genotypst: A bioinformatics Typst package for sequence analysis and visualization

`genotypst` is a bioinformatics package for Typst that enables analysis and visualization of biological data. It provides functionality for parsing FASTA and Newick files and generating publication-ready visualizations, including multiple sequence alignments, sequence logos, genome maps, and phylogenetic trees.

Contents

Working with sequence data

`genotypst` provides functions to parse sequence data and produce different visualizations.

Loading data

The `parse-fasta` function reads FASTA data and returns a dictionary mapping sequence identifiers to their corresponding sequences.

```
#let sequences = parse-fasta(read("/docs/data/dna.fna"))
```

```
(
  seq_1: "AAGGGACACTGATTTTCTCCCACAGCTGGGCCGTGGACCGTAGTGTTTCAAGAAGCCCACACAC",
  seq_2: "GCAATGGAGACAACATAGCCAACTACCTACTAGATGGCCTAGATCTGCCGCA",
  seq_3: "GGAACTGGCGTTACAGACAGTTGTGAGCCACCACATGGGCCTGGGATTTAAATATTATAAAGCTCCTC",
)
```

FASTA rendering

Use `render-fasta` to display sequences in the standard FASTA format.

```
#context {
  set text(size: 0.8em)
  render-fasta(sequences, max-width: 50)
}
```

```
>seq_1
AAGGGACACTGATTTTCTCCCACAGCTGGGCCGTGGACCGTAGTGTTTCA
AGAAGCCCACACAC
>seq_2
GCAATGGAGACAACATAGCCAACTACCTACTAGATGGCCTAGATCTGCCG
CA
>seq_3
GGAACTGGCGTTACAGACAGTTGTGAGCCACCACATGGGCCTGGGATTTA
AATATTATAAAGCTCCTC
```

In this example, `max-width` controls how many characters appear per line (default is 60).

Pairwise sequence alignment

Pairwise alignment is a method for comparing biological sequences, allowing quantification of sequence similarity and identification of evolutionary relationships between the residues of two sequences. `genotypst` supports both global alignment (end-to-end)[1] and local alignment (best-matching subsequences)[2], using dynamic programming with a user-defined scoring scheme (match/mismatch or a substitution matrix) and gap penalties. The `align-seq-pair` function performs pairwise alignment using either match/mismatch scores or a substitution matrix (see the scoring matrices section below).

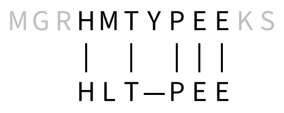
```
#let dna_pair_alignment = align-seq-pair(
  "ACT",
  "ACGT",
  match-score: 3,
  mismatch-score: -1,
  gap-penalty: -2,
  mode: "global",
)

#let protein_pair_alignment = align-seq-pair(
  "MAVHLTPEEKS",
  "HLTPEE",
  matrix: "BLOSUM62",
  gap-penalty: -4,
  mode: "local",
)
```

The DNA example uses a custom match/mismatch model for a global alignment, while the protein example uses BLOSUM62 for a local alignment. The alignment scores are 7 and 28.

Rendering pairwise alignments

Pairwise alignments can be rendered using the `render-pair-alignment` function. The example below uses the local protein alignment from the previous section, showing the unaligned regions using a light gray color.

```
// Because there may be multiple optimal alignments, `traceback-paths`
// is an array. We use `.at(0)` to visualize the path of the first alignment.
#render-pair-alignment(
  protein_pair_alignment.seq-1,
  protein_pair_alignment.seq-2,
  protein_pair_alignment.traceback-paths.at(0),
  unaligned-color: luma(75%),
)
```



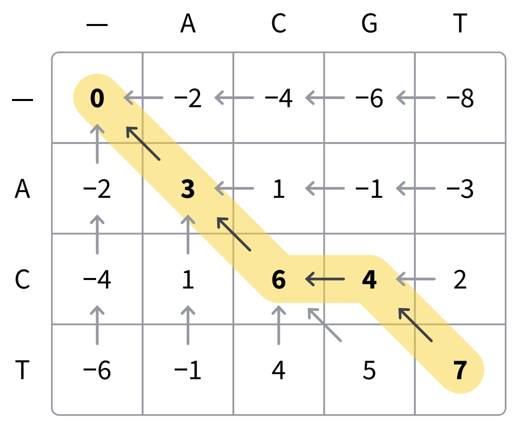
```
MGRHMTYPEEKS
   | |  |||
   HLT—PEE
```

Figure 1. Local protein alignment with unaligned regions shown in light gray.

