Searching for repeats, as an example of using the generalised Ruzzo-Tompa algorithm to find optimal subsequences with gaps

John L. Spouge*, Leonardo Mariño-Ramírez and Sergey L. Sheetlin

Computational Biology Branch,
National Center for Biotechnology Information,
Bethesda, MD 20894, USA
Email: john.spouge@nih.gov
Email: leonardo.marino-ramirez@nih.gov
Email: sergey.sheetlin@nih.gov
*Corresponding author

Abstract: Some biological sequences contain subsequences of unusual composition; e.g. some proteins contain DNA binding domains, transmembrane regions and charged regions, and some DNA sequences contain repeats. The linear-time Ruzzo-Tompa (RT) algorithm finds subsequences of unusual composition, using a sequence of scores as input and the corresponding ‘maximal segments’ as output. In principle, permitting gaps in the output subsequences could improve sensitivity. Here, the input of the RT algorithm is generalised to a finite, totally ordered, weighted graph, so the algorithm locates paths of maximal weight through increasing but not necessarily adjacent vertices. By permitting the penalised deletion of unfavourable letters, the generalisation therefore includes gaps. The program RepWords, which finds inexact simple repeats in DNA, exemplifies the general concepts by outperforming a similar extant, ad hoc tool. With minimal programming effort, the generalised Ruzzo-Tompa algorithm could improve the performance of many programs for finding biological subsequences of unusual composition.

Keywords: generalised Ruzzo-Tompa algorithm; subsequences of unusual composition; gaps; repeats.

Reference to this paper should be made as follows: Spouge, J.L., Mariño-Ramírez, L. and Sheetlin, S.L. (2014) ‘Searching for repeats, as an example of using the generalised Ruzzo-Tompa algorithm to find optimal subsequences with gaps’, Int. J. Bioinformatics Research and Applications, Vol. 10, Nos. 4/5, pp.384–408.

Biographical notes: John L. Spouge obtained an honours BSc (Mathematics) and MD from the University of British Columbia, After obtaining a DPhil (Applied Mathematics) in 1983 under John Hammersley at Trinity College, Oxford, he held post-doctoral fellowships in the T-10 Theoretical Biology Group at Los Alamos National Laboratory and the National Cancer Institute. In 1989, he became a founding member and Senior Investigator at the National Center for Biotechnology Information, a position he holds to the present. His main interests are sequence analysis statistics (particularly for pairwise sequence comparison and transcription factor binding site discovery), coalescent theory, DNA barcodes, and virological modelling.

Leonardo Mariño-Ramírez obtained a BSc (Microbiology) in 1992 from Universidad de Los Andes. After obtaining a PhD (Biochemistry and
Biophysics) in 2002 under James C. Hu at Texas A&M University, he held a postdoctoral fellowship at the National Institutes of Health (NIH). In 2004, he joined the National Center for Biotechnology Information (NCBI) as a Staff Scientist. His main interests are the study of molecular and cellular interactions controlling gene expression in eukaryotes and the evolution of eukaryotic gene regulatory elements using computational methods.

Sergey L. Sheetlin obtained his MD (Applied Mathematics) in 1993 and PhD (Physics and Mathematics) in 1997 from M.V. Lomonosov Moscow State University. After graduating he worked at the Institute of Mathematics of the Russian Academy of Sciences, Ufa city in the field of the theory of cubature formulas. Since 2003 to the present he has been working at the National Center for Biotechnology Information, National Institutes of Health, with his main interest in statistical properties of biological sequences.

This paper is a revised and expanded version of a paper entitled ‘The Ruzzo-Tompa algorithm can find the maximal paths in weighted, directed graphs on a one-dimensional lattice’ presented at the ‘2nd IEEE International Conference on Computational Advances in Bio and Medical Sciences (ICCABS)’, Las Vegas, NV, USA, 23–25 February 2012.

1 Background

1.1 Relation to previous extended abstract

This article is based on an extended abstract from ICCABS (Spouge et al., 2012), but implements the general theory in the extended abstract in the context of a specialised problem, namely identifying specific types of biological repeats. It therefore contains substantial new material describing the computer program RepWords and the corresponding empirical results. Any paragraph within quotes is quoting directly from the extended abstract. Appendix A, as a set of technical lemmas, closely paraphrases the extended abstract.

1.2 Repeats in biological sequences

Many eukaryotic genomes contain more repeats than protein-coding genes. For example, repeats occupy more than 50% of the human genome, whereas protein-coding sequences occupy only about 3% (Lander, et al., 2001). Although the terminology of repeats is not standardised, repeats are of two general types: (1) interspersed repeats, derived from transposons, and (2) simple repeats (also known as tandem repeats), which are inexact consecutive (or nearly consecutive) copies of a short oligonucleotide. Interspersed repeats are more common in the human genome (Smit, 1999), contributing to evolution in unexpected ways, notably by regulating mammalian genes (Feschotte, 2008; Marino-Ramirez and Jordan, 2006; Marino-Ramirez et al., 2005), possibly because of epigenetic modifications (Huda et al., 2010; Huda et al., 2009; Huda et al., 2011). On the other hand, simple repeats (sometimes known as microsatellites) are highly variable DNA sequences, usually less than 100 base pairs long, often composed of short tandem repeats of one to six nucleotides. Typically, simple repeats have co-dominant inheritance, making them the markers of choice in a variety of applications, such as the characterisation and certification of genetic materials in genetic mapping and breeding programs (Simbaqueba et al., 2011; Garzon-Martinez et al., 2012).
Many repeat-finding tools recognise both interspersed and simple repeats. RepeatMasker, one of the most widely used tools for repeat identification (http://repeatmasker.org), relies on local sequence alignment to compare genomic sequences with a library of known repeats (Jurka et al., 2005). The fact that both interspersed and simple repeats complicate sequence similarity searches has motivated a variety of tools for identifying and masking repeats (Wootton and Federhen, 1996; Benson, 1999; Morgulis et al., 2006a; Frith, 2011a; Saha et al., 2008; Lerat, 2010). Most tools, however, even ones for finding simple repeats, have an ad hoc basis. Our desire to provide mathematical foundations for finding simple repeats within biological sequences led us to generalise the Ruzzo-Tompa (RT) algorithm, which finds ungapped subsequences of unusual composition (Ruzzo and Tompa, 1999). Our generalisation of the RT algorithm finds gapped subsequences of unusual composition. A specialisation of the generalisation then finds gapped repeats within biological sequences.

1.3 The Ruzzo-Tompa algorithm

“The identification of unusual subsequences is a fundamental task in biological sequence analysis. Karlin and Altschul, e.g., assigned a score to each letter in a sequence, to search for contiguous subsequences with large total scores (Karlin and Altschul, 1990; Karlin et al., 1990; Karlin and Altschul, 1993). Their technique applies to proteins to find DNA-binding, transmembrane, or charged segments (Brendel et al., 1991; Karlin and Brendel, 1993; Karlin et al., 1992). In a predecessor to the present article, Ruzzo and Tompa (1999) give an example of how the technique was used to search for transmembrane segments of proteins. Transmembrane segments insert into the lipid bilayer of a cell membrane, so they tend to be more hydrophobic than the rest of the protein. Karlin and Brendel (1992) assigned to each amino acid the corresponding score $s(a)$ from the Kyte-Doolittle hydrophobicity scale (Kyte and Doolittle, 1982; Cornette et al., 1987). They then verified that contiguous high-scoring subsequences of the human β2-adrenergic receptor corresponded to known transmembrane domains. The Neyman-Pearson lemma (Neyman and Pearson, 1933) states, however, that log-odds scores are optimal statistics (according to a specific criterion), and so should improve on hydrophobicity scores. Accordingly, Karlin and Brendel derived log-odds scores $s(a)$ from the empirical composition of amino acids within and outside of known transmembrane segments and then demonstrated improved identification and delimitation of the transmembrane segments in the β2-adrenergic receptor.”

