# Estimating effective DNA database size via compression *

Martina Višňovská[1], Michal Nánási[2], Tomáš Vinař[1], and Broňa Brejová[2]

[1] Department of Applied Informatics, Faculty of Mathematics, Physics, and Informatics
Comenius University, Mlynská Dolina, 842 48 Bratislava, Slovakia
[2] Department of Computer Science, Faculty of Mathematics, Physics, and Informatics
Comenius University, Mlynská Dolina, 842 48 Bratislava, Slovakia

**Abstract.** *Search for sequence similarity in large-scale databases of DNA and protein sequences is one of the essential problems in bioinformatics. To distinguish random matches from biologically relevant similarities, it is customary to compute statistical P-value of each discovered match. In this context, P-value is the probability that a similarity with a given score or higher would appear by chance in a comparison of a random query and a random database. Note that P-value is a function of the database size, since a high-scoring similarity is more likely to exist by chance in a larger database.*
*Biological databases often contain redundant, identical, or very similar sequences. This fact is not taken into account in P-value estimation, resulting in pessimistic estimates. One way to address this problem is to use a lower effective database size instead of its real size. In this work, we propose to estimate the effective size of a database by its compressed size. An appropriate compression algorithm will effectively store only a single copy of each repeated string, resulting in a file whose size roughly corresponds to the amount of unique sequence in the database. We evaluate our approach on real and simulated databases.*

## 1 Introduction

Recent progress in genome sequencing technologies led to rapid increase in the size of DNA sequence databases. Perhaps the most common way of accessing these databases is through *sequence homology search*, where users search in the *database* for sequences similar to their *query* of interest [1]. Highly similar sequences have often evolved from a common ancestor and may also share the same function. Homology search is thus the first step in elucidating the function and evolutionary history of a newly sequenced genome.

To formally define sequence homology search as a computational problem, we consider a query $Q$ and a database $D$, which are both strings composed of nucleotides (symbols from the alphabet $\{A, C, G, T\}$). We typically introduce a scoring function that assigns a positive score to matching nucleotides in the two sequences, and negative score to mismatches. Insertion and deletion of nucleotides also invokes penalty. The goal is then to find a region in the query and a region of the sequence database with the highest possible score. This task can be accomplished either by a dynamic programming algorithm [21], or by fast heuristic methods, such as BLAST [1]. Regardless of the method, the result is a set of high scoring pairs (substring of a query matched to a substring of a database) together with their similarity scores.

Typical genomic databases are large: human genome alone has approx. 3.2 GB, and the GenBank traditional database [4] contains more than 100 GB of DNA sequences as of April 2010. In these large databases, many query sequences will have high scoring matches purely by chance. To distinguish real matches from spurious ones, we need to assess their *statistical significance*.

Traditionally, the statistical significance of a high scoring pair is assessed by *P-value*. If the high scoring pair has score $s$, its $P$-value is the probability of a match with score $s$ or better occurring in homology search of a randomly generated query in a randomly generated database. The $P$-value obviously depends on the sizes of the query and the database, but to achieve more realistic $P$-values, we can also take into account other properties of the sequences, such as frequencies of individual nucleotides [17].

Consider an illustration in Fig.1. For a given score $s$, we can visualize the sequence database $D$ as a set of points in the sequence space with the neighbourhoods that include all sequences with similarity score $s$ or higher. In this sequence space, the query $Q$ is located at the boundary of one of these neighbourhoods. If the neighbourhoods cover a large fraction of the sequence space, the P-value is high, because a randomly generated query will have a high probability of falling into one of those neighbourhoods, resulting in score larger than $s$.

While for a given database the $P$-values could be computed exactly, such a computation would be impractical. Instead, one uses various approximations assuming that the sequence in the database is randomly generated by an i.i.d. process or a Markov chain (e.g., [9]). These assumptions are often violated by real databases.

* This research is funded by European Community FP7 grants IRG-224885 and IRG-231025, VEGA grant 1/0210/10, and Comenius University grant number UK/151/2010.

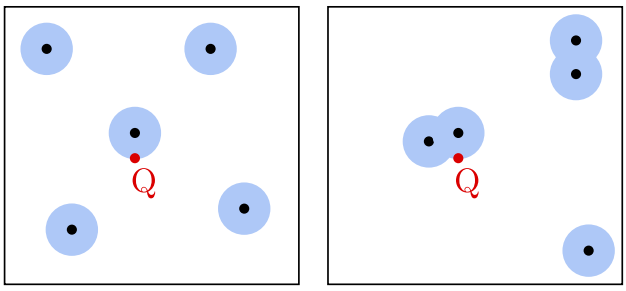


**Fig. 1.** Illustration of $P$-value computation. The points represent sequences in database $D$ and the shaded areas show their neighbourhoods with scores that are at least as good as the similarity score of query $Q$. Left: Database with randomly distributed sequences. Right: Database with clustered sequences.

