ROY MAYHALL II'S PROGRESSIVE BIT DISTRIBUTION

REVISION 21 – MAY 10, 2003

3rd Edition Documentation – May 10, 2003

Abstract

The end state of any given data sequence is the direct result of the process by which it was created. This process – the actual method of data generation – outputs data in elements or groups of elements at a time. In the case of computers, each element represents 1 byte, and each byte can have any of 256 possible states. Depending upon the data generation process, some or all of these possible states will be used at various points, and with varying degrees of frequency, as the entire data stream is progressively generated. Both the degree of utilized information, and the manner in which it is distributed, are crucial in ascertaining the exact nature of the data stream under investigation. This paper proposes Roy Mayhall II's Progressive Bit Distribution (PBD) data analysis algorithm – which can quickly and easily identify the exact nature of how the data is actually being generated – as a supplement to information-density and randomness analyses in ascertaining the nature of the complete data stream. In addition, practical applications of PBD are explored, with potential for "real-world" probability solutions.

Table of Contents

[roymayhall2]

Roy Mayhall II Software Products

I. Overview

All data is represented on a computer as collections of bits, which can each attain a state of 0 or 1. Two fundamental principles in establishing a generated data stream as being "random" are: (1) a 1-bit is just as likely to occur as a 0-bit, and (2) each bit should not be dependent upon the previous bits (each should be "independently" generated). Such a data stream qualifies as being an "independent random sequence".

Another feasible property of an independent random sequence is that since there is an equal chance of either a 0-bit or a 1-bit occurring at any given time, and since each bit is generated independently of all previous bits, than it would make sense that there is a high probability that all possible values (in this case, 8-bit bytes) have an equal chance of being generated at any given time. A random sequence that fits this criterion is described as being "uniformly distributed".

II. Uniform Distribution And Its Importance

A sequence of data that exhibits "uniform distribution" exhibits a probability that any of the full range of all possible byte values could be used at any given time. Specifically, the probability that any set value will occur is correlated to the inverse of all possible values within the current measured interval. If the value represented by "x" is uniformly distributed over the interval [a, b], it, at any time, has the probability of occurring as

$$P(x) = \frac{1}{(b-a)}*$$

In the case of byte measurement – the basic unit of storage in a computer – a uniformly distributed sequence would give each of the 256 possible bytes a probability of 1/255 for every 256 bytes in the data stream (interval [0, 255]).

Such properties are very important in distinguishing a random data sequence from a nonrandom one. Random sequences, or sequences that appear random in nature, are important in applications that rely on them in one form or the other. Specifically, the field of cryptography relies totally on random or "pseudorandom" sequences – both for utilization in cryptographic algorithms, and in exhibition within encrypted data streams or "ciphertexts".[*]

III. Testing For Randomness

In order to ensure that a data stream exhibits random characteristics, analysis is performed on the data sequence to look for specific random qualities. The types of analyses fall under two distinct categories: information density analysis, and randomness statistical analysis.

Information density analysis simply looks at the manner in which the data is distributed in a given data stream. Such analyses look at two facts in particular: (1) how many of the total possible values are used, and (2) how those values are distributed in the total sequence. The most popular information density analyses are the Entropy test, and actually viewing the data stream as a histogram of observed byte frequencies.

Randomness statistical analysis looks at the values in a data stream, checking them for "validity" by expecting certain statistical patterns in the manner and sequence in which values are generated, and "mapping" them to

[*] Christian Schiestl, *Pseudozufallszahlen in der Kryptographie*, Klagenfurt, 1999. Event x is said to be evenly distributed over the interval [a, b] if it possesses the probability density represented by the P(x) function. This situation should apply for all truly random data sequences.

known principles of random number sequences. The most popular form of randomness statistical analysis is the Chi^2 distribution test.

In many cases, both types of analyses are used together to completely ascertain the nature of the data stream in question. Information density analysis can give a general idea of the distribution of data within the data stream, and randomness statistical analysis can more fully ascertain whether or not the data is distributed in accordance with the mathematical principles of random numbers.

This paper proposes a new mathematical method for determining uniform distribution: the Progressive Bit Distribution (PBD) test. PBD is essentially an information density analysis tool, as it directly measures the distribution and utilization of feasible data within a data sequence. However, it is a very informative tool (as we shall see), and can greatly assist in the examination of a data stream to determine its probability of being random or exhibiting random qualities. Coupled with a good randomness analysis, PBD can nearly fully explain the tendencies and continuous states of the data within a data stream.

IV. Analyzing Uniform Distribution: PBD Versus Entropy

PBD combines the principles of general information density measurements with observed tendencies in actual utilization of all possible output values (bytes). One unique characteristic of PBD is that it is a function of time. Stated another way, PBD is the continuous measurement of the degree of uniform distribution in a data stream, as it is progressively generated.

Essentially, the PBD attempts to combine information regarding the utilization of the ASCII spectrum (all possible bytes) with information regarding how these obtained bytes are distributed, progressively from beginning to end, within the data stream. Although not a beat-all, ends-all test in itself, when coupled with other types of tests, the PBD can be very helpful in ascertaining the degree of uniform distribution in a data stream.

PBD operates via an information-density analysis similar to that found in the popular Entropy test. While some of the logic between PBD and Entropy is similar, there are some important differences between the two:

- Entropy measures information density as the degree of total bit utilization per byte (from 0 to 8 bits per byte) in the entire data stream. PBD measures the degree of equal utilization of all possible bytes progressively through the data stream, and is expressed as a percentage from 0 to 100.
- Entropy is calculated analyzing the entire contents of the data stream all at once. On the other hand, PBD represents the mean rate of actual distribution of bytes (as compared to the best possible distribution of bytes) as the data stream is progressively analyzed, from start to finish. With PBD, the entire data stream is not analyzed simultaneously; therefore weaknesses in overall distribution can be more readily established.
- Entropy does not take into account acute variations in the information density of the data stream at various points, as does PBD.
- Entropy is particularly sensitive to the size of the data stream, because it analyzes the entire file at once, and does not return comparable results for smaller files, as PBD calculates information density progressively, it can return comparable and accurate results even for smaller files.

Overall, PBD is a more thorough and accurate analysis of the information content of a data stream, with respect to uniform distribution and data utilization. Although the results of an Entropy analysis of a random data stream often correspond to higher PBD values for the same data stream, and vice-versa, the two are not directly correlated, and it is possible to obtain great differentials in the results of the two tests for identical types of data streams, especially if the data stream is small.

PBD is calculated *progressively* – that is, in consecutive blocks (called *cycles*) in order from the beginning of a data sequence to the end, accumulating the results of its calculations and then averaging those final results in the final PBD calculation. Therefore, PBD can be described as the analysis of the contents of a data sequence as a function of time, and can demonstrate the tendencies of the data sequence as it is being generated.

V. Calculation Of Progressive Bit Distribution

The Progressive Bit Distribution (PBD) is expressed as a percentage (from 0 to 100), and is defined by the following function in terms of a data stream s:

$$P(s) = d - dv + \frac{v \cdot \ln\left(1 + d\left(e^{e} - 1\right)\right)}{e}$$

The specific definition of PBD is:

The progressive bit distribution (PBD) of a data stream "s" is defined by the mean degree of uniform distribution "d", offset by the mean variation "v" in the progressive ratios of actual spectral utilization to the greatest feasible differential spectral utilization, with the objective of obtaining the greatest feasible balance of uniformity and utilization of data.

There are two main components to the calculation of the PBD of a data stream: the degree of uniform distribution of utilized data, and the degree of change in the content of the utilized data. The data stream in question is analyzed twice, once to determine each component.

The degree of uniform distribution of utilized data is calculated per 256-byte block (or cycle), and is referred to as the Mean Distribution Rate (MDR) for that cycle. The degree of change in the content of the utilized data is calculated per 512-byte cycle (or, if the data stream is less than 513 bytes in length, 256-byte cycle), and is referred to as the Spectral Utilization Factor (SUF) for that cycle. Together, the MDR and SUF of all the cycles in the data stream determine the final PBD value.

All of the MDR values for the data stream are calculated and accumulated, and then all of the SUF values for the data stream are calculated and accumulated. The accumulated MDR and SUF structures are then averaged, and then combined to yield the final PBD value.

In the PBD equation, "d" represents MDR, and "v" represents SUF. In the equations that describe MDR and SUF, all variables that depend upon the MDR and SUF will use these symbols as subscripts to indicate which structure (MDR or SUF) they are dependent upon. For example, an MDR cycle is denoted by c_d, and an SUF cycle is denoted by c_v. This makes it easier to understand which data set we are working with; it also makes it easier to differentiate between values that belong to the MDR calculation procedure, and values that belong to the SUF calculation procedure. With this in mind, let us discuss, step by step, how the PBD of a data stream is calculated.

Step 1: Calculating Mean Distribution Rate (MDR)

The distribution, or MDR, of a data stream is calculated based upon the principles of uniform distribution defined by the following expression of the probability P(x) of the event x occurring along the interval [a, b]:

$$P(x) = \frac{1}{(b-a)}$$

The MDR of a given data stream is calculated progressively, in blocks of up to 256 bytes at a time. The specific definition of MDR is:

The mean distribution rate (MDR) of the current cycle, when the cycle length is equal to or less than the number of total feasible data elements, is defined by the ratio of the actual degree of distribution uniformity to the optimal pattern of uniform distribution for the current cycle.

To determine the actual distribution of bits within each run-time cycle as a percentage of the optimal distribution, one must consider several factors within the current cycle to be analyzed:

- The optimal uniform distribution (defined in the previous function) must be known.
- The actual (detected) bit distribution must be derived as a deviation from optimal distribution.
- The percentage of feasible distribution must be expressed as the actual bit distribution over the optimal uniform distribution.

The manner in which the MDR is computed is actually a deviation of the current distribution of bits from the optimal or "perfect" distribution of bits. A value of 0 indicates only 1 type of byte (or bit sequence) is used in the data stream, while a value of 1 indicates all possible bytes (and all possible bit combinations) are used continuously throughout the data stream.

Each bit sequence is obtained is simply by analyzing the value of each consecutive byte in the data stream, which can range from 0 to 255. By this method, all bits actually contained within the data stream are quickly accessed and analyzed.

By definition, the length of each run-time cycle in MDR calculation is equal to the number of feasible data elements. Therefore, generally speaking, each MDR run-time cycle is 256 bytes in length. The only exception is when the end of the data stream is reached (when it will be shorter). The reason that the MDR is calculated with 256-byte lengths is because there are 2^8 or 256 possible bit combinations per byte, translating into 256 possible byte values.

We know that there is an interval of [0, 255] for each byte. In MDR calculation, we are also interested in how *often* each byte occurs in relation to the number of total bytes in the current cycle. To obtain the actual distribution of bytes within the current cycle, we consider the probability equation discussed earlier, and two distinct items of information: the current feasible *occurrence* interval (which is always [0, s], where s is the size of the current run-time cycle), and the number of occurrences for each feasible event x (in other words, each of 256 total possible bytes).

So, we can modify the probability equation for our uses to:

$$P(x) = \frac{n}{s}$$

We can use s directly as the denominator because the occurrence interval in every cycle is [0, s], and (s – 0) always equals s. The variable n represents the number of occurrences of event x, which can be no more than the value s. And, since the feasible interval for each byte value is [0, 255], there are a total of 256 possible x events (so this function could be calculated up to 256 times [once for each different occurring byte value], or as many different byte values as there are in the current cycle).

Now, we must consider that MDR calculation represents the *deviation* from the optimal rate of distribution. First, let us consider the conditions for optimal distribution:

1. The number of different byte values in the cycle must be equal to the size of the cycle (therefore as many different bytes as possible are used), and
2. All bytes that are used in the data cycle must be used equally as often, while still complying with rule 1.

You can see here that rule 2 basically is redundant to state, because if all bytes within the current cycle are unique, they will obviously occur equally as often. Additionally, it is to be expected that within a uniformly distributed cycle of any length, if each byte is to be unique, it must occur exactly once and no more, and represent 1 / s percent of the cycle's data content (with s being the length of the cycle in bytes).

Therefore, if we know that the current cycle is s bytes in length, for its data contents to be uniformly distributed, there must be exactly s number of unique bytes, and each byte must occur exactly once. But what happens when bytes in the cycle do not occur equally? Then, automatically, not all feasible bytes are used, and therefore uniform distribution is not attainable.

There are varying degrees of proximity to uniform distribution within a cycle. For instance, in a 256-byte cycle, 254 bytes may occur exactly once, and 1 byte may occur twice, thus leaving out 1 other possible byte. Another 256-byte cycle, on the other hand, may have only 1 byte occurring 256 times, thus leaving out 255 other possible bytes. These are extremes, and many times the data content within a particular cycle will fall in the middle of this range.

We do not only want to measure whether or not the cycle is uniformly distributed, but also how close the cycle is to being uniformly distributed. We can easily do this by setting up a type of "point" system. A byte should be "heavier", or receive more points, if it occurs less often (therefore enabling more possible bytes to exist in the cycle), and "lighter" (receive less points) if it occurs more often (enabling less possible bytes to exist in the cycle). A byte should receive the least amount of points if it makes up the entire cycle, and the greatest amount of points if it only occurs once in the cycle (thereby complying with the conditions for uniform distribution).

A mathematical function exists to model this situation – the natural logarithm function (log base-e or ln). It can be used to express true deviations from optimal scenarios, and can be sensitive to minor deviations. Throughout the calculation of both the MDR and the SUF values, the natural logarithm function is extensively applied.

If we consider that each byte x represents a specific percentage of the entire cycle of length s_d, we can calculate its "point value" as follows:

$$a(x) = \left| \ln\left(\frac{x}{s_d} \right) \right|$$

We take the absolute value of the natural logarithm of x / s_d (absolute value is used because we are only interested in the deviation from 0, or, in our case, the raw degree of uniformity of the byte).

You can see how useful this function can be in MDR calculation. We know that the logarithm of 1 is 0, so it follows that if a byte is the only byte in the cycle, then a(x) yields 0, meaning that the cycle is completely non-uniform (this will cause the MDR to be 0, as is discussed below).

The values a(x) returns are valid only when we consider what it is an optimal, uniformly distributed byte pattern should look like. If there are s_d bytes in the cycle, then a(x) should be calculated exactly s times, with x always being 1. Therefore, it follows that in a uniformly distributed cycle, the a(x) function calculations should total the following:

$$o_{c_d} = s_d \cdot \left| \ln\left(\frac{1}{s_d} \right) \right|$$

The function o_{cd} represents the optimal pattern of distribution in the current MDR cycle (represented by c_d), with the absolute value of the natural logarithm of 1 / s_d being calculated exactly s times.

Actually, because we are taking the absolute value of the logarithm, we know that the reciprocal of the argument of the logarithm function will yield the same end result. Therefore, the above function can be simplified to:

$$o_{c_d} = s_d \cdot \left| \ln\left(s_d \right) \right|$$

Either way, o_{cd} is defined the same.

Now, with the mathematical functions we have just defined, we can place them together in the form a(x) / o_{cd}, to yield the final MDR value as a deviation from optimal distribution of bytes within cycle c_d.

To define this function, we consider that a(x) is defined for every byte in the cycle, and o_{cd} is calculated only once. The accumulated total of all a(x) calculations is then divided by o_{cd} to obtain the final MDR percentage. If we define s_d as the length of the current cycle c_d, and x_n as the occurrence rate of each byte in the cycle (x_1 representing the first byte in the cycle, x_2 representing the second byte, x_3 the third byte, and so on), we can define MDR in terms of cycle c_d as follows:

$$d_{c_d} = \frac{\left|\ln\left(\frac{x_1}{s_d}\right)\right| + \left|\ln\left(\frac{x_2}{s_d}\right)\right| + \left|\ln\left(\frac{x_3}{s_d}\right)\right| + \ldots + \left|\ln\left(\frac{x_n}{s_d}\right)\right|}{s_d \cdot \left|\ln\left(s_d\right)\right|}$$

This function returns 0 if the data is completely non-uniform, and 1 if the data is perfectly uniformly distributed.

