Independence Diagrams: A Technique for Data Visualization

Stefan Berchtold
stb software technologie beratung gmbh

H. V. Jagadish
Univ. of Michigan*

Kenneth A. Ross†
Columbia University

Abstract

An important issue in data visualization is the recognition of complex dependencies between attributes. Past techniques for identifying attribute dependence include correlation coefficients, scatterplots, and equi-width histograms. These techniques are sensitive to outliers, and often are not sufficiently informative to identify the kind of attribute dependence present.

We propose a new approach, which we call *independence diagrams*. We divide each attribute into ranges; for each pair of attributes, the combination of these ranges defines a two-dimensional grid. For each cell of this grid, we store the number of data items in it. We display the grid, scaling each attribute axis so that the displayed width of a range is proportional to the total number of data items within that range. The brightness of a cell is proportional to the density of data items in it.

As a result, both attributes are independently normalized by frequency, ensuring insensitivity to outliers and skew, and allowing specific focus on attribute dependencies. Furthermore, independence diagrams provide quantitative measures of the interaction between two attributes, and allow formal reasoning about issues such as statistical significance.

We have addressed several technical challenges in making independence diagrams work, ranging from the treatment of categorical attributes to visual artifacts of cell-to-pixel mapping. Our experimental evaluation, using both AT&T and synthetic data, shows that independence diagrams allow the easy identification of various kinds of attribute dependence that would be difficult to identify using conventional techniques.

KEYWORDS: Visualization, Two-Dimensional, Histogram, Scatter Plot

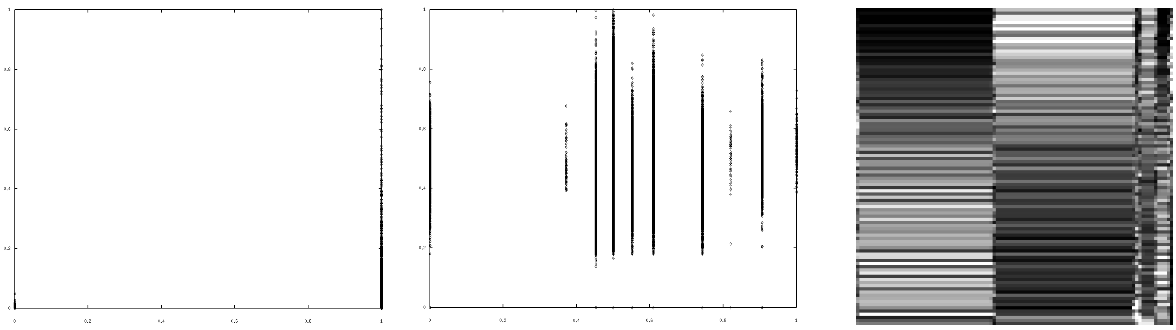
*Supported in part by NSF grant number IIS-00-02356

†Supported in part by a David and Lucile Packard Foundation Fellowship in Science and Engineering, by an NSF Young Investigator Award, by NSF grant number IIS-98-12014, and by NSF CISE award CDA-9625374.

1 Introduction

An important goal in data analysis and data mining is to understand the relationship between two attributes of a given data set. A correlation coefficient can often capture concisely the dependence between two attributes. However, it is possible for two attributes to be highly dependent, and yet have a small correlation coefficient. For example, consider a data set with data points that take values from the set $\{(-1,0),(0,1),(1,0)\}$ with equal probability. The correlation coefficient is zero. Yet, there is a strong dependence: given the value of the X-attribute, the Y-attribute is determined uniquely. There is much information obtainable through visual inspection that cannot be condensed into a single parameter, and such visual data mining is the subject are of this paper.

Given the arguments of the preceding paragraph, it is not surprising that scatterplots are a popular analysis tool among statisticians. Consider the scatterplot shown in Fig.1(a), showing the relationship between two attributes in a data set from AT&T's business.[1] Due to the skew in the distribution of the X-attribute values, most of the data points are piled up along the right edge of the figure, making any patterns very hard to discern. A first attempt at correcting the skew, by taking the logarithm of the X-attribute, results in Fig. 1(b). One can now see that there are only a few distinct values of the X-attribute that occur, but it is hard to determine how often each occurs or what the dependence of the Y-attribute may be.



(a) A scatter plot (b) Log scale on X (c) Independence Diagram

Figure 1: Finding Dependencies Between Two Attributes in a Real Data Set (100x100 pixels)

In this paper we propose a new kind of data visualization called an *independence diagram*. Figure 1(c) shows an independence diagram for the same data – as we shall explain later, it is now apparent that two values of the X-attribute (the two broad columns) account for most of the data points, and that the behavior of the Y-attribute is very different for these two values.

Independence diagrams enable the visual analysis of the dependence between any two attributes of a data set. Our technique is not affected by data skew (as in the example just seen) or by outliers (which are data points with attribute values very different from the bulk of the data, and for some anlytic techniques can unduly affect the model or visual depiction). It does not require any transformation of the data (such as the logarithm applied in Fig 1(b)) to be

[1] To protect the confidentiality of business data, we do not identify the names of the variables, or provide legends to identify absolute values of attributes, in all our examples.

specified by an expert. It provides the ability to focus purely on the dependence between two attributes, stripping away effects due to the respective univariate distributions.

In our technique, we allow both continuous and categorical data, simply because real business data contains both types of attributes. Regions in continuous attributes are natural. Regions in a categorical attribute may be meaningful if the attribute has been ordered in a hierarchical fashion. A categorical "state" attribute might be grouped by geography into "east," "central," and "west," which could be considered as regions over the categorical attribute. (Other groupings might be preferable in this example, such as grouping urban areas together, and non-urban areas together.)