For instance, the database may contain several sequences that are evolutionarily related. Such sequences will often contain only a small number of changes: for example, DNA sequences of human and chimpanzee are identical in 99% of nucleotides over most of their lengths. In addition, recently introduced next generation sequencing technologies made it financially viable to sequence multiple individuals from the same species, leading to various personal human genome projects [23]. Sequences of different individuals from the same species may differ as little as one in thousand nucleotides. Consequently, databases may contain many highly similar sequences, and it is not appropriate to model them as completely random.

Consider again the example in Fig.1. The illustration on the right shows a database with clusters of similar sequences that have overlapping neighbourhoods, covering a much smaller fraction of the sequence space than the database on the left. Consequently, the estimate based on the assumption of random distribution of sequences in the database will necessarily overestimate the $P$-value, potentially leading to rejection of high scoring pairs as matches likely to occur by chance.

One of the solutions to this problem proposed in the literature is to remove the redundancies, such as closely related sequences, from the sequence database [22]. New query can be first searched against such non-redundant database, the statistical significance of resulting matches can be evaluated, and then a secondary search can be launched against the whole database. In this paper, we propose a different solution. One of the main parameters used in $P$-value computation is the database size $n$. Instead of the real database size, we propose to use an *effective database size* $n'$ ($n' < n$) which will account for redundancies in the database. For example, if we take a random database of size $n$, and double its contents by including second exact copy of each sequence, the effective size of such new database will still be $n' = n$, even though its real size is $2n$.

Note that the notion of effective database size has been previously used to adjust for border effects in case of short queries and databases [17], and an option for setting it to an arbitrary value is included in most software tools for homology search. Analogously, statistical models in population genetics use population size as a parameter, but instead of the actual number of individuals, one typically uses effective population size to compensate for various effects that are not considered by the model, such as population size changing over time [11].

## 2 Kolmogorov complexity of DNA sequences

For our purposes, the effective database size should be an estimate of the amount of unique sequence in the database $D$, taking into account substrings that may be present in $D$ in many exact or approximate copies. One way of describing the information content of a database is its *Kolmogorov complexity* [16].

Kolmogorov complexity $K(D)$ of a sequence $D$ is the bit length of the shortest program $P$ for a fixed universal Turing machine that outputs sequence $D$. Kolmogorov complexity can be understood as a lower bound (up to a constant additive term) of compression achievable by any general-purpose algorithm. A string of length $n$ sampled uniformly at random from a fixed alphabet is on average almost incompressible [16, Section 2.8.1]. In particular, a string over a four-letter alphabet requires on average approximately $2n$ bits (with up to an $O(\log n)$ additive term). Thus if we believe that the databases with the same Kolmogorov complexity will behave similarly, we should use $n' = K(D)/2$ as an estimate of the effective database size of database $D$.

At a first glance, Kolmogorov complexity seems to be an ideal estimator to use in this context. It accounts for possible major differences between the real database and a randomly generated one. In particular, using Kolmogorov complexity would compensate for differences in frequencies of individual nucleotides (database containing only a long stretch of As will have a small effective size), and redundant sequence content (sequences that have only few differences can be described very efficiently in a small space). Moreover, the concept of Kolmogorov complexity has been successfully used in similar contexts before, in applications such as computing distance between genomes [15] (see also [10, 18] for overview).

The exact Kolmogorov complexity of database $D$ is not computable. Yet in practice, we can use various compression algorithms instead of computing the

Kolmogorov complexity. For a fixed compression algorithm, the compressed size $c(D)$ of database $D$ is an upper bound on the Kolmogorov complexity $K(D)$, and we can use value $n' = c(D)/2$ as an estimate of the effective database size. Several efficient algorithms specifically tuned to compression of DNA sequences are available (see [6, 7, 14, 3]).

As we will see in the next section, the upper bounds on $P$-values (or *conservative estimates*) are generally desirable in cases where exact $P$-values cannot be computed. Since the $P$-values increase monotonically with the database size, using an upper bound on the effective database size should not by itself lead to non-conservative bounds.

Unfortunately, using Kolmogorov complexity may not necessarily lead to conservative bounds in all instances. As an extreme case, base-4 expansion of many fundamental constants, such as $\pi$, can be generated by a constant-size program. First $n$ bits generated by such a program can be used as a sequence database of size $n$, replacing digits $0, \ldots, 3$ with nucleotides. Kolmogorov complexity of such database is $O(\log n)$ (we need a constant number of bits to represent the program, and $\log n$ bits to represent the real size of the database), yet for all practical purposes, this database behaves as a random database of size $n$ [2].

