Dynamic Programming: Theory and Practice

Alexander Rush and Nathaniel Filardo
MASC 2013

October 11, 2013

So I've been working with a student.

So I've been working with a student.

And I ask her to build a dependency parser.

So I've been working with a student.

And I ask her to build a dependency parser.

"What should I use to write that?"

A Dynamic Programming Algorithm

- Base case definition: for all $i = 1 \ldots n$, for $X \in N$

$$\pi[i, i, X] = q(X \rightarrow w_i)$$

  (note: define $q(X \rightarrow w_i) = 0$ if $X \rightarrow w_i$ is not in the grammar)

- Recursive definition: for all $i = 1 \ldots n$, $j = (i + 1) \ldots n$, $X \in N$,

$$\pi(i, j, X) = \max_{\substack{X \rightarrow Y Z \in R, \\ s \in \{i \ldots (j-1)\}}} \left(q(X \rightarrow Y Z) \times \pi(i, s, Y) \times \pi(s + 1, j, Z) \right)$$

when I am in my advisor’s class

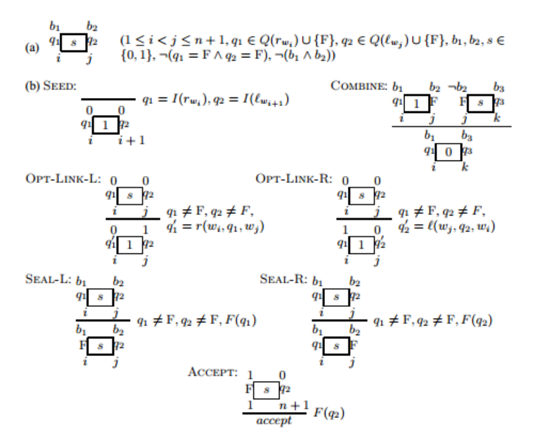


Figure 1.3 Declarative specification of an $O(n^3)$ algorithm. (a) Form of items in the parse chart. (b) Inference rules. As in Fig. 1.2b, F is a literal that means 'an unspecified final state.'

when I read the paper

```java
// static type for each edge: run time O(n^3 + Tn^2) T is number of types
public Object[][] decodeProjective(DependencyInstance inst,
                                   FeatureVector[][][] fvs,
                                   double[][][] probs,
                                   FeatureVector[][][][] nt_fvs,
                                   double[][][][] nt_probs, int K) {

    String[] forms = inst.forms;
    String[] pos = inst.postags;

    int[][] static_types = null;
    if(pipe.labeled) {
        static_types = getTypes(nt_probs,forms.length);
    }

    KBestParseForest pf = new KBestParseForest(0,forms.length-1,inst,K);

    for(int s = 0; s < forms.length; s++) {
        pf.add(s,-1,0,0.0,new FeatureVector());
        pf.add(s,-1,1,0.0,new FeatureVector());
    }

    for(int j = 1; j < forms.length; j++) {
        for(int s = 0; s < forms.length && s+j < forms.length; s++) {
            int t = s+j;

            FeatureVector prodFV_st = fvs[s][t][0];
            FeatureVector prodFV_ts = fvs[s][t][1];
            double prodProb_st = probs[s][t][0];
            double prodProb_ts = probs[s][t][1];

            int type1 = pipe.labeled ? static_types[s][t] : 0;
            int type2 = pipe.labeled ? static_types[t][s] : 0;

            FeatureVector nt_fv_s_01 = nt_fvs[s][type1][0][1];
            FeatureVector nt_fv_s_10 = nt_fvs[s][type2][1][0];
            FeatureVector nt_fv_t_00 = nt_fvs[t][type1][0][0];
            FeatureVector nt_fv_t_11 = nt_fvs[t][type2][1][1];
            double nt_prob_s_01 = nt_probs[s][type1][0][1];
            double nt_prob_s_10 = nt_probs[s][type2][1][0];
            double nt_prob_t_00 = nt_probs[t][type1][0][0];
            double nt_prob_t_11 = nt_probs[t][type2][1][1];

            double prodProb = 0.0;

            for(int r = s; r <= t; r++) {

                /** first is direction, second is complete*/
                /** _s means s is the parent*/
                if(r != t) {
```

when I look at the code

Question

We no longer manually implement

- Simplex for LPs
- SVM training for classification
- FST algorithms
- etc.

Question

We no longer manually implement

- Simplex for LPs
- SVM training for classification
- FST algorithms
- etc.

But I don't have a good answer for dynamic programming.