Now consider that if the cycle length is 1, this function translates into 0 / 0, which is undefined. The reason for this is there is no possibility for deviation from an absolute uniform distribution (remember that MDR measures the *deviation* from uniform distribution). By definition, any cycle of length 1 is automatically uniformly distributed – because only 1 byte is allowed to occur. For purposes of PBD calculation, if the MDR is ever undefined, it is defined as having the value 1 – for perfect, uniform distribution.

All MDR values are calculated using 256-byte intervals in the data stream (until the very end of the data stream is reached, when less than 256 bytes are left, at which case as many bytes as are left are used). Each MDR value for each data stream is added to an accumulator represented by d in the PBD equation; upon calculation of the final PBD value, the MDR accumulator is averaged.

Once all MDR values have been calculated for the data stream, the data stream is analyzed once again from the beginning – this time to yield the Spectral Utilization Factor, or SUF, values.

Step 2: Calculating Spectral Utilization Factor (SUF)

The SUF of the data stream represents the degree of complete spectral utilization, or to what extent the contents of the data stream "change" from cycle to cycle, such that as much different content is utilized as is possible. It is correlated with the MDR values for the current data stream, as the pattern of distribution of bytes within the data stream determines, to some extent, how much of the feasible spectrum of bytes is utilized at various points between cycles.

Unlike MDR, which is calculated using 256-byte cycles, SUF is calculated using 512-byte cycles (the exception is when the data stream is less than 513 bytes in length, in which case 256-byte cycles are used). An obvious exception to these rules is when the end of the file is reached, and only the remaining bytes can be read into the current cycle.

SUF is officially defined as follows:

The spectral utilization factor (SUF) of the current cycle, if the current cycle is at least the second cycle in the data sequence, and the cycle length is equal to a maximum of twice the number of total feasible data elements (unless the length of the data stream prohibits this length, in which case the cycle length is equal to or less than the number of total feasible data elements), represents the ratio of change in the context of the contents of the current data cycle as compared to the contents of the previous data cycle.

Why are 512-byte cycles used instead of 256-byte cycles, if there are only 256 possible byte combinations? The reason is because patterns of *repeatability* can be more readily established by analyzing two consecutive "runs" of 256 bytes (because each 256-byte block *could* utilize the entire ASCII spectrum, if it were uniformly distributed) as the data stream is progressively examined. This principle was not discovered mathematically but rather by pure observation; using 256-byte cycles with SUF analysis could not effectively reproduce the results of SUF analysis with 512-byte cycles.

Larger cycle sizes for SUF analysis were tested, but introduced inconsistencies, as over great spans of data elements, incorrect assumptions can be made about the raw distribution of data. So, by definition, the default SUF cycle size is twice the feasible number of data elements, in our case 512.

Obviously, if the data stream is less than 513 bytes in length, SUF cannot be obtained using 512-byte cycle lengths. Therefore, in this circumstance, the default cycle length must be reduced to 256 bytes.

The SUF of the current cycle is only obtained if it is at least the second cycle in the data stream. This is because SUF represents the change in the occurrence rates of bytes between the current cycle and the previous cycle. Thus, for the first cycle in the data stream, no SUF value can be obtained (therefore it is undefined, which by definition yields the value 0 for the SUF). This also means that if there are less than 257 bytes in the data stream, no SUF value is calculated (it defaults to 0 in the PBD calculation).

As was stated earlier, SUF represents the degree of "change" in the context, or the parts of the ASCII spectrum that are used, of the contents of the current data cycle as compared to the contents of the previous data cycle. This is actually a rather simple value to calculate. All we need to do is obtain the degree of difference in how often each byte in the current cycle occurs as compared to how often it occurred in the previous cycle, and divide that by the *optimal* degree of difference, which simply equates to the larger value (either the occurrence rate of the byte in the current cycle, or the occurrence rate of the same byte in the previous cycle).

We can use natural logarithms ($\log_e$) just as we did in MDR calculation to set up a fairly similar "point" system: a byte that occurs at exactly the same rate as it does in the previous cycle should receive the least amount of points (specifically, 0 points), and a byte that occurs much more often in the current cycle than it does in the previous cycle should receive more points.

Unlike with MDR calculation, SUF calculation does not take into consideration whether or not bytes actually occur in either the current cycle or the previous cycle. It does not really matter for purposes of SUF calculation because all that SUF analyzes is the *difference* in the occurrence rate between the current cycle and the previous cycle. If a certain byte does not occur in one cycle but does occur in the other cycle, then the SUF ratio would be 100% (raw value 1); this is actually the *optimal* difference in occurrence rate.

Well, we now have the foundation for the calculation of the SUF for the current cycle. Now, we must define the optimal pattern of occurrence differential, and the actual pattern of occurrence differential, between each byte in the current cycle and each corresponding byte in the previous cycle.

Before we go any further, we must recognize that each byte has 256 possible combinations. Because the default SUF cycle size is twice this (512 bytes), and because cycle sizes can vary (near the end of the data stream), we need to define a consistent system of comparing bytes, so that accurate calculations can be made, no matter what the size of the cycles being used. We can do this by setting up a proportion, to make all bytes in terms of 256 bytes. Simplified, the proportion equation takes the form:

$$x_n = \frac{ma_n}{s_{v_n}}$$

Here, x_n represents the new byte occurrence rate, m represents the total number of feasible elements (always 256 in our case), a_n represents the current byte occurrence rate, and s_{vn} represents the actual size of the cycle (usually 512). The sub-type n represents a value, either 1 or 2, that, in the equation, signifies that we are calculating occurrence rates for a byte either in the previous cycle (as in x_1) or the current cycle (as in x_2), respectively. You will see this notation used in the forming of the SUF equations, coming up.

Since we know m is always 256 for calculating x_n when analyzing a data stream, for purposes of file analysis we can simplify this equation to:

$$x_n = \frac{256 \cdot a_n}{s_{v_n}}$$

Obviously, in cases where the total number of feasible elements is not 256, we could not do this; m would have to represent some other value. However, whenever we are analyzing the SUF of a block of data represented on a digital system in 8-bit bytes, m will always be 256 (because there are 256 possible 8-bit byte combinations).

By using this equation, we can translate the occurrence rate for bytes that occur in any cycle length into the occurrence rate for the same bytes under a cycle length of 256 bytes. For example, with a cycle length of 512 bytes, plugging a 2 in as the current byte occurrence rate returns 1 as the new byte occurrence rate, to be used in SUF calculation (this makes sense when you think that 2/512 is the same as saying 1/256).

Because the logarithm function does not work for the value 0, and returns 0 for the value 1, we can obtain a more accurate SUF result by adding 1 to each x value. This way, 0 becomes 1, 1 becomes 2, etc. You can see how this can work, because now (1) the SUF routine can differentiate between the occurrence rates 0 and 1, and (2) the "true" x-value of 0 causes the routine to yield 0 (since the logarithm of 1 is 0). Therefore, we can edit the previous equation to the following:

$$x_n = \frac{ma_n}{s_{v_n}} + 1$$

Or, since we are working with data (bytes) in a computer system, we can substitute 256 for m:

$$x_n = \frac{256 \cdot a_n}{s_{v_n}} + 1$$

It turns out, as you will see, that this feature of the proportion equation makes it all the more possible to ascertain whether or not data repeats itself in the data sequence.

Now that we have defined the form of the occurrence rate for each byte, we must now form the SUF calculation. First, we consider that we are looking for the degree of difference in the occurrence rate of each byte in the current cycle as compared to the occurrence rate of each corresponding byte in the previous cycle. We do this via a simple division that takes the form x_2 / x_1 (it does not matter which is the dividend and which is the divisor, because the absolute value of the natural logarithm of the quotient will be taken), where x_2 represents the byte's occurrence rate in the current cycle and x_1 represents the same byte's occurrence rate in the previous cycle.

Since we have added 1 to the x_1 and x_2 values in the previous equation, there is no way that either value can be 0. Therefore, it is safe to divide the two values (to determine the amount of differentiation between the two), and take the natural logarithm of the result. However, it is indeed possible to obtain a final result of 0, because the natural logarithm of 1 is 0. This is perfect because according to the previous equation, the value 1 for x_n represents 0 occurrences for that byte in the cycle.

We calculate each x_1 and x_2 value for each byte that occurs in the current cycle (that is, each x_2 value that is non-zero). For each set of x_1 and x_2 values, we calculate y_n, which represents the actual differential in the occurrence rates of the two bytes between the current cycle and the previous cycle. Using the methods we have just discussed, we can define y_n as:

$$y_n = \left| \ln\left(\frac{x_2}{x_1} \right) \right|$$

Here, n represents the byte in the current cycle, for which x_2 and x_1 are obtained. This equation states that we are to take the absolute value of the natural logarithm of x_2 divided by x_1 (it does not matter which is the dividend and which is the divisor because we are taking the absolute value of a logarithm, hence x_2 / x_1 would yield the same value as x_1 / x_2, for our purposes here).

To express SUF as a percentage, as we do with MDR, we must also know the optimal pattern of differential occurrence between bytes in the current cycle and bytes in the previous cycle. Using the equation just listed, we can find this out. If x_2 is greater than 1 but x_1 is 1 (meaning it doesn't actually occur; recall that we added 1 to the x values prior to the calculation of y_n), then that byte occurs in the current cycle but doesn't occur in the previous cycle, and this would translate into $x_2 / 1$, which equals simply x_2.

SUF also applies to bytes that occur more in the *previous* cycle than in the current cycle. It is possible to have x_2 be 1 instead of x_1, or even *both* x-values to be 1. Because we are taking the absolute value of a logarithm, it translates into the same occurrence differential for purposes of SUF calculation, no matter which cycle has a higher occurrence rate.

Using the guidelines for optimal differential occurrence outlined above, for each y_n (defined just previously) we can define z_n, which represents the optimal condition that y_n is attempting to achieve:

$$z_n = \left\{ \left[x_1 > x_2 \therefore \left| \ln\left(x_1\right) \right| \right] \quad \vee \quad \left[x_1 \le x_2 \therefore \left| \ln\left(x_2\right) \right| \right] \right\}$$

This states that if x_1 is greater than x_2, the absolute value of the natural logarithm of x_1 is taken; if x_2 is greater than or equal to x_1, then the absolute value of the natural logarithm of x_2 is taken. One thing you will notice is that if the greatest occurrence rate is 1, z_n becomes 0. This makes sense because the value 1 actually represents a "real" occurrence rate of 0. If the greatest occurrence rate is 1, both x_1 and x_2 are 1, meaning that there is no change in the occurrence rate in that byte between the previous cycle and the current cycle.

Now that we have defined y_n and z_n, how do we obtain the final SUF? We simply combine all obtained y values, and all obtained z values, for as many bytes as occur in the current cycle, and divide them to obtain a percentage, which represents the degree of occurrence differential between the current cycle and the previous cycle. The SUF of cycle c_v, as compared to the previous cycle ($c_v - 1$), is defined as follows:

$$v_{c_v} = \frac{y_1 + y_2 + y_3 + \ldots + y_n}{z_1 + z_2 + z_3 + \ldots + z_n}$$

If the accumulated z-values total 0, then v_{cv} is calculated using an attempted division by zero, which is undefined. This situation means that every byte in the current cycle occurs at the same rate as every byte in the previous cycle (or *no* bytes exist in *either* cycle, a hypothetical situation at best), therefore no optimal occurrence differential can be obtained. By definition, this scenario means that no differences in the occurrence rates of the bytes between the previous cycle and the current cycle have occurred, and v_{cv} therefore should be 0. Simply put, if v_{cv} is undefined, it is defined as having the value 0.

It should be noted that in SUF calculation, the very first cycle in the data stream is understood to have an SUF of 0 because it is undefined (remember that SUF is only calculated from the second cycle onward). Nonetheless, the first cycle is still considered in the PBD calculation.

As with MDR, all SUF values for each consecutive cycle are accumulated into a value represented by v in the PBD equation; upon calculation of the PBD value, an average is needed for all SUF values.

After all SUF values have been obtained for the data stream, enough information is known to calculate the PBD for the data stream. However, before calculating the final PBD value for the data stream, both the MDR and SUF accumulated data structures must be averaged. In actual practice, this process actually occurs "on the fly", or as the MDR and SUF values for each cycle are being calculated.

Step 3: Averaging the MDR and SUF data structures

As we saw in the previous discussions of MDR and SUF calculation, for each cycle in the data stream, a new MDR/SUF value is added onto the appropriate accumulator value. In order to utilize the MDR and SUF values obtained for the data stream in the PBD equation, it is necessary to average the accumulated data structures. This is actually done not at the end of the MDR and SUF calculations, but as the MDR and SUF values are being obtained within each cycle.

We do not average the MDR and SUF data structures *exactly* in the typical way. Rather, the size of each cycle is taken into consideration, to obtain a "significance value" for each cycle, which affects how that cycle will be handled in the (simple) averaging process.

Each cycle within the data stream is usually the default length. In fact, each cycle within the data stream *is* the default length until usually the last cycle in the data stream. This is because most files and data sequences are not evenly divisible by the default cycle lengths. As a consequence of this, the MDR and SUF ratios are affected considerably.

Because the last cycle in the data stream is smaller than all other cycles in the data stream, it causes the MDR and SUF values to deviate considerably from the "norm", or the overall tendencies exhibited thus far. The smaller the cycle is, the more drastic the deviation. If we were to average all cycles in the normal way, this deviation would play a significant role in the outcome of the averaging, thus throwing off the entire PBD calculation.

To illustrate, let's say we have a 576-byte file. For MDR calculation, it is divided into 3 cycles, and we are trying to obtain the average MDR across all 3 cycles. Let's say that each cycle is as follows:

CYCLE	SIZE (bytes)	MDR
1	256	75%
2	256	78%
3	64	91%

In this example, the "traditional" average MDR would be 81.33%. This would be the case no matter *how* big the last cycle is (assuming the same MDR). However, in this example, the 3rd cycle only represents 64/576 or 1/9 of the total file. Although 8/9 of the file consists of the two 256-byte cycles, the 3rd cycle's MDR is high enough (primarily because of its smaller size) so that the average MDR *far* exceeds either of the 2 "dominate" cycles. This would cause a great inconsistency within the final PBD calculation.

The solution to the problem is very simple. We simply assign a "priority" of sorts to each cycle, as it is computed, dependent directly upon the size of the cycle. The size of the cycle in relation to the default size of the cycle is called the "significance value", and it is used to correct the inconsistency just previously discussed.

We can define the significance value as a ratio of the actual cycle size, c, to the default cycle size, s, as follows:

$$\frac{c}{s}$$

To correct the inconsistency, we simply multiply each cycle value (either MDR or SUF) by the significance value. Normally, the significance value is 1, because the cycle size throughout most of the data stream equals the default size. Therefore, in these cases, each cycle value is unaffected. However, for the very last cycle in the data stream, the significance value plays a more substantial role.

We can take the current cycle value, x (either MDR or SUF), and multiply it by the significance value. That result is added onto the appropriate accumulator value, z (either MDR or SUF), as follows:

$$z = z + \frac{xc}{s}$$

This way, the size of the cycle contributes to how much the cycle affects the overall average.

To balance the alteration of the accumulator values, the counter that keeps track of the number of cycles must also be modified. Normally, for each cycle that occurs, we add 1 to the cycle counter. This way, we can take an average by dividing the sum of all cycle values by the number of cycles indicated by the cycle counter. However, in the case of a cycle smaller than the default, we add a number *less than* 1 to the cycle counter. Specifically, we add the significance value itself to the cycle counter, n, as follows:

$$n = n + \frac{c}{s}$$

It follows, by this equation, that for most of the data stream, the cycle counter will be incremented by 1, because the vast majority of the cycles within the data stream are the default size. However, the last cycle will be affected by the significance value.