Thus using a Kolmogorov complexity and compression-based estimates of effective database size does not necessarily lead to conservative estimates of $P$-values in homology search. In the next section, we explore practical issues of using these compression-based estimates in an experimental setting.

## 3 Estimating effective database size through compression

Here, motivated by the discussion in the previous section, we explore the use of compression software for estimating the effective database size. The methodology is very simple. First, we compress the database and measure the size of the resulting file in bytes. Then, we multiply this size by four to account for the fact that in a uniformly random database, we need two bits to encode each nucleotide. In this way, we obtain an estimate of the effective database size which can be used in any formula or algorithm for estimating $P$-values on uniformly distributed databases.

The $P$-values are used to reject high scoring pairs that have a high probability of occurring by chance. In a typical search there will be many spurious matches with high $P$-values, and only a few top-scoring matches will represent genuine similarities with common evolutionary origins. Since our main task is to separate these few good matches from many spurious matches, we would like the $P$-value estimator to be *conservative*, i.e., the estimates should be strictly higher than the exact $P$-values, so that the estimated $P$-value represents an upper bound on the probability of a particular match being a false positive.

We have decided to experimentally evaluate the accuracy of the $P$-values obtained by the compression method in a simple scenario motivated by the next generation sequencing technologies. In this scenario, a sequencing machine generates many short sequences (*reads*). These reads need to be mapped as substrings to previously known reference genomes. In most cases, the reads will match the reference sequence exactly, but sometimes we will see one or two mismatches. These mismatches can be either due to sequencing errors, or (more interestingly) due to differences between individuals.

In our simplified scenario, the database $D$ is a single string of length $n$, the query $Q$ is a string of length $m$, and we are searching in $D$ for a substring of length $m$ with the smallest Hamming distance from $Q$. In contrast to the full homology search problem, we always consider the whole query (each read has to map completely to the reference), and we also disallow insertions and deletions.

We will say that the distance of query $Q$ from database $D$ is the minimum Hamming distance between $Q$ and some substring of length $m$ from database $D$. This Hamming distance will represent our score. $P$-value for a particular Hamming distance $h$ is then the number of all $m$-tuples that are at a distance of at most $h$ from $D$, divided by $4^m$ (the number of all possible $m$-tuples). Note that in this definition of $P$-value, we keep the database fixed, and only the query is random, chosen uniformly from all possible queries of length $m$. In contrast, most methods consider both query and database as random.

The main advantage of this simplified model is that we can compute these $P$-values exactly in a reasonable amount of time, and compare them to the $P$-values estimated based on the randomly generated database of corresponding effective size, thus evaluating the accuracy of our concept. The rest of this section is organized as follows. First, we introduce the algorithm for exact computation of $P$-values in our simplified scenario. Then we explore several cases of generated and real databases.

The file compression algorithms typically detect symbols or small groups of symbols that occur in the text more frequently than the others, and encode them by shorter codewords. Thus, the size of the compressed database is related to the entropy of the source generating the database. In addition to that, some methods try to detect longer strings that are repeated exactly or with mismatches multiple times, and to store only

one copy of each such string as well as differences between the approximate copies.

In our experiments, we first try to separate these two phenomena by using artificially generated databases, and in the end, we apply compression software to real DNA sequences, where both of these issues are at play.

### 3.1 Algorithm to compute exact *P*-values

We have implemented an algorithm that simultaneously computes $P$-values for a given database $D$ and all of its prefixes, and for all distances $h = 0, 1, \ldots, h_{\max}$. In experiments we use values $h_{\max} = 3$ and $m = 15$.

The algorithm proceeds along the database $D$ and at each position updates two arrays $M$ and $H$. Array $M$ of size $4^m$ stores for each $m$-tuple $Q'$ its distance to the current prefix of the database, provided that this distance is at most $h_{\max}$ (otherwise it uses a special $\infty$ value). The second array $H$ stores for each distance $h \leq h_{\max}$ the number of $m$-tuples with distance exactly $h$ from the current prefix of the database $D$.

When we read a new nucleotide of the database, we need to consider the last $m$-tuple of the current prefix. We enumerate all $m$-tuples at a distance of at most $h_{\max}$ from this new $m$-tuple, and for each we update arrays $M$ and $H$ as appropriate. Values in $H$ can be easily converted to the desired $P$-values for the current prefix at different distance thresholds. Note that this algorithm is feasible only for small values of $m$, because it requires $\Theta(4^m)$ memory.