Outline

Four views of Dynamic Programming

1. Imperative k
2. Declarative
3. Graphical
4. Optimization

Dynamic Programming for NLP

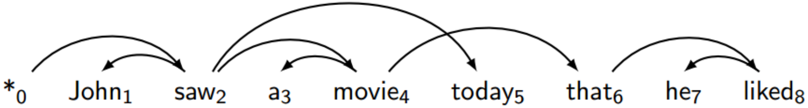
Decoding for NLP

- f; a scoring function.
- y; a possible output structure.

optimization problem:

$$\max_{y} f(y)$$

Example: Dependency Parsing



$$
\begin{aligned}
f(y) = \ & \mathit{score}(\text{head} = *_0, \text{mod} = \text{saw}_2) + \mathit{score}(\text{saw}_2, \text{John}_1) \\
& + \mathit{score}(\text{saw}_2, \text{movie}_4) + \mathit{score}(\text{saw}_2, \text{today}_5) \\
& + \mathit{score}(\text{movie}_4, \text{a}_3) + \ldots
\end{aligned}
$$

Dynamic Programming for Decoding

"Dynamic programming is a method for solving complex problems by breaking them down into simpler subproblem"

Traditionally defined as an optimization problem defined using the Bellman equations.

However these equations don't give use much insight into what to do in practice.

View 1: Imperative

In my advisor's class, we teach an imperative implementation of dynamic programming using charts.

In my advisor's class, we teach an imperative implementation of dynamic programming using charts.

Some students enjoy this, it correlates well with not having bugs.

```
Initialization: C[s][s][d][c] = 0.0    ∀s, d, c
for k : 1..n
  for s : 1..n
    t = s + k
    if t > n then break

    % First: create incomplete items
    C[s][t][←][0] = max_{s≤r<t} (C[s][r][→][1] + C[r+1][t][←][1] + s(t, s))    (*)
    C[s][t][→][0] = max_{s≤r<t} (C[s][r][→][1] + C[r+1][t][←][1] + s(s, t))

    % Second: create complete items
    C[s][t][←][1] = max_{s≤r<t} (C[s][r][←][1] + C[r][t][←][0])
    C[s][t][→][1] = max_{s<r≤t} (C[s][r][→][0] + C[r][t][→][1])

  end for
end for
```

Benefits

- Often fastest method in practice.
- Doesn't require any additional code or library.
- Great way to prove how hardcore you are.

Issue 1

- Debugging is very subtle.

$$C[s][t][\leftarrow][0] = \max_{s \le r < t}$$
$$C[s][t][\rightarrow][0] = \max_{s \le r < t}$$

% Second: create complete

$$C[s][t][\leftarrow][1] = \max_{s \le r < t}$$
$$C[s][t][\rightarrow][1] = \max_{s < r \le t}$$

Nikolay Borisov · 7 months ago

Thanks for your post, I'm also struggling with making all of this work :(. Here is what I have done so far. I have taken the test sentence:

```
STAT5A
mutations
in
the
Src
homology
2
(
SH2
)
and
SH3
domains
did
not
alter
the
BTK
-
mediated
tyrosine
phosphorylation
.
```

Sentence has 23 words so first one is at k = 0 and last one at k = 22 (it being the . character)

And when I run my code I get the following debug output:

```
k = 0 U = 0 V = 0
Calculating Pi[0, *, *] * q(0|*, *) * e(STAT5A | I_GENE)
 Taken max probability = 0.011542410287450856
k = 0 U = 0 V = 1
Calculating Pi[0, *, *] * q(1|*, *) * e(STAT5A | O)
 Taken max probability = 0.004527456817631497
k = 0 U = 1 V = 0
Calculating Pi[0, *, *] * q(0|*, *) * e(STAT5A | I_GENE)
 Taken max probability = 0.011542410287450856
k = 0 U = 1 V = 1
Calculating Pi[0, *, *] * q(1|*, *) * e(STAT5A | O)
 Taken max probability = 0.004527456817631497
```

Issue 2

- Optimized to the specific problem domain.

Cannot use MSTParser for POS tagging.

Issue 2

- Optimized to the specific problem domain.

Cannot use MSTParser for POS tagging.

Cannot really use MSTParser third-order parsing.

Issue 3

Cannot easily utilize dynamic programming extensions.

- Variant algorithms
  - K-Best
  - Loss-Augmented inference
  - Posterior Decoding

Issue 3

Cannot easily utilize dynamic programming extensions.

- Variant algorithms
  - K-Best
  - Loss-Augmented inference
  - Posterior Decoding

- Pruning algorithms
  - Max-Marginals
  - Coarse-to-Fine

Issue 3

Cannot easily utilize dynamic programming extensions.