Appendix A describes the formal computational problem, but to describe the work of Ruzzo and Tompa (1999) slightly less formally, let “:=” denote a definition. Consider any alphabet $\Sigma$ (e.g. the nucleotides or amino acids) and any scoring function $s: \Sigma \mapsto \mathbb{R}$ [e.g. the Kyte-Doolittle hydrophobicity scale above, which assigns a score $s(a)$ to each amino acid $a$]. Every amino acid sequence $L := L_1...L_n \in \Sigma^*$ then corresponds to a numerical sequence $x_i := s(L_i)$. Define a function $S$ by $S(0) := 0$ and $S(i) := x_i + ... + x_j (i = 1, 2, ..., n)$. Our interest in subsequences motivates defining the segment $[i, j] := \{i, i+1, ..., j\}$ and the segmental score $S[i, j] := S(j) - S(i) = x_i + ... + x_j$. For brevity below, ‘score’ always refers to a segmental score. Where necessary, we rely on context to disambiguate the (segmental) score $S[i, j]$ and the function $S(i)$. Below, ‘$I$’ with various adornments such as primes and subscripts denotes a segment, e.g. if $I = [i, j]$, then $S(I) := S[i, j]$. Ruzzo and Tompa (1999) considered only scores $S[i, j]$ of the form $S[i, j] := S(j) - S(i)$, which we call ‘global scores’. The applications mentioned above demonstrate that global
scores hold considerable biological interest, but our interest lies in other, more general types of scores for identifying unusual subsequences. Accordingly, let \( \bar{S}(I) \) denote any segmental score, not necessarily a global score.

Let ‘\( \subset \)' denote strict inclusion of sets and ‘\( \supset \)' strict containment; so \( B \subset B' \) iff (if and only if) \( B \supset B' \) iff \( B \subseteq B' \) and \( B \neq B' \). The segment \( I_0 \) satisfies ‘the subsegment property for \( \bar{S} \)' iff \( \bar{S}(I) < \bar{S}(I_0) \) for every strict subsegment \( I \subset I_0 \). (We sometimes omit the qualification ‘for \( \bar{S} \)' if \( \bar{S} \) is clear from context. Note that if \( I_0 \) contains no subsegments, it satisfies the subsegment property vacuously.) Intuitively, if \( \bar{S} = S \) is a global score derived from a function \( s : \mathcal{S} \to \mathbb{R} \) scoring a biological property, \( I_0 \) has more of the biological property than any of its proper subsegments. Now, consider any algorithm \( A \) whose input is a segment \( I \), and whose output is a subsegment \( I_0 \subseteq I \) having the subsegment property and a positive score \( \bar{S}(I_0) > 0 \), and maximising the score \( \bar{S} \) over all \( I_0 \subseteq I \). If there is more than one such \( I_0 \), \( A \) outputs one such \( I_0 \) arbitrarily; and if there is no such \( I_0 \), \( A \) outputs \( \emptyset \).

“From \( A \), derive the following divide-and-conquer algorithm \( A' : (0) \) set \( [i,j] \leftarrow [0,n] \); (1) apply \( A \) to the input segment \( [i,j] \) to output a segment \( [i_0,j_0] \subseteq [i,j] \), and terminate \( A' \) if the output of \( A \) is \( \emptyset \); (2) otherwise, return to Step 1 recursively, inputting into \( A' \) the segments to the left and right of \( [i_0,j_0] \), i.e., input both \( [i,j] \leftarrow [i,i_0] \) and \( [i,j] \leftarrow [j_0,j] \) into \( A' \). Call any segment in the output of \( A' \) a ‘maximal segment within \( [0,n] \)’.”

Ruzzo and Tompa pointed out that for a global score \( \bar{S} = S \), just to read a set of values \( \{S(i) : i \in [0,n]\} \), \( A' \) generally requires \( O(n) \) time. At a minimum, then, \( A' \) requires times typical of a divide-and-conquer algorithm: (1) worst-case time \( O(n^2) \) and (2) expected time \( O(n \log n) \). For a global score, Ruzzo and Tompa give an alternative to \( A' \), the RT algorithm, which finds the maximal segments in linear \( O(n) \) time. The RT algorithm can be found elsewhere (Ruzzo and Tompa, 1999); the only detail relevant to the present article is that it inputs the values \( \{S(i) : i \in [0,n]\} \) sequentially.

## 2 Results

### 2.1 Statement of the general theoretical results

The RT algorithm can be extended to solve a particular class of maximal path problems on weighted graphs. Call a graph \( G \) totally ordered iff its vertices are totally ordered. Without loss of generality, the vertices of a finite, totally ordered graph \( G \) can be labelled with the set \( [0,n] = \{0,1,2,\ldots,n\} \), under the usual total ordering ‘\(<\}'. If \( G \) contains an edge from \( i \) to \( j \) (\( 0 \leq i < j \leq n \)), denote the edge by \((i,j)\), and for convenience below, direct the edge from \( i \) to \( j \). Let \( \mathbb{R} \) denote the real numbers, and associate a weight \( W(i,j) \in \mathbb{R} \) with each edge \((i,j)\) of \( G \). A (directed) path \( \pi \) from \( i_0 \) to \( i_k \) is a sequence of edges \((i_0,i_1),\ldots,(i_{k-1},i_k)\) in \( G \) joining the vertices \( i_0 < \ldots < i_k \). (Because of the direction of the path’s edges, the path is ‘increasing’, as in the Abstract.) We write \( i \in \pi \) or \((i,j) \in \pi \) to indicate that the vertex \( i \) or the directed edge \((i,j)\) appears in the sequence of directed edges of \( \pi \). Define the weight of \( \pi \) as \( W(\pi) = \sum_{j=1}^{k} W(i_{j-1},i_j) \).
For $i < j$, let $\Pi[i,j]$ denote the set of directed paths from $i$ to $j$. Consider the following segmental score, the ‘local score’ $\hat{S}[i,j] := \max \{ W(\pi) : \pi \in \Pi[i,j] \}$, with the convention that $\hat{S}[i,i] := 0$. Define the corresponding global score $S(i) = \hat{S}[0,i]$. To avoid trivialities, assume $S(i) > -\infty$ for every $i \in [0,n]$. An edge weight $-\infty$ cannot contribute to the global score $S$, so as will be seen, in practice, the weight $-\infty$ is equivalent to omitting the corresponding edge from consideration.

In the present terminology, Ruzzo and Tompa (1999) examined the local score, in the special case $W(i-1,i) = x_i$, where $W(i,j) = -\infty$ whenever $j \geq i+2$, so $\hat{S}[i,j] = S[i,j] - S(i)$. To provide motivation for a more elaborate local score, consider a gap penalty $g : \mathbb{N} \mapsto \mathbb{R}$, where $\mathbb{N}$ denotes the natural numbers. Let $W(i-1,i) := \max \{ x_i, -g(1) \}$ and $W(i,j) := -g(j-i)$ for $i,j \in [0,n]$ with $j-i \geq 2$. The maximum defining $W(i-1,i)$ implicitly optimises the edge between $i-1$ and $i$ as either a score or a gap of length 1. Then, the local score $\hat{S}[i,j]$ modifies the global score $S[i,j]$ in Ruzzo and Tompa (1999) by permitting gaps in the segment $[i,j]$ and maximising the total gapped score over all possible arrangements of the gaps.