The algorithm can be used to compute exact $P$-value for a given real sequence database, which we consider as a reference value. It can also be applied to randomly generated databases, leading to estimates of $P$-values that would be obtained in a model, where both the database and the query are random. To emulate the proposed method, we compress a sequence database, compute its effective size, and then use the $P$-value estimates obtained from random databases of the matching size.

Finally, we also consider a simple method, where we use random databases of the same size as the original database, without compression. $P$-values computed in this way correspond to the commonly used techniques for $P$-value estimation without using the effective database size mechanism.

### 3.2 Entropy

The four nucleotides do not occur in genomes equally frequently. A commonly used measure of DNA composition is GC-content: the percentage of Cs and Gs in the given sequence. GC-content varies widely in different genomes, or even between segments of the same genome. An i.i.d. database with GC-content $g$ has entropy $H(g) = -g \lg(g/2) - (1-g) \lg((1-g)/2)$, and therefore can be encoded by approximately $H(g)n$ bits, for example by the arithmetic encoding [20].

Following our general approach, we can try to use the formulas for the uniform nucleotide frequencies and effective database size $E(n, g) = H(g)n/2$ to estimate the $P$-values for the actual database of size $n$ and GC-content $g$.

We have tested the performance of such estimates on randomly generated databases with GC-contents 75% and 90%, averaging values for five randomly generated databases. Table 1 shows the real $P$-values, $P$-values predicted by our compression method, and $P$-values obtained by the simple method of considering a database of length $n$ and GC-content 50% (disregarding the real GC-content in the $P$-value estimate). All estimates were computed by our algorithm from Section 3.1. For real P-values the algorithm was applied to the generated database with a skewed GC-content, for simple and predicted estimates to five random databases of appropriate size with GC-content 50%.

For small $P$-values the real and simple estimates are quite similar, which is expected since in a short database very few $m$-tuples occur multiple times, even if the composition of the database is skewed. As a result, the compression method gives non-conservative estimates, because it uses a much smaller database size. For larger $P$-values, the compression estimates become conservative, and are closer to the true $P$-value than the simple estimates. This is because a database with high GC-content is less likely to contain $m$-tuples with low GC-content, and thus a larger database size is required to achieve the same $P$-value. For example, among estimates for GC-content 75% shown in Table 1, compression estimates become conservative for databases larger than $10^7$ and $10^5$ nucleotides for $h = 0$ and $h = 2$, respectively.

We can study the situation analytically for the case when the query appears in the database without a mismatch. Let $X_i$ be the event that query $Q$ has an occurrence with distance at most $h$ at position $i$ in the random database of GC-content $g$, and let $X$ be the total number of occurrences of query $Q$ with at most $h$ mismatches in the whole database. For $g = 0.5$, we can compute the probability of $X_i$ by summing over the number of mismatches

$$P(X_i|Q, g = 0.5) = \sum_{k=0}^{h} \binom{m}{k} \left(\frac{3}{4}\right)^k \left(\frac{1}{4}\right)^{m-k}.$$

For general $g$ we have to distinguish between mismatches on G/C positions and mismatches on A/T

| | h=0 | | | | h=2 | | | |
|---|---|---|---|---|---|---|---|---|
| Database size | $10^4$ | $10^5$ | $10^6$ | $10^7$ | $10^4$ | $10^5$ | $10^6$ | $10^7$ |
| Real (GC 75%) | $9.3 \cdot 10^{-6}$ | $9.3 \cdot 10^{-5}$ | $9.2 \cdot 10^{-4}$ | $8.4 \cdot 10^{-3}$ | $8.8 \cdot 10^{-3}$ | $7.0 \cdot 10^{-2}$ | $3.2 \cdot 10^{-1}$ | $7.2 \cdot 10^{-1}$ |
| Predicted (GC 75%) | $8.4 \cdot 10^{-6}$ | $8.5 \cdot 10^{-5}$ | $8.4 \cdot 10^{-4}$ | $8.4 \cdot 10^{-3}$ | $8.3 \cdot 10^{-3}$ | $8.1 \cdot 10^{-2}$ | $5.7 \cdot 10^{-1}$ | 1.0 |
| Simple (GC 75%) | $9.3 \cdot 10^{-6}$ | $9.3 \cdot 10^{-5}$ | $9.3 \cdot 10^{-4}$ | $9.3 \cdot 10^{-3}$ | $9.2 \cdot 10^{-3}$ | $8.8 \cdot 10^{-2}$ | $6.0 \cdot 10^{-1}$ | 1.0 |
| Real (GC 90%) | $9.2 \cdot 10^{-6}$ | $8.7 \cdot 10^{-5}$ | $6.7 \cdot 10^{-4}$ | $3.9 \cdot 10^{-3}$ | $6.5 \cdot 10^{-3}$ | $3.3 \cdot 10^{-2}$ | $1.1 \cdot 10^{-1}$ | $2.6 \cdot 10^{-1}$ |
| Predicted (GC 90%) | $6.5 \cdot 10^{-6}$ | $6.8 \cdot 10^{-5}$ | $6.8 \cdot 10^{-4}$ | $6.8 \cdot 10^{-3}$ | $6.4 \cdot 10^{-3}$ | $6.5 \cdot 10^{-2}$ | $4.9 \cdot 10^{-1}$ | 1.0 |
| Simple (GC 90%) | $9.3 \cdot 10^{-6}$ | $9.3 \cdot 10^{-5}$ | $9.3 \cdot 10^{-4}$ | $9.3 \cdot 10^{-3}$ | $9.2 \cdot 10^{-3}$ | $8.8 \cdot 10^{-2}$ | $6.0 \cdot 10^{-1}$ | 1.0 |