Our basic idea is to divide attribute space using a grid, and represent each grid cell by a count of data points in it. The naive way to do this is to use a simple equi-spaced grid. This is a simple two-dimensional analog of an equi-width histogram, which divides each dimension into equal-sized ranges, and displays a brightness proportional to the count of each cell of the corresponding grid. However, this still leaves us vulnerable to skew and outliers. For the dataset of Figure 1, an equi-width histogram suffers from the same problems as scatterplots, and is not shown. Furthermore, such a representation does not separate one-dimensional effects from attribute dependencies. A high count for a cell may or may not reflect a dependent effect of the two attributes: it could well be the product of two independent one-dimensional distributions.

Instead, we divide the attributes independently into slices (*i.e.*, rows or columns) such that each slice contains roughly the same number of data items (popularly known as an equi-depth histogram) and additionally split slices having a large extension. This might be seen as a combination of an equi-depth and an equi-width histogram. Each intersection of row and column defines a two-dimensional bucket, with which we store a count. We call this kind of two-dimensional histogram an *equi-slice* histogram. We map the equi-slice histogram to the screen such that the width of a slice on the screen is proportional to the number of data items in the slice. Finally, the brightness of a bucket is proportional to the count of a bucket divided by its area.

This kind of visualization is amenable to interpreting dimension dependence effects, because the one-dimensional distributions have already been "normalized". Equally populated slices occupy equal space in the image, meaning that resolution is spent in proportion to the data population, and that the image is not sensitive to outliers. Returning to Figure 1(c), we see that there are two values for the X-attribute that account for roughly 40% of the data each. The left one of the two has a higher concentration of Y-attribute values in the lower half of all Y-attribute values, while the right one has a higher concentration in the upper third of Y-attribute values.

Another informative technique is to *morph* an independence diagram between its standard equi-depth representation (buckets have widths proportional to population) and an equi-width representation (buckets have widths proportional to the attribute range covered). Note that by saying "width", we mean the width of the bucket in the visualization e.g. on the monitor rather than the width in the attribute domain. This approach allows one to trace regions in the independence diagram back into the data space. It sometimes happens that single regions in the independence diagram map to more than one separated region in the data space with small intervening population.

We had to solve several technical problems during the development of our techniques. The choice of histogram boundaries can be problematic when there are several domain values with high cardinality (in a categorical or sometimes even in a continuous domain). For the images used in morphing, both equi-width and equi-depth boundaries need to be included. Mapping buckets to pixels can be difficult, particularly when there are more slices than pixels; rounding effects can induce artificial patterns. The brightness space needs to be chosen so that differences in brightness are apparent, and so that a small number of very highly populated buckets does not result in a picture with a few bright spots and darkness everywhere else.

Outline of the paper: In Section 2 we present the basic concept of a bin-based data representation technique, and describe the many engineering issues that must be resolved carefully to obtain a successful implementation of the concept introduced in the previous section. In Section 3 we present the results of some experiments we ran on a real AT&T data set. Section 4 describes related work, and we conclude in Section 5.

2 Independence Diagrams

There are several steps to generating an independence diagram. First, we determine the boundaries of the slices used in each dimension, and count the number of data items in each grid cell (and slice). Second, we scale each dimension and obtain a mapping from buckets to pixels. Third, we determine the brightness for the pixels in each grid cell. We describe these processes below.

2.1 Attribute Partitioning

There is considerable literature on choosing a good histogram [13, 12, 19]. However, "good" in this context means that the reconstruction error of the histogram is minimized, in *one dimension*. Unfortunately, for *two dimensional* visualization purposes, this may not be a good optimality criterion.

For instance, consider an attribute that has values distributed in a broad "plateau" with sharp shoulders. Most histogram bins in an optimal histogram will be devoted to the shoulders, with a single large bin for the bulk of the data in the plateau. If there are interesting patterns in the other dimension that happen to be within the range of the plateau then we will completely miss this in the visualization.

Categorical attributes are problematic with respect to a histogram representation. If the number of categories exceeds the number of buckets we can have along some axis, then multiple categories will have to be pooled together into one bucket, and there is no really good way to do this – some semantic information is required to aggregate categories into meaningful larger categories. Fortunately, the number of categories in most applications is fairly small. We restrict ourselves to the case where the number of categories is no greater than the number of buckets available. Clearly, we must have each category in a different bucket. Also, there is no point splitting the elements in a single category into multiple buckets. Therefore, the bucketization should exactly map the categories to buckets for any categorical attribute.

An extension of the categorical attribute problem occurs when we have an attribute defined over an appropriate interval scale, but with a very high peak at one (or a few) specific values. For instance, the number of minutes per month of international long distance calls made by AT&T customers is an integer valued attribute with a fairly large integer range, but with a huge peak at zero, since most customers make no international calls in any given month. In such a situation, it is beneficial to treat zero as a "categorical" attribute value, and give it its own bucket, irrespective of how we bucketize the rest.

An equi-depth histogram [17] places bucket boundaries at quantiles such that an equal number of data points fall in each bucket. Such a histogram has the nice property that each bucket is equally important (in terms of the number of points in it). However, there are some potential pitfalls, having to do with the very large ranges of attribute value being mapped into a common bucket, and with quantile boundaries that separate points with identical attribute values and assign them to distinct buckets.

We propose a modified equi-depth histogram that also handles categorical attributes appropriately. Our two criteria are:

(a) The width of any bucket should be smaller than a given constant w_{max}. A good heuristic choice for w_{max} is $w_{max} = \frac{M-m}{n}$ for some constant n, where M and m are the largest and smallest values of the attribute respectively. This is the criterion used to build an equi-width histogram.

