Instructions

1. First open **profit.nb** and evaluate all gray cells (Evaluation -> Evaluate Initialization Cells)
2. Then, on this file (**profit_nick.nb**), evaluate all initialization cells as well.
3. Now evaluate the example below by hand, by hitting shift+return on each input cell.

First example for Nick

Read the seqs from the TSS file. Don't forget the final semicolon to avoid getting flooded by a lot of output.

```
seqs = ReadSizeSeqsFromTSS[NotebookDirectory[] <> "sample.tss"];
```

```
Length[seqs]
```

Aligning 2500 traces will take a while and will be harder to visualize, so let's start by keeping just 100.

```
seqs = RandomSample[seqs, 100];
```

```
Length[seqs]
```

Build a color function and show the traces:

```
cf = MakeColorFunction[seqs];
```

```
PlotSeqs[seqs, cf]
```

This will call MAFFT with the given parameters and return an msa (multiple-sequence-alignment), which is just a big matrix of numbers with Nulls inserted between them. The msa matrix is stored as a list of rows.

```
msa = AlignSeqs[seqs, 1.5, 0.0, "--genafpair --maxiterate 1000 --thread 2"];
```

The msa is stored by rows:

```
PlotSeqs[msa, cf]
```

Let's now interactively chop this MSA. Click

```
ChopManually[seqs, msa, cf]
```

The result of chopping (stored in the chopResult variable when you click "Chop!") is now a "phases" object, i.e., a list of lists of lists. Each list is a row, each row is a list of chunks, and each chunk is a sequence of packet sizes.

Here's the number of traces:

```
Length[chopResult]
```

Number of phases in the first trace:

```
Length[chopResult[[1]]]
```

Number of packets in each of those phases:

```
Map[Length, chopResult[[1]]]
```

Now let's glue the result of chopping into one nice big block, so we can show it. This will pad each phase with Nulls on the right, and will also use a couple of Nulls as spacers between phases.

```
chopped = JoinPhases[chopResult];
```

```
PlotSeqs[chopped, cf]
```

That's a good start! You can now export those before/after images or save them to a separate notebook.
We will talk about automated chopping in the future. For now, let's gather interesting traces!
Let us know if you have any questions.

Some aux functions for Nick (definitions below)

Parse a TSS file, remove any marker packets and zero-byte packets, and make the sizes positive or negative based on their direction (whoever sent the first packet is assumed to be the "source" host).

Return a list of traces. Each trace is a sequence of packet sizes. We care about direction and order, but for now, we are ignoring the precise timing. We may add that later on. Let's align based on size/direction for now.

NOTE: To avoid re-parsing, this caches all parsed TSS files. If you want to re-parse a TSS file whose name has not changed but whose content has, try: ClearTssCache[]

```
ReadSizeSeqsFromTSS[tssPathname_] := Module[{},
  Print["Parsing TSS file"];
  {secs, ias} = ParseTssFileCaching[tssPathname];
  Print["Removing markers"];
  ias = Map[RemoveMarkersIA, ias];
  Print["Removing zero-byte packets"];
  ias = Map[RemoveZeroBytePacketsIA, ias];
  firstSrc = ias[[1]][[1]]["src"];
  Print["PosNeg-ing size using src=" <> firstSrc];
  ias = Map[PosNegIA[#, firstSrc] &, ias];
  Print["Done! Returning list of lists of sizes."];
  Return[SizesIA /@ ias];
 ]
```

Build a color function that maps each value present to a certain color, and Null to white.

```
MakeColorFunction[seqs_] := Module[{uniqueValues, colors, rules},
  uniqueValues = DeleteDuplicates[Flatten[seqs]];
  (* Special case: remove Null if it's there *)
  uniqueValues = DeleteCases[uniqueValues, Null];
  (* Assign colors *)
  colors = Map[ColorData[54], Range[Length[uniqueValues]]];
  rules = Rule @@@ Transpose[{uniqueValues, colors}];
  (* Special case: map Null to White *)
  rules = Append[rules, {Null → White}];
  Return[Association[rules]];
 ]
```

Call MAFFT with given seqs and parameters.
Return an {msa, colorFunction} tuple.

```
AlignSeqs[seqs_, mafftOP_, mafftEP_, mafftOtherArgs_] := Module[{},
  Print["Aligning..."];
  (* Build encoding: numbers <-> suitable alphabet *)
  {v2a, a2vs} =
   MakeEncoding[DeleteDuplicates@Flatten@seqs, MafftaApparentlySafeChars[]];
  (* Run MAFFT *)
  Print["Calling MAFFT at " <> DateString[]];
  {lbs, mtx} = RunMafft[seqs, v2a, mafftOP, mafftEP, mafftOtherArgs];
  Print["MAFFT done at " <> DateString[]];
  (* Build aligned matrix based on combining MAFFT output with pre-
   alignment matrix *)
  msa = SeqsUnderMtx[seqs, mtx];
  (* Return the multiple-sequence-alignment block and the color function used *)
  Return[msa];
 ]
```

