Compiling Comp Ling
Practical weighted dynamic programming and the Dyna language

Jason Eisner
Eric Goldlust
Noah A. Smith

HLT-EMNLP, October 2005

An Anecdote from ACL’05

JUST DO IT.
-Michael Jordan

Just draw a model that actually makes sense for your problem.

Just do Gibbs sampling. Um, it’s only 6 lines in Matlab…

Conclusions to draw from that talk

1. Mike & his students are great.
2. Graphical models are great.
   (because they’re flexible)
3. Gibbs sampling is great.
   (because it works with nearly any graphical model)
4. Matlab is great.
   (because it frees up Mike and his students to doodle all day and then execute their doodles)

Could NLP be this nice?

1. Mike & his students are great.
2. Graphical models are great.
   (because they’re flexible)
3. Gibbs sampling is great.
   (because it works with nearly any graphical model)
4. Matlab is great.
   (because it frees up Mike and his students to doodle all day and then execute their doodles)

Parts of it already are …

- Language modeling
- Binary classification (e.g., SVMs)
- Finite-state transductions
- Linear-chain graphical models

Toolkits available; you don’t have to be an expert

But other parts aren’t …

- Context-free and beyond
- Machine translation

Efficient parsers and MT systems are complicated and painful to write
This talk: A toolkit that’s general enough for these cases.
(stretches from finite-state to Turing machines)

“Dyna”

But other parts aren’t ...
Context-free and beyond
Machine translation

Efficient parsers and MT systems are complicated and painful to write.

Warning

- This talk is only an advertisement!
- For more details, please see the paper
  see http://dyna.org
  (download + documentation)

sign up for updates by email

How you build a system (“big picture” slide)

Cool model

practical equations

$$\beta_{i,j,k} = \sum_{i' \leq j \leq k} \beta_{i',j} g_{i',j,k} p(y_i \rightarrow y_j | N_i)$$

pseudocode (execution order)

for width from 2 to n
for i from 0 to n-width
  k = i+width
for j from i+1 to k-1
  ...

Compilation strategies (we’ll come back to this)

tuned C++
implementation (data structures, etc.)

Wait a minute ...

Didn’t I just implement something like this last month?

chart management / indexing
cache-conscious data structures
prioritize partial solutions (best-first, pruning)
parameter management
inside-outside formulas
different algorithms for training and decoding
conjugate gradient, annealing, ...
parallelization?

We thought computers were supposed to automate drudgery

How you build a system (“big picture” slide)

Cool model

practical equations

$$\beta_{i,j,k} = \sum_{i' \leq j \leq k} \beta_{i',j} g_{i',j,k} p(y_i \rightarrow y_j | N_i)$$

Dyna language specifies these equations

Most programs just need to compute some values from other values. Any order is ok.

Some programs also need to update the outputs if the inputs change:
- spreadsheets, makefiles, email readers
- dynamic graph algorithms
- EM and other iterative optimization
- leave-one-out training of smoothing params

Compilation strategies (we’ll come back to this)

tuned C++
implementation (data structures, etc.)
Writing equations in Dyna

- `int a.`
- `a = b * c.`
  - `a` will be kept up to date if `b` or `c` changes.
  - `b += y.` equivalent to `b = x+y.`
    - `b` is a sum of two variables. Also kept up to date.
- `c += z(1)`
- `c += z(2)`
- `c += z("four")`
- `c += z(foo(bar,5))`
  - `c` is a sum of all nonzero `z(...)` values.
  - At compile time, we don’t know how many!

More interesting use of patterns

- `a = b * c.`
  - Scalar multiplication
  - `a(I) = b(I) * c(I).`
    - Scalar multiplication
  - `a += b(I) * c(I).` means `a = \sum b(I) * c(I)`
    - Dot product; could be sparse
  - `a, b, c` are patterns
  - The capitalized `N` matches anything
  - `c += z(N)`
    - `c` is a sum of all nonzero `z(...)` values.

Dyna vs. Prolog

By now you may see what we’re up to!

Prolog has Horn clauses:

- `a(I,K) :- b(I,J), c(J,K).`

Dyna has “Horn equations”:

- `a(I,K) += b(I,J) * c(J,K).`

Like Prolog:

- Allow nested terms
- Syntactic sugar for lists, etc.
- Turing-complete

Unlike Prolog:

- Charts, not backtracking!
- Compile → efficient C++ classes
- Integrates with your C++ code

The CKY inside algorithm in Dyna

```cpp
using namespace cky;
chart c;
chart c[rewrite(“s”,”np”,”vp”)] = 0.7;
chart c[word(“Pierre”,0,1)] = 1;
chart c[length(30)] = true; // 30-word sentence
cin >> c; // get more axioms from stdin
cout << c[goal]; // print total weight of all parses
```

Related algorithms in Dyna?

- Viterbi parsing?
- Logarithmic domain?
- Lattice parsing?
- Earley’s algorithm?
- Binarized CKY?
- Incremental (left-to-right) parsing?
- Log-linear parsing?
- Lexicalized or synchronous parsing?
Related algorithms in Dyna?