**Table 1.** $P$-values for random databases of various lengths $n$, GC-content 75% or 90% and the query distance $h = 0$ or $h = 2$. Real P-values are computed by the algorithm described in Section 3.1. Predicted P-values take into account the entropy of the database, using effective database size $H(g)n/2$. The simple method for computing P-values disregards the GC-content and considers a random database of size $n$ and 50% GC-content.

positions in the query. Let $z$ be the number of Gs and Cs in $Q$, then

$$P(X_i|Q,g) = \sum_{i=0}^{h}\sum_{j=0}^{h-i}\binom{z}{i}\left(1-\frac{g}{2}\right)^i\left(\frac{g}{2}\right)^{z-i} \cdot\binom{m-z}{j}\left(\frac{1+g}{2}\right)^j\left(\frac{1-g}{2}\right)^{m-z-j}.$$

The expected number of occurrences can be computed by linearity of expectation (for simplicity, we ignore the edge effect at positions $n-m+2,\ldots,n$, which is negligible for large $n$):

$$E(X|Q,g) = \sum_{i=1}^{n} E(X_i|Q) = nP(X_i|Q,g).$$

We will approximate the distribution of variable $X$ by a Poisson distribution with the mean $\lambda = E(X|Q,g)$. This commonly used approximation [17] disregards dependencies between occurrences at adjacent positions and also assumes that $n$ is large and $\lambda$ relatively small. In our simulations it led to very good estimates of $P$-values (data not shown). If $X$ is from Poisson distribution, the probability of at least one occurrence of a given query $Q$ is $P(X>0|Q,g) = 1-e^{-\lambda}$. To obtain the final $P$-value, we have to consider this expression for different queries, or more precisely, for groups of queries with the same number Gs and Cs, of which $Q_z$ is one representative:

$$P_{\text{real}} = P(X>0|g) = \sum_{z=0}^{m}\binom{m}{z}2^{-m}P(X>0|Q_z,g).$$

The compression estimate $P_{\text{est}}$ uses the same formula but $g = 0.5$ and $E(n,g)$ instead of $n$. For $h = 0$, $E(X|Q,g)$ simplifies to $n2^{-m}g^z(1-g)^{m-z}$, and in particular $E(X|Q,0.5) = n4^{-m}$ does not depend on $z$, and therefore $P_{\text{est}} = 1 - e^{-E(n,g)4^{-m}}$. For small $x$, function $1-e^{-x}$ can be well approximated by $x$ [8]. Using this approximation, we obtain

$$\frac{P_{\text{est}}}{P_{\text{real}}} = \frac{E(n,g)4^{-m}}{\sum_{z=0}^{m}\binom{m}{z}2^{-m}n2^{-m}g^z(1-g)^{m-z}} = H(g)/2.$$

Therefore for small $P$-values, where the approximation of $1-e^{-x}$ is sufficiently accurate, the estimate $P_{\text{est}}$ is lower by approximately a factor of $H(g)/2$ than the real P-value. This implies that no correction for entropy is in fact necessary for small $P$-values.

On the other hand, when $h = 0$ but $n$ is sufficiently large, the compression estimates become conservative. In particular, let us assume that

$$n > \frac{m2^{-m}\ln(2)}{H(g)2^{-m-1}-(1-g)^m} = \frac{1.386}{H(g)}m4^m + o(m4^m).$$

This implies $e^{-E(n,g)4^{-m}} < 2^{-m}e^{-n2^{-m}(1-g)^m}$. The right-hand side is one of the terms of the sum

$$\sum_{z=0}^{m}\binom{m}{z}2^{-m}e^{-n2^{-m}g^z(1-g)^{m-z}},$$

and therefore the left-hand side is upper-bounded by the whole sum as well. This implies $P_{\text{est}} \geq P_{\text{real}}$.

