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, RNA secondary structures, 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-line-length: 50)
}
```

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

In this example, `max-line-length` controls how many residues are shown 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 Scoring matrices).

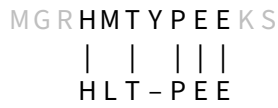
```
#let dna_pair_alignment = align-seq-pair(
  "ACT",
  "ACGT",
  match-score: 3,
  mismatch-score: -1,
  gap-penalty: -2,
  mode: "global",
)

#let protein_pair_alignment = align-seq-pair(
  "MAVHLTPEEKS",
  "HLTPEE",
  scoring-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%),
)
```

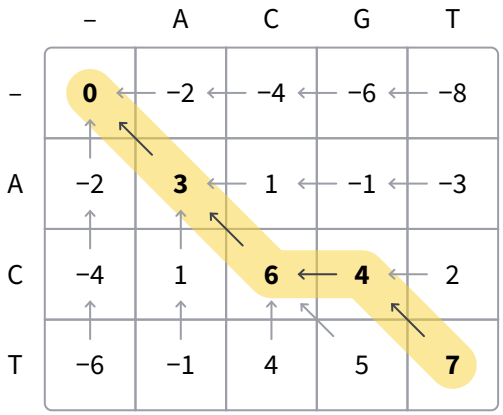


Local protein alignment with unaligned regions shown in light gray.

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 alignment.

```
// Pass traceback path directly from align-seq-pair output (end-to-start).
#render-dp-matrix(
  dna_pair_alignment.seq-1,
  dna_pair_alignment.seq-2,
  cell-values: dna_pair_alignment.dp-matrix.scores,
  path: dna_pair_alignment.traceback-paths.at(0),
  arrows: dna_pair_alignment.dp-matrix.arrows,
)
```



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.

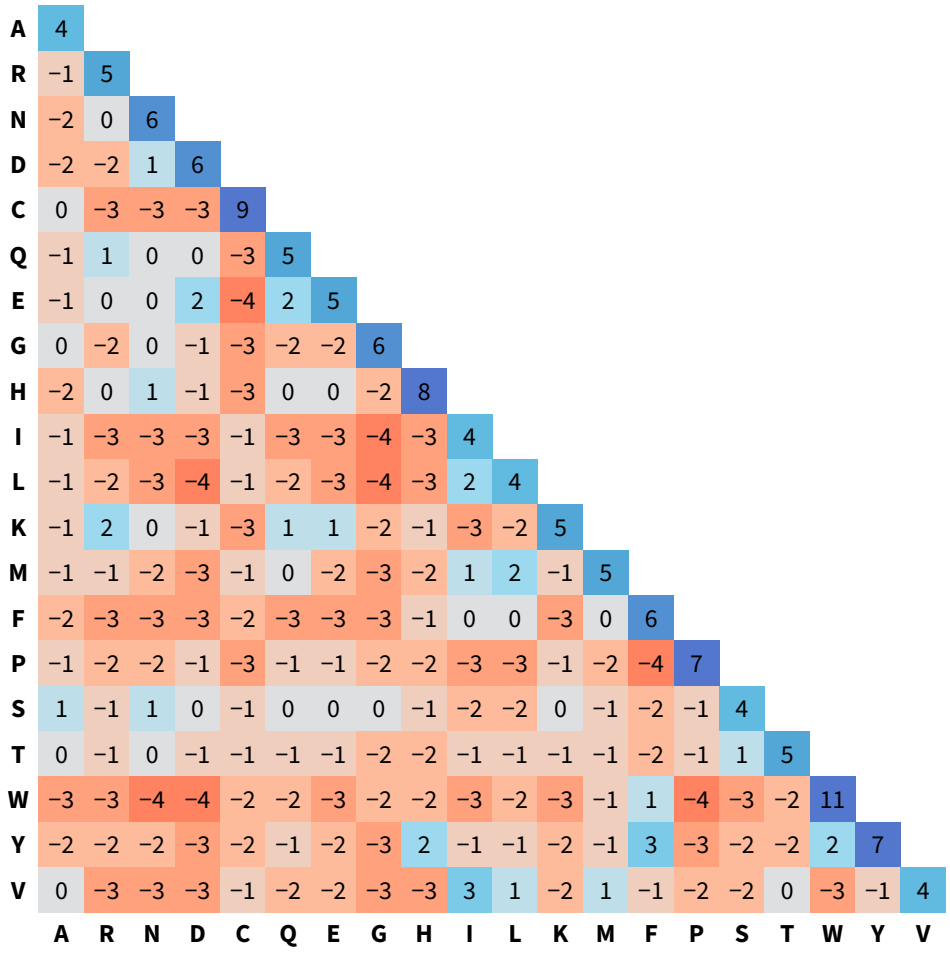
Use `get-scoring-matrix` to retrieve a matrix data and `get-score-from-matrix` to query scores for specific residue pairs.