Call a weighted, totally ordered graph $G$ decreasing iff $W(i,k) \leq W(i,j)$ and $W(i,k) \leq W(j,k)$ for every $i < j < k$. In addition, the two segments $[i_0, j_0]$ and $[i_1, j_1]$ overlap internally iff $j_0 \neq i_1$, $j_1 \neq i_0$, and $[i_0, j_0] \cap [i_1, j_1] \neq \emptyset$. Appendix A proves the following theorem, which is our main result.

**Theorem 1:** In a finite, weighted, totally ordered, decreasing graph $G$ with vertex set $[0,n]$, consider the maximal segments corresponding to any local score $\hat{S}$. Use the corresponding global scores $\{ S(i) : i \in [0,n] \}$ as input for the RT algorithm, and call the corresponding output ‘RT-maximal segments’. Then: (1) all maximal segments for $\hat{S}$ within $[0,n]$ are RT-maximal segments within $[0,n]$, and vice versa; (2) for each maximal segment $[i,j]$, $\hat{S}[i,j] = S(j) - S(i)$; and (3) if a segment $I$ does not overlap internally with a maximal segment, then $\hat{S}(I) \leq 0$.

**Remark:** By computing the values $S(i)$ $(i = 0, 1, \ldots, n)$ and using them as successive inputs to the RT algorithm, the output contains all maximal segments for $\hat{S}$ and their scores, with a guarantee that any segment not overlapping internally with a maximal segment has a non-positive score $\hat{S}$. If the computation of each $S(i) (i = 0, 1, \ldots, n)$ requires $O(1)$ (i.e. bounded) time (as in the application to repeats below), the RT algorithm then outputs the maximal segments in time $O(n)$, even though the weighted graph has $O(n^2)$ edges and $O(n^2)$ segments. Typically, given the score $\hat{S}[i,j]$ of a maximal segment $[i,j]$, the standard technique of backtracking in a dynamic programming algorithm can then determine all paths from $i$ to $j$ having score $\hat{S}[i,j]$, if desired.
Appendix A proves Theorem 1.

2.2 A specialisation of the general theoretical results to repeats

Let \( \mathcal{A} \) be any alphabet (here, the nucleotide alphabet \( \{A,C,G,T\} \)), let \( s: \mathcal{A}^2 \rightarrow \mathbb{R} \) be any similarity matrix for \( \mathcal{A} \), and let the gap penalty \( g: \mathbb{N} \rightarrow \mathbb{R} \) be any non-decreasing function. (In applications, \( g \) is usually subject to additional restrictions, like positivity \( g(i+j) \leq g(i) + g(j) \); the affine gap penalty \( g(k) = \Delta_0 + \Delta_k \) used below is common and satisfies both additional restrictions. Throughout, as in the applications, we assume \( \Delta_0 \geq 0 \).) Consider any input sequence \( L:= L_1 \ldots L_n \in \mathbb{A}^{n+w} \), where for convenience the initial index is the non-positive offset \( 1-w \). The ‘lagged scores’ \( \hat{s}(L_i, L_j) \) can detect simple repeats, as follows. As in Section 2.1, consider a totally ordered weighted graph \( G \) with vertex set \( \{0, \ldots, n\} \) and weights

\[
W(i-1,i) = \max \{s(L_{i-w}, L_i) - g(1)\} \quad \text{for} \quad 1 \leq i \leq n \quad \text{and} \quad W(i, j) = -g(j-i) \quad \text{for} \quad 0 \leq i < j \leq n
\]

(see Figure 1). Large scores \( \hat{S}[i,j] \) correspond to inexact tandem repeats, where the directed edges with weights \( W(i-k, i) = -g(k) \quad (k \geq 2) \) correspond to gapping out \( k \) letter-pairs \( (L_{j-w}, L_j) \) with unfavourable scores \( s(L_{j-w}, L_j) \) \( (j = i-k+1, \ldots, i) \). For example, if \( s(a,b) = 1 \) for \( a = b \), and \( s(a,b) = -\infty \) otherwise, gapless segments with positive scores correspond to exact tandem repeats of a single word of length \( w \).

By construction, the weights \( W(i, j) \) are decreasing, so Theorem 1 states that the RT algorithm applied to the global score \( S(j) = \max \{W(\pi): \pi \in \Pi[0, j]\} \) yields the maximal segments \( [i, j] \) for \( \hat{S} \) and the corresponding scores \( \hat{S}[i, j] = S(j) - S(i) \).

2.3 Implementation of the program RepWords for finding repeats

Our computer program ‘RepWords’ finds maximal segments associated with gapped repeats. RepWords is publicly available from the URL http://tinyurl.com/spouge-repwords. RepWords 1.0 is tuned to handle long sequences with minimal memory requirements, whereas RepWords 1.1 is faster, but requires more memory. The URL lists other differences between RepWords 1.0 and 1.1. All results and timing information presented here pertain to RepWords 1.1.

As a technical point, in a context analogous to Figure 1 and with a word-length \( w = 3 \), consider the sequence GGTGG. A maximal segment corresponding to the self-alignment

\[
\begin{align*}
\text{GGTGG} \\
\| \\
\text{GGTGG}
\end{align*}
\]

is of length \( l = 2 < w \). One could consider the entire five letters GGTGG as a repeat, but empirically, the ROC curves (presented below) improved if the repeat excluded the intermediate \( w - l = 1 \) letters (e.g. the letter ‘T’ above). Accordingly, if a maximal segment has length \( l < w \), the corresponding maximal subsequence has length \( 2l \) and
excludes the intermediate \( w-l \) letters in the corresponding subsequence. On the other hand, if the maximal segment has length \( l \geq w \), the corresponding maximal subsequence has length \( w+l \).

**Figure 1** An example of a weighted, directed finite graph that induces a total order on its vertices, in an application to tandem repeats. Figure 1 shows a sequence ACTACTGGTGG, an inexact tandem repeat of word-length \( w=3 \). For illustrative purposes, the scoring system is \( s(a,b)=4 \) for \( a=b \); \( s(a,b)=-5 \) for \( a \neq b \); and a gap penalty \( g(k)=5+2k \). To score the repeat, shift the sequence by \( w=3 \), and score the edges of length 1 (shown in red) with the score \( \max \{s(a,b),-g(1)\} \). To permit gapping, score edges of length \( k>1 \) with the length-dependent weight \(-g(k)\). The edges with a black border indicate the maximally weighted path from left to right vertex, the gaps suggesting a mismatched segment within the repeat (see online version for colours).

2.4 *Empirical timing results for RepWords and the divide-and-conquer algorithm*

Unless stated otherwise, all empirical results used the similarity matrix for the nucleotide alphabet \{A, C, G, T\} : \( s(a,b)=2 \) for \( a=b \), and \( s(a,b)=-3 \) otherwise. (When this manuscript originated, the scoring matrix was the BLASTN default, but the default has since changed.)

