that $g(x, n_1)$ and $g(m-x, n_2)$ are both strictly convex by Lemma 1. It is easy to show that the sum of two strictly convex functions is strictly convex.

Lemma 3: If $h(x)$ is a strictly convex function on $(a,b) \subseteq \Re$, and δ_1, δ_2 are positive real numbers such that $\{x - \delta_1, x + \delta_2\} \subseteq (a,b)$, then either $h(x - \delta_1) > h(x)$ or $h(x + \delta_2) > h(x)$.

Proof of Lemma 3: Assume $h(x - \delta_1) \leq h(x)$. Since h is strictly convex,

$$
(22) \quad h(x) < [\delta_2/(\delta_1 + \delta_2)]h(x - \delta_1) + [(1 - (\delta_2/(\delta_1 + \delta_2))]h(x + \delta_2).
$$

Multiplying both sides of this inequality by $\delta_1 + \delta_2$ we get

$$
(23) \quad \delta_1 h(x) + \delta_2 h(x) < \delta_2 h(x - \delta_1) + \delta_1 h(x + \delta_2).
$$

Then since $h(x-\delta_1) \leq h(x)$, it must be that $h(x+\delta_2) > h(x)$. By similar reasoning, if $h(x+\delta_2) \leq h(x)$, then $h(x-\delta_1) > h(x)$.

Proof of the Intermediate Density Property: Let P_{max} be a partition of C that maximizes W. Let blocks B_1 and B_2 be any pair of different blocks in P, and let C_1, C_2, C_3 be cells in C, with $\{C_1, C_3\} \subseteq B_1$ and $d(C_1) < d(C_2) < d(C_3)$. Assume for contradiction that C_2 is in B_2. If each cell contains a single data point then $a(C_3) < a(C_2) < a(C_1)$. Thus $\delta_1 = a(C_2) - a(C_3) > 0$ and $\delta_2 = a(C_1) - a(C_2) > 0$. We now consider two new partitions P_1 and P_2 created by swapping cell C_2 for each of C_1, C_3 in B_1. Let

$$P_1 = (P - \{B_1, B_2\}) \cup \{B_1', B_2'\} \tag{24}$$

and

$$P_2 = (P - \{B_1, B_2\}) \cup \{B_1'', B_2''\} \tag{25}$$

where

$$B_1' = (B_1 - \{C_3\}) \cup \{C_2\}, \tag{26}$$

$$B_2' = (B_2 - \{C_2\}) \cup \{C_3\}, \tag{27}$$

$$B_1'' = (B1 - \{C_1\}) \cup \{C_2\}, \tag{28}$$

$$B_2'' = (B_2 - \{C_2\}) \cup \{C_1\}. \tag{29}$$

Let P' be the partition $P_{max} - \{B_1, B_2\}$. The value of partition P_{max} in terms of $h(x) = g(x, n(B_1)) + g(a(B_1) + a(B_2) - x, n(B_2))$ is

$$W(P_{max}) = \sum_{B \in P_{max}} g(a(B), n(B)) \tag{30}$$

$$= g(a(B_1), n(B_1)) + g(a(B_2), n(B_2)) + \sum_{B \in P'} g(a(B), n(B)) \tag{31}$$

$$= h(a(B_1)) + W(P'). \tag{32}$$

Similarly $W(P_1) = h(a(B_1) - \delta_1) + W(P')$ and $W(P_2) = h(a(B_1) + \delta_2) + W(P')$. By Lemma 2, $h(x)$ is convex and by Lemma 3, either $h(a(B_1) - \delta_1) > h(a(B_1))$ or $h(a(B_1) + \delta_2) > h(a(B_1))$. Thus either $W(P_2) > W(P_{max})$ or $W(P_1) > W(P_{max})$ contradicting the fact that P_{max} maximizes W. Therefore C_2 is not in B_2 and since B_2 is an arbitrary block different from B_1, it must be that $C_2 \in B_1$.