```
#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 `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,
  )
}
```



BLOSUM62 scoring matrix.

Multiple sequence alignments

The `render-msa` function displays multiple sequence alignments and can optionally color residues, show the consensus sequence, and display residue conservation.

In the example below:

- `colors: true` enables residue coloring based on biochemical properties.
- `show-conservation: true` adds bars above the alignment columns to indicate the residue conservation at each position of 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,
    show-conservation: true,
  )
}
```

MSA visualization for positions 100–145, with residue coloring, and bars indicating residue conservation.

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.

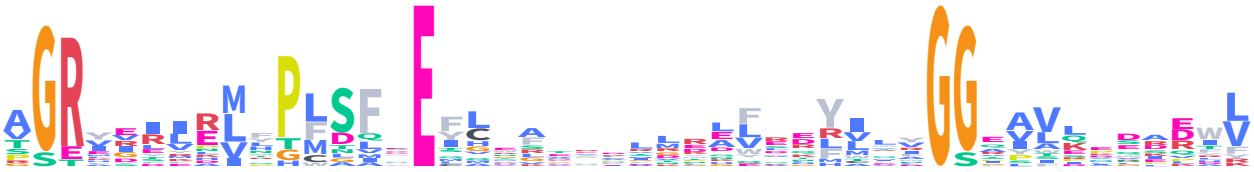
The bars above the alignment indicate the degree of residue conservation at each column.