- Viterbi parsing?
- Logarithmic domain?
- Lattice parsing?
- Earley’s algorithm?
- Binarized CKY?
- Incremental (left-to-right) parsing?
- Log-linear parsing?
- Lexicalized or synchronous parsing?

constit(X,I,J) += word(W,I,J) * rewrite(X,W).
constit(X,I,J) += constit(Y,I,Mid) * constit(Z,Mid,J) * rewrite(X,Y,Z).
goal += constit("s",0,N) if length(N).

Viterbi parsing?
Logarithmic domain?
Lattice parsing?
Earley’s algorithm?
Binarized CKY?
Incremental (left-to-right) parsing?
Log-linear parsing?
Lexicalized or synchronous parsing?

Earley’s algorithm in Dyna

constit(X,I,J) += word(W,I,J) * rewrite(X,W).
constit(X,I,J) += constit(Y,I,Mid) * constit(Z,Mid,J) * rewrite(X,Y,Z).
goal += constit("s",0,N) if length(N).

Need("s",J) = true.
need(Nonterm,J) |= ?constit(_/Nonterm|__|J).
constit(Nonterm/Needed,I) += rewrite(Nonterm,Needed) if need(Nonterm,I).
constit(Nonterm/Needed,LK) += constit(Nonterm/\[W|Needed\],I,J) * word(W,J,K).
constit(Nonterm/Needed,LK) += constit(Nonterm/X/Needed,LJ) * constit(X/I,J,K).
goal += constit("s",\[0|0|N\]) if length(N).

magic templates transformation (as noted by Minnen 1996)

Lots of equivalent ways to write a system of equations!
Transforming from one to another may improve efficiency.
(Or, transform to related equations that compute gradients, upper bounds, etc.)
Many parsing “tricks” can be generalized into automatic transformations that help other programs, too!

Program transformations

cool model

practical equations

\[ p_{ij}(k) = \sum_{l=1}^{\beta} \beta_{il} j_{ik} \]

\[ p(N \rightarrow N', [N]) \]

Lots of equivalent ways to write a system of equations!
Transforming from one to another may improve efficiency.
(Or, transform to related equations that compute gradients, upper bounds, etc.)
Many parsing “tricks” can be generalized into automatic transformations that help other programs, too!
Related algorithms in Dyna?

- Viterbi parsing?
- Logarithmic domain?
- Lattice parsing?
- Earley’s algorithm?
- Binarized CKY?
- Incremental (left-to-right) parsing?
- Log-linear parsing?
- Lexicalized or synchronous parsing?

\[
\text{constit}(X,I,J) \leftarrow \text{word}(W,I,J) \ast \text{rewrite}(X,W).
\]

\[
\text{constit}(X,I,J) \leftarrow \text{constit}(Y,I,Mid) \ast \text{constit}(Z,Mid,J) \ast \text{rewrite}(X,Y,Z).
\]

\[
\text{goal} \leftarrow \text{constit}("s",0,N) \text{ if } \text{length}(N).
\]

Rule binarization

\[
\text{constit}(X,I,J) \leftarrow \text{constit}(Y,I,Mid) \ast \text{constit}(Z,Mid,J) \ast \text{rewrite}(X,Y,Z).
\]

folding transformation: asymp. speedup!

\[
\text{constit}(X,Y,Mid,J) \leftarrow \text{constit}(Y,I,Mid) \ast \text{constit}(Z,Mid,J) \ast \text{rewrite}(X,Y,Z).
\]

\[
\text{constit}(X,I,J) \leftarrow \text{constit}(Y,I,Mid) \ast \text{constit}(X,Y,Mid,J).
\]

Related algorithms in Dyna?

- Viterbi parsing?
- Logarithmic domain?
- Lattice parsing?
- Earley’s algorithm?
- Binarized CKY?
- Incremental (left-to-right) parsing?
- Log-linear parsing?
- Lexicalized or synchronous parsing?

Just add words one at a time to the chart

Check at any time what can be derived from words so far

Similarly, dynamic grammars

Again, no change to the Dyna program

Basically, just add extra arguments to the terms above
How you build a system ("big picture" slide)

Propagate updates from right-to-left through the equations.

a.k.a. "agenda algorithm"
"forward chaining"
"bottom-up inference"
"semi-naive bottom-up"

cool model

practical equations

Propagate updates
from right-to-left

rules of program

tuned C++
implementation
(data structures, etc.)

agenda of pending updates

propagated equations

β(j,k) = \sum_{j' \neq j} β(j,j') β(j',k)

p(N_1 → N_2 | V_1)

pseudocode (execution order)

for width from 2 to n

for i from 0 to n-width

k = i+width

for j from i+1 to k-1
...

chart of derived items with current values

What's going on under the hood?

agenda of pending updates

n(I,K) += n(I,J) * n(J,K)

What's going on under the hood?

cool model

practical equations

n(5,5) = 0.2

n(5,5) = 0.3

epsilon

to make another copy of itself

n(5,5)

combining with itself

rules of program

If np(3,5) hadn't been in the chart already, we would have added it.

np(3,5)

+= 0.3

chart of derived items with current values

Parameter training

Maximize some objective function.

Use Dyna to compute the function.