Now that we have defined how the significance value affects the averaging process, we can take the correct average of the following MDR cycles:

CYCLE	SIZE (bytes)	MDR
1	256	75%
2	256	78%
3	64	91%

We can evaluate the average MDR of these 3 cycles as follows:

$$\frac{\left(75\% \cdot \frac{256}{256}\right) + \left(78\% \cdot \frac{256}{256}\right) + \left(91\% \cdot \frac{64}{256}\right)}{\frac{256}{256} + \frac{256}{256} + \frac{64}{256}}$$

The result of this expression yields an average of approximately 78.11%, which is much more appropriate.

With this method of averaging (which applies to both the MDR and SUF data structures), the final PBD result becomes much more accurate and consistent, and small variations in file sizes do not seemingly affect the calculation. Once the averages of both the MDR and SUF data structures have been obtained, the fourth and final step is to actually calculate the final PBD result itself.

Step 4: Tying MDR and SUF together to calculate Progressive Bit Distribution (PBD)

As was stated earlier, the PBD of the current data stream is calculated as follows:

$$P(s) = d - dv + \frac{v \cdot \ln\left(1 + d\left(e^{e} - 1\right)\right)}{e}$$

Here, d represents the averaged MDR values for the data stream, and v represents the averaged SUF values for the data stream. The value c_d represents the number of MDR cycles, and c_v represents the number of SUF cycles. It is understood by this equation that the accumulated MDR and SUF values are averaged before calculating the PBD of the data stream.

Here, we will discuss what this equation means, and how it is derived. First, let us reexamine the actual definition of PBD:

The progressive bit distribution (PBD) of a data stream "s" is defined by the mean degree of uniform distribution "d", offset by the mean variation "v" in the progressive ratios of actual spectral utilization to the greatest feasible differential spectral utilization, with the objective of obtaining the greatest feasible balance of uniformity and utilization of data.

(It should be noted that initially, as MDR and SUF are undefined, they are defined as the values 1 and 0, respectively, when 0 bytes have been analyzed. However, only the MDR and SUF values that are generated from analyzing actual data in the file [from byte 1 onward] are used in PBD calculation. This is easy to comprehend from a computer-programming standpoint, but mathematically it is important to clarify the distinction between analysis of 0 bytes of information and analysis of 1 or more bytes of information [actual data]. Analyses of 0 bytes of information are always undefined mathematically, but are not applicable to real-world problems.)

Calculating the PBD of a data stream helps us to understand the nature of the data stream's contents. From the PBD, we know how well the data stream utilizes the full range of possible byte values, and how well it distributes the range of byte values that it uses. In summary, PBD is a measurement of how well the data stream distributes the full range of possible byte values, or how well the contents of the data stream are essentially unpredictable, so that each byte is essentially independent of its predecessors. PBD can tell an analyst, in a nutshell, the degree to which the data stream is completely uniform in distributing the *complete* range of feasible byte (or ASCII) values.

So, we know how well the data in the data stream is uniformly distributed (MDR, represented by d), and how well the contents of the data stream change with time (SUF, represented by v).

The basis of the PBD calculation is the degree of uniform distribution, represented by d. Therefore, we start out simply with P(s) defined as d:

$$P(s) = d$$

Now, we want to couple the uniform distribution rate with the actual degree of change within the data stream. Although different sets of bytes may generally exhibit the same degree of uniform distribution in a random data sequence, the true information density of the sequence (and hence the capabilities of the random number generator) is partially related to the degree by which the entire feasible spectrum of bytes is consistently utilized at various points throughout the data stream. Although the uniform distribution rate shows this to some extent, actually analyzing the rate of change in the contents of the data stream is much more productive. Hence, we bring in the SUF value, represented by v.

Our objective in calculating PBD is to obtain the perfect rate of uniform distribution of the entire feasible range of bytes. Since SUF and MDR are intertwined to some extent (because the data they both represent is identical), we aim for them, together, to accumulate to the value 1 (the raw PBD value is an absolute ratio from 0 to 1). Specifically, we want the rate of uniform distribution to be offset by the variance in data utilization, such that a perfect variance would attain the PBD value 1. To model this scenario, we define the following for P(s):

$$P(s) = d + v(x - d)$$

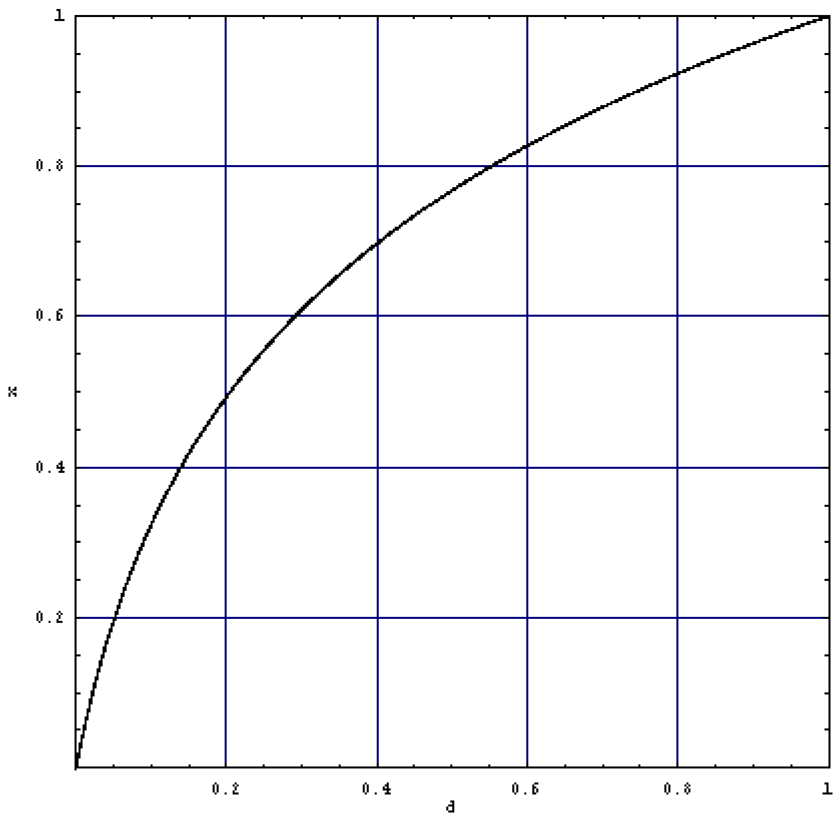
Here, we multiply v by an amount which, when added to d, will make x – naturally, x – d. Therefore, it follows that no matter what d is, if v is 1, then P(s) will equal x. Conversely, no matter what v is, the absolute *minimum* value of P(s) will always be d.

What does x represent? This value represents the allowable maximum that the PBD value is allowed to obtain, given the current MDR (d) and SUF (v) values. Optimally, this value should be 1. However, the SUF serves as an *offset* to the MDR. If the MDR is substantially low, a high SUF should not be able to offset the MDR such that the PBD becomes 1, because the MDR serves as the *basis* for the PBD calculation. In other words, the MDR should serve as the primary foundation of the PBD calculation and should limit the maximum permissible value that the PBD can obtain.

We know that the minimum value for x should be 0, and the maximum value for x should be 1. We do not want this function to be linear based on the MDR, because *d + v(d – d)* always cancels out to just *d*, therefore causing the SUF to be negligible. To carry out this operation, we let x equal a logarithmic curve that takes the form *ln d*. However, we need to offset the logarithmic curve such that its minimum value is 0 and its maximum value is 1 (the standard natural logarithm function does not do this). In other words, we need to modify it so that it always yields 0 when the MDR (d) is 0, and so that it always yields 1 when the MDR (d) is 1. We do this by transforming the logarithmic curve represented by x to the following form:

$$x = \frac{\ln\left(1 + d\left(e^e - 1\right)\right)}{e}$$

These transformations will yield a logarithmic curve that sets its lower and upper boundaries to those we desire for PBD calculation. A graph of this curve would resemble the following across the domain [0,1] and the range [0,1]:



We now have the maximum permissible limit for the PBD equation. Plugging this curve back into the original PBD equation would give us the following:

$$P(s) = d + v\left(\frac{\ln\left(1 + d\left(e^e - 1\right)\right)}{e} - d\right)$$

The above form of P(s) can therefore be simplified into the final PBD equation:

$$P(s) = d - dv + \frac{v \cdot \ln\left(1 + d\left(e^{e} - 1\right)\right)}{e}$$

The result is a raw value between 0 and 1. To express the ratio that it represents as a percentage, we simply multiply the result by 100. Now, we have the final PBD value for data stream s.

We now have defined the manner in which PBD is calculated based on the overall "general" tendencies in the MDR and SUF through the file. Though this is excellent for most purposes, sometimes it is more helpful to know exactly what is going on at various points throughout the data stream. Beyond simply the PBD value in itself, the actual tendencies of the contents of the data stream can be examined by looking at the MDR and SUF data structures themselves.

VI. Observing Actual Tendencies in Data Distribution

Although the PBD of a data stream is very informative about the nature of its contents, further information about the data stream can be obtained by analyzing the tendencies exhibited in the actual distribution of the contents of the data stream, progressively through the file. This can be done by gathering and comparing the cumulative (non-averaged) MDR and SUF values. These values can be visually represented in a line graph, as will be demonstrated here. This helps significantly in picturing the data stream in its entirety.

Here, we will analyze several different types of files that exhibit various tendencies in data distribution – from completely repetitive data to highly unorganized, "random" data – by graphing the MDR and SUF data structures and actually computing the PBD, for each file. Some of the files used here are part of the Encryption Torture Test (ETT) package provided by Marathon Computer Press, which also includes the PBD algorithm as part of the analysis suite.

We will be using the standard PBD algorithm executable, coded in C, to obtain the average MDR and SUF values (employing the "significance value" method discussed previously), and the final PBD result, and outputting the MDR and SUF data structures for graphing.

First of all, let us recap on how MDR and SUF should be able to reflect actual contents of the data stream in question. MDR, or mean distribution rate, is an analysis of the actual degree of uniform distribution within each cycle in the data stream. SUF, or spectral utilization factor, is an analysis of the actual degree of variation in the utilization of data between cycles in the data stream. Therefore, it follows that if a data stream contains highly repetitive data, the MDR and SUF values should be lower than if it were to contain highly unpredictable data. Our goal is to obtain both high MDR values and high SUF values on a continual basis throughout the data stream.

Sometimes, high MDR values can be obtained with corresponding lower SUF values. This can happen if the data exhibits tendencies towards evenly distributing only a finite number of different bytes, with some bytes not being used often or not at all. Conversely, higher SUF values can be obtained with lower MDR values, if the data exhibits tendencies of utilizing many different bytes but poorly distributes them. In any case, the MDR and SUF data structures are related enough such that both influence each other to some extent, as will become apparent in the graphs that are to follow.

Also, for certain types of data streams, MDR and SUF values can fluctuate between cycles. Sometimes, a period of higher distribution and byte utilization is followed by a period of lower distribution and/or byte utilization, and vice-versa, sometimes in easily discernable patterns. Generally, in most data streams, there will be some fluctuation, but optimally both the MDR and SUF values should remain close to some higher constant value.

Now that we have a basic understanding of the tendencies of the MDR and SUF values in data streams, let us visually observe their tendencies in various types of files. We will start out with a file called "TEST", which contains highly repetitive sequences of data. For about the first third or so of the file, the following data appears continuously:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890
abcdefghijklmnopqrstuvwxyz0987654321
```

For the remainder of the file, the following data continuously appears:

```
111111111111111111111111111111111111111111111111111111111111111
```

We can predict already that the degree of uniform distribution and change in byte utilization are both very low. Therefore, the PBD of this file should be very low as well.

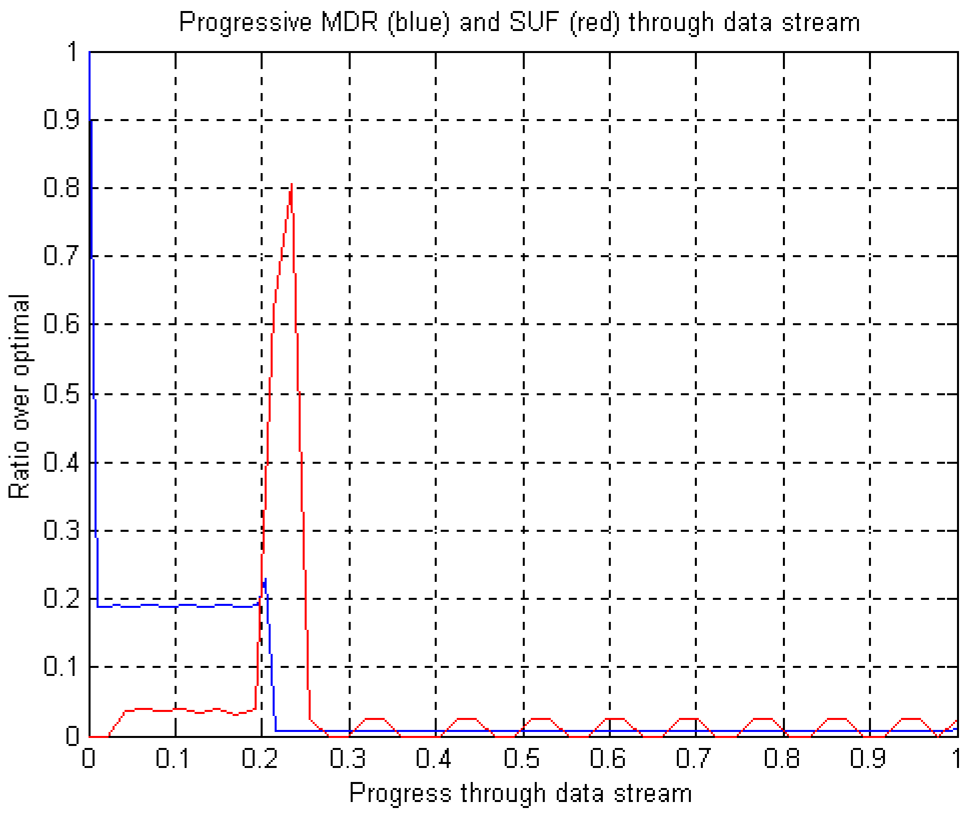
First, let us compute the average MDR and SUF values, and obtain the final PBD value, for this file. For the file "TEST", we get the following results:

```
File size                          = 23720 bytes
Average mean distribution rate     = 4.407771%
Average spectral utilization factor  = 4.636863%