Sequence logos

Sequence logos[5] are graphical summaries of residue variation and conservation across positions in 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 individual letters are scaled according to 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)
```



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

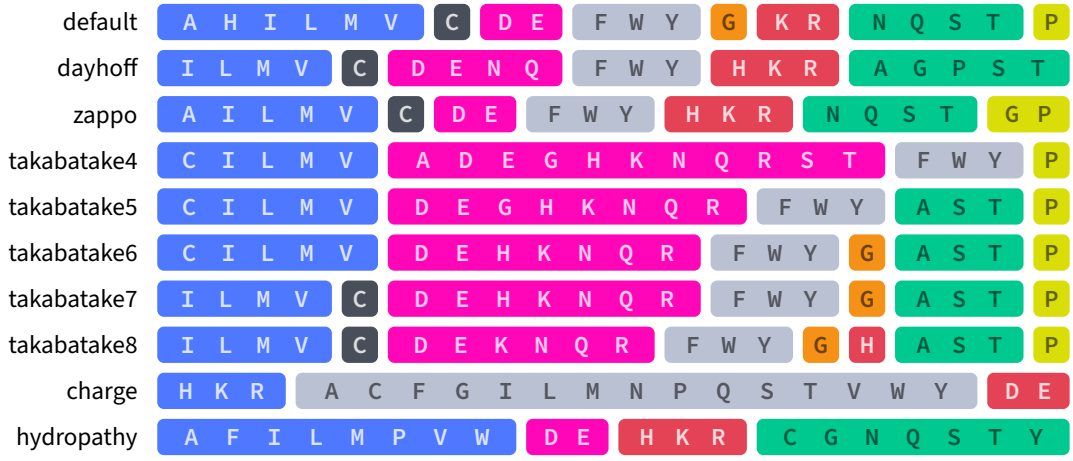
Like `render-msa`, `render-sequence-logo` automatically applies the appropriate color palette based on the sequence alphabet.

Residue palettes

`genotypst` uses predefined color palettes to assign colors to sequence residues. These palettes can be used to customize residue colors across different visualizations, as described in Specifying residue palettes.

Amino acid palettes

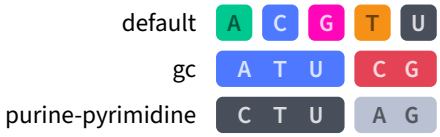
Amino acids are colored according to their physicochemical properties. Grouping residues by color helps reveal the chemical nature of conserved positions (e.g., whether a position is consistently hydrophobic or charged), which is often important for understanding protein structure, function, and evolution.



The ten amino acid palettes available in `genotypst`. The labels on the left indicate the palette name, letters within the colored boxes indicate the amino acids, and the colors represent their color in the palette.

Nucleic acid palettes

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



The four nucleic acid palettes available in `genotypst`. The labels on the left indicate the palette name, letters within the colored boxes indicate the nucleotides, and the colors represent their color in the palette.

RNA secondary structures

`genotypst` can predict RNA secondary structures from a sequence and render them. The `predict-rna-structure` function takes in a sequence and returns a dictionary containing the predicted structure in dot-bracket notation and scoring metadata.

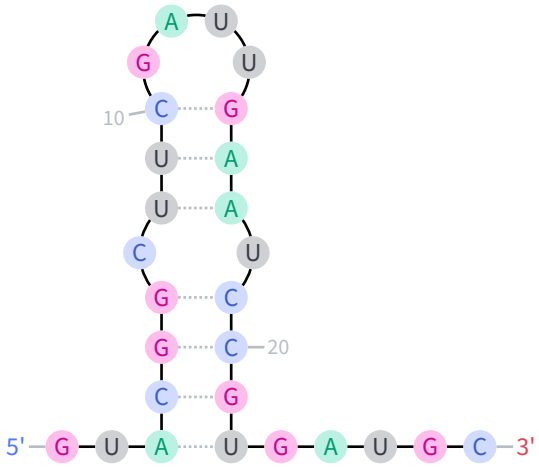
```
#let sequence = "GUACGGCUUCGAUUGAAUCCGUGAUGC"
#let rna-structure-prediction = predict-rna-structure(sequence)
#raw(repr(rna-structure-prediction), block: true)
```

```
(
  structure: "..(((((.(((....))).))))).....",
  score: (model: "ContraFold", value: 0.74309313923),
)
```

To draw the predicted structure, pass the sequence and the returned dot-bracket structure to `render-rna-structure`. In the example below:

- `layout: "rna_puzzler"` selects the RNApuzzler[6] layout algorithm.
- `show-nucleotide-circles: true` draws a circle around each nucleotide.
- `palette: residue-palette.rna.default` colors the nucleotides with the default RNA palette.
- `position-interval: 10` adds position ticks at intervals of 10 nucleotides.
- `show-terminal-labels: true` adds labels for the 5′ and 3′ ends.

```
#render-rna-structure(
  sequence,
  rna-structure-prediction.structure,
  width: 70mm,
  layout: "rna_puzzler",
  show-nucleotide-circles: true,
  palette: residue-palette.rna.default,
  position-interval: 10,
  show-terminal-labels: true,
)
```



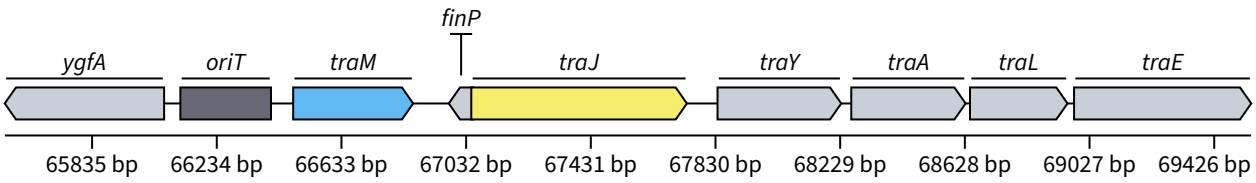
RNA secondary structure with colored nucleotides, position ticks, and terminal labels.

Rendering genome maps and parsing GFF3 files

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",
)
```