(b) The depth (count of items) in any bucket should be smaller than a given constant p_{max}. A good heuristic choice for p_{max} is $p_{max} = \frac{N}{n}$ for some constant n, where N is the cardinality of the data set. This is the criterion used to build an equi-depth histogram and therefore corresponds to bucketizing into n quantiles. Exceptions to this rule are allowed when the number of data items with a single value of the attribute exceeds this limit.

Furthermore, if the number dis_j of distinct values observed in attribute j in the database is smaller than n, we should output no more than dis_j buckets, and we should give each distinct value a single bucket.

The basic idea of the algorithm is first to sort the database according to attribute, j, of interest. Then, in one pass, we consider each data item in turn, and place it in the current bucket, if we can do so without violating the criteria above. When either criterion is violated, a new bucket must be started and immediately becomes the new current bucket). To handle the situation that a value occurs more frequently than the specified minimum depth, we delay finalizing the bucket boundary until we see a larger value of the attribute. At this point, we can move the bucket boundary to the left or right, if needed, to obtain buckets closest to even without splitting points with identical values. For example, consider a data set with values (0, 1, 2, 2, 2, 2, 2, 4, 5) and a target depth, $p_{max} = 3$. Without adjustment, the buckets obtained would be (0, 1, 2), (2, 2, 2), (2, 4, 5), and the value 2 would be contained in all three buckets. Our algorithm refuses to split after having seen the first 2 and instead waits until it sees the 4 to output buckets (0,1) and (2,2,2,2,2). It then continues processing and produces the final bucket (4,5).

As a consequence, the number of buckets produced by our algorithm is not fixed. In addition, some special care is required to avoid creating fully n buckets when there are less than n distinct values (or populated categories).

For the description of the algorithm, we assume the following prerequisites: We are given a database `DB` containing N d-dimensional data items. Each of the d attributes of the data items may be of type *categorical*, *integer*, or *float*. If an attribute is categorical, we transform the set of categories into an equivalent set of integers. For example, the categories $\{red, green, blue\}$ are mapped into the set of integers $\{0, 1, 2\}$. This allows us to focus on numeric values in the bucketization algorithm. We denote the j-th attribute of a data item v by $v[j]$. Let M_j be the maximum value of the domain of attribute j (for an integer attribute, this would be MAXINT). Let m_j be the minimum, accordingly. The algorithm of Figure 2 can be applied if the number of distinct values in the database is relatively large i.e. larger than n, the number of buckets. This will be true for most numerical data and some categorical data.

In order to speed-up the bucketization process, the sorting step can be replaced by an adequate algorithm for finding (or approximating) quantiles. This quantile selection step can be done with a single pass through the data set [3, 2]. In this case, the algorithm slightly changes and is given in Figure 3.

This algorithm is of linear complexity and, in fact, only requires a single pass through the data set. Nevertheless, in case of categorical data, where we frequently only have a few categories, a much simpler and even more efficient algorithm, given in Figure 4, can be applied. Note that the fact that an attribute has only a few distinct values is known in advance so that it is clear which algorithm to apply.

2.2 Determining the 2-dimensional Populations

As a next step of our visualization algorithm, we have to determine the 2-dimensional distributions of the database. Thus, for each pair of dimensions u and w, $0 \leq u < w < d$, we compute a 2-dimensional array $P_{u,w}$ such that each entry $P_{u,w}[x][y]$ contains the number of data items that fall into a corresponding two-dimensional range.

For each data point, a simple binary search along each attribute is used to determine the bucket along this attribute dimension that this data point falls into. Once we have determined the buckets for each attribute of the data point, we can increment the counts for the corresponding grid cell for each pair of dimensions, *all in one pass*.

2.3 Scaling

Data distributions are often skewed, and it is not clear that having each axis in the visualization linearly represent the attribute value space is ideal. It is common for a statistician to transform an attribute value by applying some mathematical function, such as taking the logarithm, the square root, or the exponential of a data value, so that attribute values are "properly scaled" to produce the maximum information value to the (visual) analyst. The specific function used is selected subjectively based on the circumstances – there is limited formal procedure.

A better alternative is an *equi-depth* scaling of each axis. The idea is that each unit of length along the axis represents an equal number of data points, rather than an equal range in attribute value. Thus, outliers get squeezed into a small portion of the visual space, and the bulk of the area is devoted to the bulk of the points, even if these are clustered tightly. A desirable consequence is that patterns and correlations within this bulk of points now become easily visible – something that is just not possible in the presence of outliers. Another benefit is that all dimensions can be interpreted in a similar fashion.

The scaling of an axis for visualization is completely independent of bucket boundary selection – one could have an equi-depth scaling of the type proposed here and yet have equi-width histograms for bucket boundary placement. With equi-depth scaling, the boundaries of an equi-depth histogram are all equally spaced whereas the boundaries of an equi-width histogram are spaced non-uniformly depending on the density of data points in the bucket. In terms of an

```
float * bucketization(Database *DB, int j, int n, float max_depth, float
max_width)
// the number of distinct values in attribute j is larger than n
{
    int i, k, depth, width;
    float *split_val, act_v, old_v;

    split_val = new float[n+1];

    // sort database ascending according to attribute j
    DB->sort_ascending(j);

    k = 1;
    split_val[0] = old_v = DB->m(j); //min.  value of attribute j
    depth = 0;
    // scan through the database
    for (i = 0; i < DB->N; i++)
    {
        act_v = (DB->get(i))[j];
        width = act_v - old_v;
        depth ++;

        // check if a new bucket shall begin
        if (width > max_width || depth > max_depth)
        {
            // scan to the end of this value
            while ((i+1) < DB->N && (DB->get(i+1))[j] == act_v)
              i++;

            // create new bucket
            split_val[k] = old_v = act_v;
            depth = 0;
            k++;
            if (k >= n)
              return NULL; // ERROR!
        }
    }
    split_val[k] = DB->M(j); //max.  value of attribute j
    return split_val;
}
```

