Complex PRISM models for analyzing very large biological sequence data
– plus a few notes on probabilistic abductive logic programming

Henning Christiansen
Roskilde University, Denmark
henning@ruc.dk, http://www.ruc.dk/~henning
This talk: Probabilistic tools which *may* be useful for systems biology

- Experiences and adaptation of PRISM (Sato & al) for sequence data
  - Developed in the LoSt project, funded by Danish Strategic Research Council
  - Thanks especially to PhD students, Christian Theil Have, Ole Torp Lassen, postdoc Matthieu Petit; to the PRISM group, Taisuke Sato, Yoshitaka Kameya, Neng-Fa Zhou

- (Probabilistic) abductive logic programming developed with Constraint Handling Rules (here: only brief overview)
PRISM (Sato & al) and the LoSt project

- Chosen for the LoSt project because
  - Declarative: Firm, theoretical basis
  - Flexible: A full programming language
  - Instrumented with powerful probabilistic inference methods
  - LoSt project goal: investigate to which extent “such models” are useful for bio sequence analysis as compared with “traditional tools”, e.g. HMM software written in C

- Most of our effort
  - Cope with inherently high complexity of PRISM models
  - Increase scaleability
  - (No revolutionary biological results yet)
  - Learned quite a lot about writing different models in PRISM
    - E.g. (Christiansen, Have, Lassen, Petit. Taming the Zoo of discrete HMM subspecies & some of their relatives. In Biology, Computation and Linguistics, New Interdisciplinary Paradigms, volume 228 of Frontiers in Artificial Intelligence and Applications, IOS Press, 2011)
Sequence analysis with PRISM
Example: HMM + study scaleability

Hidden Markov Model

- Well-known probabilistic model for sequential phenomena, e.g., genomes
- Probabilistic, finite state machine with probabilistic emissions

Viterbi path

- \( \approx \) the most probable sequence of states for observed sequence
- aka explanation, description, annotation
- Linear time Viterbi algorithm – dynamic programming (DP)
- PRISM has generalized Viterbi algorithm, DP effect obtained by B-Prolog’s tabling

Our example: Simple 2-state HMM adapted from PRISM manual
values(init, [s0, s1]).
values(out(_), [a, b]).
values(tr(_), [s0, s1]).

hmm(L):-
    msw(init, S0),
    hmm(S0, L).

hmm(_, []).

hmm(S, [Ob | Obs]):-
    msw(out(S), Ob),
    msw(tr(S), Next),
    hmm(Next, Obs).

?- viterbif(hmm([b, a, a, b]))
hmm([a, a, b, b])
    <= hmm(s1, [a, a, b, b]) & msw(init, s1)
hmm(s1, [a, a, b, b])
    <= hmm(s1, [a, b, b]) & msw(out(s1), a)
        & msw(tr(s1), s1)
hmm(s1, [a, b, b])
    <= hmm(s0, [b, b]) & msw(out(s1), a)
        & msw(tr(s1), s0)
hmm(s0, [b, b])
    <= hmm(s1, [b]) & msw(out(s0), b)
        & msw(tr(s0), s1)
hmm(s1, [b])
    <= hmm(s0, []) & msw(out(s1), b)
        & msw(tr(s1), s0)
hmm(s0, [])

Viterbi_P = 0.008470728000000

Problem:
- we want an explicit representation of the Viterbi path
- so let’s add it ....
values(init,[s0,s1]).
values(out(_),[a,b]).
values(tr(_),[s0,s1]).

hmm(L,Ss):-
  msw(init,S0),
  hmm(S0,L,Ss).

hmm(S,[],[S]).

hmm(S,[Ob|Obs],[S|Ss]):-
  msw(out(S),Ob),
  msw(tr(S),Next),
  hmm(Next,Obs,Ss).

?- viterbig(hmm([b,a,a,b],Path)).
Path = [s1,s0,s1,s0,s1]

Problem:
– PRISM not design with this in mind
– The history argument destroys tabling

Runtime more than exponential

<table>
<thead>
<tr>
<th>Length</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.022 sec</td>
</tr>
<tr>
<td>20</td>
<td>&gt; 1 min</td>
</tr>
<tr>
<td>21</td>
<td>???</td>
</tr>
</tbody>
</table>
Version 2: Remove non-discriminating arguments
(Christiansen, Gallagher, ICLP 2009)

values(init,[s0,s1]).
values(out(_),[a,b]).
values(tr(_),[s0,s1]).

hmm(L,--Ss):-
    msw(init,S0),
    hmm(S0,L,--Ss).

hmm(S,[]),--[S]).

hmm(S,[Ob|Obs],--[S|Ss]):-
    msw(out(S),Ob),
    msw(tr(S),Next),
    hmm(Next,Obs,--Ss).