This simple bound is not very interesting, since it works only for very large $n$, where P-values are very close to one. Nonetheless, it agrees with our observation that the compression estimate is appropriate for sufficiently large $n$. Perhaps a tighter bound on $n$ can be obtained by considering additional elements of the sum.

### 3.3 Redundancy

Next, we consider artificial databases that are concatenation of many mutually similar sequences of the same length $k$. In the experiments we use $k = 10^4$. To generate the database, we first sample a string $S = s_1 s_2 \ldots s_k$ of length $k$ uniformly at random. This string will be the center of the cluster of similar sequences.

The $i$-th sequence in the concatenated database is obtained from $S$ by randomly mutating several nucleotides of $S$ so that the nucleotide $j$ is the same

| | h=0 | | | | h=2 | | | |
|---|---|---|---|---|---|---|---|---|
| Database size | $10^4$ | $10^5$ | $10^6$ | $10^7$ | $10^4$ | $10^5$ | $10^6$ | $10^7$ |
| Real | $9.3 \cdot 10^{-6}$ | $8.1 \cdot 10^{-5}$ | $6.5 \cdot 10^{-4}$ | $4.3 \cdot 10^{-3}$ | $9.2 \cdot 10^{-3}$ | $6.8 \cdot 10^{-2}$ | $3.4 \cdot 10^{-1}$ | $8.8 \cdot 10^{-1}$ |
| GenCompress | $9.3 \cdot 10^{-6}$ | $7.7 \cdot 10^{-5}$ | $6.8 \cdot 10^{-4}$ | $6.5 \cdot 10^{-3}$ | $9.2 \cdot 10^{-3}$ | $7.3 \cdot 10^{-2}$ | $4.9 \cdot 10^{-1}$ | 1.0 |
| bzip2 | $1.0 \cdot 10^{-5}$ | $9.2 \cdot 10^{-5}$ | $8.1 \cdot 10^{-4}$ | $8.0 \cdot 10^{-3}$ | $1.0 \cdot 10^{-2}$ | $8.7 \cdot 10^{-2}$ | $5.5 \cdot 10^{-1}$ | 1.0 |
| Simple | $9.3 \cdot 10^{-6}$ | $9.3 \cdot 10^{-5}$ | $9.3 \cdot 10^{-4}$ | $9.3 \cdot 10^{-3}$ | $9.2 \cdot 10^{-3}$ | $8.8 \cdot 10^{-2}$ | $6.0 \cdot 10^{-1}$ | 1.0 |

**Table 2.** $P$-values for the artificial clustered database. Real P-values were computed by the algorithm from Section 3.1 applied directly to the clustered database. GenCompress and bzip2 estimates use different compression tools to compute effective database size. The simple method for computing P-values considers a uniformly generated random database of size $n$.

| | h=0 | | | | h=2 | | | |
|---|---|---|---|---|---|---|---|---|
| Database size | $1.6 \cdot 10^4$ | $2.1 \cdot 10^5$ | $1.5 \cdot 10^6$ | $1.1 \cdot 10^7$ | $1.6 \cdot 10^4$ | $2.1 \cdot 10^5$ | $1.5 \cdot 10^6$ | $1.1 \cdot 10^7$ |
| Real | $1.1 \cdot 10^{-5}$ | $1.4 \cdot 10^{-4}$ | $9.4 \cdot 10^{-4}$ | $6.0 \cdot 10^{-3}$ | $1.0 \cdot 10^{-2}$ | $1.1 \cdot 10^{-1}$ | $4.6 \cdot 10^{-1}$ | $9.1 \cdot 10^{-1}$ |
| GenCompress | $9.3 \cdot 10^{-6}$ | $1.2 \cdot 10^{-4}$ | $8.5 \cdot 10^{-4}$ | $6.0 \cdot 10^{-3}$ | $9.2 \cdot 10^{-3}$ | $1.1 \cdot 10^{-1}$ | $5.7 \cdot 10^{-1}$ | 1.0 |
| GC corrected | $1.1 \cdot 10^{-5}$ | $1.4 \cdot 10^{-4}$ | $9.6 \cdot 10^{-4}$ | $6.8 \cdot 10^{-3}$ | $1.1 \cdot 10^{-2}$ | $1.3 \cdot 10^{-1}$ | $6.1 \cdot 10^{-1}$ | 1.0 |
| bzip2 | $1.5 \cdot 10^{-5}$ | $1.9 \cdot 10^{-4}$ | $1.3 \cdot 10^{-3}$ | $9.4 \cdot 10^{-3}$ | $1.5 \cdot 10^{-2}$ | $1.7 \cdot 10^{-1}$ | $7.2 \cdot 10^{-1}$ | 1.0 |
| Simple | $1.5 \cdot 10^{-5}$ | $1.9 \cdot 10^{-4}$ | $1.4 \cdot 10^{-3}$ | $1.1 \cdot 10^{-2}$ | $1.5 \cdot 10^{-2}$ | $1.8 \cdot 10^{-1}$ | $7.4 \cdot 10^{-1}$ | 1.0 |