PROGRESSIVE BIT DISTRIBUTION (PBD)   = 5.030403%
```

Our assumptions are apparently correct. Notice that the MDR and SUF are very low, and the PBD is only about 5.03% (0.0503).

Now that we have a general idea about the contents of the file, let us now visually examine the tendencies within the file itself. Analysis of the individual MDR and SUF values for every cycle in the file yields the following line graph:



Here, you can see that the overall tendencies in the MDR and SUF are continuously low values; this was to be expected. But as you can see here, we can now look at the *trends* of these data structures through the file. The scales on the graph are in raw decimals, but can be translated into percents by multiplying by 100. Remember that all values are ratios of actual conditions over optimal conditions.

First, notice that the MDR values start at 1, and the SUF values start at 0. This is how these are both defined initially (recall the explanations of MDR and SUF in the previous section), because they are both undefined before the data is analyzed.

Notice how the MDR hovers at about 0.2 until about 20% through the file, at which point it spikes up briefly, and then plummets down to close to 0, where it stays for the duration of the file. Likewise, notice that the SUF hovers at about 0.05 until close to the point where the MDR briefly spikes (at about 20% through the file). It then spikes *way* up, and then plummets down to 0. But notice that through the remainder of the file, it periodically rises and falls in a cyclical pattern, but never gets higher than about 0.025.

The explanations of these tendencies are rational once we understand the actual contents of the TEST file. For the first portion of the file, the letters of the alphabet and the 10 digits are used in repeating sequences. There is some degree of distribution and byte utilization variance, but not a lot. The lines here are fairly straight because the data remains fairly constant. Then, the contents of the data stream change suddenly to repeating strings of 1's and an every once-in-a-while carriage return and line feed (between lines of 1's). Because of the sudden change in the contents of the data, the SUF shoots up to over 0.8, and because of a slightly higher degree of uniform distribution of bytes, the MDR rises very slightly. But then, the sequence of 1's becomes very predictable and orderly, so the SUF and MDR both plummet substantially. Because there is a periodic carriage return and line feed sequence, the SUF rises slightly at these points, and then returns to 0 when the continuous and unchanging pattern of 1's resumes. This explains the rising and falling motion of the SUF line. But since the data is very constant and unchanging, the MDR value remains very low, but never reaches 0 because *some* uniform distribution is taking place (within the bounds of the utilized bytes).

You can see here how the MDR and SUF values are somewhat tied into each other. Although the MDR checks specifically for uniform distribution, it does depend on the degree of feasible bytes that are used in the sequence. And although SUF checks specifically for variations in byte utilization, it does depend to some extent how these bytes are distributed in the sequence (because this is directly proportional to the number of different bytes that are actually used).

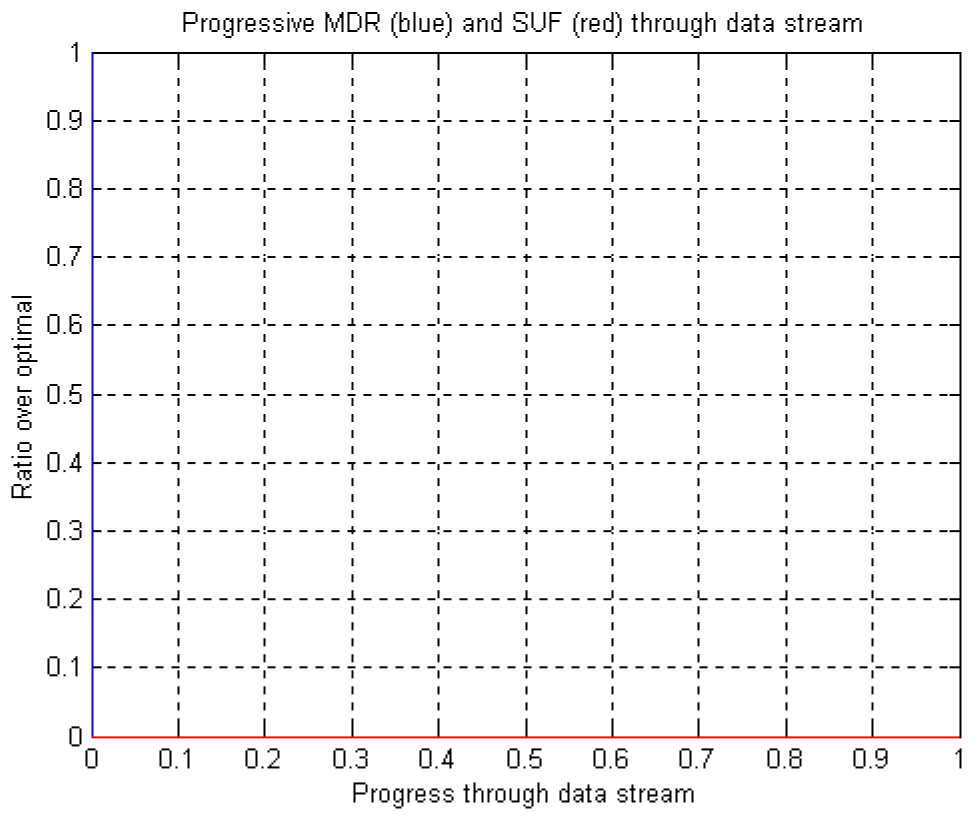
Now, let us look at a different type of file, "BIGNULL". This file is over a megabyte in length, and contains nothing but null bytes (ASCII code 0).

As should be expected, the MDR, SUF, and PBD values all turn out to be 0:

```
File size                          = 1433600 bytes
Average mean distribution rate     = 0.000000%
Average spectral utilization factor  = 0.000000%

PROGRESSIVE BIT DISTRIBUTION (PBD)   = 0.000000%
```

The graph of this file should be very predictable as well:



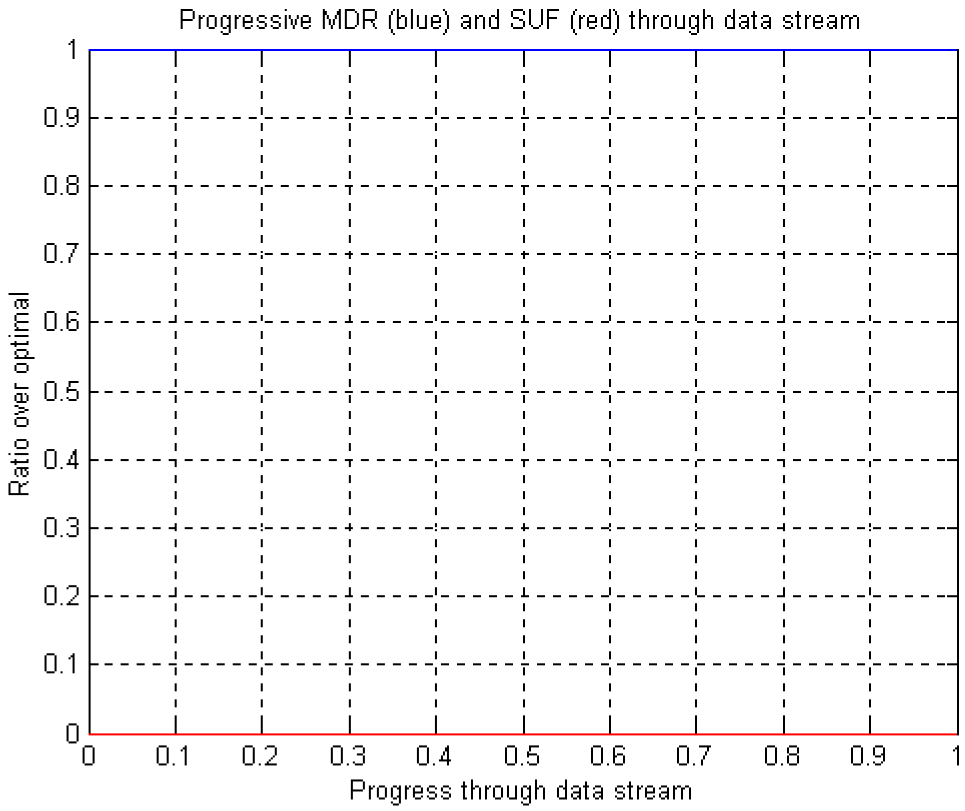
Although the MDR starts at 1 because initially it is undefined (as usual), no bytes occur. Therefore everything cancels to 0. It should be noted that the initial MDR and SUF values are not calculated into the final PBD result; only the MDR and SUF values that are generated from actual data in the data stream are used.

Let us now look at a file that is *perfectly* uniformly distributed. The file is 256 bytes in length, and consists of the full spectrum of bytes 0 through 255, respectively. In other words, every byte occurs in sequence, exactly one time. Since every possible byte is used equally as often, the file falls under the conditions of optimal uniform distribution. PBD analysis of this file yields:

```
File size                            = 256 bytes
Average mean distribution rate       = 100.000000%
Average spectral utilization factor  = 0.000000%

PROGRESSIVE BIT DISTRIBUTION (PBD)   = 100.000000%
```

A graphical analysis of the MDR and SUF data structures yields:



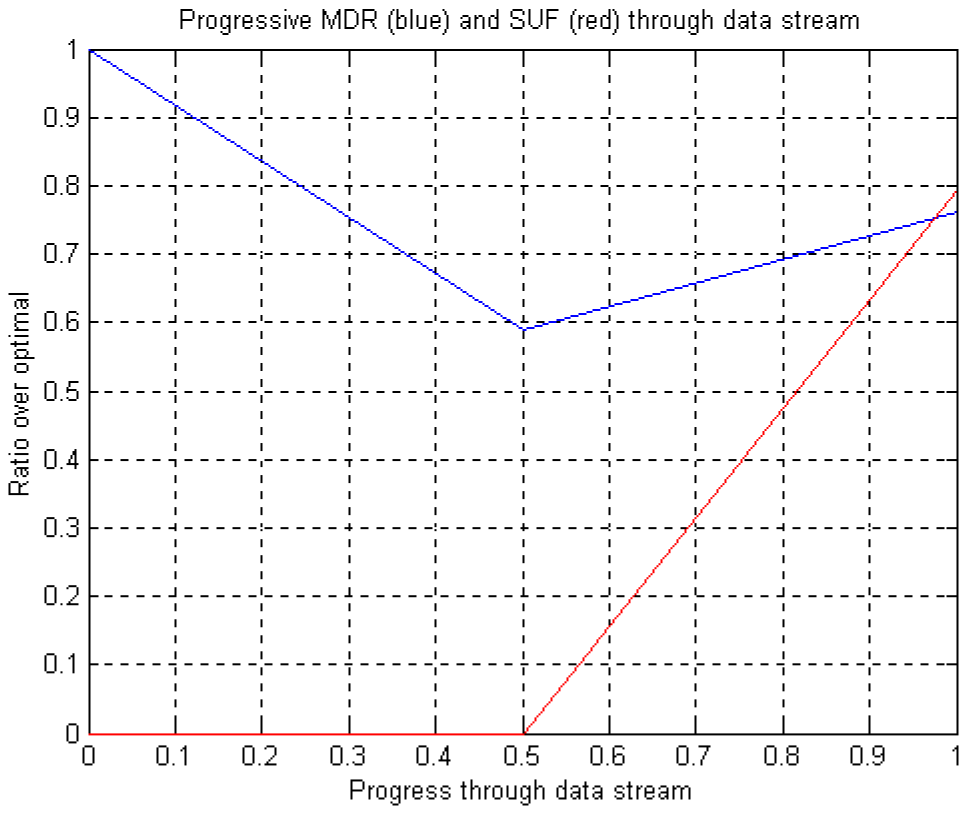
As is expected, the MDR for the file is 1. The SUF of the file is 0 because there are only 256 bytes in the file, and SUF calculation requires at least 257 bytes (even if the file were larger and exhibited the same pattern of uniform distribution, the SUF would still be inconsequential to the PBD calculation, because the data would be completely unchanging and perfectly distributed [yielding an MDR of 1]).

Let us encrypt this "perfect" file using the Rijndael (AES) algorithm (128-bit cipher strength) and see what happens to it. Using a random key, we get the following PBD analysis results:

```
File size                            = 332 bytes
Average mean distribution rate       = 62.810363%
Average spectral utilization factor  = 18.147408%

PROGRESSIVE BIT DISTRIBUTION (PBD)   = 66.710491%
```

A graphical analysis of the MDR and SUF values yields:



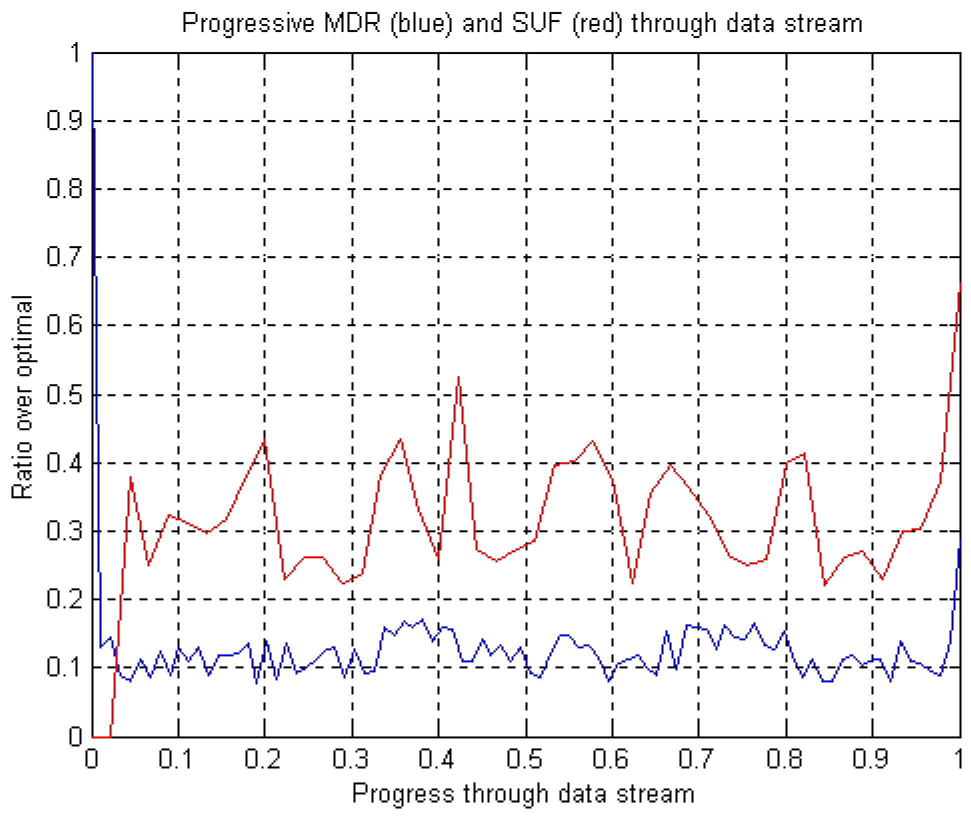
By encrypting this "perfect" file, we have actually lowered the PBD, because an encryption algorithm cannot yield a perfect uniformly distributed file (note that because the file is less than 513 bytes in length, the standard SUF cycle size becomes 256 bytes, so SUF values are attainable. The first cycle's SUF is 0, and the second cycle's SUF is about 0.79. Recall that even though the first cycle's SUF is undefined [0], it is still considered in the PBD equation.).

Now, we will analyze a text file – ABOUT.TXT, included in ETT 6.0, which describes the package in detail. We do not know the actual contents of the file, this time. But we do know that it is a text file, specifically; therefore, we know that only a limited range of bytes will occur (no extended ASCII characters will be used, for example). We do not anticipate that the PBD will be very high. We should expect that the MDR be lower because not too many bytes are used and they are not completely unpredictable (because English words need to be formed), but that the SUF should be comparatively higher because there should be considerable variations in the contents of the data (the same patterns of letters should not be used over and over again in a typical document), although of course it should also be limited because only a finite number of different bytes actually occur. Analysis of ABOUT.TXT yields the following results:

```
File size                            = 22606 bytes
Average mean distribution rate       = 12.073004%
Average spectral utilization factor  = 31.340138%