To compare RepWords’ empirical computational times against the divide-and-conquer algorithm mentioned in Section 1 (adapted for repeat finding), we generated random nucleotide sequences of different lengths under uniform background frequencies \( (0.25,0.25,0.25,0.25) \) for the alphabet \{A, C, G, T\}. ROC curves (presented later) suggested that the gap penalty \( g(k)=1+k \) was superior to the gap penalty \( g(k)=5+2k \), so the timing tests used \( g(k)=1+k \) was superior to the gap penalty \( g(k)=5+2k \), so the timing tests used \( g(k)=1+k \). (For \( a \neq b \) and \( g(k)=1+k \), \( \max \{s(a,b),-g(1)\} = \max \{-3,-2\} = -2 \), so the mismatch score could be changed to \( s(a,b)=-2 \) without influencing results.)

The timing results in Figure 2 show that for each repeat word-length \( w \), RepWords determines the maximal segments in approximate time \( O(n) \); the divide-and-conquer
algorithm, in approximate time $O(n \log n)$. (Algorithm $A'$ does not backtrack to find optimal paths, so the timing results do not include backtracking. The backtrack passes beyond the direct purview of this article. Note in passing, however, that a backtrack to find just one out of possibly many optimal paths requires time $O(n)$, because of its constant time per node.)

**Figure 2** Computational time per letter in a sequence vs. the number of letters in the sequence. Figure 2 plots average timing results for random nucleotide sequences for the divide-and-conquer algorithm $A$ in red circles and for the RT algorithm in blue squares (see online version for colours)

### 2.5 ROC curves for repeat-finding by RepWords and RepSeek

All empirical results involving real DNA were based on the full length of human chromosome 19. In the following, percentages refer to the fraction of the length of chromosome 19, 59, 128, 983 bases, of which 5.61% are Ns.

We adopted RepeatMasker (RepeatMasker at http://repeatmasker.org) as our standard of truth for repeat-finding. Although the standard it provides is imperfect (and is arguably a matter of definition anyway), no clearly superior standard is available. Under its default settings, RepeatMasker annotated repeats covering 54.55% of the length of chromosome 19, classifying the repeats as follows: SINEs, 27.00%; LINEs, 12.97%; LTRs, 8.17%; DNA repeats, 2.04%; satellite repeats, 2.01%; simple repeats, 1.25%; low complexity repeats, 0.76%; and miscellaneous repeats, 0.37%.

We chose to compare RepWords to the repeat-finding program RepSeek (Achaz et al., 2007), because they both use the lagged scores $s(L_{i-n}, L_i)$ mentioned in Section 2.2. The fact that the lagged scores have been used independently at least twice (Spouge, 2007) suggests that they provide a natural method of finding simple repeats. RepSeek incorporates seed and gapped extension heuristics, however, unlike RepWords but somewhat like gapped BLAST (Altschul et al., 1997). The comparison between RepWords and RepSeek examines the relative practical merits of specialising the generalised RT algorithm and a similar approach tailored to a specific application.
We compared the empirical performance of RepWords and RepSeek by using ROC curves (Gribskov and Robinson, 1996). Various types of repeats called by our standard of truth, RepeatMasker, sometimes contain simple repeats within them. Consider any particular repeat-type of interest. Then (1) any nucleotide base not in a RepeatMasker repeat was considered a negative (i.e. a non-repeat base); (2) any base within a RepeatMasker repeat of interest, a positive (i.e. a base of that type); and (3) because of overlapping repeats, all other bases were discarded from consideration, to avoid contamination of negatives with cryptic positives. As shown in Figure 3, RepWords decisively dominated RepSeek for finding simple repeats and low-complexity repeats. Appendix B contains Figures B1–B6 showing, however, that RepWords performed poorly on other types of repeats, compared either to RepSeek or in absolute terms (partly reflecting our intent, to develop a tool for finding simple inexact repeats). Figures B1–B6 also suggest, however, that when RepWords performs poorly, RepSeek is rarely (if ever) a practical alternative.

The linear gap penalties $g(k) = k$ and $g(k) = 2k$ are equivalent to ungapped alignment mismatch scores of $s(a,b) = -1$ and $s(a,b) = -2$, respectively. In Figure 3, therefore, the advantages of a gap penalty, though distinct, are therefore modest, much as they are in sequence alignment.

**Figure 3** ROC curves comparing RepSeek and RepWords for different gap penalties. Figure 3A displays ROC curves for simple repeats and Figure 3B, for low complexity repeats. All repeats are classified according to RepeatMasker. The black curves show results for RepSeek with its default settings. RepWords (with the word-length $w_{\text{max}} = 200$) used the match-mismatch scoring matrix $s(a,b) = 2$ for $a = b$ and $s(a,b) = -3$ for $a \neq b$ with a variety of gap penalties: $g(k) = 5+k$ (solid red), so repeats are ungapped; $g(k) = 5+k$ (solid orange); $g(k) = 5+2k$ (dashed orange); $g(k) = 3+k$ (solid green); $g(k) = 3+2k$ (dashed green); $g(k) = 1+k$ (blue); $g(k) = 1+2k$ (dashed blue); $g(k) = k$ (solid purple); and $g(k) = 2k$ (dashed purple). (Some curves, in particular the solid red curve, are partially obscured by others.) (see online version for colours)
2.6 Empirical timing results for RepWords and RepSeek

As Section 5 describes, the use of RepSeek required splitting chromosome 19 into pieces of approximately equal length, with 160 pieces corresponding to RepSeek’s best ROC performance in our hands. Piece #10 of the 160 pieces was taken arbitrarily as a typical piece, to time programs. The estimated CPU time RepSeek required for Piece #10 was about 262 s (or about 11.7 hours for the entire chromosome). For comparison, RepWords should be slowest when its output contains all maximal segments, corresponding to a threshold score \( y = 1 \). Accordingly, for the timing results, we set \( y = 1 \). The ROC curves in Figure 3 suggest that RepWords’ repeat-finding performance is enhanced with the gap penalty \( g(k) = 1 + k \), used accordingly for the timing results. On Piece #10 of chromosome 19, RepWords output the union of all maximal segments for all word-lengths \( w \) up to the maximum word-length \( w_{\text{max}} = 10 \) in 0.13 s CPU time; up to \( w_{\text{max}} = 20 \), in 0.18 s; up to \( w_{\text{max}} = 100 \), in 0.66 s; and up to \( w_{\text{max}} = 200 \), in 1.33 s.

2.7 The spectrum of word-lengths composing a repeat as determined by RepWords

Figure 4 suggests that simple and low-complexity repeats are composed of a spectrum of word-lengths \( w \). On the one hand, the ROC curve for \( w_{\text{max}} = 10 \) rises most rapidly from the origin, suggesting that many simple and low-complexity repeats are dominated by short word-lengths \( w \). The results displayed in Figure 4 also suggest that decomposing repeats into a spectrum of word-lengths might be informative.