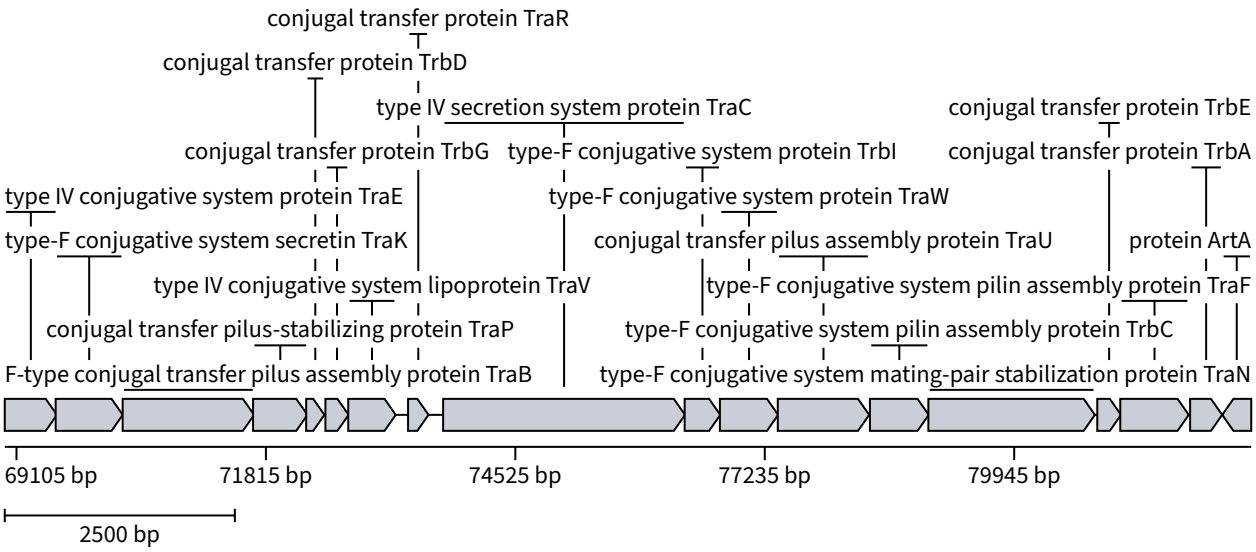
Genome map showing the genes within the 65,556–69,544 bp region of the F plasmid of *Escherichia coli* K-12 (GenBank: AP001918.1).

If your genomic features are stored in a GFF3 file, you can use the `parse-gff` function to parse the file and return an array of dictionaries suitable as input to `render-genome-map`.

In the example below:

- `feature-types`: Return only features of the specified types.
- `range`: Return only features within the specified genomic range, defined as `(accession, start, end)`.
- `exclude-partial`: Exclude genes that are not fully contained within the specified range.
- `label-attribute`: Use this feature attribute as the label in the genome map.

```
#let parsed_gff3_data = parse-gff(
  read("data/NC_002483.gff3"),
  feature-types: ("CDS", "pseudogene"),
  range: ("NC_002483.1", 68960, 82550),
  exclude-partial: true,
  label-attribute: "product",
)

#render-genome-map(
  parsed_gff3_data,
  coordinate-axis: true,
  scale-bar: true,
  unit: "bp",
)
```