PROGRESSIVE BIT DISTRIBUTION (PBD)   = 19.778603%
```

A graphical analysis of the MDR and SUF data structures yields:



Our assumptions appear to be correct. The MDR is quite low, between 0.1 and 0.16, generally. But the SUF is much higher, overall, spiking up to 0.525 near the middle of the file. The logic for this is simple, as discussed just previously. There are only a finite number of bytes that are permitted to occur, as it is a standard text file, so the MDR is limited in how high it can reach. The MDR remains quite low, because the letters being used in the document do not vary by too much and are not completely unpredictable; otherwise they would not be able to properly form common English words (but notice that the MDR fluctuates somewhat as patterns in character utilization change). The SUF is much higher than the MDR because the patterns change throughout the document as different sets of words and letter/number/punctuation patterns are used. Notice the greater fluctuations in SUF values; this indicates that some parts of the file utilize a greater degree of changing character patterns than others.

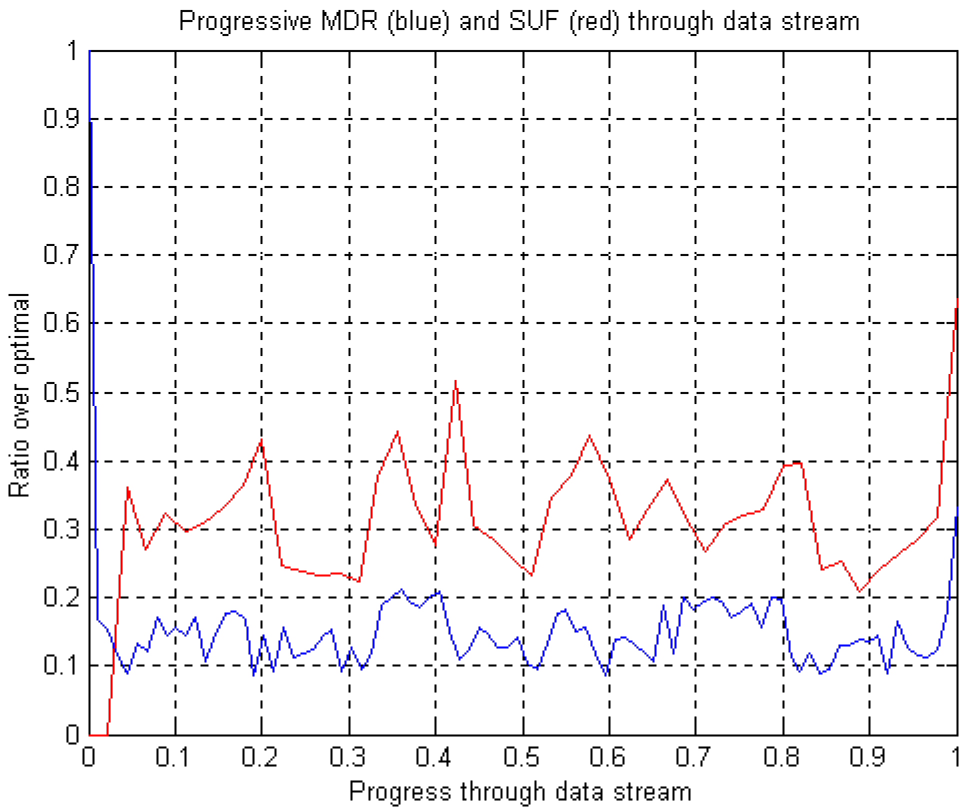
Also notice how the rises and falls in the SUF line correspond, to some extent, to much smaller rises and falls in the MDR line. This is yet another example of the correlation between the two values. Additionally, notice how at the very end of the file, both values rise substantially. This is because less than a full cycle's length of bytes occurs at the very end, therefore less possible outcomes for data distribution are possible. Most files will exhibit this tendency, because their lengths are not exactly divisible by 256. This is exactly the reason why the "significance values" are used in the averaging of the MDR and SUF data structures, prior to final PBD calculation.

Now, let us see what happens when we scramble the contents of the ABOUT.TXT file. We will encrypt the file using the Hill cipher with a 10x10 random-generated matrix. The resulting file will be ABOUT.HIL. Only the letters are changed; all punctuation, spacing, and numbers remain the same. Analyzing the ABOUT.HIL file, we get the following:

```
File size                            = 22613 bytes
Average mean distribution rate       = 14.517090%
Average spectral utilization factor  = 30.944375%
```

```
PROGRESSIVE BIT DISTRIBUTION (PBD)    = 22.737268%
```

You can see that the PBD is higher, as are the average MDR and SUF values. But they are not that much higher, because the formatting of the file remains similar to that of the ABOUT.TXT file. A graphical analysis follows:



You can visually see that the overall MDR values are slightly higher, but follow a similar pattern. In fact, the maximum SUF value never gets higher than it does in the ABOUT.TXT file; in fact, the average SUF is actually slightly *lower* than the average SUF for ABOUT.TXT. The reason for the similarities can be explained because the Hill cipher works upon a type of substitution system – certain letters are substituted for corresponding opposing letters in the alphabet. Therefore, the general tendencies in character distribution in the document have not changed. On the other hand, overall results are higher (particularly with the MDR values), because the data no longer is orderly enough to produce easily recognizable character patterns (or words). But the finite range of bytes still remains, and the formatting remains the same, so the change can only be minimal. The overall SUF is slightly lower because the pattern of bytes is more uniformly distributed and does not change as frequently (but almost).

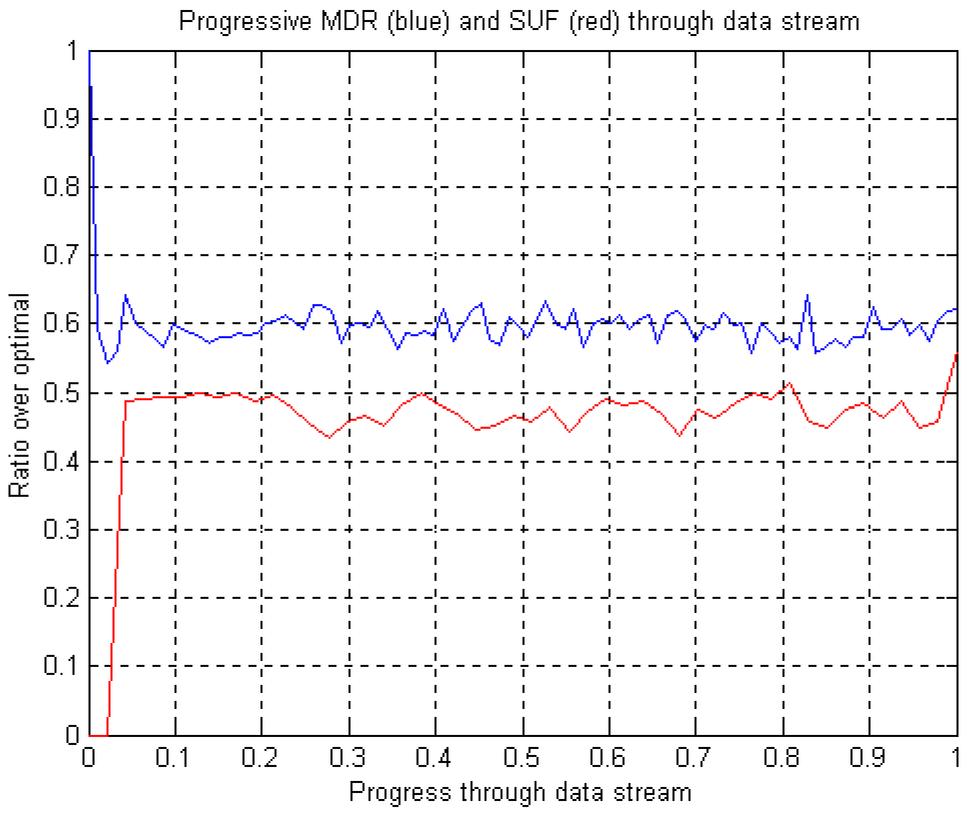
Now, let us look at files that exhibit the full feasible range of byte values – binary or data files. In these files, there should be absolutely no restrictions on how data should be utilized or distributed, as there are with text files. We will now examine case scenarios with seemingly random data sequences, and data sequences that are more predictable with repeating patterns of characters.

First, we will encrypt the file TEST (analyzed earlier) with the AES algorithm (in the same manner as we did previously with the "perfect" uniform file). This will yield a binary file (with all 256 byte combinations used at some point in the file), which we will call TEST.AES. We expect the PBD (and the average MDR and SUF) to be very high with this file. Using a random key to encrypt the file, PBD analysis of the TEST.AES file returns the following:

```
File size                             = 23788 bytes
Average mean distribution rate        = 59.398353%
Average spectral utilization factor   = 46.462706%

PROGRESSIVE BIT DISTRIBUTION (PBD)    = 70.113440%
```

Graphical analysis of the MDR and SUF values yields the following:



You will notice that this file exhibits high levels of uniform distribution and data utilization – higher than all of the previous files we have examined. In fact, this is about as high as the MDR and SUF values can get for an essentially random data sequence (although, of course, not *truly* random; a software algorithm can never generate an actual random sequence, just a sequence that *appears* random in many respects). This makes sense when you consider that if there is no chance of bytes occurring in any particular sequence or order, then *absolute* uniform distribution is impossible. Technically, the content of the data stream should be at least 50% uniformly distributed (an MDR of at least 0.5), because at least half of the bytes should be used equally as often. The same applies for the degree of change in the data content between cycles; the SUF should be at least 0.4 but should approach 0.5. This makes sense when you consider that the SUF should "compensate" the MDR; that is, the pattern of distribution should change just enough such that as many bytes as possible are evenly distributed.

These rules apply because of the principles of the basic probability density equation covered earlier:

$$P(x) = \frac{1}{(b-a)}$$

If the interval [0, 255] is used, then the expected byte to be encountered is halfway through this interval (128 in this case)[*]. Therefore, it follows that in a random data stream, for every 256 bytes, generally about half of the total spectrum of feasible bytes will be used.

So, we know through general probability theory that in a random data sequence, the data should not be perfectly uniformly distributed, but should be relatively close. In fact, the higher the MDR and SUF values are, together, the more "random" the file should be. But there are a couple of limitations to this principle:

- Both the MDR and SUF must be high *together*, and the SUF must follow a general tendency of being just less than the MDR. This is because the MDR must demonstrate a constant state of good uniform distribution, and the contents of the data stream should be changing on an order that effectively "compensates" for bytes that do not get uniformly distributed, so that *all* bytes get a chance to be uniformly distributed at various parts of the data stream (to make up for the missing byte distribution indicated by the MDR). If the MDR is high, and the SUF is significantly lower than the MDR continuously, the data stream in question probably is not random in nature. Important exception: when the data stream is less than 257 bytes in length, the SUF is always 0, so the SUF restrictions do not apply. Still, the MDR must *not* be 1.
- The data stream is probably not random if the SUF is generally *higher* than the MDR, because this indicates that more bytes are actually being used than are being evenly distributed, and this would normally indicate a substantial deficit in a "balanced" distribution rate. Because a significant change in the pattern of the bytes occurs, not many bytes were utilized and evenly distributed in the first place, so a reasonable conclusion can be drawn that the data stream is acceptably not random.
- In accordance with the laws of probability, and with general principles of random data, the MDR must be continuously above 0.5 in order for randomness to even be considered a possibility (but *not* a guarantee) in the data stream. Higher values indicate an even better chance of the data stream being random. The SUF must lie just below the MDR, on the order of at least 0.4 but closer to 0.5.
- There is an absolute upper limit, defined by the laws of probability, which the MDR can attain for a random data sequence. This can be generalized by saying that because of the unpredictable nature of a random data sequence, perfect uniform distribution is impossible.
- The degree of variance (from a graphical standpoint, the degree of rises and falls in the line) within the MDR data structure must remain minimal (because the degree of uniform distribution must be consistently high; variations indicate problems with the nature of the distribution of the data in the data stream. An ideal random file would display very consistent tendencies in uniform distribution because the conditions for the state of data generation do not change at any point in the file.). The SUF data structure is allowed more flexibility; the primary restriction on SUF is that it should always remain *slightly* lower (but not significantly lower) than the MDR at any given point, in order for randomness to be considered in the data stream (unless, of course, the data stream is less than 257 bytes in length).

Here is an example of a binary file that exhibits non-random characteristics as defined by the preceding guidelines. The TEST file is to be encrypted using the Bazeries encryption algorithm, into a file named TEST.BAZ. The Bazeries cipher exhibits a tendency of leaving "digital fingerprints", or repeating sequences of bytes that correspond to repeating characters in the plaintext (unencrypted) file.

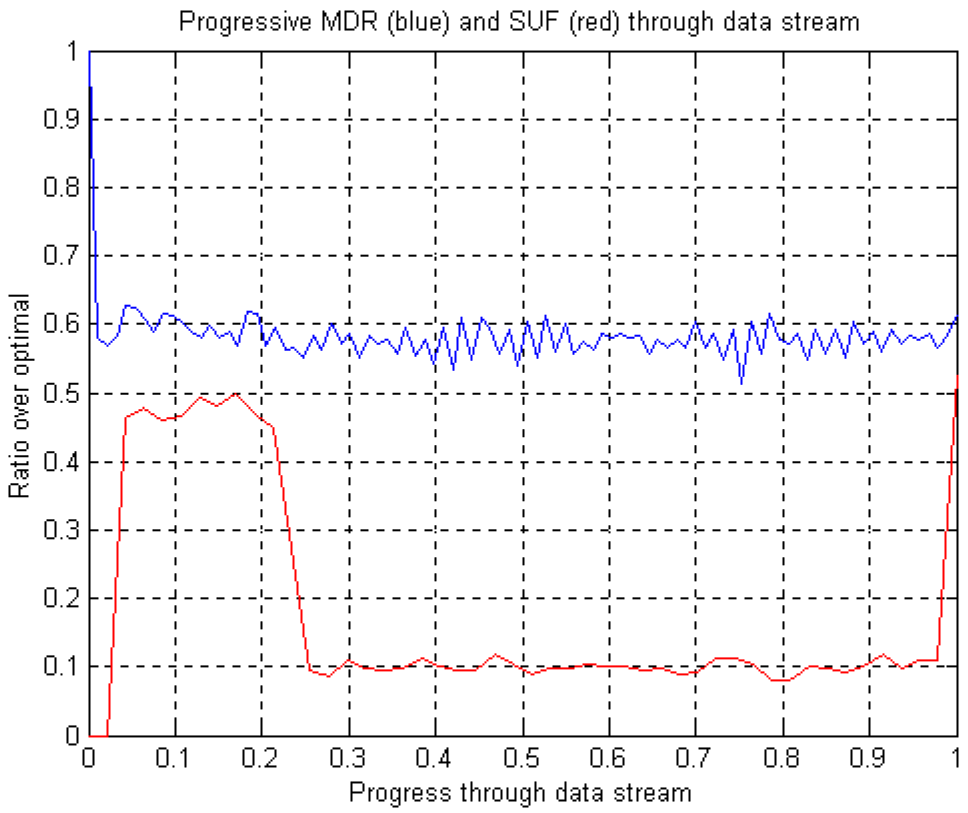
PBD analysis of TEST.BAZ yields the following results:

```
File size                          = 23736 bytes
Average mean distribution rate     = 57.981826%
Average spectral utilization factor  = 17.719142%

PROGRESSIVE BIT DISTRIBUTION (PBD)   = 62.178727%
```

A graphical analysis yields the following:

[*] Christian Schiestl, ***Pseudozufallszahlen in der Kryptographie***, Klagenfurt, 1999. If P(x) applies to a random data sequence and x falls within the interval [a, b], the expected value for x is calculated as E(x) = (a + b) / 2.



You can clearly see the SUF plummet far below the MDR after about 20% to 25% progress through the file. You will notice that the MDR is consistently above 0.5 (actually closer to 0.6), and remains fairly steady. However, although the bytes are uniformly distributed along random-sequence guidelines, a deficit of bytes exists (as indicated by the very low SUF values) – so the file cannot be random, because not all bytes are used to "make up" for the distribution deficit indicated by the MDR. This graph indicates that there are continuously repeating sequences of bytes throughout most of the file (indicating a cryptographic weakness in the Bazeries algorithm).

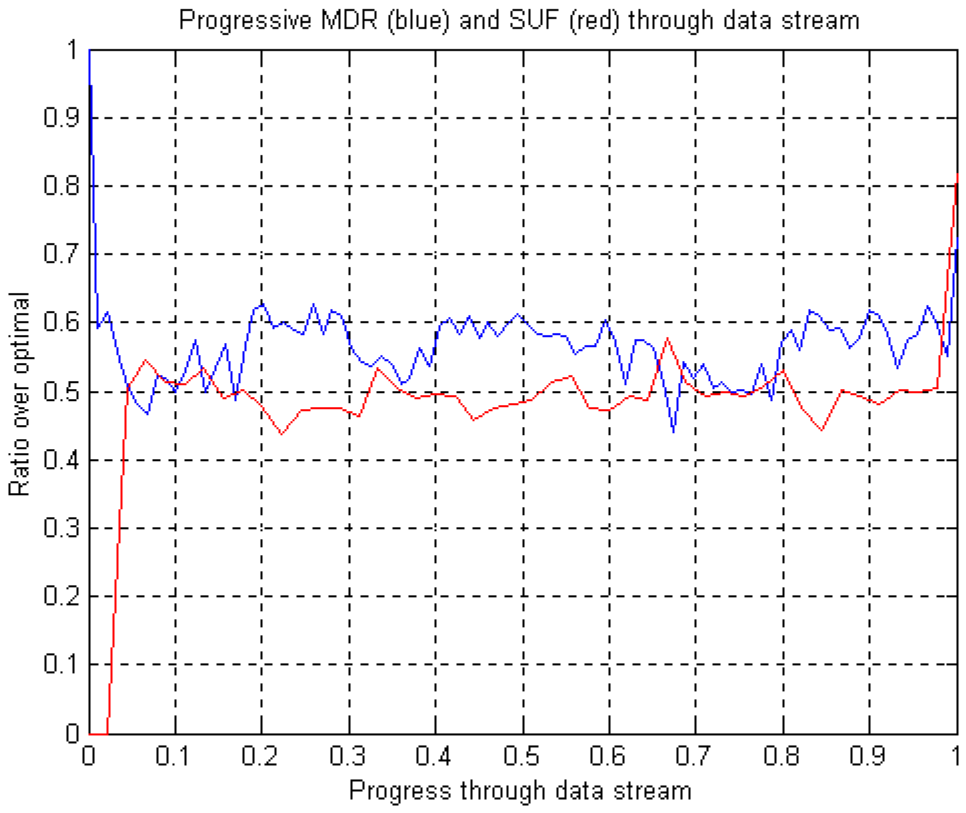
Now, let us take an example that is a bit harder to determine as being random from PBD analysis. We will encrypt the ABOUT.TXT file (referenced earlier in this section) using the DES algorithm in ECB mode, to a binary file named ABOUT.DES. This most certainly creates repetitive patterns of bytes in repetitive plaintexts (even more so than with the Bazeries algorithm). However, since ABOUT.TXT is not very repetitive in nature (as compared to the TEST file), clearly distinguishable fingerprints will not be present – although sequences of characters will still probably repeat to some extent.

PBD analysis of ABOUT.DES yields:

```
File size                           = 22608 bytes
Average mean distribution rate      = 56.251236%
Average spectral utilization factor = 48.495744%

PROGRESSIVE BIT DISTRIBUTION (PBD)  = 68.095966%
```

A graphical analysis yields:



As you can see, this file exhibits tendencies towards being random. The MDR is generally above 0.5, and the SUF generally remains just below the MDR. But notice that both the MDR and SUF lines in this graph are lower than those from the TEST.AES file, referenced earlier in this section. This is a clue that the TEST.AES file is more random in nature than the ABOUT.DES file.

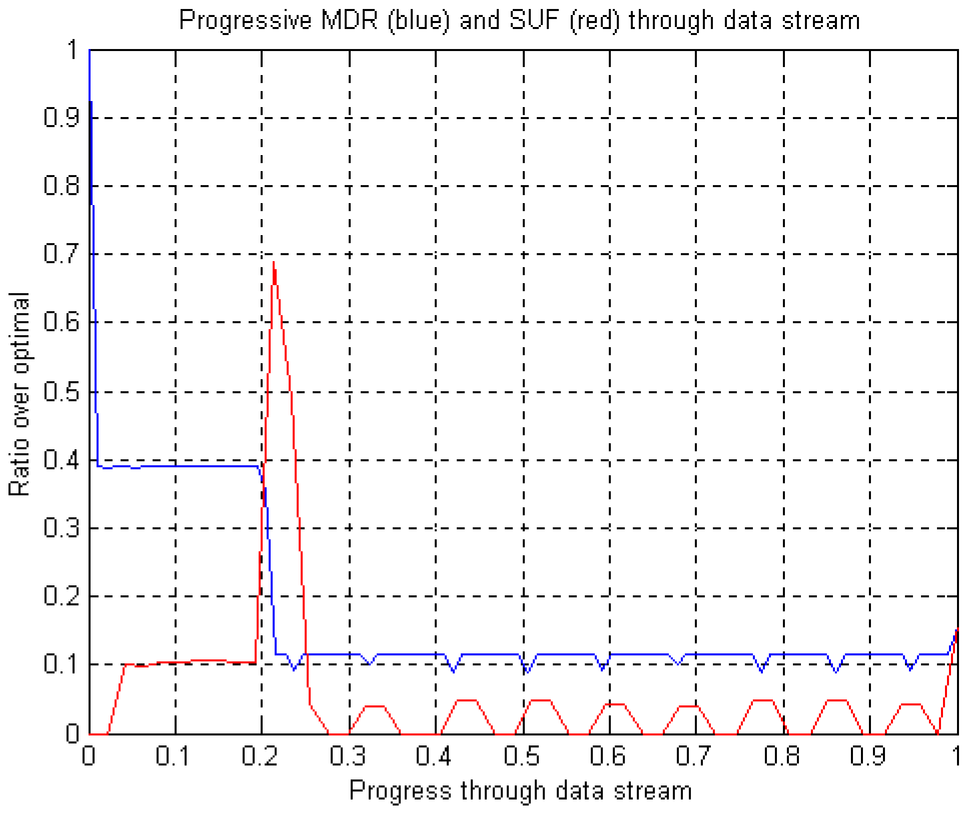
But without comparing ABOUT.DES to any other files, how would we determine whether or not it is random? Unfortunately, there are no clear-cut guidelines for this, and arguments could be made either way. But perhaps the biggest clue is the fact that the MDR is not consistent – it rises and falls rather dramatically throughout the file. It even drops to 0.45 at one point. As was pointed out in the previous guidelines, the MDR should not change too much through the file. Apparently, ABOUT.DES is not consistent in its tendencies of uniform distribution (as stated earlier, an ideal random file would display very consistent tendencies in uniform distribution because the conditions for the state of data generation do not change at any point in the file). So, more than likely, the ABOUT.DES file is not random – or at least, not ideally random.

Encrypting the file TEST with the DES (ECB) algorithm will certainly generate a wildly non-random file. This file, which we will call TEST.DES, yields the following PBD analysis results:

```
File size                            = 23720 bytes
Average mean distribution rate       = 17.011016%
Average spectral utilization factor  = 6.090122%

PROGRESSIVE BIT DISTRIBUTION (PBD)   = 18.721931%
```

Graphical analysis yields:



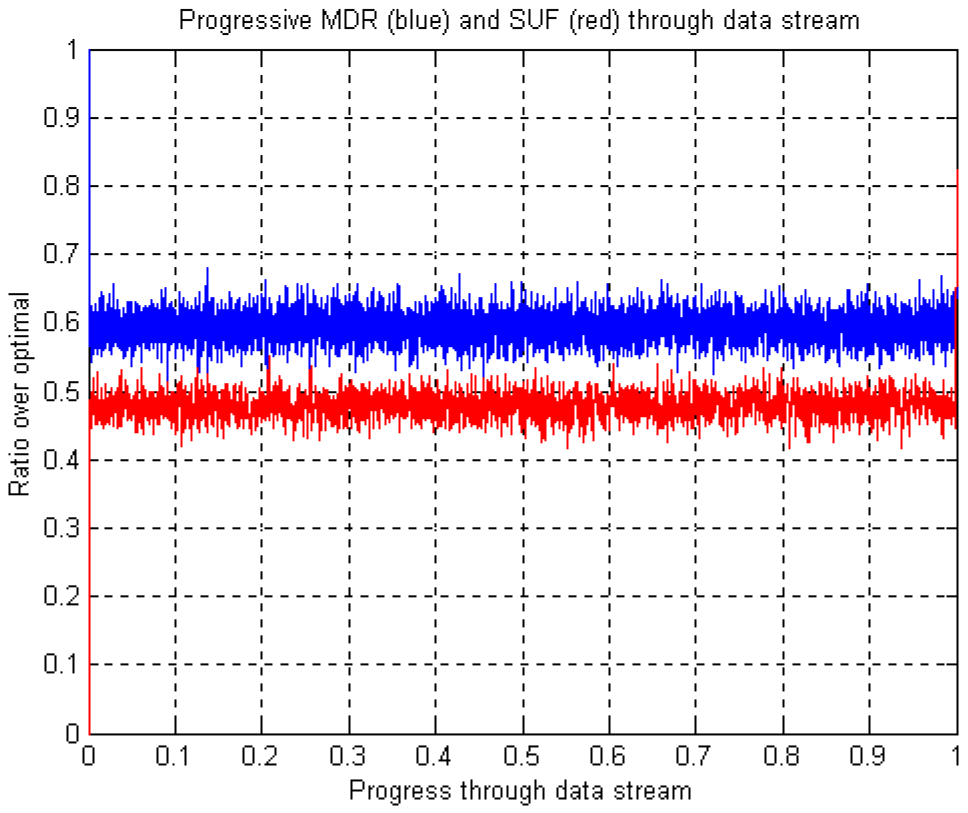
Notice the similarities in this graph to the graph of TEST, shown earlier in this section. The similarities are not coincidental. The TEST.DES file demonstrates the exact same tendencies as the TEST file – only with the contents of the file changed. The only reason the MDR and SUF values are higher in this graph is because proportionally more different bytes are used in TEST.DES. Other than that, the format of the file remains nearly identical to that of the TEST file.

Now, let us analyze a larger file and view its graph. We will start by encrypting the BIGNULL file, mentioned earlier in this section, with the AES algorithm. We will call this file BIGNULL.AES. PBD analysis of this file yields:

```
File size                           = 1433676 bytes
Average mean distribution rate      = 59.324304%
Average spectral utilization factor = 47.790929%

PROGRESSIVE BIT DISTRIBUTION (PBD)  = 70.361491%
```

A graphical analysis of this file yields:



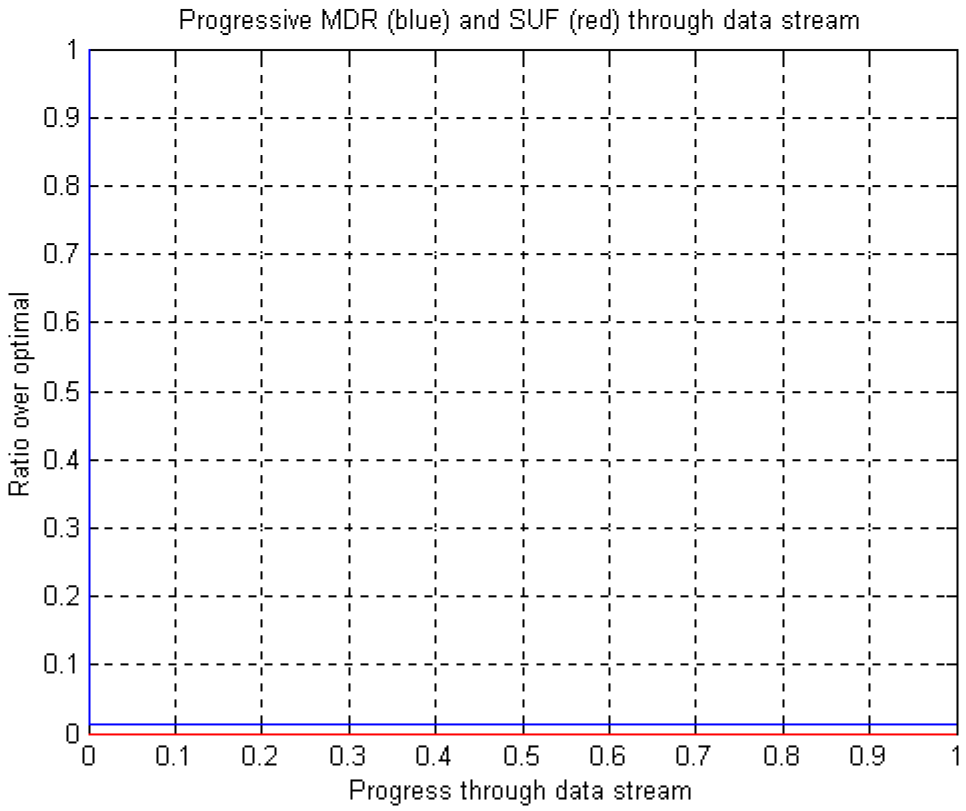
You can see that since the file is much larger than the files we have looked at up to this point, the MDR and SUF lines are much more tightly compacted. But notice that overall, the tendencies of this graph are similar to the tendencies of the graph of the TEST.AES file. Because the MDR values are consistently above 0.5 and the SUF values are just below the MDR values, and because the MDR and SUF values do not fluctuate too much (keep in mind that as the file size increases, the fluctuation of the MDR and SUF values will also generally increase just because more data is being analyzed), it can be safely assumed that the BIGNULL.AES file is probably random in nature (remember that when we say "random", we actually mean that the file *appears* random, not that it actually is random. It is impossible to generate a truly random sequence on a computer using a software algorithm.).

Now, if we encrypt the BIGNULL file using the DES algorithm, and call it BIGNULL.DES, we would come up with a file that exhibits highly repetitive sequences (especially since only 1 type of byte occurs in the BIGNULL file). PBD analysis of the first megabyte of BIGNULL.DES yields:

```
File size                          = 1048575 bytes
Average mean distribution rate     = 1.171876%
Average spectral utilization factor  = 0.000046%

PROGRESSIVE BIT DISTRIBUTION (PBD)   = 1.171878%
```

A graphical analysis yields:



The MDR is *very* low, and the SUF is very close to 0 because the contents of the file almost never change. The MDR is a perfectly straight line because the degree of uniform distribution never changes (because the data never changes). This file is acceptably non-random in nature (obviously), and in fact contains very recognizable repeating byte sequences.

Let us now "break away" from encrypted files and examine other types of files, to determine their PBD rates and graphs. We have already examined one document file – ABOUT.TXT – and saw that since a finite number of bytes are permitted to occur in a text file, the PBD, MDR, and SUF values are all limited.

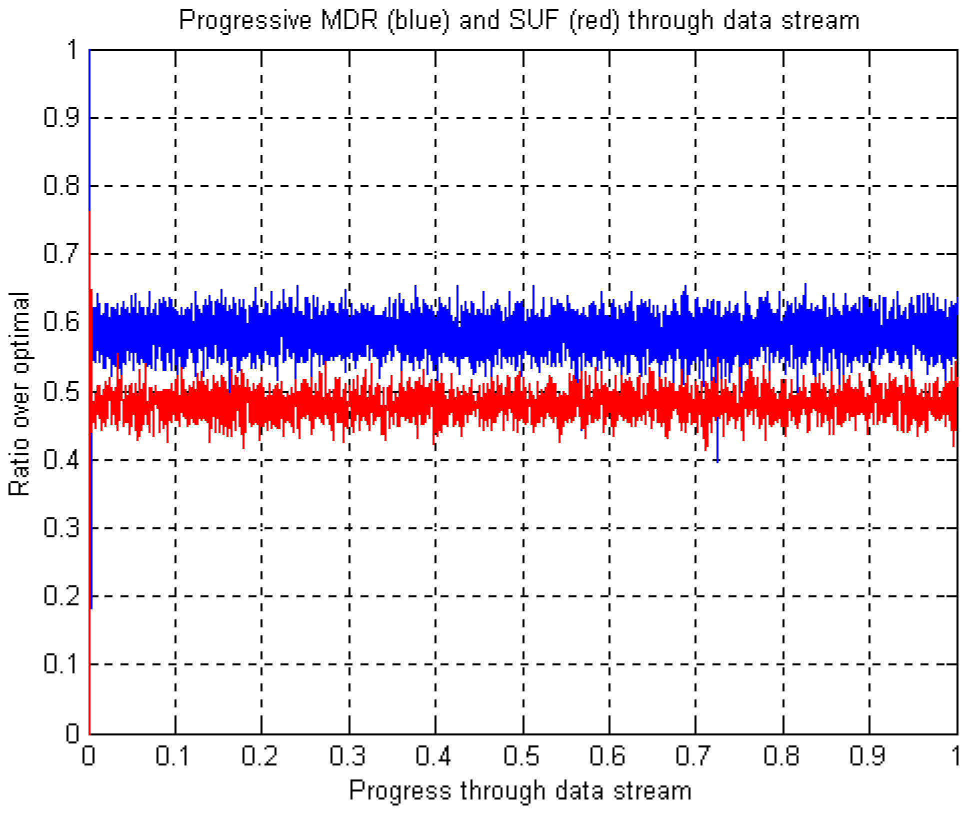
We will first examine a JPEG picture file. A JPEG file contains a high information density, since it is a highly compressed picture. So, we would expect the PBD of a JPEG file to be rather high. However, from common knowledge we already know that a JPEG file cannot have truly random tendencies, because the data that it works upon is fairly orderly. But because of the high amount of information that is contained in the file, it will take on many characteristics of a randomly generated data stream, such as good uniform distribution and content utilization. We will have to closely examine the graph of the JPEG file to ascertain whether or not it can be considered random-like.

PBD analysis of our JPEG file yields the following:

```
File size                           = 1666980 bytes
Average mean distribution rate      = 58.321983%
Average spectral utilization factor = 48.102651%

PROGRESSIVE BIT DISTRIBUTION (PBD)  = 69.644100%
```

The graph of the MDR and SUF values of this file yields:



By quickly glancing at the graph, you can notice that it has many of the properties of a random data stream. It looks quite similar to the graph of the BIGNULL.AES file shown earlier in this section. It can be argued that this JPEG file exhibits random tendencies.

However, if you more closely analyze this graph, you will notice some tendencies in the MDR line that can help to disprove the argument that it is random. Notice that the degree of variation in the MDR line is quite substantial at certain points. For example, it drops below 0.5 several times and even drops as low as 0.4. If you look closely, at the very beginning of the file, the MDR actually is lower than 0.2. Because of not only the degree of variation in the MDR values through the file, but also the fact that it periodically drops below the "random threshold" of 0.5, it is very likely that this JPEG file does not exhibit random characteristics.

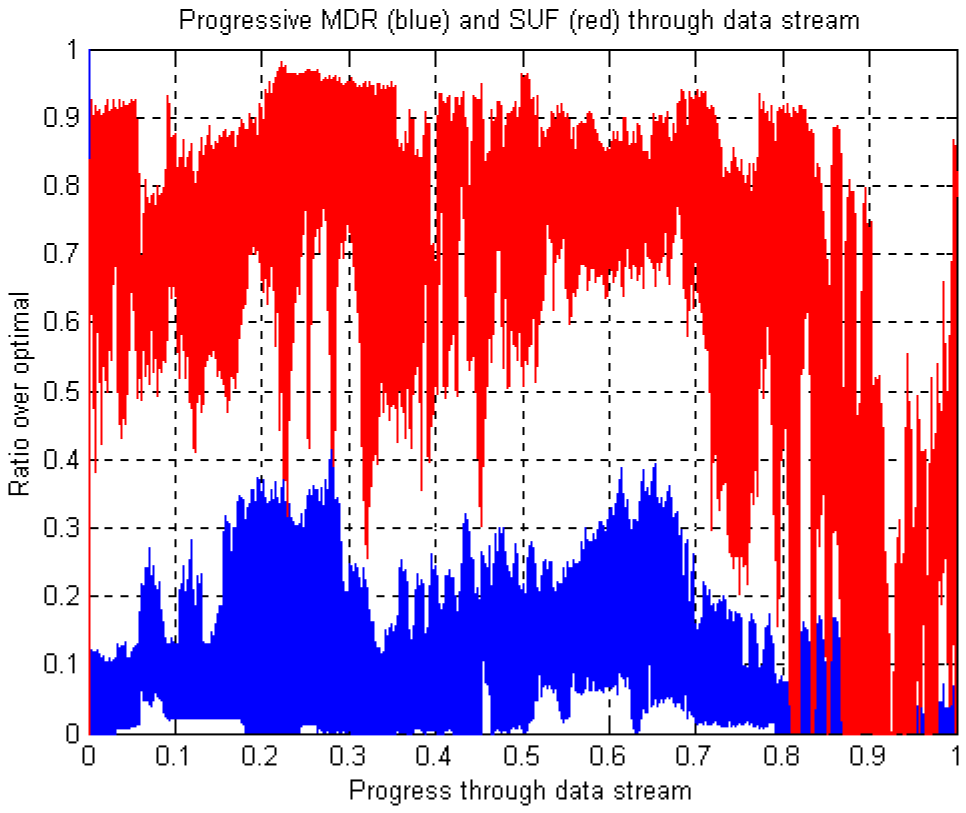
Now, let us examine a regular Windows bitmap (BMP) picture file, to compare it to the JPEG file we just examined. Unlike a JPEG file, a bitmap is not compressed, and therefore contains relatively repetitive and predictable data (with the degree of repeating sequences dependent upon the complexity of the picture). A bitmap is also, comparatively speaking, many times larger than a corresponding JPEG file.

PBD analysis of a 24-bit, rather complex bitmap file yields the following results:

```
File size                            = 2359352 bytes
Average mean distribution rate       = 8.969994%
Average spectral utilization factor  = 67.301487%

PROGRESSIVE BIT DISTRIBUTION (PBD)   = 23.225882%
```

A graphical analysis yields the following:



You can see immediately that this graph looks wildly different from the other graphs we have examined. Notice that the MDR is very low, but the SUF is *very* high (almost approaching 1 in some places). The reason that the SUF is so high is because the bitmap has a tendency of exhibiting one type of repeating sequence of characters, followed immediately by a completely different set of repeating characters, and so on throughout the file – which helps to make up for the deficit exhibited by the weak distribution rate (MDR). But notice that the SUF drops off towards the end of the file, as the bytes repeat themselves without changing as much.

Notice that even though the SUF indicates a high rate of variations in the *pattern* of distribution of data, the MDR is still very low because the data is not very evenly distributed. Therefore, the final PBD result is a quite low value. Everything indicates that this bitmap file is acceptably non-random.

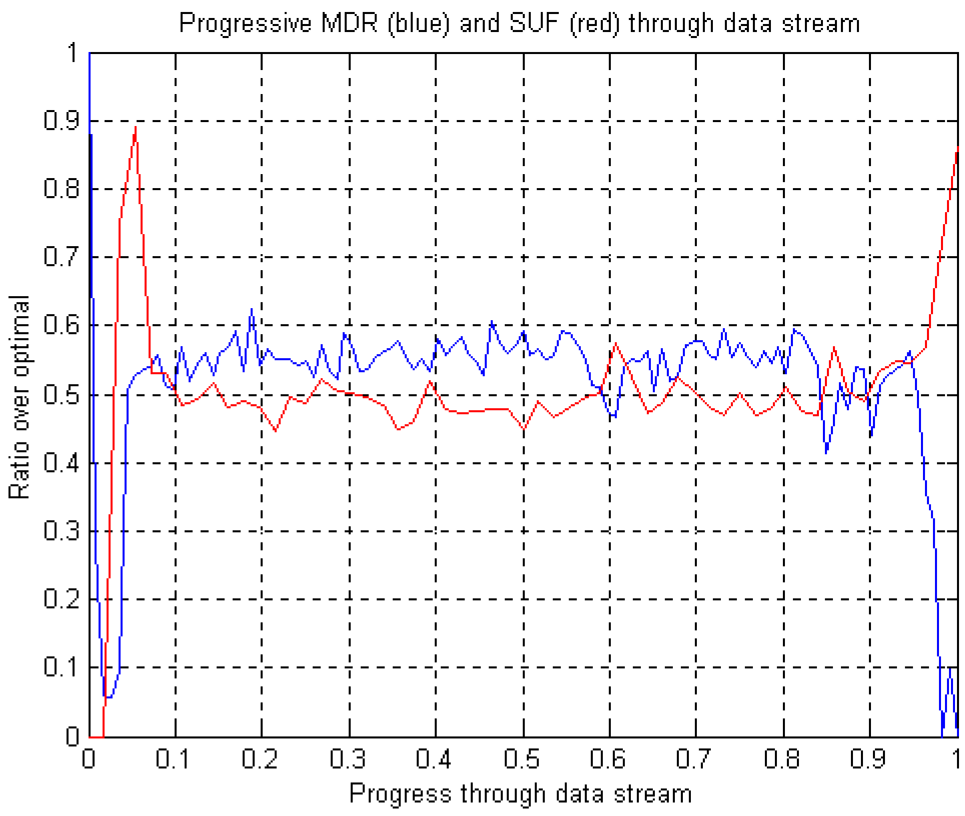
We will examine one last file in this section: an executable (EXE) file. We will examine the PBD.EXE file (a compiled version of the C source code implementation of the PBD algorithm), which is a Win32 console application that has been compressed by an EXE-packaging utility. Because it is compressed, it exhibits a high information density and should therefore exhibit some random characteristics. However, logic tells us that it cannot be acceptably random in nature.

PBD analysis of PBD.EXE yields:

```
File size                            = 28672 bytes
Average mean distribution rate       = 51.417101%
Average spectral utilization factor  = 51.012922%