Program transformation for PRISM programs:
– remove such arguments
– run viterbi on reduced program
– reconstruct arguments by deterministic run directed by proof tree.
– runtimes as Version 0 :)

?- prismAnnot(hmm2).
?- viterbiAnnot(hmm([b,a,a,b],Path),Prob)
Path = [s1,s0,s1,s0,s1]
Prob = 0.008470728 ?
Runtimes still not good enough

<table>
<thead>
<tr>
<th>Length</th>
<th>Version 1: With annot</th>
<th>Version 2+autoannot = Version 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.022 sec</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>&gt; 1 min</td>
<td>0</td>
</tr>
<tr>
<td>21</td>
<td>???</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>1,000</td>
<td>–</td>
<td>0.07 sec</td>
</tr>
<tr>
<td>5,000</td>
<td>–</td>
<td>1.6 sec</td>
</tr>
<tr>
<td>10,000</td>
<td>–</td>
<td>6 sec</td>
</tr>
<tr>
<td>20,000</td>
<td>–</td>
<td>25 sec</td>
</tr>
<tr>
<td>30,000</td>
<td>–</td>
<td>1 min</td>
</tr>
</tbody>
</table>

Tests made with PRISM 2.0 on iMac 2.8GHzs Intel Core i5 with 12 GB ram

≈ Quadratic time complexity :(

* B-Prolog’s tabling copies and compares structure
* No optimization for ground structures - where in principle storing and comparing pointers would do
Version 3: As Version 2 but now simulating pointers
(Have, Christiansen, PADL 2011)

hmmTop(L,--S):-
    store_list(L,Index),
    hmm(Index,--S).

hmm(S,[],--[S]):-!.

hmm(S,ObY,--[S|Ss]):-
    retrieve_list(ObY,Ob,Y),
    msw(out(S),Ob),
    msw(tr(S),Next),
    hmm(Next,Y,--Ss).

Program trans. for PRISM:
– translate structured args. into pointer representation

:- store_list([b,a,a,b],Idx).

May result in

retrieve_list(1, b, 2).
retrieve_list(2, a, 3).
retrieve_list(3, a, 4).
retrieve_list(4, b, 5).
retrieve_list(5, _, []).

?- prismAnnot(hmm3).
?- viterbiAnnot(hmmTop([b,a,a,b],Path),Prob)
Path = [s1,s0,s1,s1,s0,s1]
Prob = 0.008470728 ?
Runtimes, finally

<table>
<thead>
<tr>
<th>Length</th>
<th>V. 1: With annot</th>
<th>V. 2 = V.1 + autoannot</th>
<th>V. 3 = V. 2 + pointers</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.022 sec</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>&gt; 1 min</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>???</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>1,000</td>
<td>–</td>
<td>0.07 sec</td>
<td>0.016 sec</td>
</tr>
<tr>
<td>5,000</td>
<td>–</td>
<td>1.6 sec</td>
<td>0.052 sec</td>
</tr>
<tr>
<td>10,000</td>
<td>–</td>
<td>6 sec</td>
<td>0.11 sec</td>
</tr>
<tr>
<td>20,000</td>
<td>–</td>
<td>25 sec</td>
<td>0.24 sec</td>
</tr>
<tr>
<td>30,000</td>
<td>–</td>
<td>1 min</td>
<td>0.4 sec</td>
</tr>
<tr>
<td>100,000</td>
<td>–</td>
<td>–</td>
<td>2.9 sec</td>
</tr>
</tbody>
</table>

Linear time complexity :)

- ... crashes around length = 150,000 :/
- independently of memory settings, 32 vs. 64 bit machine with extreme amount of RAM
Runtimes, finally