Dynamic programming matrix visualization

Dynamic programming is the core procedure used by the pairwise alignment algorithm: it fills a matrix of optimal scores for all prefix pairs of the two sequences, where each cell stores the best score achievable at that position and arrows indicate the traceback directions that can lead to an optimal alignment. The `render-dp-matrix` function renders the DP matrix of a given alignment, overlaying the traceback path used to produce the final alignmkent.

```
#render-dp-matrix(
  dna_pair_alignment.seq-1,
  dna_pair_alignment.seq-2,
  dna_pair_alignment.dp-matrix.values,
  path: dna_pair_alignment.traceback-paths.at(0),
  arrows: dna_pair_alignment.dp-matrix.arrows,
)
```



$$\begin{array}{c|ccccc} & — & A & C & G & T \\ \hline — & \mathbf{0} & -2 & -4 & -6 & -8 \\ A & -2 & \mathbf{3} & 1 & -1 & -3 \\ C & -4 & 1 & \mathbf{6} & \mathbf{4} & 2 \\ T & -6 & -1 & 4 & 5 & \mathbf{7} \end{array}$$

Figure 2. Dynamic programming matrix for the DNA alignment, with the optimal path highlighted.

Scoring matrices

Substitution matrices assign scores for aligning residues in pairwise and multiple sequence alignments, rewarding likely substitutions and penalizing unlikely ones. They represent substitution preferences as log-odds scores derived from observed evolutionary changes. In this way, they guide alignment algorithms by quantifying how plausible it is for one residue to replace another over time, helping produce biologically meaningful alignments. `genotypst` provides scoring matrices from the BLOSUM[3] and PAM[4] families for protein sequences, as well as the EDNAFULL matrix for DNA and RNA sequences.

Use `get-scoring-matrix` to retrieve a matrix data and `get-score-from-matrix` to query scores for specific residue pairs.

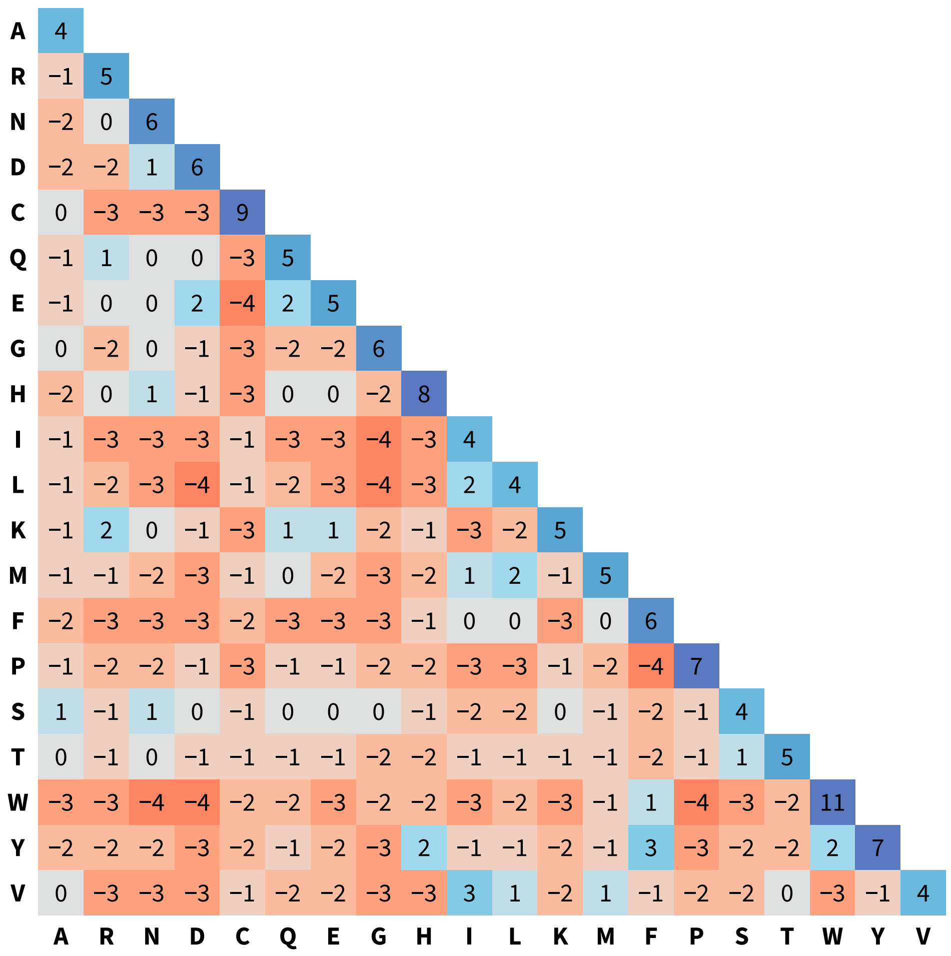
```
#let blosum62 = get-scoring-matrix("BLOSUM62")
#let ar_score = get-score-from-matrix(blosum62, "L", "D")
#text[The substitution score for L vs D in BLOSUM62 is: #ar_score.]
```

The substitution score for L vs D in BLOSUM62 is: −4.