PROGRESSIVE BIT DISTRIBUTION (PBD)   = 64.852204%
```

Graphical analysis yields the following:



The tendencies in the MDR and SUF values for this file lean towards being random. However, you will notice that there is a great deal of variation in the MDR line. It sinks far below the 0.5 threshold on several occasions, and even drops down to near 0 at the beginning and end of the file. So, this file is acceptably non-random in nature. You will notice that the overall PBD rate is lower than the average PBD rate for the files that were "acceptably" random in nature.

Notice how the SUF shoots up over 0.89 when the MDR is close to about 0.06, near the beginning of the file. This is because the data contents change substantially at this point. A similar pattern occurs at the end of the file, and for the same reason.

You can now see how close examination of the tendencies of the MDR and SUF values through a data stream, coupled with the final PBD result, can assist in ascertaining the state of the data stream – whether it is random, perfectly uniform, completely non-uniform, or somewhere in-between. Obviously, however, PBD by itself cannot totally determine whether or not a data stream is random. For that, a randomness-analysis test (such as Chi^2 distribution) is necessary. But still, PBD can be a very helpful tool in the analysis of any given data stream.

But beyond analyzing binary data on a computer system, PBD can also be applied to real-world probability problems, by simply substituting a few parameters that we have used up to now in the analysis of digital data.

VII. A Sample Application of PBD in a Real-World Problem

PBD was designed originally to assist in the analysis of files generated by encryption algorithms and random number generators. However, the concepts of PBD can be applied to situations outside of the computer realm, to look for the same information – the degree of uniform distribution and utilization of all feasible data elements.

As PBD is a new concept, its exact practicality in situations outside of the computer realm is unknown at this time. More research is necessary to verify its usefulness in real-world scenarios, and productive research will be performed once PBD is proven first in the digital realm (this document will be updated with any additional information that is obtained). For now, we will examine a very simple probability problem and obtain a single PBD rate, in order to demonstrate how a theoretical application of PBD in such a situation might work.

The first thing that should be done in utilizing PBD for real-world application is to recognize that the parameters that define the MDR and SUF cycle sizes and the total number of feasible elements must be redefined to fit the situation at hand. Additionally, when referring to the data to be analyzed, we no longer consider it as a "file" or "data stream", but rather as a "data sequence". Also, each item in the data is not a "byte", but rather a "data element". So, we will refer to the content we will be analyzing as data sequences, and we will be adjusting the cycle sizes and total number of feasible data elements in accordance with the data sequences.

Before we begin looking at a simple case scenario, we must overview how the total number of feasible data elements, standard MDR cycle size, and standard SUF cycle size are defined. Obviously, the total number of feasible data elements (which we will represent with the variable "m") is known before the PBD is calculated. From that, we know that the default MDR cycle size is always equivalent to m, and the default SUF cycle size is always equivalent to twice m (2m), unless there are less than or equal to 2m data elements in the data sequence, in which case the default SUF cycle size becomes m.

Mathematically, the MDR and SUF default cycle sizes can be represented in terms of s_d and s_v, respectively, as follows:

$$s_d = m \qquad\qquad s_v = \{[l > 2m \therefore 2m] \quad \vee \quad [l \leq 2m \therefore m]\}$$

This is simply the mathematical equivalent of what we just discussed.

Now, with this in mind, let us look at a very simple problem. It may not have any practical meaning, but it demonstrates the use of PBD in a non-computer situation.

Here is the problem:

A fabric company makes a line of swatches using only the four primary pigments cyan (C), magenta (M), yellow (Y), and black (K), without combining any of them. The line is 14 swatches long, and is made up of the following pattern:

K, M, Y, M, C, Y, K, K, M, M, Y, M, C, K

What is the Progressive Bit Distribution of this swatch sequence?

By looking at this problem, we know (a) how many feasible data elements there are, and (b) how many total data elements there are in the data sequence. There are 4 colors (C, M, Y, and K), so there are 4 feasible data elements total. The default length of each MDR cycle will therefore be 4 data elements. There are a total of 14 data elements altogether, which is more than twice the number of feasible elements, so the default length of each SUF cycle will be 8 data elements.

First, we calculate all of the MDR values. Breaking down the data elements in the sequence, we come up with the following MDR cycles to be analyzed (one at a time):

K, M, Y, M
C, Y, K, K
M, M, Y, M
C, K

Now we take into account the general MDR equation:

$$d_{c_d} = \frac{\left|\ln\left(\frac{x_1}{s_d}\right)\right| + \left|\ln\left(\frac{x_2}{s_d}\right)\right| + \left|\ln\left(\frac{x_3}{s_d}\right)\right| + \ldots + \left|\ln\left(\frac{x_n}{s_d}\right)\right|}{s_d \cdot \left|\ln\left(s_d\right)\right|}$$

First, let us find the MDR for cycle 1. There are 3 of the 4 feasible values used (K, M, and Y). K occurs once, M occurs twice, and Y occurs once. Plugging in all the values in this equation yields the following:

$$d_1 = \frac{\left|\ln\left(\frac{1}{4}\right)\right| + \left|\ln\left(\frac{2}{4}\right)\right| + \left|\ln\left(\frac{1}{4}\right)\right|}{4 \cdot \left|\ln\left(4\right)\right|} = 0.625$$

If we solve this for d_1, we get 0.625.

For the second cycle, there are 3 of the 4 feasible values used (C, Y, and K). C occurs once, Y occurs once, and K occurs twice. We solve for the following:

$$d_2 = \frac{\left|\ln\left(\frac{1}{4}\right)\right| + \left|\ln\left(\frac{1}{4}\right)\right| + \left|\ln\left(\frac{2}{4}\right)\right|}{4 \cdot \left|\ln\left(4\right)\right|} = 0.625$$

Solving for this, we get 0.625 for d_2.

For the third cycle, there are 2 of the feasible 4 values used (M and Y). M occurs three times, and Y occurs once. We solve for the following:

$$d_3 = \frac{\left|\ln\left(\frac{3}{4}\right)\right| + \left|\ln\left(\frac{1}{4}\right)\right|}{4 \cdot \left|\ln\left(4\right)\right|} \approx 0.3019$$

Solving for d_3, we get about 0.3019.

For the last cycle, there are 2 values used (C and K). C and K both occur once. There are only 2 feasible values for this cycle because it is only 2 data elements long. We solve for the following:

$$d_4 = \frac{\left|\ln\left(\frac{1}{2}\right)\right| + \left|\ln\left(\frac{1}{2}\right)\right|}{2 \cdot \left|\ln\left(2\right)\right|} = 1$$

Solving for d_4, we get 1 (perfectly uniform). However, since the length of this cycle is less than the default (2 values instead of 4), we must apply the rules of significance in its calculation. So, we take 1 and multiply it by the significance value for this cycle (2 / 4 or 0.5). Therefore, the value of d_4, for purposes of PBD calculation, is 0.5.

Also, the number of MDR cycles is affected by the rules of significance. The last cycle (technically, cycle 4) has a length of 2. Therefore, the number of MDR cycles is only incremented by 0.5 (2 / 4) for this cycle. As a result, the number of MDR cycles, for purposes of PBD calculation, is 3.5.

Now that we've found all of the MDR values for the data sequence, we must now find all of the SUF values for the data sequence. We break down the data sequence into the following SUF cycles to be analyzed sequentially:

K, M, Y, M, C, Y, K, K
M, M, Y, M, C, K

There are only 2 SUF cycles; only 1 SUF value is obtained (for the second cycle in relation to the first cycle). We assume v_1 = 0, because for the first SUF cycle, the SUF is undefined (hence it becomes 0).

The first thing we do is consider the SUF equation:

$$v_{c_v} = \frac{y_1 + y_2 + y_3 + \ldots + y_n}{z_1 + z_2 + z_3 + \ldots + z_n}$$

We must obtain each y and z value. Recall that each y is defined as:

$$y_n = \left| \ln\left(\frac{x_2}{x_1} \right) \right|$$

Likewise, each z is defined as:

$$z_n = \left\{ \left[x_1 > x_2 \therefore \left| \ln(x_1) \right| \right] \quad \vee \quad \left[x_1 \le x_2 \therefore \left| \ln(x_2) \right| \right] \right\}$$

So the next thing we have to do is calculate each y and z value. In order to do this, we must calculate each x_1 and x_2, using the following equation:

$$x_n = \frac{ma_n}{s_{v_n}} + 1$$

In the current cycle, all 4 elements (C, M, Y, and K) exist. C occurs 1 time in both the current cycle and the previous cycle. M occurs 3 times in the current cycle and 2 times in the previous cycle. Y occurs 1 time in the current cycle and 2 times in the previous cycle. K occurs 1 time in the current cycle and 3 times in the previous cycle.

We will use y_1 through y_4 and z_1 through z_4 to calculate the SUF of the current cycle in relation to the previous cycle as follows:

$$v_2 = \frac{y_1 + y_2 + y_3 + y_4}{z_1 + z_2 + z_3 + z_4}$$

To calculate y_1 and z_1 (which will represent the C values), we do the following:

First, we convert both the C occurrence rate in the previous cycle (a_1) and the C occurrence rate in the current cycle (a_2) into x_1 and x_2 values (respectively) that represent identical and comparable figures. Plugging in all values, we get the following:

$$x_1 = \frac{4 \cdot 1}{8} + 1 = 1.5 \qquad\qquad x_2 = \frac{4 \cdot 1}{6} + 1 \approx 1.6667$$

Plugging x_1 and x_2 to solve y_1 and z_1, we get the following:

$$y_1 = \left|\ln\left(\frac{1.6667}{1.5}\right)\right| \approx 0.1054 \qquad z_1 = \left\{\left[1.5 \le 1.6667 \therefore \left|\ln\left(1.6667\right)\right|\right]\right\} \approx 0.5108$$

Now, we will calculate y_2 and z_2 (which will represent the M values). Following the same sequence of calculations, we solve for y_2 and z_2 as follows:

$$x_1 = \frac{4 \cdot 2}{8} + 1 = 2 \qquad x_2 = \frac{4 \cdot 3}{6} + 1 = 3$$

$$y_2 = \left|\ln\left(\frac{3}{2}\right)\right| \approx 0.4055 \qquad z_2 = \left\{\left[2 \le 3 \therefore \left|\ln\left(3\right)\right|\right]\right\} \approx 1.0986$$

Calculating y_3 and z_3 (which will represent the Y values), we get the following:

$$x_1 = \frac{4 \cdot 2}{8} + 1 = 2 \qquad x_2 = \frac{4 \cdot 1}{6} + 1 \approx 1.6667$$

$$y_3 = \left|\ln\left(\frac{1.6667}{2}\right)\right| \approx 0.1823 \qquad z_3 = \left\{\left[2 > 1.6667 \therefore \left|\ln\left(2\right)\right|\right]\right\} \approx 0.6931$$

Calculating y_4 and z_4 (which will represent the K values), we get the following:

$$x_1 = \frac{4 \cdot 3}{8} + 1 = 2.5 \qquad x_2 = \frac{4 \cdot 1}{6} + 1 \approx 1.6667$$

$$y_4 = \left|\ln\left(\frac{1.6667}{2.5}\right)\right| \approx 0.4055 \qquad z_4 = \left\{\left[2.5 > 1.6667 \therefore \left|\ln\left(2.5\right)\right|\right]\right\} \approx 0.9163$$

Now that we have all y values and all z values, we can plug them all back into the original equation to find the final SUF for the current cycle:

$$v_2 = \frac{0.1054 + 0.4055 + 0.1823 + 0.4055}{0.5108 + 1.0986 + 0.6931 + 0.9163} \approx 0.3413$$

So, the SUF of cycle 2 is approximately 0.3413.

The second SUF cycle is less than the default (6 values as opposed to 8), so the rules of significance apply. The value of cycle 2 (0.3413) is multiplied by the significance value (6 / 8 or 0.75), to yield approximately 0.256 as the correct value for v_2.

Also, the number of SUF cycles is affected by the rules of significance. The 2nd SUF cycle is 6 values instead of 8, so 6 / 8 or 0.75 gets added to the number of SUF cycles instead of 1. Therefore, the total number of SUF cycles, for purposes of PBD calculation, is 1.75.

Now that we have all MDR and SUF values for the entire data sequence, we can calculate the PBD of the data sequence. First, we combine all MDR values as follows:

$$d = 0.625 + 0.625 + 0.3019 + 0.5 = 2.0519$$

Now, we combine all SUF values as follows:

$$v = 0 + 0.256 = 0.256$$

Next, we divide d by the computed number of MDR cycles, and v by the computed number of SUF cycles, to average both data structures. We get the following:

$$d = \frac{2.0519}{3.5} \approx 0.5863 \qquad v = \frac{0.256}{1.75} \approx 0.1463$$

Finally, we plug the averaged MDR and SUF values back into the original PBD equation, which you may recall has the following form:

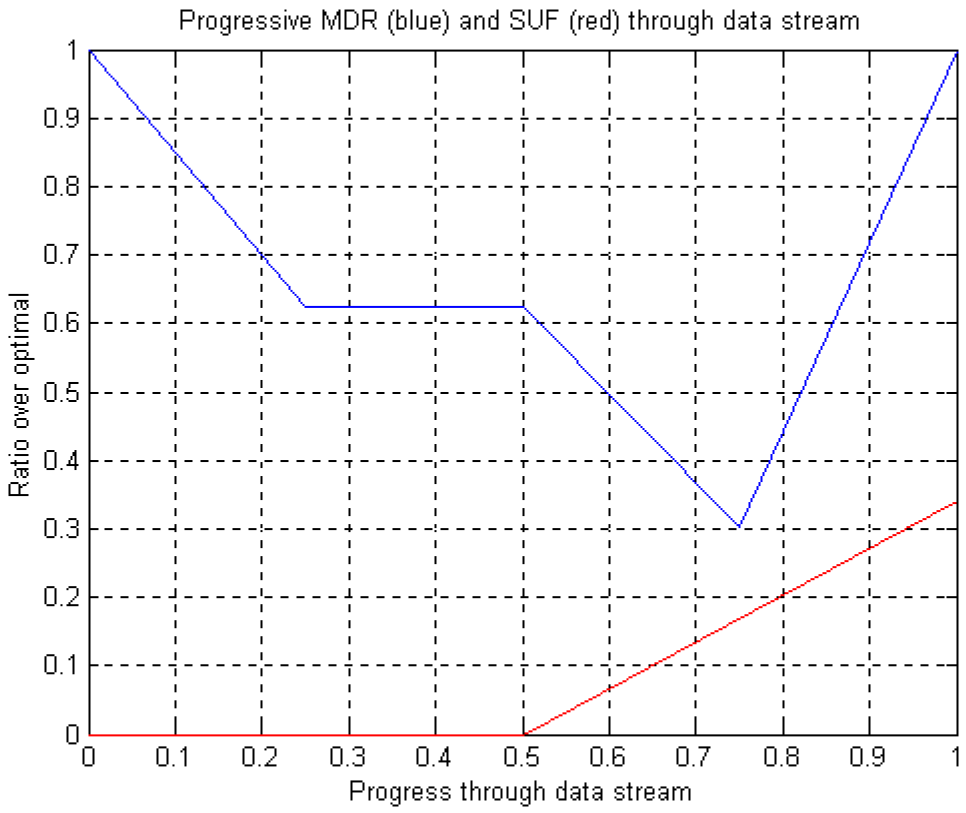
$$P(s) = d - dv + \frac{v \cdot \ln\left(1 + d\left(e^e - 1\right)\right)}{e}$$

Plugging in our values, the PBD of this data sequence is calculated as follows:

$$P(s) = 0.5863 - (0.5863)(0.1463) + \frac{0.1463 \cdot \ln\left(1 + 0.5863\left(e^e - 1\right)\right)}{e}$$

Therefore, the PBD of this data sequence is approximately 61% (0.61 x 100 for final percentage).

A graph of the MDR and SUF data structures for this data sequence resembles the following:



Because the MDR progressively varies considerably and drops below the 0.5 threshold, it can be ascertained that this data sequence is probably not random-like – that is, for the amount of data that occurred. Because the sequence is so small, no definite conclusions can be made, because only a finite number of data elements were allowed. Whenever a small number of data elements exist, no definite conclusions can be made about whether or not the data is random.

So, what exactly does this all mean for this data sequence? The feasible data is 61% evenly distributed in the actual data sequence. But as stated earlier, at present no practical application of this data is known. Through research, surely a useful solution to and explanation of certain probability problems can be found through the use of PBD. PBD is a brand new concept, and it will take some time for it to be accepted enough to be cross-examined for practical real-world application.

PBD is sufficiently acceptable for the analysis of binary data. More tests and analyses on the algorithm may yield potentially new and innovative applications of PBD for a variety of different uses and fields. Most of all, intensive experimentation and breakdowns of the PBD algorithm will be needed to completely verify its validity in the proper analysis of the uniform distribution of data. It is hoped that eventually, PBD may become a well-recognized standard in the area of data analysis.

VIII. Progressive Bit Distribution Mathematical Summary

The computation of the progressive bit distribution (PBD) of data sequence s, applying the averaged MDR and SUF data structures (the average of all obtained d_{cd} and v_{cv} values), represented by d and v respectively, understanding there to be more than 1 feasible element in the data sequence, is derived as follows:

$$P(s) = d - dv + \frac{v \cdot \ln\left(1 + d\left(e^e - 1\right)\right)}{e}$$

Prior to averaging the MDR and SUF data structures, for each cycle in the data sequence as it is computed, a significance value is obtained representing the length of the current cycle c divided by the default length of the cycle s (c / s). The number of cycles n is incremented by the significance value, and the current cycle value x (MDR or SUF) is multiplied by the significance value before being added onto the accumulated data structure z (MDR or SUF), as follows:

$$n = n + \frac{c}{s} \qquad\qquad z = z + \frac{xc}{s}$$

The maximum size of the MDR cycle s_d and the maximum size of the SUF cycle s_v are expressed in terms of the total number of feasible elements m within the data sequence, with l representing the total size of the data sequence, as follows:

$$s_d = m \qquad\qquad s_v = \left\{\left[l > 2m \therefore 2m\right] \quad \vee \quad \left[l \le 2m \therefore m\right]\right\}$$

In MDR calculation, given a cycle c_d of length s_d, and with x_n denoting the element occurrence rate for each element in the cycle, understanding x_n to be greater than zero, the mean distribution rate (MDR) of the data within this cycle can be expressed as

$$d_{c_d} = \frac{\left|\ln\left(\frac{x_1}{s_d}\right)\right| + \left|\ln\left(\frac{x_2}{s_d}\right)\right| + \left|\ln\left(\frac{x_3}{s_d}\right)\right| + \ldots + \left|\ln\left(\frac{x_n}{s_d}\right)\right|}{s_d \cdot \left|\ln\left(s_d\right)\right|}$$ (If d_{cd} is undefined [division by 0], it is defined as 1)

In SUF calculation, if the current cycle c_v is at least the second cycle in the data sequence (the first cycle's SUF is undefined or 0), then given two cycles (the previous and the current), of lengths s_{v1} and s_{v2} respectively, and with element occurrence rates a_1 and a_2 respectively, with m as the total number of feasible elements within both cycles, and deriving each x_n for each element a_n with each cycle size s_{vn} (where n denotes the subtype 1 or 2) as

$$x_n = \frac{ma_n}{s_{v_n}} + 1$$

then the spectral utilization factor (SUF) of the data within cycle c_v as compared to the previous cycle (c_v - 1) can be expressed as

$$v_{c_v} = \frac{y_1 + y_2 + y_3 + \ldots + y_n}{z_1 + z_2 + z_3 + \ldots + z_n}$$ (If v_{cv} is undefined [division by 0], it is defined as 0)

where each y value is computed as

$$y_n = \left| \ln\left(\frac{x_2}{x_1}\right) \right|$$

and each z value is computed as

$$z_n = \left\{ \left[x_1 > x_2 \therefore \left|\ln(x_1)\right| \right] \quad \vee \quad \left[x_1 \le x_2 \therefore \left|\ln(x_2)\right| \right] \right\}$$

for as many elements as exist in the current cycle.

IX. Conclusion

Progressive bit distribution represents a new concept in data analysis. Through its examination of the progressive tendencies of the contents of a data sequence, it can clearly demonstrate much of the nature of that sequence. PBD can be used to measure relative randomness, information density, or simply the context of the distribution of feasible data in the sequence. Graphical analysis of the progressive MDR and SUF data structures can more visually identify key clues about the nature of the data sequence in question, at various selected points throughout its structure.

PBD excels in the analysis of data files on a digital computer system. Even in its early incarnations, PBD has proven very effective in the raw analysis of encrypted data. Marathon Computer Press (www.marcompress.com) currently offers PBD as part of its Encryption Torture Test (ETT) ciphertext-analysis suite. The company was actually very helpful in the creation of the PBD algorithm; through their analytical assistance, PBD has progressed to a point never before anticipated by its creator. Now, PBD is on the brink of becoming a well-accepted new means of information density analysis. Soon, PBD may even find use in the real world, outside of computer files.

It is hoped that you have enjoyed this discussion of Roy Mayhall II's PBD data analysis algorithm. Every effort has been made to make this document as insightful and informative as possible. Please feel free to comment on this document or PBD itself. We even welcome you to lend a hand in its evolution as a new standard in data analysis. PBD needs extensive evaluation, and comments and suggestions are mandatory if the algorithm is to progress. Please see the contact information below and get in touch! We welcome your input!

X. Acknowledgements and Contact Information

Roy Mayhall II designed and wrote the PBD algorithm in March – April 2003. Revision 21, the latest revision, was completed on May 10, 2003. Roy Mayhall II initially created this document on March 27, 2003, and updated it May 10, 2003, to include the new logic of the PBD 21 algorithm.

Roy Mayhall II would like to extend his gratitude to John B. Holder, vice president of the development division at Marathon Computer Press (http://www.marcompress.com), for his insightful assistance in the creation of PBD and for including the PBD product in the company's ETT package. Without his assistance, PBD would have had no chance of becoming what it is today.

The simplification of the PBD equations and the PBD logarithmic curve graph were made possible through the Mathematica 3.0 for Windows software application. Mathematica 3.0 is copyrighted © 1988-1996 by Wolfram Research, Inc. All rights are reserved. Visit the Mathematica homepage at http://www.wolfram.com.

The various graphs of the MDR and SUF data structures used herein were created using a macro function in MATLAB 6.1. MATLAB 6.1 is copyrighted © 1984-2001 by The MathWorks, Inc. All rights are reserved. Visit the MATLAB homepage at http://www.mathworks.com.

Roy Mayhall II can be contacted via E-mail at platinumd@netzero.net.
Visit the PBD web site at http://kickme.to/pbd for the latest PBD release, including free source code downloads.

Please also check out Marathon Computer Press' ETT 7.0 package at
http://www.marcompress.com/AboutETT7.htm.

Remember to send your questions, comments, suggestions and the like to Roy Mayhall II via E-mail. Also remember to contact Roy Mayhall II if you would like to assist in the evaluation or development of PBD.