Then how do you differentiate it?

for gradient ascent, conjugate gradient, etc.

gradient also tells us the expected counts for EM!

model parameters

as axiom values

objective function

as a theorem's value

e.g., inside algorithm computes likelihood of the sentence

DynaMITE: training toolkit

Two approaches:

Program transformation – automatically derive the "outside" formulas.

Back-propagation – run the agenda algorithm "backwards."

works even with pruning, early stopping, etc.
What can Dyna do beyond CKY?

- Context-based morphological disambiguation with random fields (Smith, Smith & Tromble EMNLP'05)
- Parsing with constraints on dependency length (Eisner & Smith IWPT'05)
- Unsupervised grammar induction using contrastive estimation (Smith & Eisner GIA'05)
- Grammar induction with annealing (Smith & Eisner ACL'05)
- Synchronous cross-lingual parsing (Smith & Smith EMNLP'04)
- Loosely syntax-based MT ... (Smith & Eisner in prep.)
- Partly supervised grammar induction ... (Dreyer & Eisner in prep.)
- Teaching (Eisner JHU'05; Smith & Tromble JHU'04)
- Unsupervised log-linear models using contrastive estimation (Smith & Eisner ACL'04)
- More finite-state stuff ... (Tromble & Eisner in prep.)
- Most of my own past work on trainable (in)finite-state machines, parsing, MT, phonology ...

Can it express everything in NLP? 😊

- Remember, it integrates tightly with C++, so you only have to use it where it's helpful, and write the rest in C++. Small is beautiful.
- We're currently extending the class of allowed formulas "beyond the semiring"
  - will be able to express smoothing, neural nets, etc.
- Of course, it is Turing complete ... 😆

Smoothing in Dyna

- mle_prob(X,Y,Z)  % context = count(X,Y,Z)/count(X,Y).
- smoothed_prob(X,Y,Z) = lambda*mle_prob(X,Y,Z) + (1-lambda)*mle_prob(Y,Z).  % for arbitrary n-grams, can use lists
- count_count(N) += 1 whenever N is count(Anything).  % updates automatically during leave-one-out jackknifing

Neural networks in Dyna

- out(Node) = sigmoid(in(Node)).
- in(Node) += input(Node).
- in(Node) += weight(Node,Kid)*out(Kid).
- error += (out(Node)-target(Node))**2 if ?target(Node).

Game-tree analysis in Dyna

- goal = best(Board) if start(Board).
  - best(Board) max= stop(player1, Board).
  - best(Board) max= move(player1, Board, NewBoard) + worst(NewBoard).
- worst(Board) min= stop(player2, Board).
- worst(Board) min= move(player2, Board, NewBoard) + best(NewBoard).

Weighted FST composition in Dyna (epsilon-free case)

- :- bool item=false.
  - start (A o B, Q x R) |= start (A, Q) & start (B, R).
  - stop (A o B, Q x R) |= stop (A, Q) & stop (B, R).
  - arc (A o B, Q1 x R1, Q2 x R2, In, Out) |= arc (A, Q1, Q2, In, Match) & arc (B, R1, R2, Match, Out).

- Inefficient? How do we fix this?
**Constraint programming (arc consistency)**

- :- bool item=false.
- :- bool consistent=true. % overrides prev line

variable(Var) |= in_domain(Var:Val).
possible(Var:Val) & in_domain(Var:Val).
possible(Var:Val) & support(Var:Val, Var2) whenever variable(Var2).
support(Var:Val, Var2) |= possible(Var2:Val2) & consistent(Var:Val, Var2:Val2).

---

**Is it fast enough? (sort of)**

- Asymptotically efficient
- 4 times slower than Mark Johnson’s inside-outside
- 4-11 times slower than Klein & Manning’s Viterbi parser

---

**Are you going to make it faster? (yup!)**

- Currently rewriting the term classes to match hand-tuned code
- Will support “mix-and-match” implementation strategies
  - store X in an array
  - store Y in a hash
  - don’t store Z (compute on demand)
- Eventually, choose strategies automatically by execution profiling

---

**Synopsis: today’s idea → experimental results fast!**

- Dyna is a language for computation (no I/O).
- Especially good for dynamic programming.
- It tries to encapsulate the black art of NLP.

- Much prior work in this vein …
  - Deductive parsing schemata (preferably **weighted**)
    - Goodman, Nederhof, Pereira, Warren, Shieber, Schabes, Sikkel…
  - Deductive databases (preferably with aggregation)
    - Ramakrishnan, Zukowski, Freitag, Specht, Ross, Sagiv, …
  - Probabilistic programming languages (**implemented**)
    - Zhao, Sato, Pfeffer … (also: efficient Prologish languages)

---

**Contributors!**

- **Jason Eisner**, Eric Northup, Johnny Graettinger
  (compiler backend)
- **Noah A. Smith**, Markus Dreyer, David Smith
  (compiler frontend)
- **Eric Goldlust**, Mike Kombluh, George Shafer, Gordon Woodhull
  (visual debugger)
- **John Blatz**
  (program transformations)
- **Asheesh Laroia**
  (web services)

---

http://www.dyna.org