You can render the entire scoring matrix using `render-scoring-matrix` function. In the example below:

- `context { set text(size: 0.9em) ... }` reduces the size of the text.
- `scale-limit: 7` set the color scale limits to −7 to 7.

```
#context {
  set text(size: 0.9em)
  render-scoring-matrix(
    blosum62,
    scale-limit: 7,
  )
}
```



A	4																			
R	−1	5																		
N	−2	0	6																	
D	−2	−2	1	6																
C	0	−3	−3	−3	9															
Q	−1	1	0	0	−3	5														
E	−1	0	0	2	−4	2	5													
G	0	−2	0	−1	−3	−2	−2	6												
H	−2	0	1	−1	−3	0	0	−2	8											
I	−1	−3	−3	−3	−1	−3	−3	−4	−3	4										
L	−1	−2	−3	−4	−1	−2	−3	−4	−3	2	4									
K	−1	2	0	−1	−3	1	1	−2	−1	−3	−2	5								
M	−1	−1	−2	−3	−1	0	−2	−3	−2	1	2	−1	5							
F	−2	−3	−3	−3	−2	−3	−3	−3	−1	0	0	−3	0	6						
P	−1	−2	−2	−1	−3	−1	−1	−2	−2	−3	−3	−1	−2	−4	7					
S	1	−1	1	0	−1	0	0	0	−1	−2	−2	0	−1	−2	−1	4				
T	0	−1	0	−1	−1	−1	−1	−2	−2	−1	−1	−1	−1	−2	−1	1	5			
W	−3	−3	−4	−4	−2	−2	−3	−2	−2	−3	−2	−3	−1	1	−4	−3	−2	11		
Y	−2	−2	−2	−3	−2	−1	−2	−3	2	−1	−1	−2	−1	3	−3	−2	−2	2	7	
V	0	−3	−3	−3	−1	−2	−2	−3	−3	3	1	−2	1	−1	−2	−2	0	−3	−1	4
	A	**R**	**N**	**D**	**C**	**Q**	**E**	**G**	**H**	**I**	**L**	**K**	**M**	**F**	**P**	**S**	**T**	**W**	**Y**	**V**

Figure 3. BLOSUM62 scoring matrix.

Multiple sequence alignments

The `render-msa` function displays multiple sequence alignments with optional residue coloring and conservation bars.

In the example below:

- `colors: true` enables residue coloring based on biochemical properties.
- `conservation: true` adds conservation bars above the alignment.
- `start: 100` and `end: 145` limit the display to a specific region of interest (residues 100 to 145).

```
#let protein_msa = parse-fasta(read("/docs/data/msa.afa"))

#context {
  set text(size: 0.8em)
  render-msa(
    protein_msa,
    start: 100,
    end: 145,
    colors: true,
    conservation: true,
  )
}
```