**Table 3.** $P$-values for genomic data from human, chimpanzee, and rhesus. Real P-values were computed by the algorithm from Section 3.1 applied directly to the genomic sequences. GenCompress and bzip2 estimates use different compression tools to compute effective database size. GC corrected shows the GenCompress estimate corrected for the average database entropy. The simple method for computing P-values considers a uniformly generated random database of size $n$.

as $s_j$ with the probability of 90%, and with the probability of 10% it changes to another nucleotide chosen randomly from the remaining three. This way, we get a *clustered database* of sequences that differ from the center of the cluster on 10% positions on average.

For clustered databases of various sizes, we compute the real $P$-value simple estimate, which uses effective size equal to the real size of the database without compression, and two estimates based on two different lossless compression programs GenCompress [6] and bzip2. The results are shown in Table 2.

Bzip2 is based on Burrows–Wheeler transform [5] which tends to create blocks of identical symbols if the input contains repeated substrings. The transformed text is then encoded by other compression techniques, such as Huffman encoding. To save memory, bzip2 divides a file into blocks and processes each block separately, which may have negative effect on the compressed size.

GenCompress [6] is an algorithm developed specifically for compressing DNA sequences. It finds approximate repeats in the compressed sequence and encodes them by a sequence of edit operations. GenCompress is a single-pass algorithm that proceed along the input sequence and in each step it finds the best prefix that can be encoded as an approximate repeat of some substring of already encoded input sequence.

In this experiment, estimates based on data compression are mostly conservative and better than the estimates obtained by the simple methods.

### 3.4 Real data

Finally, we have applied the compression method of estimating the effective database size to real genomic data from human, chimpanzee, and rhesus macaque. Our set consisted of a portion of human chromosome 22 and corresponding portions of genome from the two other primates. We have omitted larger blocks that did not have counterparts in neither chimpanzee, nor macaque. The sequence data and the genome alignments were obtained from the UCSC genome browser [13].

The database contains a short block of the human sequence followed by the corresponding block in the two other genomes, followed by another block in human, etc. Therefore, similar sequences are close to each other which improves compression with algorithms such as bzip2 that always consider only a block of the whole file.

The results of the test for different input sizes are shown in Table 3. As we can see, with GenCompress we often underestimate real $P$-values, while bzip2 achieves only a very small improvement compared to the simple method without compression.

Earlier, we have reached a conclusion that skewed GC-content should not automatically imply lower effective database sizes since this would lead to underestimation of small $P$-values. However, compression algorithms estimate the effective database size based on both sequence redundancy and lower sequence entropy in case of locally high or low GC-contents. That could be the reason why GenCompress estimates are non-conservative.

We have attempted to further correct for this issue by computing an average database entropy $H'$. The effective database size estimated by GenCompress was then multiplied by the correction factor of $2/H'$. This approach leads to surprisingly good $P$-value estimates that were also conservative in our experiments (Table 3, line GC corrected).

To estimate the value $H'$ for this experiment, we have computed entropy separately for non-overlapping windows of size 1000 to capture different properties of individual genomic regions. We have estimated a Markov chain of second order from each window of the sequence and computed an entropy of this Markov chain, that is, entropy where the probability of each nucleotide is conditioned on the two previous nucleotides to capture local dependencies in DNA sequences. The average entropy $H'$ of the whole database was then computed as an average of entropy values from all windows.

## 4 Conclusion

In this paper, we have considered methods for more accurate estimation of $P$-values in the context of sequence homology search. In particular, we propose to adjust the size of the database to compensate for the structure present in the database due to the fact that individual sequences are related by evolution.

We have explored the idea of using compression to estimate the effective database size, and we have demonstrated by experiments that the use of the compression algorithms leads to non-conservative $P$-value estimates for small $P$-values. This is at least partially caused by the fact that besides identifying longer repeated substrings, compression algorithms also compensate for sequences with low entropy. We have shown that such compensation should not be considered, at least in the case of small $P$-values. We have suggested a simple way to disentangle the portion of the compression coming from locally low entropy and shown that the correction leads to better $P$-value estimates.

The compression would be a fast and efficient way of estimating the effective database size. Even though most of the general purpose compression algorithms (such as bzip2) cannot handle distant large blocks of similarity common in DNA sequence databases, one could use fast methods for identifying such blocks [12, 19] to speed up the algorithms developed specifically for compression of DNA sequences [6, 7, 14, 3].