![Figure 4](image_url) ROC curves comparing RepSeek and RepWords for different word-lengths. Figure 4A displays ROC curves for simple repeats; Figure 4B, for low complexity repeats. The black curves show results for RepSeek with its default settings. RepWords used the match-mismatch scoring matrix \( s(a,b) = 2 \) for \( a = b \) and \( s(a,b) = -3 \) for \( a \neq b \) and the gap penalty \( g(k) = 1 + k \) with all word-lengths \( w \) up to the following maximum word-lengths: \( w_{\text{max}} = 10 \) (red); \( w_{\text{max}} = 20 \) (orange); \( w_{\text{max}} = 100 \) (green); and \( w_{\text{max}} = 200 \) (blue) (see online version for colours).
Accordingly, Figure 5 decomposes two types of repeats into a spectrum of word-lengths \( w \). On the one hand, Figure 5A for SINE repeats shows abrupt rises in coverage, indicating that the corresponding word-lengths \( w \) contribute disproportionately to SINE repeats. On the other hand, Figure 5B for LINE repeats shows a steady rise, indicating an absence of word-lengths \( w \) contributing disproportionately to LINE repeats. (Corresponding plots for simple and low-complexity repeats were less striking, rising from 0.0 to 1.0 rapidly, indicating that the underlying spectrum of word-lengths \( w \) consists mostly of short word-lengths.)

**Figure 5** Curves giving the spectrum of word-lengths underlying repeats up to word-length 1000. Figure 5A displays the curve for SINE repeats; Figure 5B, for LINE repeats. RepWords used the match-mismatch scoring matrix \( s(a,b) = 2 \) for \( a = b \) and \( s(a,b) = -3 \) for \( a \neq b \) and the gap penalty \( g(k) = 1 + k \). The maximal segment score threshold chosen was \( y = 17 \). In the sense mentioned in Section 5, results are robust against the exact choice of \( y \) (see online version for colours)

Accordingly, Figure 5 decomposes two types of repeats into a spectrum of word-lengths \( w \). On the one hand, Figure 5A for SINE repeats shows abrupt rises in coverage, indicating that the corresponding word-lengths \( w \) contribute disproportionately to SINE repeats. On the other hand, Figure 5B for LINE repeats shows a steady rise, indicating an absence of word-lengths \( w \) contributing disproportionately to LINE repeats. (Corresponding plots for simple and low-complexity repeats were less striking, rising from 0.0 to 1.0 rapidly, indicating that the underlying spectrum of word-lengths \( w \) consists mostly of short word-lengths.)

### 3 Discussion

The generalisation of the Ruzzo-Tompa (RT) algorithm in this article holds the possibility of improving the performance of tools for finding biological subsequences of unusual composition, because minimal programming effort is now required to add and to tune gap penalties for individual biological applications. One simply writes a program...
Searching for repeats

for the gapped global scores \( S(i) = S[0, i] \) specific to the application, feeds the program output \( \{ S(i) : i \in [0, n] \} \) into the RT algorithm and then collects the output of the RT algorithm, namely the maximal segments of the gapped local score. A single subroutine implementing the RT algorithm therefore solves the general problem of finding maximal segments for any gapped scoring scheme.

In fact, our original desire was to develop a formal mathematical basis for finding simple repeats, which led us to generalise the RT algorithm (Ruzzo and Tompa, 1999). Unlike the \textit{ad hoc} methods underlying most tools for finding repeats (Benson, 1999; Morgulis et al., 2006a; Frith, 2011a,b; Morgulis et al., 2006b), the generalised RT algorithm provides our tool, RepWords, with a systematic approach generalising some methods of repeat detection already extant in the literature (Achaz et al., 2007; Spouge, 2007).

Figure 2 shows that although the more natural divide-and-conquer algorithm for locating subsequences of unusual composition requires time \( O(n \log n) \) to find simple repeats, RepWords only requires time \( O(n) \) for each word-length \( w \). Although RepWords searches individually for repeats of each word-length of interest, word-match heuristics (like those used in BLAST; Altschul et al., 1997; Smith and Waterman, 1981; Altschul et al., 1990) might accelerate simultaneous searches across different word-lengths \( w \). We are currently investigating the incorporation of word-match heuristics into RepWords.

Within RepWords, Gotoh’s (1982) algorithm calculates in constant time each of the global scores \( S(i) \) \((i = 0, 1, \ldots, n)\) as inputs for the RT algorithm, limiting RepWords to affine gap penalties \( g(k) = \Delta_0 + \Delta k \). Although affine gap penalties suffice for most repeat-finding applications, efficient algorithms are also available to calculate global scores for sublinear or superlinear gap penalties \( g(k) \) (Miller and Myers, 1988; Eppstein et al., 1988; Galil and Giancarlo, 1989). Such algorithms can provide the corresponding input for the RT algorithm, just as Gotoh’s algorithm did for affine gap penalties. In fact, the generalised RT algorithm presented here requires only global scores \( S(i) \) as input; it even removes the need for any explicit algorithm to compute gapped segmental (local) scores \( \hat{S}[i, j] \).

As exemplified by finding simple repeats within DNA sequences, therefore, this article provides a proof of principle, suggesting that the generalised RT algorithm has broad implications for the methodical development of tools for finding subsequences of unusual composition.

4 Conclusions

This article generalises the Ruzzo-Tompa (RT) algorithm for finding ungapped subsequences of unusual composition, effectively making it an algorithm for finding gapped subsequences of unusual composition. In a specific case (perhaps not the simplest, but one reflecting the original motivation of the theory), we applied the generalisation, to develop a tool for finding simple repeats. With repeat classes
determined by using RepeatMasker as a gold standard, when finding the classes of simple and low-complexity repeats, the resulting tool (‘RepWords’) performs well when compared to a tool with a similar but less general basis.

5 Methods

All computations were carried out on a 64-bit CentOS 5.8 operating system with 48G RAM and an Intel(R) Xeon(R) CPU X5660 (2.80GHz, 2 processors x 6 cores / 2 threads).

5.1 The Ruzzo-Tompa algorithm applied to finding inexact simple repeats with gaps permitted

In the context of Theorem 1 and Section 2.2, consider the affine gap penalty \( g(k) = \Delta_0 + \Delta_k \) common to many sequence applications. If \( \Delta_i \geq 0 \), then the weights \( W(i,j) \) are decreasing, as required. The global score can be computed efficiently with the Gotoh (1982) affine gap algorithm, as follows. Initialise with \( S_0 = 0 \) and \( I_0 = -\infty \). For \( j = 1, \ldots, n \),

\[
\begin{align*}
S_j &= \max\{S_{j-1}, I_{j-1}\} + s(B_{j-w}, B_j) \\
I_j &= \max\{S_{j-1} - \Delta_0, I_{j-1}\} - \Delta_i
\end{align*}
\]

(1)

The Gotoh affine gap algorithm computes \( S(j) := \max\{S_j, I_j\} \) and the successive inputs \( S(i) \ (i \in [0,n]) \) are passed into the RT algorithm.

5.2 ROC curves for RepSeek

To produce an ROC curve, the number of false positives predicted by a program must vary. Although the default mode of RepSeek has no free parameters, RepSeek does permit input of seed minimum length and repeat minimum score, similar to parameters in gapped BLAST (Morgulis et al., 2006b). As an option favourable to RepSeek’s performance, we varied the repeat minimum score parameter across its full range. When applied to the full length of chromosome 19, RepSeek returned an error. To avoid computational problems, we split chromosome 19 into pieces and then combined the results from RepSeek. Figures display results from splitting chromosome 19 into 160 pieces, corresponding to RepSeek’s best performance in our hands.