<table>
<thead>
<tr>
<th>Length</th>
<th>V. 1: With annot</th>
<th>V. 2 = V.1 + autoannot</th>
<th>V. 3 = V. 2 + pointers</th>
<th>V. 4 = V3 + log_scale</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.022 sec</td>
<td>0</td>
<td>0.016 sec</td>
<td>0.018 sec</td>
</tr>
<tr>
<td>20</td>
<td>&gt; 1 min</td>
<td>0</td>
<td>0.052 sec</td>
<td>0.08 sec</td>
</tr>
<tr>
<td>21</td>
<td>???</td>
<td>0</td>
<td>0.11 sec</td>
<td>0.19 sec</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>1,000</td>
<td>–</td>
<td>0.07 sec</td>
<td>0.016 sec</td>
<td>0.018 sec</td>
</tr>
<tr>
<td>5,000</td>
<td>–</td>
<td>1.6 sec</td>
<td>0.052 sec</td>
<td>0.08 sec</td>
</tr>
<tr>
<td>10,000</td>
<td>–</td>
<td>6 sec</td>
<td>0.11 sec</td>
<td>0.19 sec</td>
</tr>
<tr>
<td>20,000</td>
<td>–</td>
<td>25 sec</td>
<td>0.24 sec</td>
<td>0.44 sec</td>
</tr>
<tr>
<td>30,000</td>
<td>–</td>
<td>1 min</td>
<td>0.4 sec</td>
<td>0.66 sec</td>
</tr>
<tr>
<td>100,000</td>
<td>–</td>
<td>–</td>
<td>2.9 sec</td>
<td>3.8 sec</td>
</tr>
</tbody>
</table>

Linear time complexity :)  
* ... crashes around length = 150,000 :/  
* independently of memory settings, 32 vs. 64 bit machine with extreme amount of RAM
Our approach to complex models: Bayesian Annotation Networks
(Christiansen, Have, Lassen, Petit, ICLP 2011)

Divide complex model into sub-models (= separate PRISM models) organized in a Bayesian network

- each model possibly parameterized by outcome of other models

\[
\text{m}_i( +\text{Sequence}, -\text{Annot}, +\text{Annot}_1, +\text{Annot}_2, \ldots ) : -
\]
\[
\ldots
\]
\[
\text{msw}( \text{xxx}( \text{part-\text{Annot}}_1, \text{part-\text{Annot}}_2), \text{part-\text{Annot}} )
\]
\[
\ldots
\]

- A distinguished top-model
- Viterbi computations done one submodel at a time in topological order, thus reducing degrees of freedom (≈no of states) in each step
- Training done in a similar way
- Implemented as “The LoSt Framework” with its own script language for dependencies
- To be released spring 2012, integrated with the previous PRISM optimizations
Our approach to complex models: Bayesian Annotation Networks
(Christiansen, Have, Lassen, Petit, ICLP 2011)

Divide complex model into sub-models (= separate PRISM models) organized in a Bayesian network
- each model possibly parameterized by outcome of other models

\[
m_i(\text{Sequence}, \neg\text{Annot}, +\text{Annot}_1, +\text{Annot}_2, \ldots)
\]

- A distinguished top-model
- Viterbi computations done one submodel at a time in topological order, thus reducing degrees of freedom (≈ no of states) in each step
- Training done in a similar way
- Implemented as “The LoSt Framework” with its own script language for dependencies
- To be released spring 2012, integrated with the previous PRISM optimizations
Overview of probabilistic abduction, inspired by PRISM and Constraint Handling Rules

State of the art Probabilistic Abductive Logic Programming:

(Christiansen, 2008, in “Constraint Handling Rules, Current Research Topics”, LNCS 5388)

* An LP language with possibly non-ground abducibles and integrity constraints
* A nice semantics (possible worlds; assumed independent abducibles)
* Prototype implementations in CHR, including with best-first search

Probabilistic Abductive Logic Programming with dependencies in simult. probability distr. over abducibles specified using CHRIsm (Sneyers,...).

(Christiansen, Saleh, CHR-Workshop, 2011)

* Nice semantics (possible worlds)
* Slow prototype implementation in CHR+CHRIsm

Efficient implementation of non-prob. abduction, with powerful ICs

Conclusions

(Probabilistic) Logic programming technology apply to biological sequence analysis

- Clean semantics: (Probabilistic) Herbrand models, ...
- Transparency, modifiability, easy experiments, high expr. power
- Flexibility of a full programming language (incl. dirty tricks)

It does scale

- Our program transformation based optimizations obvious to implement at low level
- If you want n>100.000 in LoSt Framework, use a chunker as submodel ;-) 

Newer logic programming paradigms add forward chaining rules, (state --> state)

- CHR, CHRI\textsc{sm} (\textsc{chr*prism})

(P)L\textsc{p} technology demonstrated here for sequence analysis, so obvious in the toolbox for systems biology