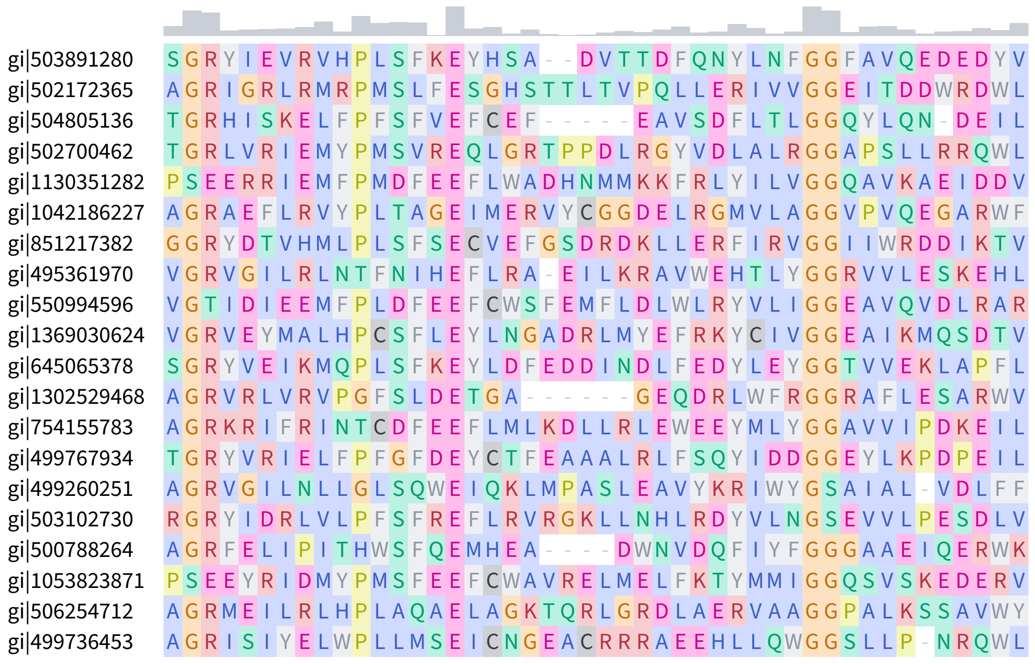

gi|503891280 SGRYIEVRVHPLSFKEYHSA--DVTTDFQNYLNFGGFAVQEDEDYV
gi|502172365 AGRIGRLRMRPMSLFESGHSTTLTVPQLLERIVVGGEITDDWRDWL
gi|504805136 TGRHISKELFPFSFVEFCEF-----EAVSDFLTLGGQYLQN-DEIL
gi|502700462 TGRLVRIEMYPMSVREQLGRTPPDLRGYVDLALRGGAPSLLRRQWL
gi|1130351282 PSEERRIEMFPMDFEEFLWADHNMMKKFRLYILVGGQAVKAEIDDV
gi|1042186227 AGRAEFLRVYPLTAGEIMERVYCGGDELRGMVLAGGVPVQEGARWF
gi|851217382 GGRYDTVHMLPLSFSECVEFGSDRDKLLERFIRVGGIIWRDDIKTV
gi|495361970 VGRVGILRLNTFNIHEFLRA-EILKRAVWEHTLYGGRVVLESKEHL
gi|550994596 VGTIDIEEMFPLDFEEFCWSFEMFLDLWLRYVLIGGEAVQVDLRAR
gi|1369030624 VGRVEYMALHPCSFLEYLNGADRLMYEFRKYCIVGGEAIKMQSDTV
gi|645065378 SGRYVEIKMQPLSFKEYLDFEDDINDLFEDYLEYGGTVVEKLAPFL
gi|1302529468 AGRVRLVRVPGFSLDETGA------GEQDRLWFRGGRAFLESARWV
gi|754155783 AGRKRIFRINTCDFEEFLMLKDLLRLEWEEYMLYGGAVVIPDKEIL
gi|499767934 TGRYVRIELFPFGFDEYCTFEAAALRLFSQYIDDGGEYLKPDPEIL
gi|499260251 AGRVGILNLLGLSQWEIQKLMPASLEAVYKRIWYGSAIAL-VDLFF
gi|503102730 RGRYIDRLVLPFSFREFLRVRGKLLNHLRDYVLNGSEVVLPESDLV
gi|500788264 AGRFELIPITHWSFQEMHEA----DWNVDQFIYFGGGAAEIQERWK
gi|1053823871 PSEEYRIDMYPMSFEEFCWAVRELMELFKTYMMIGGQSVSKEDERV
gi|506254712 AGRMEILRLHPLAQAELAGKTQRLGRDLAERVAAGGPALKSSAVWY
gi|499736453 AGRISIYELWPLLMSEICNGEACRRRAEEHLLQWGGSLLP-NRQWL

Figure 4. MSA visualization for positions 100–145, with residue coloring and conservation bars enabled.

Residue coloring represents amino acid physicochemical properties. The sequence alphabet (amino acid, DNA, or RNA) is determined automatically and a suitable color palette is applied.

The bars above the alignment indicate the degree of conservation at each column.

Sequence logos

Sequence logos[5] summarize conservation patterns within a sequence alignment and are commonly used to visualize binding sites, motifs, and functional domains. In a sequence logo, the total height of each stack represents the information content (in bits) at that position, while the height of individual letters reflects their relative frequencies.