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: text(weight: "bold")[Root],
  length: none,
  children: (
    (
      name: text(fill: orange)[Internal node],
      length: 0.3,
      children: (
        (name: text[_*Leaf* A_], length: 0.2, children: none),
        (name: "Leaf B", length: 0.1, children: none),
      ),
    ),
    (name: text(fill:blue)[Leaf C], length: 0.6, children: none),
  ),
)
```

Visualizing trees

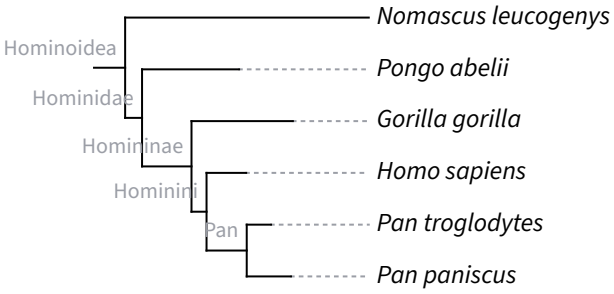
`genotypst` can produce visualizations of phylogenetic trees. To illustrate this, we will read and render a Newick file containing a phylogeny of the *Hominoidea* superfamily, which was extracted from the Ensembl Compara species tree[7].

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

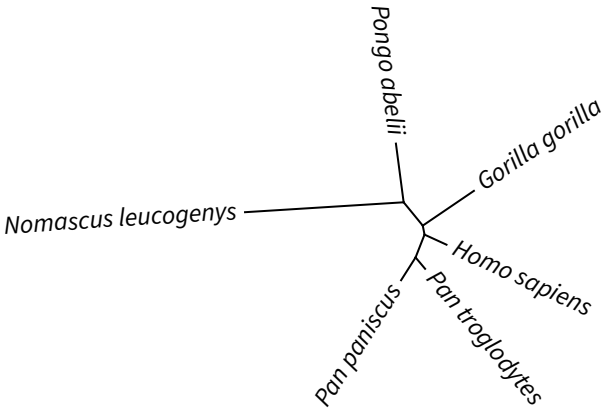
Two types of trees can be produced: rectangular trees (`render-rectangular-tree`) and unrooted trees (`render-unrooted-tree`). In the examples below:

- `tip-label-italics: true` renders tip labels in italics.
- `orientation: "horizontal"` sets the rectangular tree to be oriented horizontally. It can also be set to `"vertical"` for a vertical layout.
- `align-tip-labels: true` aligns tip labels and connects them to branches with dotted leader lines.
- `hide-internal-labels: true` hides internal node labels.
- `layout: "daylight"` applies the daylight layout algorithm to the unrooted tree. The default layout is `"equal-angle"`.

```
#context {
  set text(size: 0.9em)
  render-rectangular-tree(
    hominoidea_tree,
    tip-label-italics: true,
    orientation: "horizontal",
    align-tip-labels: true,
  )
  render-unrooted-tree(
    hominoidea_tree,
    tip-label-italics: true,
    hide-internal-labels: true,
    layout: "daylight",
  )
}
```



Horizontal rectangular tree.



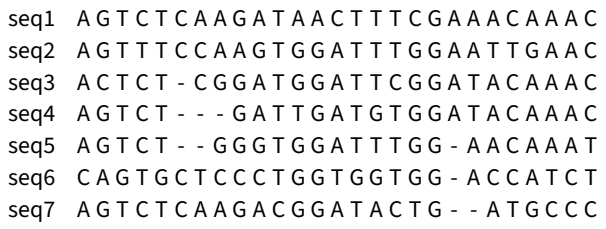
Unrooted tree using the daylight layout.

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.

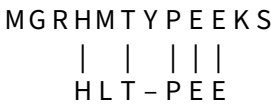
```
#let dna_msa = (
  "seq1": "AGTCTCAAGATAACTTTCGAAACAAAC",
  "seq2": "AGTTTCCAAGTGGATTTGGAATTGAAC",
  "seq3": "ACTCT-CGGATGGATTCGGATACAAAC",
  "seq4": "AGTCT---GATTGATGTGGATACAAAC",
  "seq5": "AGTCT--GGGTGGATTTGG-AACAAAT",
  "seq6": "CAGTGCTCCCTGGTGGTGG-ACCATCT",
  "seq7": "AGTCTCAAGACGGATACTG--ATGCCC",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

#context {
  set text(font: "Maple Mono", size: 0.785em)
  render-msa(dna_msa)
}
```



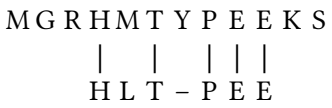
Default document font.

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),
  )
}
```



Default document font.