Figure 2: Algorithm for many distinct attribute values.

```
float * no_sort_bucketization(Database *DB, int j, int n, float max_depth,
float max_width)
// the number of distinct values in attribute j is larger than n
{
    int i, k, l, depth, width;
    float *split_val, *quantiles, num, act_v, old_v;

    split_val = new float[n+1];

    // determine approximate quantiles according to attribute j
    quantiles = DB->determine_approximate_quantiles(j, n);

    split_val[0] = old_v = DB->m(j);
    /* for each quantile, check if it has a width smaller than max_width.
    If not, then insert an additional split value in the middle of the
    bucket.  If two quantiles are identical because of a larger number of
    identical values than ignore one.  */
    for (i = 0, k = 1; i < n; i++)
    {
       // check for identical quantiles
       if (quantiles[i] == quantiles[i+1])
          continue;
       width = quantiles[i] - old_v;
       // check for max_width criterion
       if (width > max_width)
       {
          num = width / max_width;
          for (l = 0; l < (int) num; l++)
            split_val[k++] = quantiles[i] + l * (quantiles[i+1] -
            quantiles[i]) / num ;
       }
       else
          split_val[k++] = quantiles[i];
       old_v = quantiles[i];
    }
    split_val[k] = DB->M(j);
    return split_val;
}
```

Figure 3: Algorithm variant with quantile estimation.

```
float * categorical_bucketization(Database *DB, int j, int n)
// the number of distinct values in attribute j is smaller than n
{
    int i, dist_n;
    float *split_val, *dist_val;

    split_val = new float[n+1];

    // determine distinct values of attribute j and store them in dist_val
    DB->distinct_values(j, &dist_n, dist_val);

    // the first split value of the bucketization must be m_j
    split_val[0] = DB->m(j);
    for (i = 0; i < dist_n; i++)
        split_val[i+1] = dist_val[i];

    return split_val;
}
```

Figure 4: Algorithm for few distinct attribute values.

effective user interface, it is critical that axes are labeled clearly when there is non-uniform scaling.

2.4 Quantization Problems

Given a bucketization for an attribute, we would like to display the bucket with a width proportional to the number of data points in it. However, we are constrained in electronic displays to widths measurable as integral numbers of pixels. Utmost care is required in this quantization, particularly since one is often working in a regime where the number of buckets along an attribute dimension is comparable to the number of pixels.

A naive quantization would simply perform "rounding" moving each bucket boundary to the closest pixel boundary. The problem is that buckets could "disappear" after rounding (for example, a bucket that went from 1.6 to 2.4 would be rounded to go from 2 to 2). At the same time, a much smaller bucket (such as one from 2.4 to 2.6) could grab an entire pixel (being rounded to go from 2 to 3).

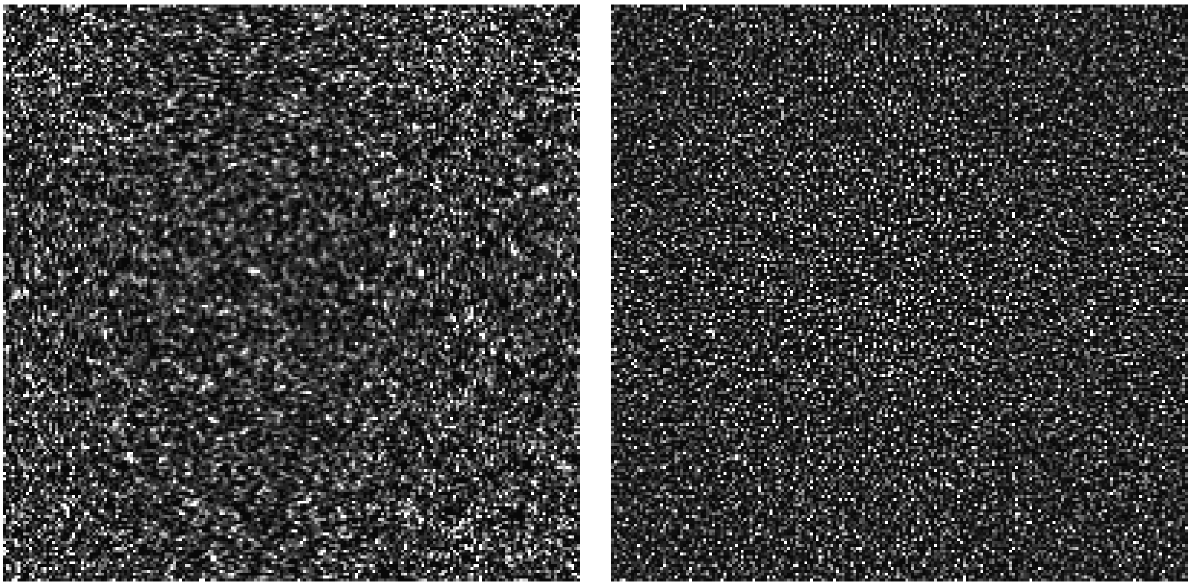
An alternative we considered seriously was to round "up" consistently, devoting a fraction of an additional pixel to each bucket. There is no information loss, and we found that the eye did not pay too much attention to the increased widths, even for very small buckets, in most cases. However, for highly skewed distributions, all the small buckets occur clustered together, and in this case, the cumulative error can become quite significant.

