Advanced Dynamic Programming in CL: Theory, Algorithms, and Applications

Liang Huang
University of Pennsylvania
A Little Bit of History...
A Little Bit of History...

- Who invented Dynamic Programming? and when was it invented?
A Little Bit of History...

- Who invented Dynamic Programming? and when was it invented?
  - R. Bellman (1940s-50s)
  - A. Viterbi (1967)
  - E. Dijkstra (1959)
  - Hart, Nilsson, and Raphael (1968)
    - Dijkstra => A* Algorithm
  - D. Knuth (1977)
    - Dijkstra on Grammar (Hypergraph)
A Little Bit of History...

- Who invented Dynamic Programming and when was it invented?
  - R. Bellman (1940s-50s)
  - A. Viterbi (1967)
  - E. Dijkstra (1959)
  - Hart, Nilsson, and Raphael (1968)
    - Dijkstra => A* Algorithm
  - D. Knuth (1977)
    - Dijkstra on Grammar (Hypergraph)
Dynamic Programming

- Dynamic Programming is everywhere in NLP
  - Viterbi Algorithm for Hidden Markov Models
  - CKY Algorithm for Parsing and Machine Translation
  - Forward-Backward and Inside-Outside Algorithms
- Also everywhere in AI/ML
  - Reinforcement Learning, Planning (POMDP)
  - AI Search: Uniform-cost, A*, etc.
- This tutorial: a unified theoretical view of DP
- Focusing on Optimization Problems
Review: DP Basics

- DP = Divide-and-Conquer + Two Principles:
  - [required] Optimal Subproblem Property
  - [recommended] Sharing of Common Subproblems

- Structure of the Search Space
  - Incremental
  - Graph
  - Knapsack, Edit Dist., Sequence Alignment

- Branching
  - Hypergraph
  - Matrix-Chain, Polygon Triangulation, Optimal BST
## Two Dimensional Survey

<table>
<thead>
<tr>
<th>Search Space</th>
<th>Traversing Order</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Graphs with semirings</strong></td>
<td><strong>Topological (acyclic)</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Best-first (superior)</strong></td>
</tr>
<tr>
<td></td>
<td>Viterbi</td>
</tr>
<tr>
<td></td>
<td>Dijkstra</td>
</tr>
<tr>
<td><strong>Hypergraphs with weight functions</strong></td>
<td>Generalized Viterbi</td>
</tr>
<tr>
<td></td>
<td>Knuth</td>
</tr>
</tbody>
</table>

Liang Huang (Penn)
Graphs in NLP

part-of-speech tagging

lattice in speech
Semirings on Graphs

- in a weighted graph, we need two operators:
  - **extension** (multiplicative) and **summary** (additive)
  - the weight of a path is the **product** of edge weights
  - the weight of a vertex is the **summary** of path weights

\[
\begin{align*}
    d(\pi_1) &= \bigotimes_{e_i \in \pi_1} w(e_i) = w(e_1) \otimes w(e_2) \otimes w(e_3) \\
    d(t) &= \bigoplus_{\pi_i} w(\pi_i) = w(p_1) \oplus w(p_2) \oplus \cdots
\end{align*}
\]
A monoid is a triple \((A, \otimes, 1)\) where

1. \(\otimes\) is a closed associative binary operator on the set \(A\),
2. \(1\) is the identity element for \(\otimes\), i.e., for all \(a \in A\), \(a \otimes 1 = 1 \otimes a = a\).

A monoid is commutative if \(\otimes\) is commutative.
A **monoid** is a triple \((A, \otimes, \bar{1})\) where

1. \(\otimes\) is a closed **associative binary operator** on the set \(A\),

2. \(\bar{1}\) is the **identity element** for \(\otimes\), i.e., for all \(a \in A\), \(a \otimes \bar{1} = \bar{1} \otimes a = a\).

A monoid is **commutative** if \(\otimes\) is commutative.

\[
\begin{align*}
([0, 1], +, 0) \\
([0, 1], \times, 1) \\
([0, 1], \text{max}, 0)
\end{align*}
\]
A **monoid** is a triple \((A, \otimes, 1)\) where

1. \(\otimes\) is a closed **associative binary operator** on the set \(A\),

2. \(1\) is the **identity element** for \(\otimes\), i.e., for all \(a \in A\), \(a \otimes 1 = 1 \otimes a = a\).

A monoid is **commutative** if \(\otimes\) is commutative.

A **semiring** is a 5-tuple \(R = (A, \oplus, \otimes, 0, 1)\) such that

1. \((A, \oplus, 0)\) is a commutative monoid.

2. \((A, \otimes, 1)\) is a monoid.

3. \(\otimes\) distributes over \(\oplus\): for all \(a, b, c\) in \(A\),

   \[
   (a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c),
   \]

   \[
   c \otimes (a \oplus b) = (c \otimes a) \oplus (c \otimes b).
   \]

4. \(0\) is an **annihilator** for \(\otimes\): for all \(a\) in \(A\), \(0 \otimes a = a \otimes 0 = 0\).