In [Scargle(1998)] we also have the following global likelihood for data that is prebinned into evenly spaced intervals (with constant rate per bin equal to Λ),

$$\int_0^\infty \Lambda^N e^{(-M+1)\Lambda} d\Lambda = \Gamma(N+1)/(M+1)^{N+1} \tag{33}$$

for a block of N data points in M bins. For prebinned data, the data cells in the starting partition are taken to be the bins which can start with any number of data points. As before the likelihood of a partition is assumed to be the product of the likelihoods of its blocks and taking the logarithm we get a function that satisfies the additive property.

Also in [Scargle(1998)] we have a similar likelihood function for time to spill (TTS) data on an interval. Assuming only every Sth photon is recorded and that $\tau_1, \tau_2, \ldots, \tau_{n-1}$ are the lengths of the data cells (intervals between spill events) then the likelihood that the intensity is constant over a block is

$$[(\prod_{n=1}^{N-1} \tau_n)^{S-1}/\Gamma(S)^{N-1}] \cdot [\Gamma(S(N-1)+1)/(M+1)^{S(N-1)+1}] \tag{34}$$

where $M = \sum_{n=1}^{N-1} \tau_n$ is the length of this block and $S(N-1)$ is equal to the number of data points in this block. The likelihood for a partition of data cells into blocks is thus a constant (depends only on S and N and not on the partition), multiplied by a function that is equal to the likelihood function for the binned data.

Note that the proof of the intermediate density property given here requires that the number of data points in each cell is 1. A similar, though slightly more complicated, proof shows that the intermediate density property is still true for an arbitrary starting partition (cells can have any number of data points). A proof quite similar to that in Lemma 1 shows that the likelihood function for binned data is strongly convex as well and since the likelihood function for binned data is strongly convex we see that the likelihood function for TTS data is also strongly convex. We deduce that the intermediate density property holds for both binned data and TTS data. Thus the algorithms described in this paper can be used to find the optimal partitions for data in equal-spaced bins and for TTS data as well.

Another extension of the intermediate density property shows that if two cells of the starting partition are equal in density, then they are in the same block of the optimal partition. Unfortunately we haven't yet been able to use either of these extensions of the intermediate density property to speed up any of the algorithms described in this paper. The complexity of the branch-and-bound algorithm we described earlier, for finding the optimal partition of a set of data points into connected blocks, is exponential. We suspect that this problem is NP-complete in dimension 2 and higher, but we have not yet been able to prove it.

We conclude with another example, in this case of optimal partitioning in 3D. Figure 3 is based on 3D positions of 146,000 galaxies in the redshift sample of the Sloan Digital Sky Survey [York et al.(2000)], currently the largest survey with over 1 million redshifts measured thus far. Voronoi cells were computed and the optimal partition into constant-density blocks was obtained with the algorithm described above, using a maximum likelihood fitness function for the blocks. The figures shows only a few of the highest density blocks, so that it is easier to see the so-called skeleton of the cosmic web.

5. Future Work

The $O(N^2)$ 1D dynamic programming algorithm described in [**?**]Jackson, and which forms the basis of our approach to higher dimensional prolems, is usable if significant computation power is available for problems with N approaching $1,000,000$. However it would be useful to construct say a N log N implementation. We are pursuing ideas related to the empirical result that currently arriving information does not much, or at all, affect changepoints earlier found earlier in the data stream. This suggests that some sort of sliding-window approach to the analysis will work and be much faster. There would be considerable use for a scheme to optimally partition data on a circle, or a sphere, or other topologies. It would also be of some interest to elucidate the general conditions under which the intermediate density properties holds for more general classes of fitness functions than those considered here.

The authors gratefully acknowledge the Henry Woodward Fund of San Jose State University and the Applied Information Systems Research Program of NASA for funding, and the Institute for Pure and Applied Mathematics of UCLA and the Banff International Research Station for kind hospitality. Jay Norris and James Chiang provided valuable comments.