Among some alternatives, the following algorithm showed the best results: For each bucket, we compute the exact (fractional) pixel width, and the corresponding density (of data points per pixel). Pixels completely included in the bucket all are assigned the same data point count. Pixels at the boundary get partial contributions from two (or more) buckets. The result is to "soften" the bucket boundaries by one-pixel wide "transitions".

However, even this approach has a drawback. Consider the following example: Given the split values (0, 1, 2, 3, 4, 5), the one-dimensional bucket populations (10, 10, 10, 10, 10), and an image size of 4 pixels along each dimension. Thus, the bucket widths (measured in pixels) are (0.8, 0.8, 0.8, 0.8, 0.8). Consider an equi-slice histogram shown below that has to be mapped to the 4 by 4 pixel matrix. The resulting pixel matrix is shown on the right.

$$\begin{array}{cc} \text{Equi-slice Histogram} & \text{Image} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 10 \\ 0 & 0 & 0 & 10 & 0 \\ 0 & 0 & 10 & 0 & 0 \\ 0 & 10 & 0 & 0 & 0 \\ 10 & 0 & 0 & 0 & 0 \end{bmatrix} & \begin{bmatrix} 0 & 0 & 1.9 & 10.6 \\ 0 & 2.5 & 8.1 & 1.9 \\ 1.9 & 8.1 & 2.5 & 0 \\ 10.6 & 1.9 & 0 & 0 \end{bmatrix} \end{array}$$

If we focus on the main diagonal of the image, we observe the values (10.6, 8.1, 8.1, 10.6). Obviously, this is not the desired result because there is a systematic bias introduced which could lead to the assumption that the data is less well correlated in the middle of the data space. This is an intrinsic problem of trying to map k buckets to l pixels and any visualization technique will have to handle the problem. To see how problematic this single effect can be, consider Figure 5(a), which shows a uniformly distributed data set with two perfectly independent attributes. Due to bucket-pixel synchronization, we can observe a smooth area at the center of the image and high contrast areas in the corners.



(a) Artifact (b) Uniform Distributions

Figure 5: Potential artifacts in an independence diagram (100x100 pixels).

We therefore dither, randomly shifting a bucket from its original position by at most half a pixel. This operation removes the bias without introducing any significant error; see Figure 5(b) which is the dithered version of Figure 5(a). This leads us to the pixel mapping algorithm in Figure 6.

```
float ** map_to_pixels(int n_x, int n_y, int n_u, int n_w, float
*split_val_u, float *split_val_w, int **buckets)
{
    int a, b, i, j, x, y, ratio;
    float **image = ...; // malloc 2-dim.  array and initialize with 0

    // for each bucket
    for (i = 0; i < n_u; i++) {
        for (j = 0; j < n_w; j++) {
            // randomly shift bucket buckets[i][j]
            ...;

            // determine the pixels affected by the bucket
            lx = ...; ux = ...; // from pixel lx to pixel ux
            ly = ...; uy = ...; // from pixel ly to pixel uy

            // for each affected pixel
            for (a = lx; a ≤ ux; a++) {
              for (b = ly; b ≤ uy; b++) {
                /* determine ratio of overlap based on the split values and
                the total number of buckets vs.  the total number of pixels
                */
                ratio = ...;
                image[a][b] = ratio * buckets[i][j];
              }
            }
        }
    }

    return **image;
}
```

Figure 6: Algorithm for mapping buckets to pixels.

2.5 Shading Pixels

We now need to map a count value for a cell into a grey-scale value for the display. Our goals are as follows. (a) There should be a single, simple grey-scale map for the whole picture. (b) This grey-scale map should adapt to the distribution of the relative populations in the buckets. (c) The grey-scale value should be more perceptible if the relative population is higher or lower than the average relative population. (d) Grey-scale values should not be wasted: an all-black image with a single white spot should not occur.

The most straightforward way to scale from counts to pixels is to map the highest count to white, the lowest count to black, and scale all other counts in linear proportion. In practice, this approach leads to mostly-black images because often there are counts that are outliers in the count-space. To overcome this effect, we sort the counts in an image, and use the 5% and 95% quantile count values as the darkest and lightest values respectively. The precise quantiles can be adjusted; in a production-quality interactive tool these values should be adjustable via the user interface. The 5% and 95% quantile values gave good results on most (but not all) of our data sets.

A more sophisticated approach might be to use a grey-scale mapping table as proposed in [16]. Such tables have been constructed so that visual gradations in brightness correspond subjectively to humans as linear gradations in some scale. We tried such mappings, but found that they did not give sufficiently good grey-scale distinction between buckets. Good grey-scale distinction is more critical in this application than subjective interpretation of grey-scales. Similar issues arise if we attempt to create a color image rather than a grey-scale image.

A legend is generated to explain the numerical value of each grey-scale value. This legend associates a grey-scale value with a ratio of the number of data points in a bucket to the number of points that would be expected in that bucket if the dimensions were independent. Both high ratios (white) and low ratios (black) are potentially interesting. Fig. 7 shows an independence diagram for a synthetic data set, along with its legend. The attributes are independent, except for a range of X-attribute values. The black "finger" in the upper half of the range indicates the very low ratio in that region. The slight brightening of the lower half shows that there are somewhat more points than expected.