- Variant algorithms
  - K-Best
  - Loss-Augmented inference
  - Posterior Decoding

- Pruning algorithms
  - Max-Marginals
  - Coarse-to-Fine

- Constrained algorithms
  - Beam Search
  - Dual Decomposition
  - Lagrangian Relaxation

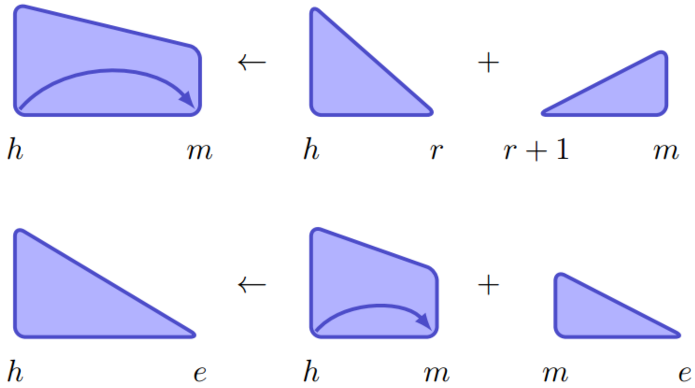
....

Opinion Slide

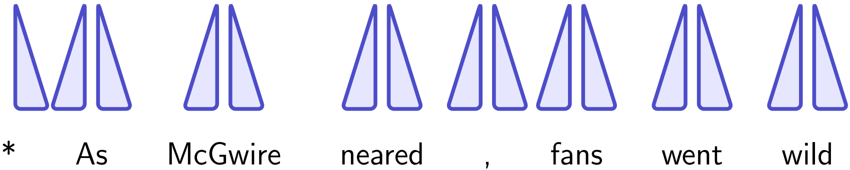
Do not write imperative dynamic programming code.