5.3 ROC curves for RepWords

All maximal segments \([i,j]\) correspond to a local score \( \hat{S}[i,j] \). ROC curves for RepWords were generated by discarding all maximal segments with scores \( \hat{S}[i,j] < y \) and then varying the threshold \( y \).
5.4 Determining the spectrum of word-lengths \( w \) composing a repeat-type

Consider all bases in chromosome 19 that RepeatMasker designated as, e.g., a SINE repeat. Call them ‘SINE bases’. Fix some arbitrary threshold \( y \). Define RepWords’ coverage of RepeatMasker SINEs to be the fraction of SINE bases in at least one maximal subsequence (defined immediately above) whose word-length is no more than \( w_{\text{max}} \) and whose score is no less than \( y \). Arbitrarily, we chose \( y = 17 \) as a threshold that includes most SINE bases for sufficiently large \( w_{\text{max}} \). In Figure 5, an increase in \( y \) multiplies the coverage by an approximate constant less than 1.0, so (within limits) the results are robust against the choice of \( y \), other than a change in the scale of the \( y \)-axis.

Acknowledgements

All authors contributed to the design of the study. John L. Spouge and Sergey L. Sheetlin contributed to the theory. Leonardo Mariño-Ramírez prepared the sequence data. Sergey L. Sheetlin implemented the computer programs and carried out the computer study.

This research was supported by the Intramural Research Program of the NIH, National Library of Medicine.

References


Appendix A

Appendix A proves our main results, Theorems 2 and 3, through a series of technical lemmas. (The lemmas themselves are of very limited interest.) Theorems 2 and 3 together imply Theorem 1 as a special case.

Sufficient conditions for disjoint maximal segments

Define \( (n) = [1,n] \) (a standard notation) and \( (n)_0 = [0,n] \), and fix a (segmental) score \( \tilde{S} \). A segment \( I \subseteq I' \) satisfies ‘the supersegment property within \( I' \) for \( \tilde{S} \)’ iff no supersegment \( I_0 \supseteq I \) such that \( I_0 \subseteq I' \) has the subsegment property. (Thus, the subsegment property is intrinsic to a segment \( I \), whereas the supersegment property has dependencies extrinsic to it.) A segment \( I_0 \subseteq I' \) is a n ‘RT-maximal segment within \( I' \) for \( \tilde{S} \)’ iff \( \tilde{S}(I_0) > 0 \) and \( I_0 \) has both the subsegment property and the supersegment property within \( I' \) for \( \tilde{S} \). The following omits ‘for \( \tilde{S} \)’ if the score \( \tilde{S} \) is clear from context; it sometimes omits the phrase ‘within \( I' \)’ if \( [0,n] \).

For global scores, Ruzzo and Tompa proved that every RT-maximal segment is a maximal segment (i.e. one of the segments in the output of the divide-and-conquer algorithm \( A' \) ) and vice versa. To generalise their proofs to scores other than global scores, call \( \tilde{S} \) an ‘RT score’ iff it has two properties. The first is the reverse triangle inequality: \( \tilde{S}[i,j] + \tilde{S}[j,k] \leq \tilde{S}[i,k] \) whenever \( i < j < k \). The second is the positivity property: if \( [i_0,j_0] \) has the subsegment property for \( \tilde{S} \), then \( \tilde{S}[i_0,j_0] > 0 \) whenever \( i_0 < i < j_0 \). The proofs below use the convention that \( \tilde{S}[i,j] = 0 \), to include the trivial inequalities \( \tilde{S}[i,j] + \tilde{S}[j,k] \leq \tilde{S}[i,k] \) and \( \tilde{S}[i,j] + \tilde{S}[k,k] \leq \tilde{S}[i,k] \) in the reverse triangle inequality, thereby avoiding the separate analysis of boundary cases.

The present subsection essentially paraphrases the corresponding proofs in Ruzzo and Tompa (1999) as a sequence of technical lemmas. In particular, the lemmas ensure that RT-maximal segments remain disjoint for general RT scores (Lemma 4 below), not just for global scores. Lemma 4 is crucial to the main result, Theorem 2, which asserts the equivalence of RT-maximal segments and maximal segments, demonstrating that the RT algorithm and algorithm \( A' \) have equivalent outputs for an RT score.

Lemma 1 shows that RT segmental scores generalise the notion of a global score.

Lemma 1: Every global score \( S \) is an RT score.

Proof: The reverse triangle inequality is immediate (and reduces to equality for global scores). To prove the positivity property, if \( [i_0,j_0] \) has the subsegment property for \( S \), \( S[i,j] < S[i_0,j_0] \) for every \( [i,j] \subseteq [i_0,j_0] \). The special case \( j = j_0 \) shows that \( S(i) < S(j_0) \) whenever \( i_0 < i < j_0 \). Similarly, the special case \( i = i_0 \) shows that \( S(j) < S(j_0) \) whenever \( i_0 < j < j_0 \). The definition \( S[i,j] := S(j) - S(i) \) then proves the Positivity Property of \( S \).

Lemma 2: For every score \( \tilde{S} \), if \( \tilde{S}(I) > 0 \), then some \( I_0 \subseteq I \) has the subsegment property and a positive score \( \tilde{S}(I_0) > 0 \).
Proof: Either \( I \) itself has the subsegment property or else there is some \( I_i \subset I \) such that \( \tilde{S}(I_i) \geq \tilde{S}(I) > 0 \). Continue recursively. By the principle of infinite descent, the recursion terminates at some \( I_0 \subset I \) having the subsegment property (possibly vacuously) and a positive score \( \tilde{S}(I_0) > 0 \).

Lemma 3: Let \( \tilde{S} \) be an RT score. Let \( I_0 = [i_0, j_0] \subset I' \) have the subsegment property and a positive score \( \tilde{S}(i_0, j_0) > 0 \), and let \( I = [i, j] \) be an RT-maximal segment within \( I' \). If \( I_0 \cap I \neq \emptyset \), then \( I_0 \subseteq I \).

Proof: The subsegment property of \( I_0 \) and the supersegment property of \( I \) within \( I' \) exclude the possibility \( I_i \subset I_0 \), so except for the case \( I_0 \subseteq I \), there are only two symmetric cases remaining: (1) \( i_0 < i \leq j_0 < j \) and (2) \( i < i_0 \leq j < j_0 \). Using the positivity property, we eliminate Case 1 explicitly below; a symmetric argument eliminates Case 2.

In Case 1, \( i_0 < i \leq j_0 \) implies \( i_0 < i \leq j_0 \) or \( i < i \leq j_0 \) (or both), which leads to a contradiction, as follows. Whenever \( i < i_0 \), the subsegment property of \( I_0 \) and its positivity property show that \( \tilde{S}(i_0, i) > 0 \). Moreover, \( \tilde{S}(i_0, j_0) > 0 \). Similarly, whenever \( i_0 < i \leq j_0 \), the reverse triangle inequality, the inequality \( \tilde{S}(i_0, i) \geq 0 \) and the positivity property of \( I \) (or its positive score \( \tilde{S}(i, j_0) > 0 \)) imply that \( \tilde{S}(i_0, i) \geq \tilde{S}(i, j) + \tilde{S}(j, j_0) > 0 \).

Thus, \( \tilde{S}(i_0, i) > 0 \) whenever \( i < i \leq j \). By symmetry, \( \tilde{S}(j, j_0) > 0 \) whenever \( i_0 < i \leq j_0 \). Consequently, for every \( [i, j] \subset [i_0, j_0] \),

\[
\tilde{S}(i_0, i) + \tilde{S}(i, j) + \tilde{S}(j, j_0) > \tilde{S}(i, j) \quad (A1)
\]

Thus, \( [i_0, j_0] \) has the subsegment property, which contradicts the supersegment property of \( [i, j_0] \subset [i_0, j_0] \) within \( I' \). As stated above, the only remaining possibility is \( I_0 \subseteq I \).

