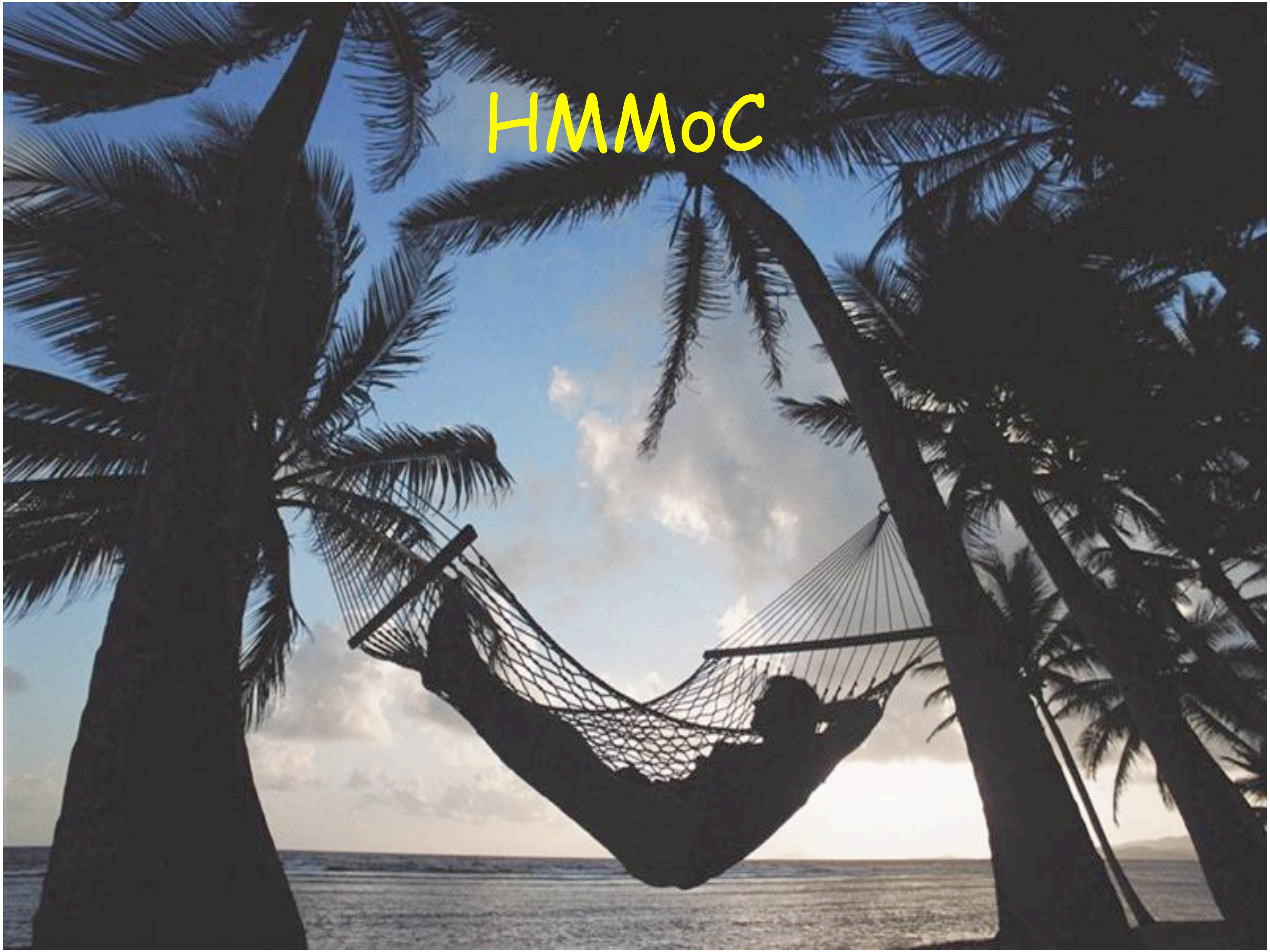
HMMoC



HMMs – powerful but tedious…

Implementing from scratch

- + Flexibility
- + Efficient
- – Optimal efficiency usually not reached
- – Expertise required
- – Long implementation time
- – Bugs
- – Pitfall: hard-code model/optimization choices