Plot given seqs (or msa) with the given color function.

```
PlotSeqs[seqs_, colorFun_] := Module[{},
  ArrayPlot[
   Map[colorFun, seqs, {2}],
   ImageSize → {480, Automatic},
   ImagePadding → None,
   ImageMargins → None,
   PlotRangePadding → None,
   Mesh → All,
   MeshStyle → {{White}, {White}}
  ]
 ]
```

This allows you to chop an MSA manually, by interactively clicking.
When you click “Chop!”, a variable named **chopResult** is set with the resulting phases.

```
ChopManually[seqs_, msa_, colorFun_] := DynamicModule[{
   choppingPointVector = {},
   choppingPoints = {},
   ranges = {}
  },
  choppingPointVector = ConstantArray[0, Length[First[msa]] + 1];
  Clear[chopResult];
  Manipulate[
   Column[{
     Row[{
       Ceiling[uu][[1]],
       "  pts: ",
       choppingPoints,
       "  ranges: ",
       ranges
      }],
     ArrayPlot[{Drop[choppingPointVector, -1]}, ImageSize → {600, Automatic}],
     ArrayPlot[Map[colorFun, msa, {2}], ImageSize → {600, Automatic}]
    }],
   {uu, Locator},
   Button["Chop!",
    chopResult = Transpose@SeqsMsaFromTosToListOfSeqLists[seqs, msa, ranges];
    chopRanges = ranges;
   ],
   Button["Add chopping point",
    choppingPointVector[[Ceiling[uu][[1]]]] = 1;
    choppingPoints = Flatten@Position[choppingPointVector, 1];
    ranges = Transpose[{Drop[choppingPoints, -1], Drop[choppingPoints, 1] - 1}],
    ImageSize → {500, 50}
   ]
  ]
 ]
```

Given a list of seqs, their msa, and a list of n {from, to} pairs, return n lists of subseqs, one per phase

```
SeqsMsaFromTosToListOfSeqLists[seqs_, msa_, fromTos_] :=
 Module[{paps, listOfListsOfSubseqs},
  paps = MsaRangesToPreAlignmentPositions[msa, fromTos];
  listOfListsOfSubseqs = Table[
    seqs[[i]][[paps[[i]][[j]]]],
    {j, 1, Length[fromTos]},
    {i, 1, Length[seqs]}
   ];
  Return[listOfListsOfSubseqs];
 ]
```

Given an MSA with numeric values and Nulls, return a matrix of the same size where all non-Null values have been replaced by indices starting at 1 on each line. For example: {{8, Null, 9, 74}, {Null, 7, 7, Null}} becomes {{1, Null, 2, 3}, {Null, 1, 2, Null}}.

```
MsaToPositions[msa_] := Map[MsaRowToPositions, msa];

MsaRowToPositions[msaRow_] :=
 Module[{copyOfRow = msaRow, nonNullPositions = PositionsThatAreNot[msaRow, Null]},
  copyOfRow[[nonNullPositions]] = Range[Length[nonNullPositions]];
  Return[copyOfRow];
 ]
```

Given an MSA with numeric values and Nulls, and a list of {from, to} index pairs within the aligned MSA, return a list of lists of positions in the non-aligned sequences within those endpoints (inclusive).

In other words, for each row of the MSA and the given ranges, we get a list of lists of indices, each representing the pre-alignment points located within the given post-alignment endpoints. (Note that some lists could be empty.)

```
MsaRangesToPreAlignmentPositions[msa_, fromTos_] :=
  Transpose[Map[MsaRangeToPreAlignmentPositions[msa, #] &, fromTos]];

MsaRangeToPreAlignmentPositions[msa_, {from_, to_}] :=
  Map[MsaRowRangeToPreAlignmentPositions[#, {from, to}] &, msa];

MsaRowRangeToPreAlignmentPositions[row_, {from_, to_}] :=
 DeleteCases[MsaRowToPositions[row][[from ;; to]], Null]
```

Given a chopping result (phases), this pads each phase with Nulls on the right side, glues everything together with a few blank spacers, and returns a new matrix.

```
JoinPhases[phases_] := JoinWithNulls /@ PadPhases[phases]
```

Given a chopping result, this pads each phase with Nulls on the right side.

```
PadPhases[phases_] := Module[{tphases, widths, paddedtphases, paddedphases},
  tphases = Transpose[phases]; (* make each phase a list of rows *)
  widths = Map[Max, Map[Length, tphases, {2}]];
  paddedtphases = Table[
    Map[PadRight[#, widths[[i]], Null] &, tphases[[i]]],
    {i, 1, Length@tphases}
   ];
  paddedphases = Transpose[paddedtphases];
  Return[paddedphases]
 ]
```

This concatenates a list of lists, putting a couple of Nulls between each two adjacent sublists.

```
JoinWithNulls[listOfLists_] := Riffle[
   listOfLists,
   ConstantArray[{Null, Null}, Length@listOfLists]
  ] // Catenate
```