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

International Symposium On Symbolic Systems Biology (ISSSB'11) Shonan Village Centre, Japan 13-17 November, 2011

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

- ≈ 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

Version 0

```
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 ....

Version 1: explicit annotation

```
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

Length	Runtime
10	0.022 sec
20	> 1 min
21	???

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

Length	Version 1: With annot	Version 2+autoannot ≈ Version 0
10	0.022 sec	0
20	> 1 min	0
21	???	0
...	...	...
1,000	–	0.07 sec
5,000	–	1.6 sec
10,000	–	6 sec
20,000	–	25 sec
30,000	–	1 min

Tests made with PRISM 2.0 on iMac 2.8GHs 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,s0,s1]
Prob = 0.008470728 ?
```

Runtimes, finally

Length	V. 1: With annot	V. 2 = V.1 + autoannot	V. 3 = V. 2 + pointers
10	0.022 sec	0	
20	> 1 min	0	
21	???	0	
...	...	...	
1,000	–	0.07 sec	0.016 sec
5,000	–	1.6 sec	0.052 sec
10,000	–	6 sec	0.11 sec
20,000	–	25 sec	0.24 sec
30,000	–	1 min	0.4 sec
100,000	–	–	2.9 sec

Linear time complexity :)

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

Runtimes, finally

Length	V. 1: With annot	V. 2 = V.1 + autoannot	V. 3 = V. 2 + pointers	V. 4 = V3 + log_scale
10	0.022 sec	0		
20	> 1 min	0		
21	???	0		
…	…	…		
1,000	–	0.07 sec	0.016 sec	0.018 sec
5,000	–	1.6 sec	0.052 sec	0.08 sec
10,000	–	6 sec	0.11 sec	0.19 sec
20,000	–	25 sec	0.24 sec	0.44 sec
30,000	–	1 min	0.4 sec	0.66 sec
100,000	–	–	2.9 sec	3.8 sec

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

```
m_i(+Sequence, –Annot, +Annot1, +Annot2, … ):-
   ….
   msw(xxx(part-Annot1, part-Annot2), part-Annot)
   ….  .
```

- 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**

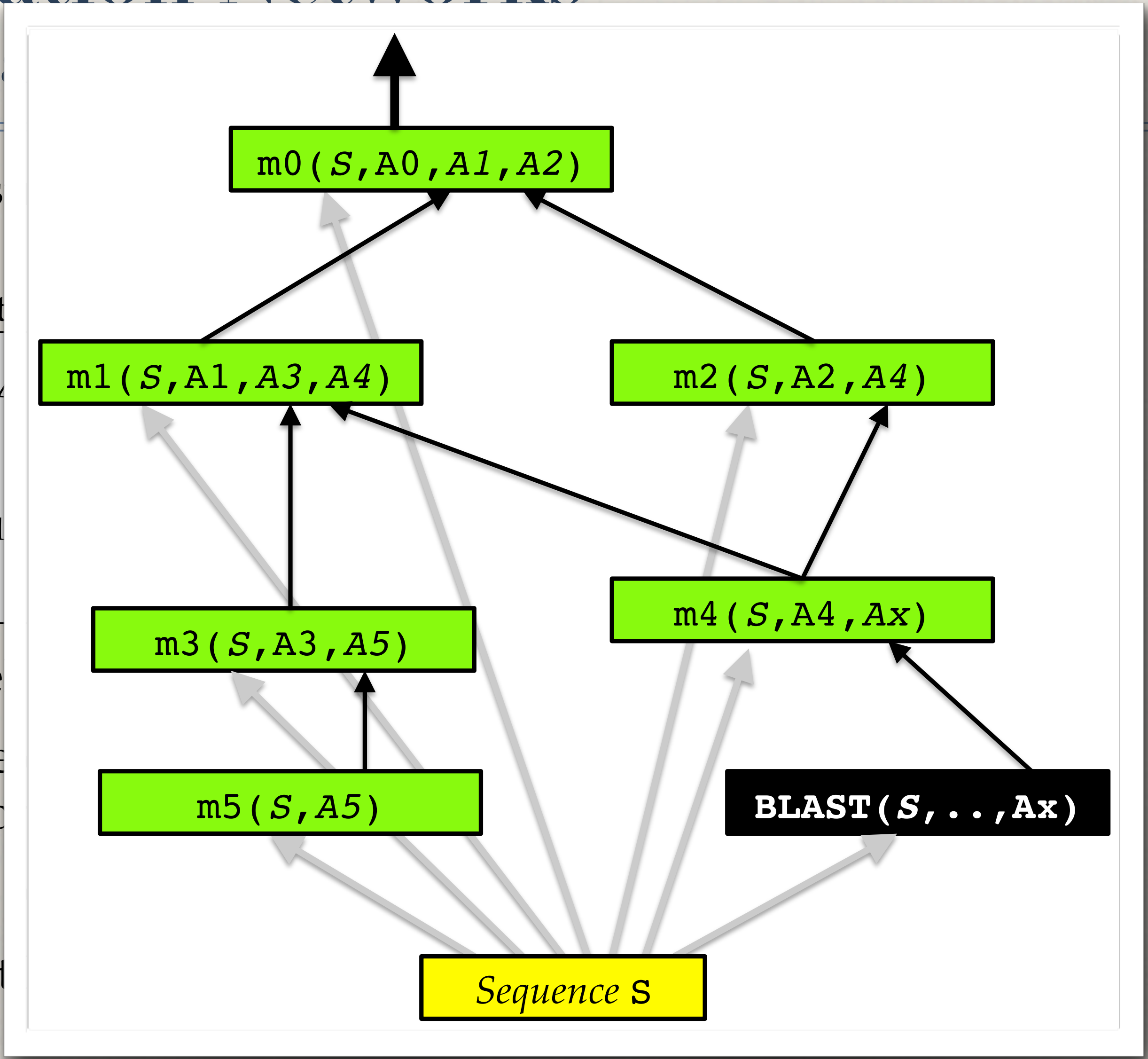
(Christiansen, H

Divide complex model into s
Bayesian network

- each model possibly paramet

```
mi(+Sequence, –Annot, +
    ....
    msw(xxx(part-Annot1
    ....  .
```

- A distinguished top-mode
- Viterbi computations done
  reducing degrees of freedo
- Training done in a similar
- Implemented as "The LoS
  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
(Christiansen. *Executable specifications for hypothesis-based reasoning with Prolog and Constraint Handling Rules*, Journal of Applied Logic, vol 7, 2009) SEE EXAMPLE IN SEPARATE FILE)

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, CHRiSM (= CHR*PRISM)

(P)LP technology demonstrated here for sequence analysis, so obvious in the toolbox for systems biology