Libraries

- + Lower level of expertise required
- + Short implementation time
- + Bug free
- + Model change easy
- – Restricted to library features
- – Suboptimal efficiency

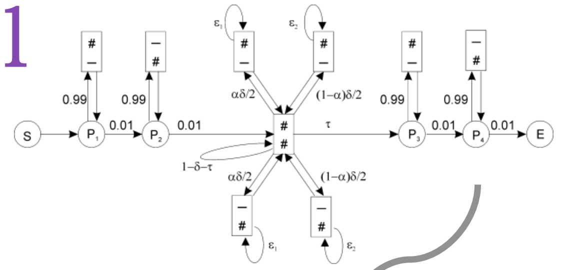
HMMoC

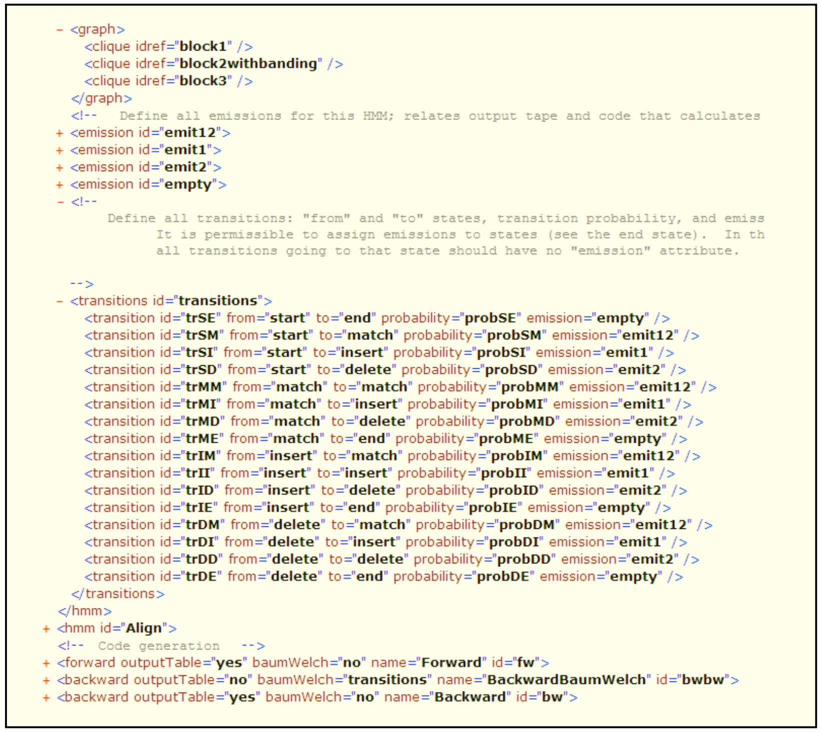
Aims to combine the advantages of both

“Hidden Markov Model Compiler” (really, a parser-generator):

- Describe structure of HMM in XML
- Transition and emission probabilities: C snippets
- Generates C++ code / header file
- Feature-rich
- Documentation + examples