Lemma 4: For an RT score \( \tilde{S} \), if \( I_0 \) and \( I_i \) are distinct RT-maximal segments within \( I' \), they are disjoint.

Proof: Every RT-maximal segment within \( I' \) has the subsegment property. Lemma 3 implies, therefore, if \( I_0 \cap I_i \neq \emptyset \), then \( I_0 \subseteq I_i \) and \( I_i \subseteq I_0 \), so \( I_0 = I_i \).

Lemma 5: For an RT score \( \tilde{S} \), let \( [i_0, j_0] \) be RT-maximal within \( [i, j] \). Then, (1) \( I_i \subset [i, i_0] \) is RT-maximal within \( [i, j] \) iff \( I_i \) is RT-maximal within \( [i, i_0] \) and (2) \( I_i \subset [i_0, j] \) is RT-maximal within \( [i, j] \) iff \( I_i \) is RT-maximal within \( [i_0, j] \).

Proof: We prove Assertion (1) explicitly; Assertion (2) then follows by a symmetric argument. All hypotheses and conclusions state that \( I_i \) has the subsegment property and a positive score, so the proof considers only the relevant supersegment properties. (1. \( \Rightarrow \) )
If no supersegment \( I_2 \) such that \( I_1 \subset I_2 \subseteq [i,j] \) has the subsegment property, then a fortiori, no supersegment \( I_2 \) such that \( I_1 \subset I_2 \subseteq [i,i_0] \) has the subsegment property. (1. \( \Rightarrow \) ) If \( I_1 \) were RT-maximal within \([i,i_0]\) but not within \([i,j]\), then \( I_1 \subset I_2 \), where \( I_2 \) has the subsegment property, \( S(I_2) > S(I_1) > 0 \), and \( I_2 \cap [i_0,j_0] \neq \emptyset \). By Lemma 3, \([i_0,j_0]\) cannot be maximal within \([i,j]\), a contradiction.

Two segments \([i_0,j_0]\) and \([i_1,j_1]\) overlap internally iff \( j_0 \neq i_1, j_1 \neq i_0 \), and \([i_0,j_0]\cap[i_1,j_1]\neq\emptyset\).

Theorem 2: For any RT score \( \tilde{S} \), (1) if a segment \( I \) has a positive score \( S(I) > 0 \), then it overlaps internally with an RT-maximal segment and (2) all maximal segments within \([0,n]\) are RT-maximal segments within \([0,n]\) and vice versa.

Proof of Conclusion (1): By Lemma 2, any segment with a positive score contains a segment having the subsegment property and a positive score. Thus, it overlaps internally with an RT-maximal segment.

Proof of Conclusion (2): Given input \( I \), Algorithm \( \tilde{A} \) outputs an RT-maximal segment \( I_o \) within \( I \) (\( I_o \) has the subsegment property and a positive score, and no segment within \( I \) has a strictly larger score than \( I_o \), so no supersegment \( I' \supset I_o \) within \( I \) can have the subsegment property). Inductive application of Lemma 5 shows that whenever \( \tilde{A} \) finds an RT-maximal segment within \( I \subset [0,n] \), in fact it finds another RT-maximal segment within \([0,n]\). Thus, \( \tilde{A}' \) outputs only segments RT-maximal in \([0,n]\), so every maximal segment is an RT-maximal segment. Lemma 4 shows that \( \tilde{A}' \) terminates only after all RT-maximal segments appear in its output, so every RT-maximal segment is also a maximal segment.

A path optimisation with disjoint maximal segments

Section 1 introduced the local score \( \hat{S}[i,j] := \max \{W(\pi) : \pi \in \Pi[i,j] \} \), with the convention that \( \hat{S}[i,i] = 0 \). Again, a sequence of technical lemmas precedes the main result, Theorem 3 and its Remark, which together assert that the maximal segments of \( \hat{S} \) and \( S \), along with their scores, are the same.

Before proceeding, let us list some properties of any local score \( \hat{S} \).

Lemma 6: For any local score \( \hat{S} \), the following properties hold: (1) the extended reverse triangle property, such that \( \hat{S}[i,j] + \hat{S}[j,k] \leq \hat{S}[i,k] \) whenever \( 0 \leq i \leq j \leq k \leq n \); (2) the edge maximisation property, such that \( \hat{S}[i,j] + W(j,j^*) + \hat{S}[j^*,k] \leq \hat{S}[i,k] \) whenever \( [j,j^*] \subseteq [i,k] \subseteq [0,n] \); and (3) the internal point maximisation property: if \( j^* \in \pi \) for any \( \pi \in \Pi[i,k] \) satisfying \( W(\pi) = \hat{S}[i,k] \), then \( \hat{S}[i,k] = \hat{S}[i,j^*] + \hat{S}[j^*,k] \).
Searching for repeats

Proof: Any cases not handled in the proof follow implicitly from the convention that \( \hat{S}[i,i] = 0 \). (1) The reverse triangle inequality holds whenever \( 0 \leq i < j < k \leq n \), because both sides maximise path weights over paths \( \pi \) from \( i \) to \( k \), but the one on the left includes the extra restriction that \( j \in \pi \). (2) The edge maximisation property holds whenever \( [j,j'] \subseteq [i,k] \subseteq [0,n] \), because both sides maximise path weights over paths \( \pi \) from \( i \) to \( k \), but the one on the left includes the extra restriction that \( (j,j') \in \pi \). (3) The internal point maximisation property holds whenever \( i < j < k \) because maximising path weights over the segments \([i,j]\) and \([j,k]\) separately cannot improve on the maximum weight \( W(\pi) = \hat{S}[i,k] \).

As a special case of the extended reverse triangle inequality, for every \( 0 \leq i \leq j \leq n \),
\[
\hat{S}[i,j] \leq \hat{S}[0,j] - \hat{S}[0,i] = S(j) - S(i) = S[i,j]
\]  \( \text{(A2)} \)

Lemma 6 specialised to produce other statements involving the global score \( S \), the following uses the implicit specialisations without comment. Lemma 6 shows that the reverse triangle inequality follows from the definition of \( \hat{S} \). The positivity property for \( \hat{S} \), however, requires an additional restriction on \( W(i,j) \). Call the weights ‘decreasing’ iff \( W(i,k) \leq W(i,j) \) and \( W(i,k) \leq W(j,k) \) for every \( 0 \leq i < j < k \leq n \).

Lemma 7: For decreasing weights, \( \hat{S} \) is an RT score.

Proof: To prove that \( \hat{S} \) has the positivity property, let \([i_0,j_0] \) have the subsegment property for \( \hat{S} \), i.e. \( \hat{S}[i,j] < \hat{S}[i_0,j_0] \) for every \([i,j] \subseteq [i_0,j_0] \). Let \( W(\pi) = \hat{S}[i_0,j_0] \) for \( \pi \in \Pi[i_0,j_0] \), to prove by contradiction that \( \hat{S}[i_0,i] > 0 \) whenever \( i_0 < i < j_0 \).