Finally, we would like to extend our work to more complex scenarios of homology search. Longer query sequences will require handling of insertions, deletions, and matches that involve only portions of the query sequence. More complex scoring schemes on both nucleotide and protein sequences also need to be examined.

## References


1. S.F. Altschul, W. Gish, W. Miller, E.W. Myers, and D.J. Lipman: *Basic local alignment search tool.* Journal of Molecular Biology, **215** (3), 1990, 403–410.
2. D. Bailey and R. Crandall: *Random generators and normal numbers.* Experimental Mathematics, **11** (4), 2002, 527–546.
3. B. Behzadi and F. Le Fessant: *DNA compression challenge revisited: a dynamic programming approach.* In: Combinatorial Pattern Matching (CPM 2005), Springer, 2005, 190–200.
4. D.A. Benson, I. Karsch-Mizrachi, D.J. Lipman, J. Ostell, and E.W. Sayers: *GenBank.* Nucleic Acids Research, **37** (Database issue), 2009, D26–31.
5. M. Burrows and D.J. Wheeler: *A block-sorting lossless data compression algorithm.* Technical Report 124, Digital Equipment Corporation, Palo Alto, California, 1994.
6. X. Chen, S. Kwong, and M. Li: *A compression algorithm for DNA sequences and its applications in genome comparison.* In: Genome Informatics (GIW'99), Universal Academy Press, 1999, 51–61.
7. X. Chen, M. Li, B. Ma, and J. Tromp: *DNACompress: fast and effective DNA sequence compression.* Bioinformatics, **18** (12), 2002, 1696–1698.
8. T.H. Cormen, C.E. Leiserson, and R.L. Rivest: *Introduction to algorithms.* MIT Press, 1990.
9. A. Dembo, S. Karlin, and O. Zeitouni: (1994). *Limit distribution of maximal non-aligned two-sequence segmental score.* The Annals of Probability, **22** (4), 1994, 2022–2039.
10. R. Giancarlo, D. Scaturro, and F. Utro: *Textual data compression in computational biology: a synopsis.* Bioinformatics, **25** (13), 2009, 1575–1576.
11. D.L. Hartl and A.G. Clark: *Principles of population genetics*, Fourth Edition. Sinauer Associates, 2006.
12. W.J. Kent, R. Baertsch, A. Hinrichs, W. Miller, and D. Haussler: D. *Evolution's cauldron: duplication, deletion, and rearrangement in the mouse and human genomes.* Proceedings of the National Academy of Sciences of the United States of America, **100** (20), 2003, 11484–9.
13. W.J. Kent, C.W. Sugnet, T.S. Furey, K.M. Roskin, T.H. Pringle, A.M. Zahler, and D. Haussler: *The human genome browser at UCSC.* Genome Research, **12** (6), 2002, 996–1006.

14. G. Korodi and I. Tabus: *An efficient normalized maximum likelihood algorithm for dna sequence compression.* ACM Transactions on Information Systems, **23** (1), 2005, 3–34.
15. M. Li, J.H. Badger, C. Xin, S. Kwong, P. Kearney, and H. Zhang: *An information based sequence distance and its application to whole mitochondrial genome phylogeny.* Bioinformatics, **17** (2), 2001, 149–154.
16. M. Li and P. Vitányi: *An introduction to Kolmogorov complexity and its applications.* Springer, 2008.
17. A.Y. Mitrophanov and M. Borodovsky: (2006). Statistical significance in biological sequence analysis. *Briefings in Bioinformatics*, 7(1), 2006, 2–24.
18. O.U. Nalbantoglu, D.J. Russell, and K. Sayood: Data compression concepts and algorithms and their applications to bioinformatics. *Entropy (Basel, Switzerland)*, **12** (1), 2010, 34.
19. B. Paten, J. Herrero, S. Fitzgerald, K. Beal, P. Flicek, I. Holmes, and E. Birney: *Genome-wide nucleotide-level mammalian ancestor reconstruction.* Genome Research, **18** (11), 2008, 1829–1833.
20. Rissanen, J. and Langdon, G. (1979). Arithmetic coding. *IBM Journal of Research and Development*, 23(2):149–162.
21. T.F. Smith and M.S. Waterman: *Identification of common molecular subsequences.* Journal of Molecular Biology, **147** (1), 1981, 195–197.
22. B.E. Suzek, H. Huang, P. McGarvey, R. Mazumder, and C.H. Wu: *UniRef: comprehensive and non-redundant UniProt reference clusters.* Bioinformatics, **23** (10), 2007, 1282–1288.
23. J.C. Venter: *Multiple personal genomes await.* Nature, **464** (7289), 2010, 676–677.