In the example below, we visualize the same region as the MSA of the previous section.

```
#render-sequence-logo(protein_msa, start: 100, end: 145)
```

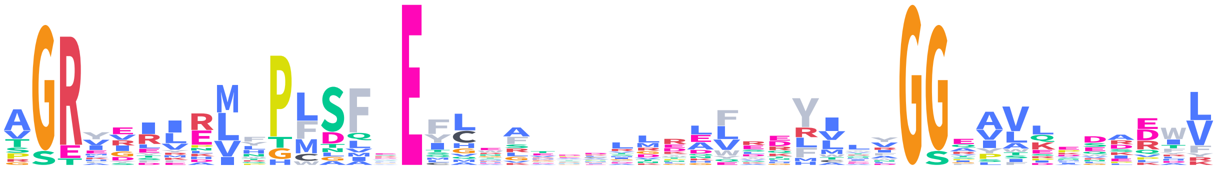


Figure 5. Sequence logo for positions 100–145, showing conservation and residue frequency.

Like `render-msa`, `render-sequence-logo` automatically applies the appropriate color palette based on the sequence alphabet.

Color palettes

`genotypst` uses predefined color palettes to assign colors to sequence residues.

Amino acid palette

Amino acids are colored according to their physicochemical properties. Grouping residues by color helps reveal the chemical nature of conserved positions (e.g., whether a position is consistently hydrophobic or charged), which is often important for understanding protein structure, function, and evolution.

Hydrophobic

Alanine	Ala	A	#4d78ff
Histidine	His	H	#4d78ff
Isoleucine	Ile	I	#4d78ff
Leucine	Leu	L	#4d78ff
Methionine	Met	M	#4d78ff
Valine	Val	V	#4d78ff

Polar

Serine	Ser	S	#00c990
Threonine	Thr	T	#00c990
Glutamine	Gln	Q	#00c990
Asparagine	Asn	N	#00c990

Aromatic

Phenylalanine	Phe	F	#bac1d2
Tryptophan	Trp	W	#bac1d2
Tyrosine	Tyr	Y	#bac1d2

Negatively charged

Aspartic acid	Asp	D	#ff07b8
Glutamic acid	Glu	E	#ff07b8

Positively charged

Lysine	Lys	K	#e44356
Arginine	Arg	R	#e44356

Cysteine

Cysteine	Cys	C	#494e5b

Glycine

Glycine	Gly	G	#f59116

Proline

Proline	Pro	P	#d9de09

Nucleic acid palettes

The DNA and RNA palettes assign a distinct color to each nucleotide.

DNA palette		
Adenine	A	#00c990
Cytosine	C	#4d78ff
Guanine	G	#ff07b8
Thymine	T	#f59116

RNA palette		
Adenine	A	#00c990
Cytosine	C	#4d78ff
Guanine	G	#ff07b8
Uracil	U	#f59116

Visualizing genomic loci with genome maps

Genome maps enable visualization of the genes and other genomic elements within a locus, highlighting their order, orientation, and length. `genotypst` provides a `render-genome-map` function that produces a genome map from an array of dictionaries, each representing a genomic feature that will be plotted:

- `start` (required): Start coordinate (1-indexed integer).
- `end` (required): End coordinate (1-indexed integer).
- `strand`: Feature orientation (`1` or `"+"` for the positive strand, `-1` or `"-"` for the negative strand). `none` draws an undirected block.
- `label`: Feature label
- `color`: Fill color.

```
#let f_plasmid_locus = (
  (start: 65556, end: 66065, strand: -1, label: [_ygfA_]),
  (start: 66118, end: 66407, label: [_oriT_], color: rgb("#696975")),
  (start: 66479, end: 66862, strand: 1, label: [_traM_], color: rgb("#62B9F2")),
  (start: 66977, end: 67055, strand: -1, label: [_finP_]),
  (start: 67049, end: 67738, strand: 1, label: [_traJ_], color: rgb("#F7ED6C")),
  (start: 67837, end: 68232, strand: 1, label: [_traY_]),
  (start: 68265, end: 68630, strand: 1, label: [_traA_]),
  (start: 68645, end: 68956, strand: 1, label: [ _traL_]),
  (start: 68978, end: 69544, strand: 1, label: [_traE_]),
)

#render-genome-map(
  f_plasmid_locus,
  coordinate-axis: true,
  unit: "bp",
)
```

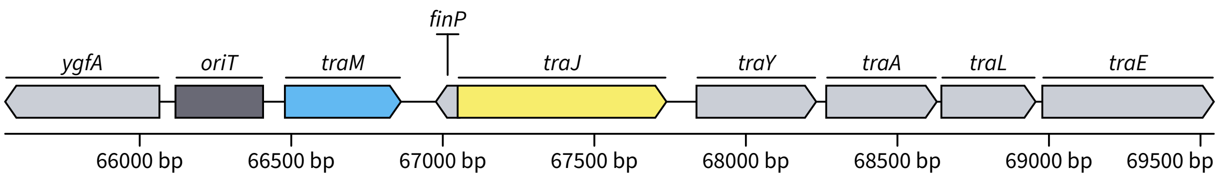


Figure 6. Genome map showing the genes within the 65,556–69,544 bp region of the F plasmid of *Escherichia coli* K-12 (GenBank: AP001918.1).