Model



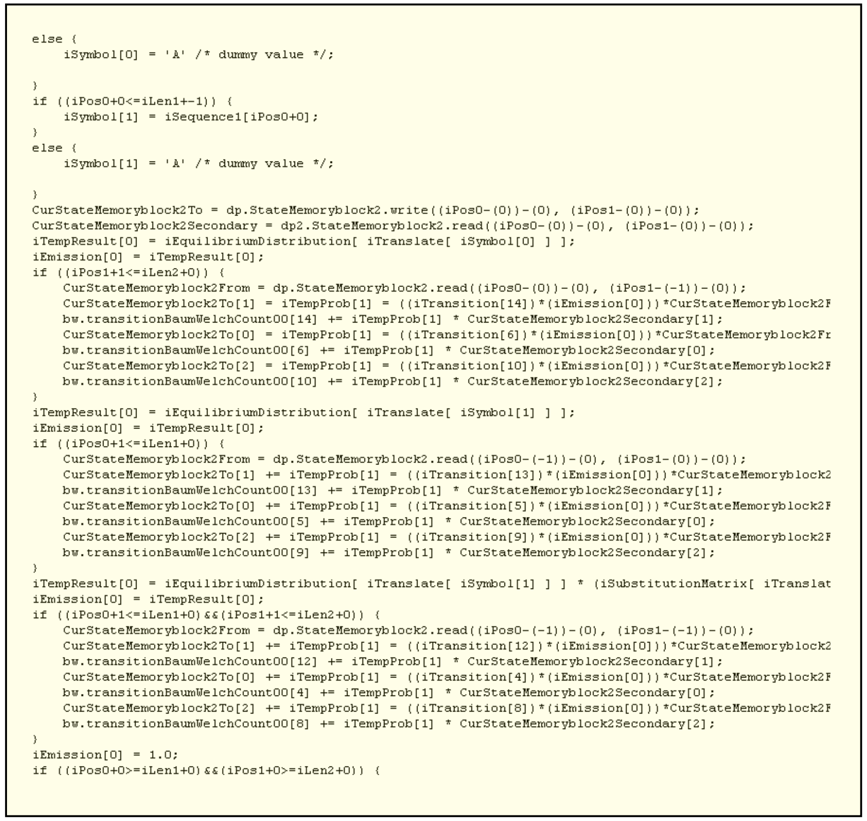


```
  - <graph>
      <clique idref="block1" />
      <clique idref="block2withbanding" />
      <clique idref="block3" />
    </graph>
    <!--   Define all emissions for this HMM; relates output tape and code that calculates
  + <emission id="emit12">
  + <emission id="emit1">
  + <emission id="emit2">
  + <emission id="empty">
  - <!--
          Define all transitions: "from" and "to" states, transition probability, and emiss
                It is permissible to assign emissions to states (see the end state).  In th
                all transitions going to that state should have no "emission" attribute.

    -->
  - <transitions id="transitions">
      <transition id="trSE" from="start" to="end" probability="probSE" emission="empty" />
      <transition id="trSM" from="start" to="match" probability="probSM" emission="emit12" />
      <transition id="trSI" from="start" to="insert" probability="probSI" emission="emit1" />
      <transition id="trSD" from="start" to="delete" probability="probSD" emission="emit2" />
      <transition id="trMM" from="match" to="match" probability="probMM" emission="emit12" />
      <transition id="trMI" from="match" to="insert" probability="probMI" emission="emit1" />
      <transition id="trMD" from="match" to="delete" probability="probMD" emission="emit2" />
      <transition id="trME" from="match" to="end" probability="probME" emission="empty" />
      <transition id="trIM" from="insert" to="match" probability="probIM" emission="emit12" />
      <transition id="trII" from="insert" to="insert" probability="probII" emission="emit1" />
      <transition id="trID" from="insert" to="delete" probability="probID" emission="emit2" />
      <transition id="trIE" from="insert" to="end" probability="probIE" emission="empty" />
      <transition id="trDM" from="delete" to="match" probability="probDM" emission="emit12" />
      <transition id="trDI" from="delete" to="insert" probability="probDI" emission="emit1" />
      <transition id="trDD" from="delete" to="delete" probability="probDD" emission="emit2" />
      <transition id="trDE" from="delete" to="end" probability="probDE" emission="empty" />
    </transitions>
  </hmm>
+ <hmm id="Align">
  <!--   Code generation   -->
+ <forward outputTable="yes" baumWelch="no" name="Forward" id="fw">
+ <backward outputTable="no" baumWelch="transitions" name="BackwardBaumWelch" id="bwbw">
+ <backward outputTable="yes" baumWelch="no" name="Backward" id="bw">
```