View 2: Declarative

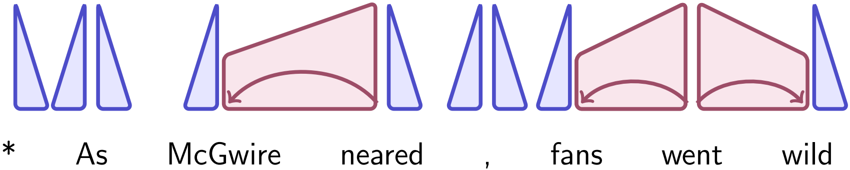
Dependency Parsing Setup



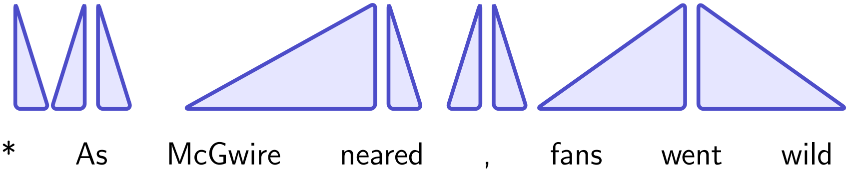
First-Order Parsing



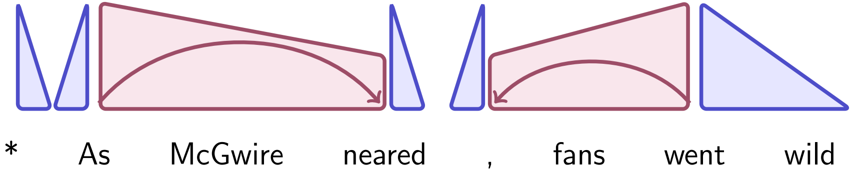
First-Order Parsing



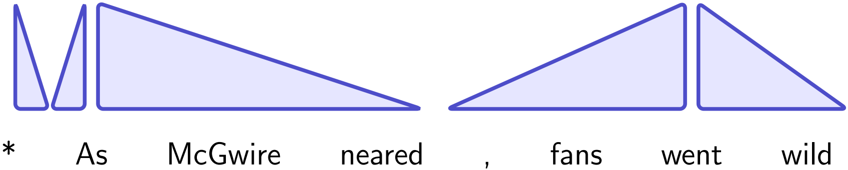
First-Order Parsing



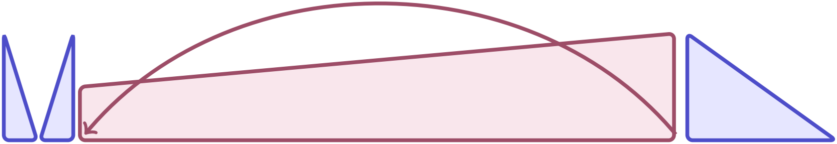
First-Order Parsing



First-Order Parsing

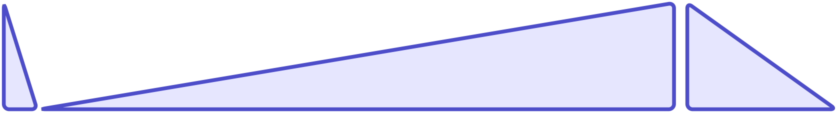


First-Order Parsing



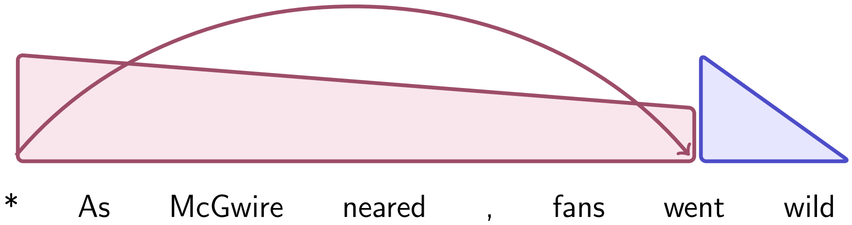
* As McGwire neared , fans went wild

First-Order Parsing

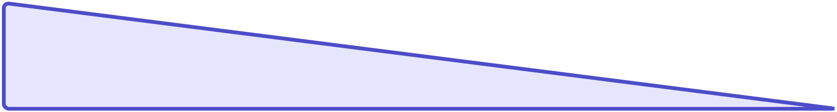


* As McGwire neared , fans went wild

First-Order Parsing



First-Order Parsing



* As McGwire neared , fans went wild

Declarative code is clean, often less buggy.

```
:- item(item, double, 0).
:- item([lhalf,rhalf,end], bool, false).
constit(A,H,H,H) += rewrite(A:D,D) if word(D,H).
left(A/C,I,J,H) += constit(B,I,H2,J) * rewrite(A:D,B:D2,C:D)
if lhalf(C,=J+1,H) & word(D,H) & word(D2,H2).
right(A\C,H,I,J) += constit(B,I,H2,J) * rewrite(A:D,C:D,B:D2)
if rhalf(C,H,=J-1) & word(D,H) & word(D2,H2).
lhalf(A,I,H) |= constit(A,I,H,J)!=0.
rhalf(A,H,J) |= constit(A,I,H,J)!=0.
constit(A,I,H,K) += left(A/C,I,J,H) * constit(C,=J+1,H,K).
constit(A,I,H,K) += constit(C,I,H,=J-1) * right(A\C,H,J,K).
goal += constit(s,1,_,N) if end(N).
```

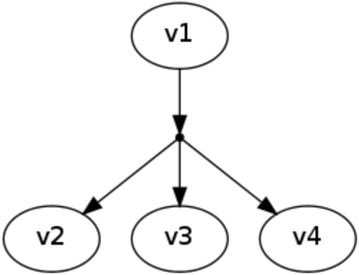
Declarative code is clean, often less buggy.

```
:- item(item, double, 0).
:- item([lhalf,rhalf,end], bool, false).
constit(A,H,H,H) += rewrite(A:D,D) if word(D,H).
left(A/C,I,J,H) += constit(B,I,H2,J) * rewrite(A:D,B:D2,C:D)
if lhalf(C,=J+1,H) & word(D,H) & word(D2,H2).
right(A\C,H,I,J) += constit(B,I,H2,J) * rewrite(A:D,C:D,B:D2)
if rhalf(C,H,=J-1) & word(D,H) & word(D2,H2).
lhalf(A,I,H) |= constit(A,I,H,J)!=0.
rhalf(A,H,J) |= constit(A,I,H,J)!=0.
constit(A,I,H,K) += left(A/C,I,J,H) * constit(C,=J+1,H,K).
constit(A,I,H,K) += constit(C,I,H,=J-1) * right(A\C,H,J,K).
goal += constit(s,1,_,N) if end(N).
```

I am not smart enough to make this efficient.