Working with phylogenetic trees

`genotypst` includes functions to parse and render phylogenetic trees. Trees can be created by parsing Newick-formatted strings with `parse-newick` or by manually constructing nested dictionary structures.

```
#let parsed_newick_tree = parse-newick(
  "(('Leaf A':0.2,'Leaf B':0.1)'Internal node':0.3,'Leaf C':0.6)Root;"
)

#let manual_tree = (
  rooted: true,
  name: "Root",
  length: none,
  children: (
    (
      name: "Internal node",
      length: 0.3,
      children: (
        (name: "Leaf A", length: 0.2, children: none),
        (name: "Leaf B", length: 0.1, children: none),
      ),
    ),
    (name: "Leaf C", length: 0.6, children: none),
  ),
)
```

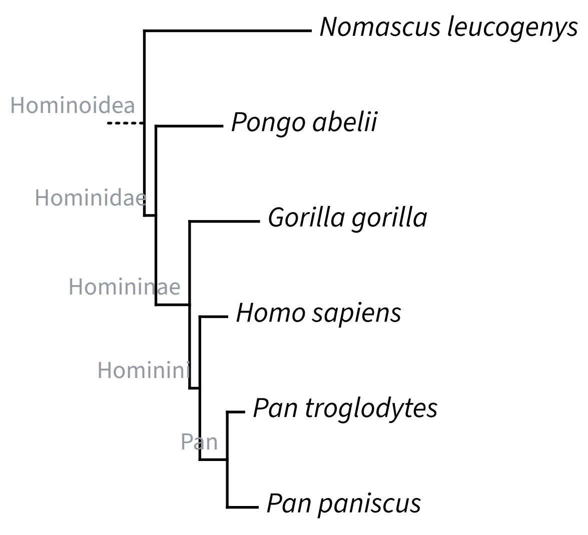
Visualizing trees

`genotypst` can produce visualizations of phylogenetic trees. To illustrate this, we will read and render a Newick file containing a phylogeny of the *Hominoidea* superfamily, which was extracted from the Ensembl Compara species tree[6].

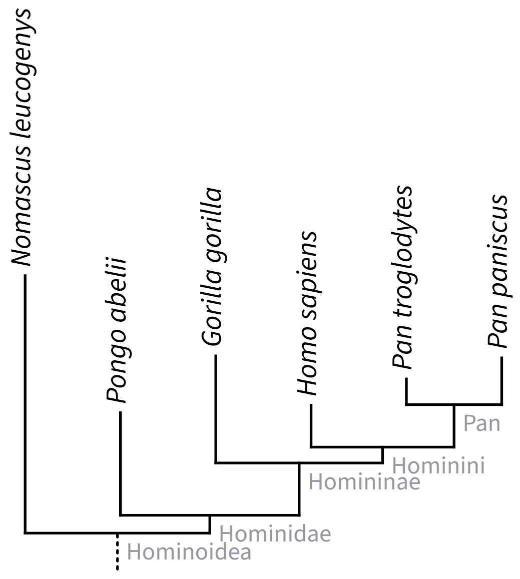
```
#let hominoidea_tree = parse-newick(read("/docs/data/hominoidea.nwk"))
```

To render the tree, use the `render-tree` function. By default, it produces a horizontal rectangular tree, but a vertical layout can be specified using the `orientation: "vertical"` option.

```
#render-tree(
  hominoidea_tree,
  tip-label-italics: true,
  orientation: "horizontal",
)

#render-tree(
  hominoidea_tree,
  tip-label-italics: true,
  orientation: "vertical",
)
```