2.6 Morphing

We could choose to map a given two-dimensional equi-slice histogram in an equi-width manner, or an equi-depth manner. In fact, we could map to an image in which the width for the given bucket is interpolated between two visualization modes. If we use a linear interpolation of the bucket width, a single parameter is sufficient to generate a sequence of images between an equi-width and an equi-depth mapping. By showing the series of images as a movie, we can "morph" one visualization mode into another.

Morphing the images in this way can aid the interpretation of the data. Morphing helps when the equi-depth picture shows a high concentration around a certain quantile range, but the equi-width picture indicates that this concentration actually comes from two separate subranges in the data space with low intervening population. Morphing also helps identify some small-scale effects that get swamped by larger effects in the equi-depth picture. Examples of the use of morphing to understand these phenomena can be downloaded from "http://www.research.att.com/~berchtol/KDD98/example2.html".

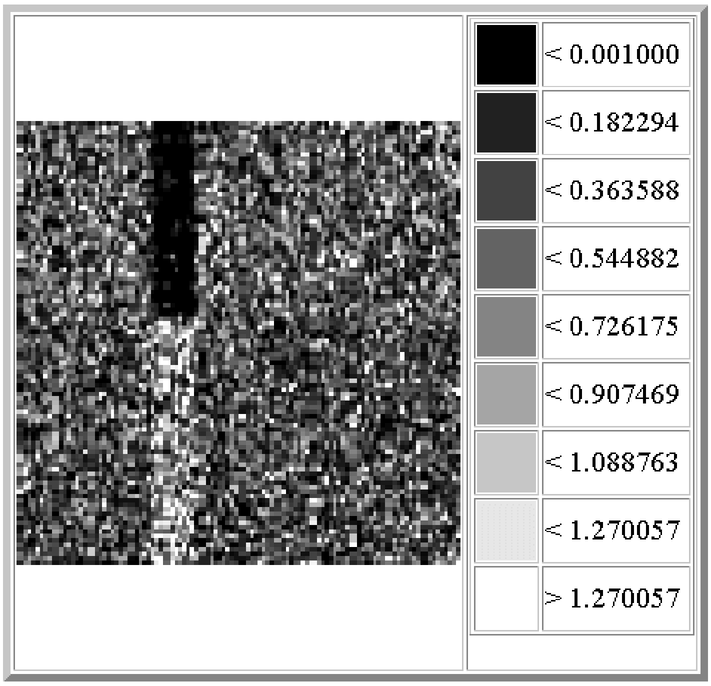


Figure 7: An independence diagram with legend (100x100 pixels).

3 Experiments

A formal experimental investigation of any visualization technique would require experimenting with human subjects. Such experiments are prone to biases due to differences in ability and experience between subjects, and are notoriously difficult to conduct rigorously. Such work is beyond the scope of the current paper.

Instead, we present several examples, including an extended example based on a large AT&T data set of more than 800,000 tuples. We use these examples to demonstrate that effective data visualization is possible using our techniques.

Our system was implemented in `C++` on an SGI Unix machine. The time to compute the one-dimensional partitions of the 13-dimensional data set was about 5 minutes (including a sort and one pass through the data set for each dimension, followed by one pass to compute counts for all grid cells in all 78 (=13*12/2) independence diagrams). After this preprocessing step, the time to produce a single independence diagram and write it to disk in PPM format was about 0.23 seconds. Thus, it took about 5.5 minutes to produce *all* 78 independence diagrams. Displaying a single image on screen can be done in less than a second.

To interpret independence diagrams, we must remember that each vertical slice of the image having width w (measured as a fraction of the image width) exactly represents the same number of data items, namely a proportion w of the total data set. A similar property holds for horizontal slices. This makes independence diagrams amenable to a quantitative analysis and can even be used to extract rules of the form:

> "While a fraction x of all points lie in range R_A in attribute A, among those data points lying in range R_B in attribute B, the fraction is y."

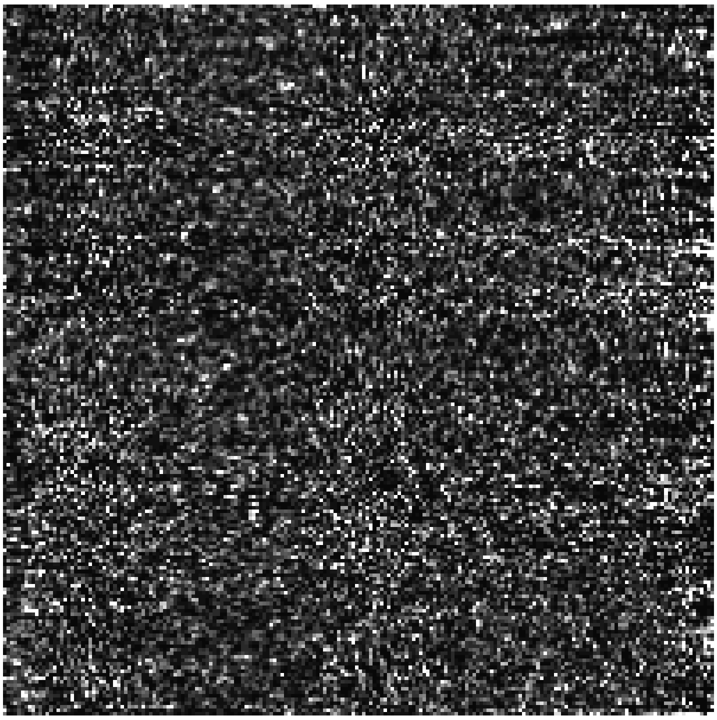


Figure 8: An independence diagram showing real independent data. (100x100 pixels)

In order to get such a rule, a user selects – based on visual characteristics – a rectangular "interesting" region of the image; for example one might choose R_A and R_B to define a region around a bright spot (or a dark spot) in the image. The brightness of this region directly gives us $Pr(R_A \cap R_B)/(Pr(R_A) * Pr(R_B)) = Pr(R_A|R_B)/Pr(R_A) = y/x$. The width of the region gives us $x = Pr(R_A)$. Thus, one can also generate the counts for each range and the intersection region, which allows one to estimate the statistical significance of the difference between x and y. (See [8] for an example of a technique for evaluating statistical significance in a two-dimensional matrix.)