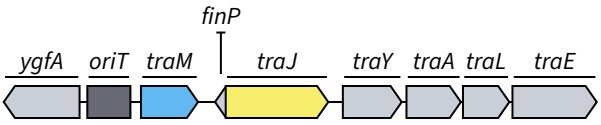
Custom font (Libertinus Serif).

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

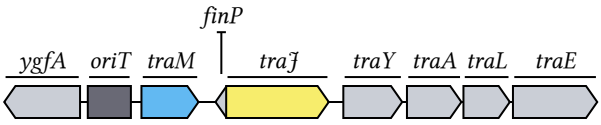
Default document font.

Custom font (Libertinus Serif).

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

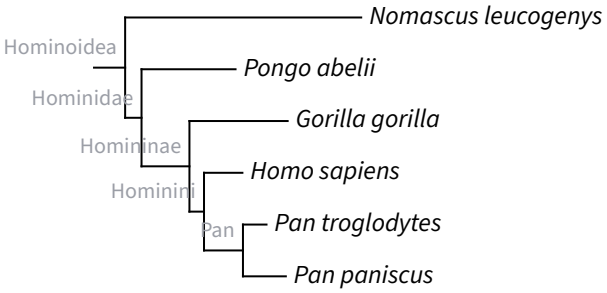


Default document font.

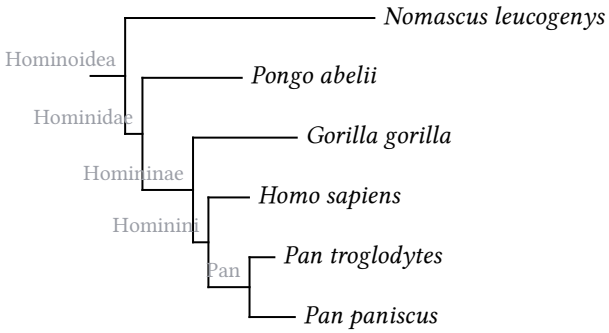


Custom font (Libertinus Serif).

```
#let hominoidea_tree = parse-newick(read("/docs/data/hominoidea.nwk"))
#context {
  show text: set text(font: "Libertinus Serif", size: 0.9em)
  #render-rectangular-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`.

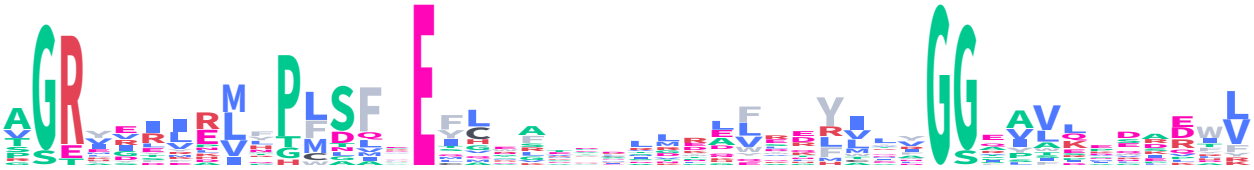
For example, to color residues in a MSA according to their charge, you can use the `charge` palette:

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

MSA visualization for positions 100–145 using the `charge` amino acid color palette.

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,
)
```



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. in *Atlas of protein sequence and structure* (National Biomedical Research Foundation, Washington, D.C, 1979).
5. Schneider, T. D. & Stephens, R. Sequence logos: a new way to display consensus sequences. *Nucleic Acids Research* **18,** 6097–6100 (1990).
6. Wiegreffe, D., Alexander, D., Stadler, P. F. & Zeckzer, D. RNApuzzler: efficient outerplanar drawing of RNA-secondary structures. *Bioinformatics* **35,** 1342–1349 (2019).
7. Herrero, J. *et al.* Ensembl comparative genomics resources. *Database* **2016,** bav96 (2016).