View 3: Hypergraph

Hyperedge

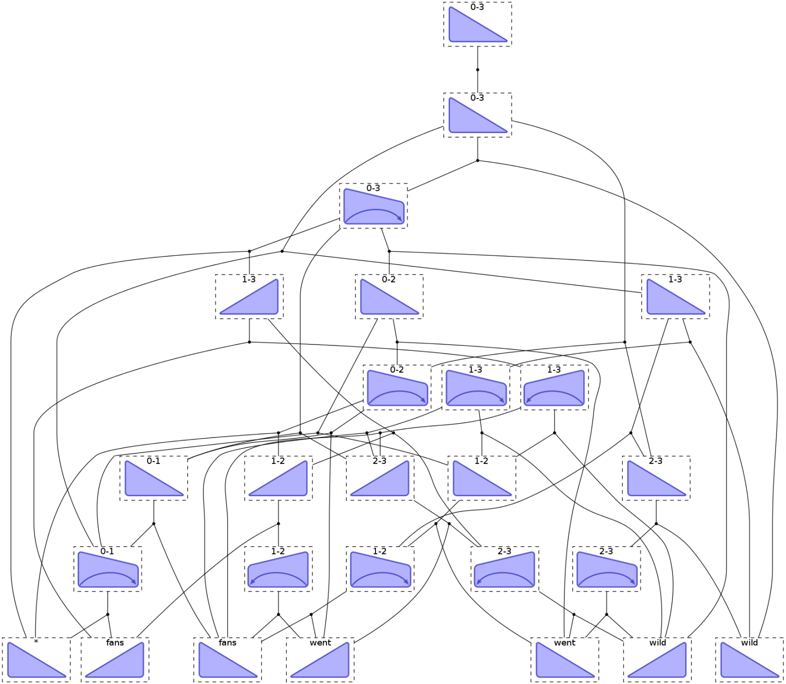


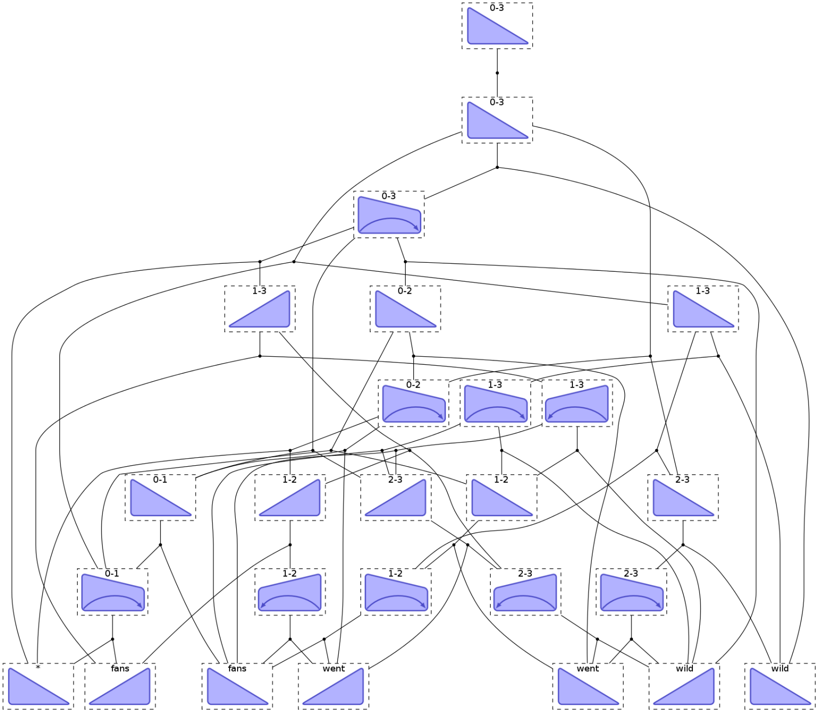
Hypergraph

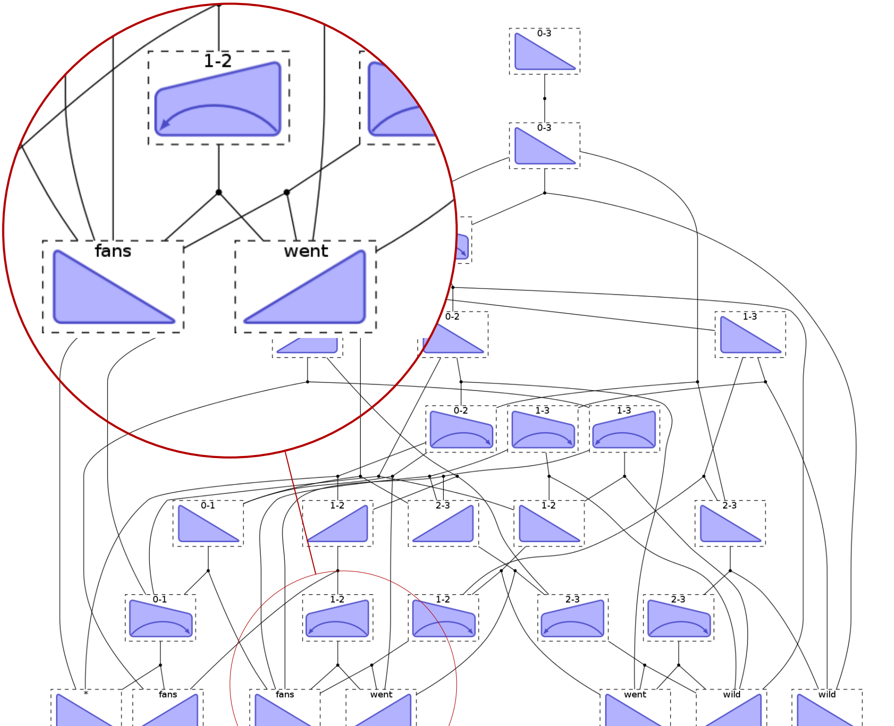
$\mathcal{V}$	set of vertices	1	root node
$\mathcal{E}$	set of hyperedges	$h(e)$	head vertex of edge
		$t(e)$	tail vertex of edge