XML

HMMoC



```
else {
    iSymbol[0] = 'A' /* dummy value */;

}
if ((iPos0+0<=iLen1+-1)) {
    iSymbol[1] = iSequence1[iPos0+0];
}
else {
    iSymbol[1] = 'A' /* dummy value */;

}
CurStateMemoryblock2To = dp.StateMemoryblock2.write((iPos0-(0))-(0), (iPos1-(0))-(0));
CurStateMemoryblock2Secondary = dp2.StateMemoryblock2.read((iPos0-(0))-(0), (iPos1-(0))-(0));
iTempResult[0] = iEquilibriumDistribution[ iTranslate[ iSymbol[0] ] ];
iEmission[0] = iTempResult[0];
if ((iPos1+1<=iLen2+0)) {
    CurStateMemoryblock2From = dp.StateMemoryblock2.read((iPos0-(0))-(0), (iPos1-(-1))-(0));
    CurStateMemoryblock2To[1] = iTempProb[1] = ((iTransition[14])*(iEmission[0]))*CurStateMemoryblock2F
    bw.transitionBaumWelchCount00[14] += iTempProb[1] * CurStateMemoryblock2Secondary[1];
    CurStateMemoryblock2To[0] = iTempProb[1] = ((iTransition[6])*(iEmission[0]))*CurStateMemoryblock2Fr
    bw.transitionBaumWelchCount00[6] += iTempProb[1] * CurStateMemoryblock2Secondary[0];
    CurStateMemoryblock2To[2] = iTempProb[1] = ((iTransition[10])*(iEmission[0]))*CurStateMemoryblock2F
    bw.transitionBaumWelchCount00[10] += iTempProb[1] * CurStateMemoryblock2Secondary[2];
}
iTempResult[0] = iEquilibriumDistribution[ iTranslate[ iSymbol[1] ] ];
iEmission[0] = iTempResult[0];
if ((iPos0+1<=iLen1+0)) {
    CurStateMemoryblock2From = dp.StateMemoryblock2.read((iPos0-(-1))-(0), (iPos1-(0))-(0));
    CurStateMemoryblock2To[1] += iTempProb[1] = ((iTransition[13])*(iEmission[0]))*CurStateMemoryblock2
    bw.transitionBaumWelchCount00[13] += iTempProb[1] * CurStateMemoryblock2Secondary[1];
    CurStateMemoryblock2To[0] += iTempProb[1] = ((iTransition[5])*(iEmission[0]))*CurStateMemoryblock2F
    bw.transitionBaumWelchCount00[5] += iTempProb[1] * CurStateMemoryblock2Secondary[0];
    CurStateMemoryblock2To[2] += iTempProb[1] = ((iTransition[9])*(iEmission[0]))*CurStateMemoryblock2F
    bw.transitionBaumWelchCount00[9] += iTempProb[1] * CurStateMemoryblock2Secondary[2];
}
iTempResult[0] = iEquilibriumDistribution[ iTranslate[ iSymbol[1] ] ] * (iSubstitutionMatrix[ iTranslat
iEmission[0] = iTempResult[0];
if ((iPos0+1<=iLen1+0)&&(iPos1+1<=iLen2+0)) {
    CurStateMemoryblock2From = dp.StateMemoryblock2.read((iPos0-(-1))-(0), (iPos1-(-1))-(0));
    CurStateMemoryblock2To[1] += iTempProb[1] = ((iTransition[12])*(iEmission[0]))*CurStateMemoryblock2
    bw.transitionBaumWelchCount00[12] += iTempProb[1] * CurStateMemoryblock2Secondary[1];
    CurStateMemoryblock2To[0] += iTempProb[1] = ((iTransition[4])*(iEmission[0]))*CurStateMemoryblock2F
    bw.transitionBaumWelchCount00[4] += iTempProb[1] * CurStateMemoryblock2Secondary[0];
    CurStateMemoryblock2To[2] += iTempProb[1] = ((iTransition[8])*(iEmission[0]))*CurStateMemoryblock2F
    bw.transitionBaumWelchCount00[8] += iTempProb[1] * CurStateMemoryblock2Secondary[2];
}
iEmission[0] = 1.0;
if ((iPos0+0>=iLen1+0)&&(iPos1+0>=iLen2+0)) {
```