In a first example, we show what an uninteresting independence diagram looks like. Recall that uninteresting in our sense means "contains no attribute dependencies". Figure 8 contains an example of a real data set having two attributes, one of which has some outliers, the other is very skewed. The image, however, does not show any interesting patterns, only random noise, because the attributes are independent.

In contrast to other techniques, even exotic correlations and arbitrarily shaped dependencies can be observed. Figure 9 shows an example: Obviously, the two attributes are strongly correlated and this could be captured even by means of a simple correlation coefficient. However, there

Figure 9: Independence diagram (100x100 pixels).

is more information contained in the image. As one can see, the correlation is much stronger at the boundary of the data space. Furthermore, there is a second process appearing as a smooth line. These complex dependencies are not likely to be captured by any simple metric.

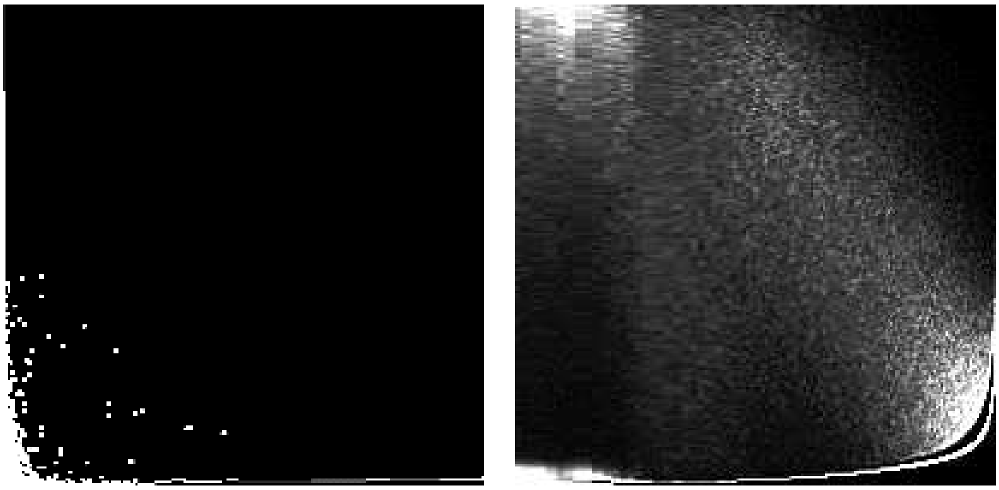
Even more complex distributions exist. Look at Figure 10(a). From a scatterplot or even an equi-width presentation, one would get the impression that we have a simple Zipfian[2] distribution with some outliers. However, as one can see from the independence diagram in Figure 10(b), the dependency is much more complex. Note that one can see: clusters, empty regions in space, a lower bound for a process as a sharp line (right bottom of the image), *etc.* Little of this information could be conveyed by a scatterplot or equi-width histogram under presence of outliers and skew.

In a final experiment, we demonstrate the advantages of animated images morphing an independence diagram from an equi-width to an equi-depth representation. The corresponding images are depicted in Figure 11. There are two clusters in the data. The clusters are realized as a single cluster in an equi-depth presentation. On the other hand, the clusters can hardly be seen in an equi-width presentation. Thus, an animated presentation is more informative than either presentation alone.

4 Related Work

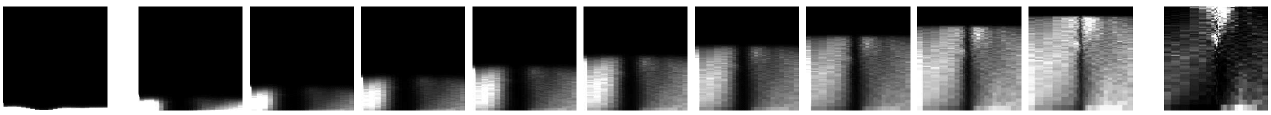
A variety of data visualization techniques such as *scatterplots* [4], *hyperslice* [21], *parallel coordinates* [11] and *circle segments* [5] have been proposed. *Variable resolution bivariate plots* dynamically transition from scatterplot format to a greyscale format based on density estimates for each cell [10]. See [15] for a survey of data visualization techniques. No previous technique specifically addressed visualizing attribute dependence effects while normalizing out the one-dimensional distributions.

[2]Zipf distributions are very frequently observed in practice for highly skewed data sets. In such distributions, the bulk of the data points all have similar value (near the lower left corner in this example) but a few points have very different values.



(a) Equi-width histogram (b) Independence diagram

Figure 10: A complex distribution (100x100 pixels).



Equi-width Equi-depth

Figure 11: A morphed image sequence. (100x100 pixels)

There is much literature on finding histogram bucket boundaries [12, 19, 13]. There is even some literature on creating histogram bins in multiple dimensions [18, 6]. However, all of this work is focused on finding histograms that minimize the error in the estimate of the attribute value, and our objective is quite different in this paper.