Tree with horizontal orientation



Tree with vertical orientation

Customizing visualizations

Font selection

By default, the visualizations produced by `genotypst` are rendered using the default document font. To control font size or font family for these visualizations, wrap the rendering function in a `context` block and set the desired text properties.

```
#let dna_msa = (
  "seq1": "AGTCTCAAGATAACTTTCGAAACAAAC",
  "seq2": "AGTTTCCAAGTGGATTTGGAATTGAAC",
  "seq3": "ACTCT-CGGATGGATTCGGATACAAAC",
  "seq4": "AGTCT---GATTGATGTGGATACAAAC",
  "seq5": "AGTCT--GGGTGGATTTGG-AACAAAT",
  "seq6": "CAGTGCTCCCTGGTGGTGG-ACCATCT",
  "seq7": "AGTCTCAAGACGGATACTG--ATGCCC",
)

#context {
  set text(font: "Maple Mono", size: 0.76em)
  render-msa(dna_msa)
}
```

```
seq1  A G T C T C A A G A T A A C T T T C G A A A C A A A C
seq2  A G T T T C C A A G T G G A T T T G G A A T T G A A C
seq3  A C T C T - C G G A T G G A T T C G G A T A C A A A C
seq4  A G T C T - - - G A T T G A T G T G G A T A C A A A C
seq5  A G T C T - - G G G T G G A T T T G G - A A C A A A T
seq6  C A G T G C T C C C T G G T G G T G G - A C C A T C T
seq7  A G T C T C A A G A C G G A T A C T G - - A T G C C C
```

Default document font

```
seq1  AGTCTCAAGATAACTTTCGAAACAAAC
seq2  AGTTTCCAAGTGGATTTGGAATTGAAC
seq3  ACTCT-CGGATGGATTCGGATACAAAC
seq4  AGTCT---GATTGATGTGGATACAAAC
seq5  AGTCT--GGGTGGATTTGG-AACAAAT
seq6  CAGTGCTCCCTGGTGGTGG-ACCATCT
seq7  AGTCTCAAGACGGATACTG--ATGCCC
```

Custom font (Maple Mono)

```
#context {
  show text: set text(font: "Libertinus Serif")
  render-pair-alignment(
    protein_pair_alignment.seq-1,
    protein_pair_alignment.seq-2,
    protein_pair_alignment.traceback-paths.at(0),
  )
}
```

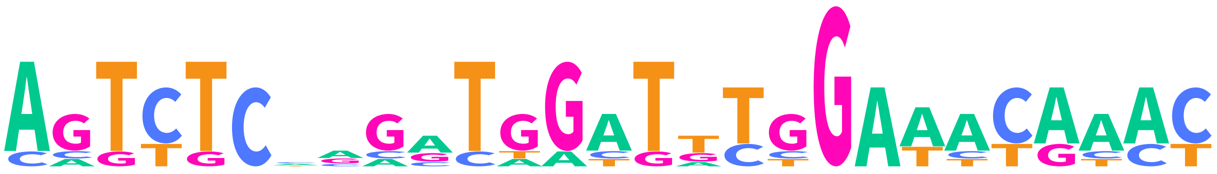
```
MGRHMTYPEEKS
   | | |||
   HLT—PEE
```

Default document font

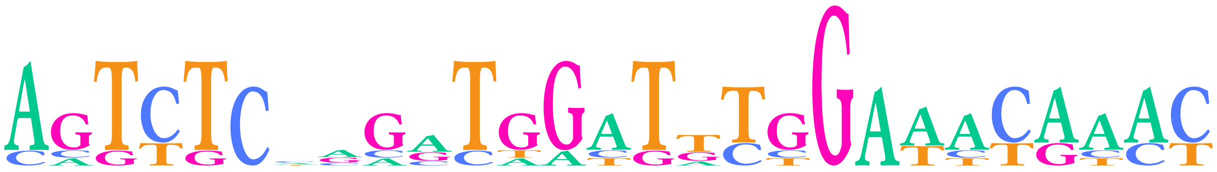
```
M G R H M T Y P E E K S
      |   |   | | |
      H L T — P E E
```

Custom font (Libertinus Serif)

```
#context {
  show text: set text(font: "Libertinus Serif")
  render-sequence-logo(dna_msa)
}
```

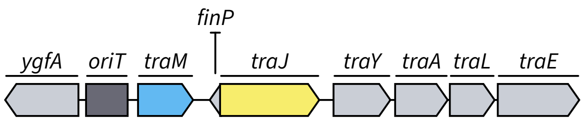


Default document font

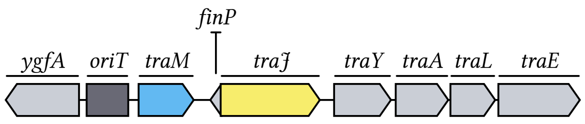


Custom font (Libertinus Serif)

```
#context {
  show text: set text(font: "Libertinus Serif")
  render-genome-map(f_plasmid_locus)
}
```