C++

HMMoC - Features

- Single, pair, triple, quadruple HMMs
- Forward, Backward, Viterbi, posterior sampling, Baum-Welch
- Silent states (“wing folding”) fully supported.
- Higher-order HMMs
- Emissions associated to states (Moore) or transitions (Mealy); mixing allowed.
- Position-dependent transition and emission probabilities
- Banded recursion by providing a DP table iterator
- Transparent memory+time efficient implementation of banded DP table
- Probabilities: double, logspace, or extended-exponent float (“bfloats”)
- Both reals and DP tables are implemented by templates; can be adapted if desired.

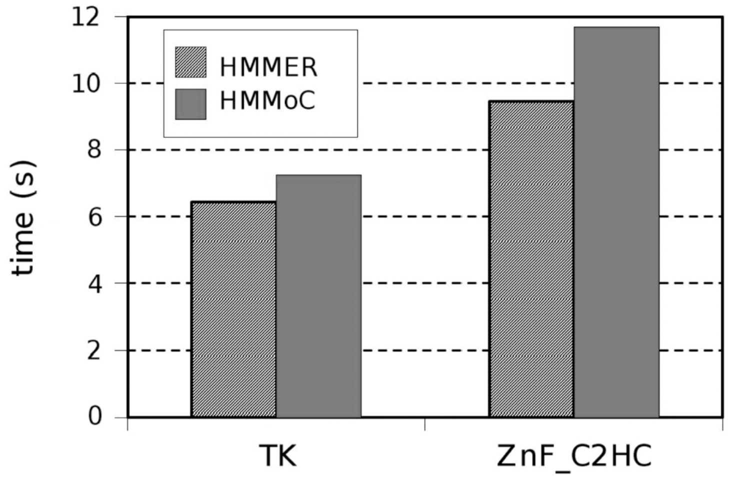
HMMoC - Efficiency

Time:

- No table lookups; loops over transitions/emissions are unrolled
- Probabilities are computed early and re-used
- Calculations are ordered to minimize DP table lookups
- Silent state ordering to minimize dimension of matrix inversions

Memory:

- Special states (start, end, …) have their own DP tables
- Supports folded DP tables if not required for output



HMMoC – Handy features

- Macro facility to reduce repetitiveness of XML
- DP table / Baum-Welch counts access by name and numerical ID
- Python interface (prototype, using pyrex/cython)

- Negative probabilities cause run-time warnings
- Informative compiler errors
- Sanity checks on HMM

  e.g. consistent order of states;

  consistent mixing of Mealy/Moore views
- Reasonably readable (indented) C++ output

HMMoC - Examples

- Published:
  - Probabilistic whole-genome re-alignment
    Lunter, Rocco, Mimouni, Heger, Caldeira, Hein; Gen Res 18; 2008
  - Identification of viral overlapping reading frames
    de Groot et al., BMC Bioinformatics (accepted)
- Toy examples included with HMMoC:
  - Occasionally dishonest casino
  - CpG island detection
  - Simple pairwise aligner
  - HMMER implementation
  - Several more
- Download
  - http://genserv.anat.ox.ac.uk/downloads/software/hmmoc
  - Google “hmmoc”