The quantitative rules we can generate resemble association rules [1, 20]. A fundamental difference between our approach and the association-rule approach is that we aim to communicate regions that are either above *or below* the expected cardinality of a cell assuming independence. Association-rule algorithms try to find regions R_A and R_B such that $Pr(R_A|R_B)$ (the "confidence") is above a threshold, whether or not the confidence measure is close to the expected value assuming independence. Other differences include the fact that our approach is primarily visual, and that we analyze pairs of different attributes rather than arbitrary subsets of a single attribute domain.

Visual techniques for finding association rules are presented in [14, 9]. These techniques also look for confidence above a certain fixed threshold, and do not normalize the space to focus on attribute dependence effects alone.

A preliminary version of this paper appeared as [7].

5 Conclusions

We introduced independence diagrams as a means to visually determine dependencies between two attributes. Finding such dependencies is at the heart of most data mining.

Data sets frequently have more than two attributes. In such situations, our approach is to generate images for *all* combinations of attributes. Given d attributes there are $(d^2 - d)/2$ images. These images could be shown together as thumbnails, lining up the images along the same attributes. This is our preferred mode for initial analysis. Images that show interesting effects can be enlarged and analyzed further. Cross-figure comparisons are possible since each attribute is scaled and bucketized identically in each figure in which it occurs.

Our approach is computationally tractable, even when computing images for all combinations of dimensions. We have implemented it and evaluated it using a large AT&T data set, from which the images in this paper are drawn. It is easy to identify when no two-dimensional effects are present. This task may be performed by a non-expert. Large effects occupy a correspondingly large fraction of an image. With a little training in how to read the pictures, it is possible to make quantitative judgements based on the images without extensive domain knowledge. These quantitative judgements can be supported by measures of statistical significance. Our approach is insensitive to outliers, and is robust under extreme distributions.

By aligning many images from the same data set, we facilitate data analysis across dimensions. (To achieve this benefit, we are careful to select the same slice boundaries in a dimension for all images in which that dimension appears.)

One can morph between different bucket mappings to obtain additional visual information.

References

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining Association Rules Between Sets of Items in Large Databases. *Proceedings of ACM SIGMOD*, pages 207–216, June 1993.

[2] R. Agrawal and A. Swami. A One-Pass Space-Efficient Algorithm for Finding Quantiles. *Proceedings of COMAD*, December 1995.

[3] K. Alsabti, S. Ranka, and V. Singh. A One-Pass Algorithm for Accurately Estimating Quantiles for Disk-Resident Data. *Proceedings of VLDB*, pages 346–355, August 1997.

[4] D.F. Andrews. Plots of High-Dimensional Data. *Biometrics*, 29:125–136, 1972.

[5] M. Ankerst, D.A. Keim, and H.-P. Kriegel. Circle Segments: A Technique for Visually Exploring Large Multidimensional Data Sets. *Visualization '96*, 1996.

[6] S. Berchtold, H. V. Jagadish, and K. A. Ross. Approximate High-Dimensional Aggregate Queries Using Boxes. *unpublished mansucript*, February 1998.

[7] S. Berchtold, H. V. Jagadish, and K. A. Ross. Independence diagrams: A technique for visual data mining. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, 1998.

[8] W. DuMouchel, C. Friedman, G. Hripcsak, S. B. Johnson, and P. D. Clayton. Two Applications of Statistical Modeling to Natural Language Processing. *AI and Statistics V*, 1996. (Chapter 39).

[9] T. Fukuda, Y. Morimoto, S. Morishita, and T. Tokuyama. Data Mining Using Two-Dimensional Optimized Association Rules: Scheme, Algorithms and Visualization. *Proceedings of ACM SIGMOD*, pages 13–23, June 1996.

[10] Chisheng Huang, John Alan McDonald, and Werner Stuetzle. Variable Resolution Bivariate Plots. *Journal of Computational and Graphical Statistics*, 6(4):383–396, 1997.

[11] A. Inselberg. The Plane with Parallel Coordinates, Special Issue on Computational Geometry. *The Visual Computer*, 1:29–97, 1985.

[12] Y. Ioannidis and Viswanath Poosala. Balancing Histogram Optimality and Practicality for Query Result Size Estimation. *Proceedings of ACM SIGMOD*, pages 233–244, June 1995.

[13] H. V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. C. Sevcik, and Torsten Suel. Optimal Histograms with Quality Guarantees. *Proceedings of VLDB*, August 1998.

[14] D. Keim, H. Kriegel, and T. Seidl. Supporting Data Mining of Large Databases by Visual Feedback Queries. *Proceedings of ICDE*, pages 302–313, 1994.

[15] D.A. Keim. Visual Data Mining. *Proceedings of VLDB*, 1997.

[16] H. Levkowitz. *Color Theory and Modeling for Computer Graphics, Visualization, and Multimedia Applications.* Kluwer Academic Publishers, 1997.

[17] M. Muralikrishna and D.J. DeWitt. Equi-Depth Histograms For Estimating Selectivity Factors For Multi-Dimensional Queries. *Proceedings of ACM SIGMOD*, pages 28–36, June 1988.

[18] V. Poosala and Y. Ioannidis. Selectivity Estimation Without the Attribute Value Independence Assumption. *Proceedings of VLDB*, pages 486–495, August 1997.

[19] V. Poosala, Y. Ioannidis, P. Haas, and E. Shekita. Improved Histograms for Selectivity Estimation of Range Predicates. *Proceedings of ACM SIGMOD*, pages 294–305, June 1996.

[20] R. Srikant and R. Agrawal. Mining Quantitative Association Rules in Large Relational Tables. *Proceedings of ACM SIGMOD*, pages 1–12, June 1996.

[21] J.J. van Wijk and R.D. van Liere. Hyperslice. *Visualization '93*, pages 119–125, 1993.