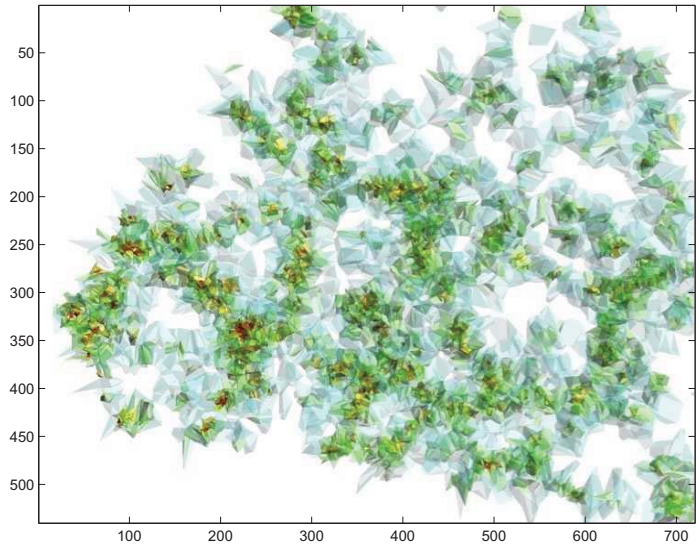

FIGURE 3. Partition of the Universe. Only the highest density blocks are shown, in order to reveal the connectivity of the extremely varied structural features.

REFERENCES

[Alpert,Kahng(1997)] Alpert, C. J. and Kahng, A. B., Splitting Orderings into Multi-way Partitionings to Minimize the Maximum Diameter, Journal of Classification, (14), 1997, pp. 51-74.

[Barry,Hartigan(1992)] Barry, D. and Hartigan, J.A., Product partition models for change point problems, J. Amer. Statist. Assoc., 20, 1992, 260-279.

[Bellman(1957)] Bellman, R., Dynamic Programming, Princeton University Press, Princeton, 1957.

[Garey,Johnson(1979)] Garey, M. and Johnson, D., Computers and Intractability W.H. Freeman and Company, New York, 1979.

[Hubert(1997)] Hubert, P., Change points in meteorological time series, Applications of Time Series Analysis in Astronomy and Meteorology, Rao, T., Priestly, M., and Lessi, O., eds., 1997, Chapman and Hall.

[Jackson,Scargle,et.al.(2003)] Jackson, B., Scargle J., Barnes, D., Arabhi, S., Alt, A., Gioumousis, P., Gwin, E., Sangtrakulcharoen, P., Tan, L., Tun Tao Tsai, An algorithm for optimal partitioning of data on an interval, IEEE Signal Processing Letters, Vol. 12, 105-108.

[Kay(1998)] Kay, S. M., Fundamentals of Statistical Signal Processing: Detection Theory, Englewood Cliffs. NJ: Prentice-Hall, 1998.

[Kehagias,Nicolau,Fragkou,Petridis(2004)] Kehagias, A., Nicolau, A., Fragkou, P., Petridis, V., Text Segmentation by Product Partition models and Dynamic programming, Mathematical and Computer modeling, 39, 2004, 209-217.

[Lee(1997)] Lee, Peter M., Bayesian Statistics: An Introduction, 2nd edition, Arnold, London, 1997.

[Quintana,Iglesias(2003)] Quintana, F., and Iglesias, P., Bayesian Clustering and Product Partition Models, Journal of the Royal Statistical Society, Series B, 65, 557-574, 2003.

[Scargle(1998)] Scargle, J., Studies in Astronomical Time Series Analysis. V. Bayesian Blocks, A New Method to Analyze Structure in Photon Counting Data, The Astrophysical Journal, (504), 1998, pp. 405-418.

[Vidal(1993)] Vidal, R., Optimal Partition of an Interval, Applied Simulated Annealing, Springer-Verlag, New York, 1993, 277-291.

[York et al.(2000)] York, D.G. et al. 2000, Astronomical Journal, 120, 1579