- $\mathcal{X}$ - set of valid "hyperpaths" y
  - $y \in \{0,1\}^{|\mathcal{V}|+|\mathcal{E}|}$ has $y(v) = 1$ iff node v is in path

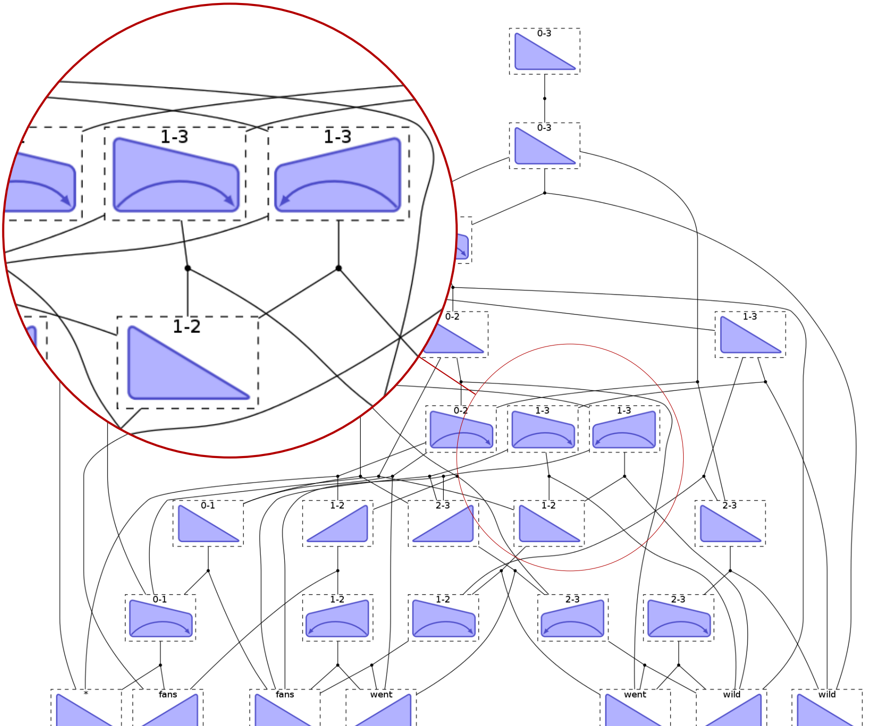
Any dynamic program can be represented as a hypergraph.


0-3
0-3
0-3
1-3
0-2
1-3
0-2
1-3
1-3
0-1
1-2
2-3
1-2
2-3
0-1
1-2
1-2
2-3
2-3
*
fans
fans
went
went
wild
wild


0-3
0-3
0-3
1-3
0-2
1-3
0-2
1-3
1-3
0-1
1-2
2-3
1-2
2-3
0-1
1-2
1-2
2-3
2-3
*
fans
fans
went
went
wild
wild



0-3
0-3
0-2
1-3
0-2
1-3
1-3
0-1
1-2
2-3
1-2
2-3
0-1
1-2
1-2
2-3
2-3
*
fans
fans
went
went
wild
wild
1-2
fans
went



0-3
0-3
1-3
1-3
1-2
0-2
1-3
0-2
1-3
1-3
0-1
1-2
2-3
1-2
2-3
0-1
1-2
1-2
2-3
2-3
*
fans
fans
went
went
wild
wild

Weights

- $\theta \in R^{|\mathcal{E}|}$ - the weights associated with each hyperedge.

Hypergraph problem:

$$\max_{y \in \mathcal{X}} \theta^\top y$$

Dynamic Programming on Hypergraphs

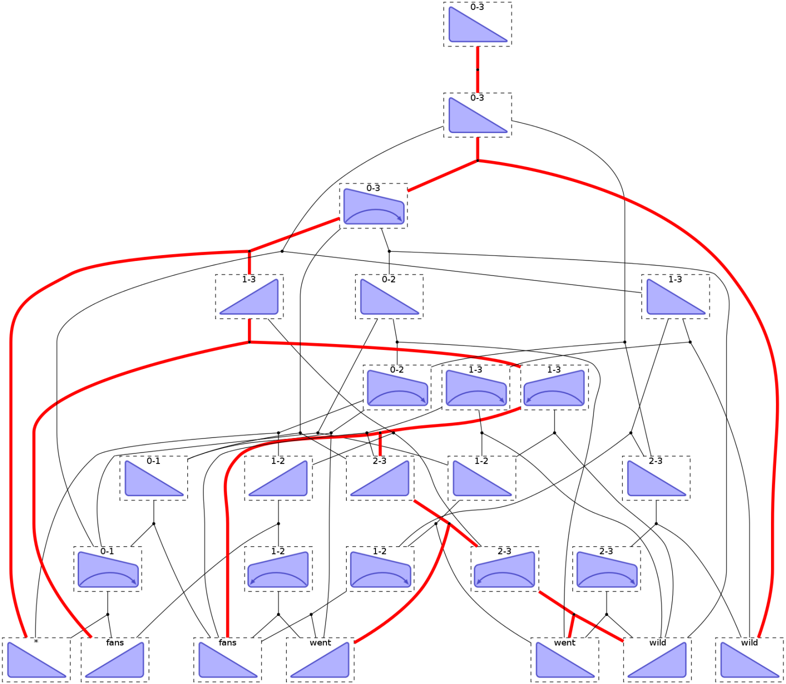
procedure $\textsc{BestPath}(\mathcal{V}, \mathcal{E}, \theta)$

  $\pi(v) \leftarrow 0$ for all terminals

  for $v \in \mathcal{V}$ in bottom-up order **do**

    $\pi(v) \leftarrow \max_{e \in \mathcal{E} : h(e) = v} \theta(e) + \pi(t(e))$

  return $\pi(1)$


0-3
0-3
0-3
1-3
0-2
1-3
0-2
1-3
1-3
0-1
1-2
2-3
1-2
2-3
0-1
1-2
1-2
2-3
2-3
*
fans
fans
went
went
wild
wild

procedure $\textsc{Outside}(\mathcal{V}, \mathcal{E}, \theta)$

$\rho[1] \leftarrow 0$

for $e \in \mathcal{E}$ in top-down order **do**

$\langle\langle v_2, \ldots, v_k\rangle, v_1\rangle \leftarrow e$

$$t \leftarrow \theta(e) + \sum_{i=2}^{k} \pi[v_i]$$

for $i = 2 \ldots k$ **do**

$s \leftarrow \rho[v_1] + t - \pi[v_i]$

if $s > \rho[v_i]$ **then** $\rho[v_i] \leftarrow s$

return ρ

Benefits

- Helps debugging, visualizing, pruning, and explaining.
- Can optimize the hell out of the inner loop code.
- Useful for other downstream tasks.

Caveats

- Need to create edges in memory.
- Requires construction, often still imperative.

View 4: Optimization

Linear Program

- Any dynamic program can be expressed as a linear program.
- Can be derived from the Bellman equations.
- Also directly from the hypergraph representation.

Linear Constraints Conversion

$$
\begin{aligned}
\mathcal{X} = \{y : y(1) &= 1, \\
y(v) &= \sum_{e : h(e) = v} y(e) \;\forall\; v \in \mathcal{V} \text{ except root}, \\
y(v) &= \sum_{e : t(e) = v} y(e) \;\forall\; v \in \mathcal{V} \text{ except terminals}\}
\end{aligned}
$$

Linear Constraints Conversion

$$\begin{aligned}\mathcal{X} = \{y : y(1) &= 1, \\ y(v) &= \sum_{e:h(e)=v} y(e) \; \forall \; v \in \mathcal{V} \text{ except root,} \\ y(v) &= \sum_{e:t(e)=v} y(e) \; \forall \; v \in \mathcal{V} \text{ except terminals}\}\end{aligned}$$

- $O(|\mathcal{V}|)$ constraints

Linear Program

$$\max_{y} \theta^{\top} y$$

$$\begin{array}{rcl}
y(1) & = & 1 \\
y(v) & = & \displaystyle\sum_{e:h(e)=v} y(e) \ \forall \ v \in \mathcal{V} \text{ except root} \\
y(v) & = & \displaystyle\sum_{e:t(e)=v} y(e) \ \forall \ v \in \mathcal{V} \text{ except terminals} \\
0 \leq & y(e) & \leq 1 \ \ \forall e \in \mathcal{E} \\
0 \leq & y(v) & \leq 1 \ \ \forall v \in \mathcal{V}
\end{array}$$

```
\* Hypergraph Problem *\
Minimize
OBJ: 0.865309927772 edge_0 + 0.805027827013 edge_1 + 0.719704686
 + 0.824844977148 edge_3 + 0.668153201232 edge_4 + 0.93283382422
 + 0.551267246091 edge_6
Subject To
_C1: node_13_tri_right_0_3 = 1
_C10: - edge_0 + node_1_tri_right_1_1 = 0
_C11: - edge_1 + node_2_tri_left_1_1 = 0
_C12: - edge_2 + node_3_tri_right_2_2 = 0
_C13: - edge_0 + node_4_tri_left_2_2 = 0
_C14: - edge_3 + node_5_tri_right_3_3 = 0
_C15: - edge_2 + node_6_tri_left_3_3 = 0
_C16: - edge_1 + node_7_trap_left_1_2 = 0
_C17: - edge_4 + node_8_tri_left_1_2 = 0
_C18: - edge_3 + node_9_trap_right_2_3 = 0

...
```