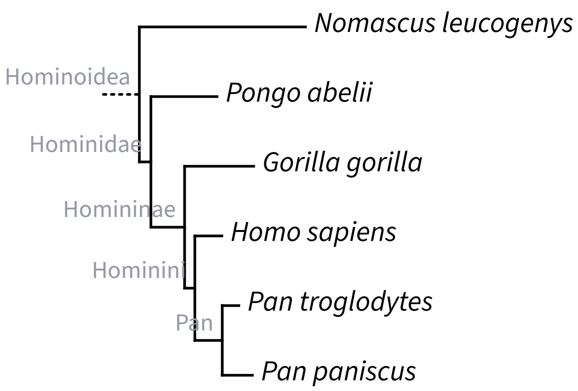


Default document font

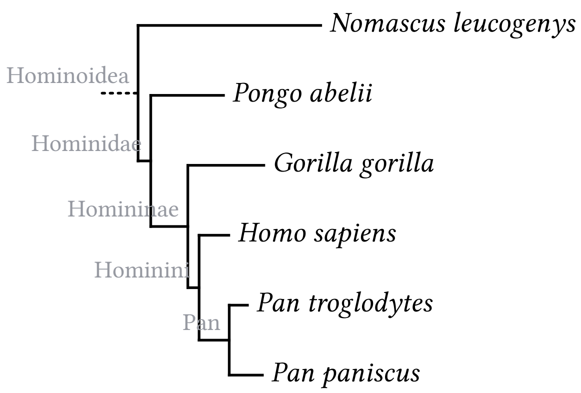


Custom font (Libertinus Serif)

```
#let hominoidea_tree = parse-newick(read("/docs/data/hominoidea.nwk"))
#context {
  show text: set text(font: "Libertinus Serif")
  #render-tree(hominoidea_tree, tip-label-italics: true)
}
```



Default document font



Custom font (Libertinus Serif)

Specifying residue palettes

If you want to use an alternative palette to color residues, you can provide a dictionary to the `palette` parameter of `render-msa` and `render-sequence-logo`. The provided palette can be either custom or one provided by `genotypst` under `residue-palette`.

The following palettes are available:

- **Protein:** `default` (8 colors), `dayhoff` (6), `zappo` (7), `takabatake4` (4), `takabatake5` (5), `takabatake6` (6), `takabatake7` (7), and `takabatake8` (8)[7].
- **DNA:** `default` (4)
- **RNA:** `default` (4)

For example, to use the Dayhoff amino acid color palette in a sequence logo:

```
#render-sequence-logo(
  protein_msa,
  start: 100,
  end: 145,
  palette: residue-palette.aa.dayhoff,
)
```

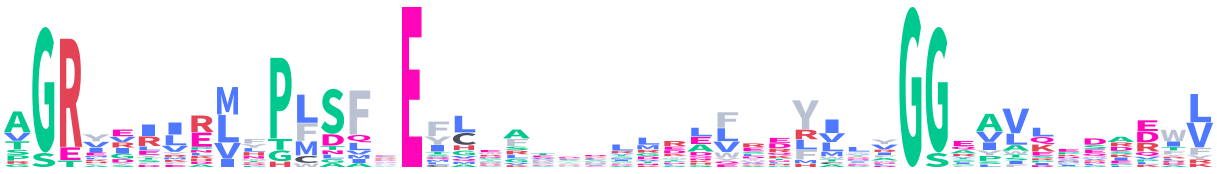


Figure 19. Sequence logo for positions 100–145 using the Dayhoff amino acid color palette.

Bibliography

1. Needleman, S. B. & Wunsch, C. D. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology* **48,** 443–453 (1970).
2. Smith, T. & Waterman, M. Identification of common molecular subsequences. *Journal of Molecular Biology* **147,** 195–197 (1981).
3. Henikoff, S. & Henikoff, J. G. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences* **89,** 10915–10919 (1992).
4. Dayhoff, M., Schwartz, R. & Orcutt, B. A model of evolutionary change in proteins. *Atlas of protein sequence and structure* (1979).
5. Schneider, T. D. & Stephens, R. Sequence logos: a new way to display consensus sequences. *Nucleic Acids Research* **18,** 6097–6100 (1990).
6. Herrero, J. *et al.* Ensembl comparative genomics resources. *Database* **2016,** bav96 (2016).
7. Takabatake, K. *et al.* Improved Large-Scale Homology Search by Two-Step Seed Search Using Multiple Reduced Amino Acid Alphabets. *Genes* **12,** 1455 (2021).