If not, then \( \hat{S}[i_0,i] \leq 0 \) for some \( i \) satisfying \( i_0 < i < j_0 \). If \( i \in \pi \), the internal point maximisation property in Lemma 6 yields
\[
\hat{S}[i_0,j_0] = \hat{S}[i_0,i] + \hat{S}[i,j] \leq \hat{S}[i,j_0]
\]  \( \text{(A3)} \)
contradicting the subsegment property of \([i_0,j_0] \), thereby implying \( i \notin \pi \). Then, however, \((i',j') \in \pi \) for some \( i_0 \leq i' < j' \leq j_0 \). Because \( W(i',j') \leq W(i',i) \),
\[
\hat{S}[i_0,j_0] = \hat{S}[i_0,i'] + W(i',j') + \hat{S}[j',j_0]
\leq \hat{S}[i_0,i'] + W(i',i) + \hat{S}[j',j_0]
\leq \hat{S}[i_0,i] + \hat{S}[j',j_0]
\]  \( \text{(A4)} \)
where the equality follows from the edge maximisation property in Lemma 6; the first inequality, from decreasing weights; and the final inequality, from a degenerate case of the edge maximisation property. If \( j' = j_0 \), then \( \hat{S}[i_0,j_0] \leq \hat{S}[i_0,i] \); otherwise...
\[ \hat{S}[i_0, i] \leq 0 \text{ implies } \hat{S}[i_0, j_0] \leq \hat{S}[j', j_0] \text{ for } j' < j_0. \] In either case, equation (A4) contradicts the subsegment property of \([i_0, j_0]\). Thus, \(\hat{S}[i_0, i] > 0\) whenever \(i_0 < i < j_0\).

A symmetric argument yields \(\hat{S}[i, j_0] > 0\) whenever \(i_0 < i < j_0\). Thus, \(\hat{S}\) is an RT score.

Now, we show that for decreasing weights, the RT-maximal segments for the local score \(\hat{S}\) and the global score \(S\) are in fact identical. Recall throughout the proofs that Lemma 6 has implications for \([\hat{S}, i, j]\). Thus,

\[ \hat{S}[i, j_0] > 0 \text{ whenever } i_0 < i < j_0. \]

Lemma 8: For decreasing weights, given any segment \([i_0, j_0]\), if \(S(i_0) < S(i)\) whenever \(i_0 < i \leq j_0\), then \(\hat{S}[i_0, i] = S(i) - S(i_0) > 0\) whenever \(i_0 < i \leq j_0\).

Proof: Let \(W(\pi) = S(i)\) for \(\pi \in \Pi[0, i]\) and \(i_0 < i \leq j_0\). Let us prove that \(i_0 \in \pi\), so the internal point maximisation property in Lemma 6 then yields the required conclusion. If \(i_0 = 0\), then \(i_0 \in \pi\). Otherwise, if \(i_0 \notin \pi\), then some \((i', j') \in \pi\), where \(0 \leq i' < i_0 < j' \leq i \leq j_0\). A degenerate case of the internal edge maximisation property shows that \((i') + W(i', i_0) \leq S(i_0)\), so

\[
S(j') = S(i') + W(i', j') \\
\leq S(i') + W(i', i_0) \leq S(i_0) < S(j')
\]

because \(W(i', j') \leq W(i', i_0)\). The contradiction proves \(i_0 \in \pi\), so Lemma 8 follows.

Theorem 3: For decreasing weights, \([i_0, j_0]\) has the subsegment property for \(S\) iff it has the subsegment property for \(\hat{S}\). Moreover, any segment \([i_0, j_0]\) possessing the subsegment properties for \(S\) and \(\hat{S}\) satisfies \(\hat{S}[i_0, j_0] = S[i_0, j_0] = S(j_0) - S(i_0)\).

Remark: Theorem 3 implies the supersegment properties for \(S\) and \(\hat{S}\) are also equivalent, so every RT-maximal segment for \(S\) is also an RT-maximal segment for \(\hat{S}\) and vice versa. Both \(\hat{S}\) and (more particularly) \(S\) are RT scores, so Theorem 2 pertains.

Proof: (\(\Rightarrow\)) If \([i_0, j_0]\) has the subsegment property for \(S\), then \(S[i, j] < S[i_0, j_0]\) for every \([i, j] \subset [i_0, j_0]\). An examination first of \([i, j_0] \subset [i_0, j_0]\) and then of \([i_0, i] \subset [i_0, j_0]\) shows that \(S(i_0) < S(i_0) < S(j_0)\) whenever \(i_0 < i < j_0\). Thus, for every \([i, j] \subset [i_0, j_0]\), because \(S(i) + \hat{S}[i, j] \leq S(j)\),

\[
\hat{S}[i, j] \leq S(i) + \hat{S}[i, j] - S(i_0) \\
\leq S(j) - S(i_0) \leq S(j_0) - S(i_0) \leq \hat{S}[i_0, j_0]
\]

where Lemma 8 justifies the final equality. Because either the first or the third inequality is strict, equation (A6) implies the subsegment property for \(\hat{S}\) within \([i_0, j_0]\).
Searching for repeats

(⇐) If \([i_0, j_0]\) has the subsegment property for \(\hat{S}\), then the positivity property shows that \(\hat{S}[i_0, i] > 0\) and \(\hat{S}[i, j_0] > 0\) whenever \(i_0 < i < j_0\). Equation (A2) then yields \(S[i_0, i] > 0\) and \(S[i, j_0] > 0\), whenever \(i_0 < i < j_0\). Thus, \(S(i_0) < S(i) < S(j_0)\) for every \(i\) satisfying \(i_0 < i < j_0\), implying

\[
S[i_0, j_0] = S(j_0) - S(i_0) > S(j) - S(i) = S[i, j]
\]

for every \([i, j] \subseteq [i_0, j_0]\), i.e., \([i_0, j_0]\) has the subsegment property for \(S\).

Together, Theorems 2 and 3 imply Theorem 1.

Appendix B

ROC curves comparing RepSeek and RepWords for different gap penalties. Figure B1 displays ROC curves for SINE repeats; Figure B2, LINE repeats; Figure B3, LTR repeats; Figure B4, DNA repeats; Figure B5, satellite repeats; and Figure B6, miscellaneous (all other) repeats. All repeats are classified according to RepeatMasker. The black curves show results for RepSeek with its default settings. RepWords (with the word-length \(W_{\text{max}} = 200\)) used the standard BLASTN match-mismatch \(s(a, b) = 2\) for \(a = b\) and \(s(a, b) = -3\) for \(a \neq b\) with a variety of gap penalties: 

- \(g(k) = \infty\) (solid red), so repeats are ungapped;
- \(g(k) = 5 + k\) (solid orange);
- \(g(k) = 5 + 2k\) (dashed orange);
- \(g(k) = 3 + k\) (solid green);
- \(g(k) = 3 + 2k\) (dashed green);
- \(g(k) = 1 + k\) (blue);
- \(g(k) = 1 + 2k\) (dashed blue);
- \(g(k) = k\) (solid purple); and
- \(g(k) = 2k\) (dashed purple).

Figure B1 ROC curves comparing RepSeek and RepWords for different gap penalties – SINE repeats
Figure B2 ROC curves comparing RepSeek and RepWords for different gap penalties – LINE repeats

Figure B3 ROC curves comparing RepSeek and RepWords for different gap penalties – LTR repeats
Searching for repeats

Figure B4 ROC curves comparing RepSeek and RepWords for different gap penalties – DNA repeats

Figure B5 ROC curves comparing RepSeek and RepWords for different gap penalties – satellite repeats
Figure B6 ROC curves comparing RepSeek and RepWords for different gap penalties – miscellaneous repeats