Benefits

- Can solve with general-purpose LP solver. (Gurobi)
- There are tricks for outside, K-Best, etc.
- Real benefit: Adding additional constraints.

Constrained Paths

problem: constrain maximization to valid paths

- $A \in^{|b| \times |\mathcal{E}|}$; a matrix of linear constraints
- $b \in^{|b|}$; a constraint vector

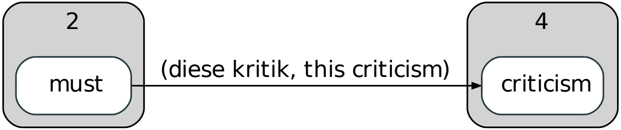
Constrained paths

$$\mathcal{X}' = \{y \in \mathcal{X} : Ay = b\}$$

Constrained best path

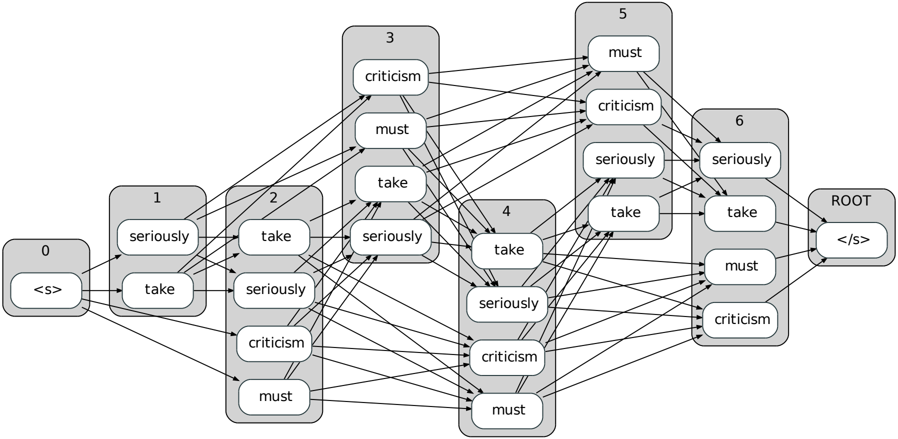
$$\max_{y \in \mathcal{X}'} \theta^\top y$$

Individual Edge (e)

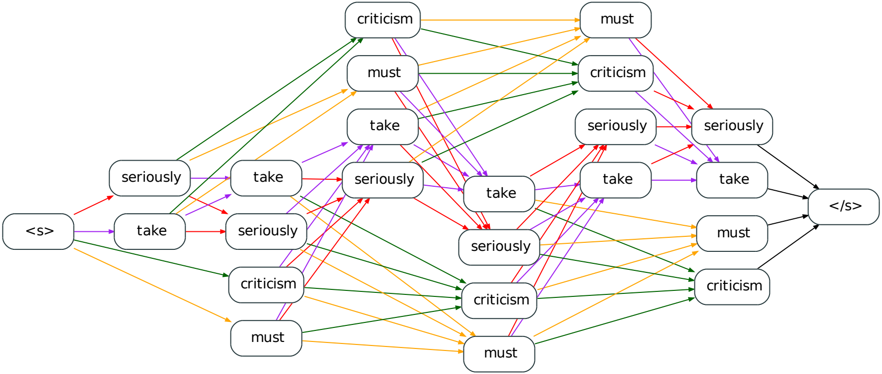


$$\begin{aligned}\theta(e) &= \text{score}(\text{diese kritik}, \text{this criticism}) + \\ &\quad \text{score}(\text{must}, \text{this}) + \text{score}(\text{this}, \text{criticism})\end{aligned}$$

Constraints



Constraints



Pitch: PyDecode

A pragmatic library for building dynamic programming.

- Easy interface in Python
- Hypergraph code in C++
  - Inside/Outside Viterbi
  - Pruning with max-marginals
  - Constraint specification
  - Dual Decomposition/Lagrangian Relaxation
  - Export to LP
- Visualization tools supporting IPython Notebook