\([0, 1], +, 0\)
\([0, 1], \times, 1\)
\([0, 1], \text{max}, 0\)
A **monoid** is a triple \((A, \otimes, \bar{1})\) where

1. \(\otimes\) is a closed **associative binary operator** on the set \(A\),

2. \(\bar{1}\) is the **identity element** for \(\otimes\), i.e., for all \(a \in A\), \(a \otimes \bar{1} = \bar{1} \otimes a = a\).

A monoid is **commutative** if \(\otimes\) is commutative.

A **semiring** is a 5-tuple \(R = (A, \oplus, \otimes, 0, \bar{1})\) such that

1. \((A, \oplus, 0)\) is a commutative monoid.

2. \((A, \otimes, \bar{1})\) is a monoid.

3. \(\otimes\) distributes over \(\oplus\): for all \(a, b, c\) in \(A\),

\[
(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c),
\]

\[
c \otimes (a \oplus b) = (c \otimes a) \oplus (c \otimes b).
\]

4. \(0\) is an **annihilator** for \(\otimes\): for all \(a\) in \(A\), \(0 \otimes a = a \otimes 0 = 0\).
A **monoid** is a triple \((A, \otimes, \bar{1})\) where

1. \(\otimes\) is a closed **associative binary operator** on the set \(A\),
2. \(\bar{1}\) is the **identity element** for \(\otimes\), i.e., for all \(a \in A\), \(a \otimes \bar{1} = \bar{1} \otimes a = a\).

A monoid is **commutative** if \(\otimes\) is commutative.

A **semiring** is a 5-tuple \(R = (A, \oplus, \otimes, 0, \bar{1})\) such that

1. \((A, \oplus, 0)\) is a commutative monoid.
2. \((A, \otimes, \bar{1})\) is a monoid.
3. \(\otimes\) distributes over \(\oplus\): for all \(a, b, c\) in \(A\),
   \[
   (a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c),
   \]
   \[
   c \otimes (a \oplus b) = (c \otimes a) \oplus (c \otimes b).
   \]
4. \(0\) is an **annihilator** for \(\otimes\): for all \(a\) in \(A\), \(0 \otimes a = a \otimes 0 = 0\).
## Examples

<table>
<thead>
<tr>
<th>Semiring</th>
<th>Set</th>
<th>$\oplus$</th>
<th>$\otimes$</th>
<th>$\bar{0}$</th>
<th>$\bar{1}$</th>
<th>Intuition/Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>${0, 1}$</td>
<td>$\lor$</td>
<td>$\land$</td>
<td>0</td>
<td>1</td>
<td>Logical deduction, recognition</td>
</tr>
<tr>
<td>Viterbi</td>
<td>$[0, 1]$</td>
<td>max</td>
<td>$\times$</td>
<td>0</td>
<td>1</td>
<td>Prob. of the best derivation</td>
</tr>
<tr>
<td>Inside</td>
<td>$\mathbb{R}^+ \cup {+\infty}$</td>
<td>+</td>
<td>$\times$</td>
<td>0</td>
<td>1</td>
<td>Prob. of a string</td>
</tr>
<tr>
<td>Real</td>
<td>$\mathbb{R} \cup {+\infty}$</td>
<td>$\min$</td>
<td>+</td>
<td>$+\infty$</td>
<td>0</td>
<td>Shortest-distance</td>
</tr>
<tr>
<td>Tropical</td>
<td>$\mathbb{R}^+ \cup {+\infty}$</td>
<td>$\min$</td>
<td>+</td>
<td>$+\infty$</td>
<td>0</td>
<td>With non-negative weights</td>
</tr>
<tr>
<td>Counting</td>
<td>$\mathbb{N}$</td>
<td>+</td>
<td>$\times$</td>
<td>0</td>
<td>1</td>
<td>Number of paths</td>
</tr>
</tbody>
</table>

Liang Huang (Penn)
Ordering

- **idempotent**

  A semiring \((A, \oplus, \otimes, 0, 1)\) is **idempotent** if for all \(a\) in \(A\), \(a \oplus a = a\).
Ordering

- **idempotent**
  A semiring \((A, \oplus, \otimes, 0, 1)\) is **idempotent** if for all \(a\) in \(A\), \(a \oplus a = a\).

- **comparison**
  
  \[(a \leq b) \iff (a \oplus b = a)\] defines a partial ordering.

- **examples: boolean, viterbi, tropical, real, ...**
  
  \[
  \begin{align*}
  (\{0, 1\}, \lor, \land, 0, 1) & \quad (\mathbb{R}^+ \cup \{+\infty\}, \text{min}, +, +\infty, 0) \\
  ([0, 1], \max, \otimes, 0, 1) & \quad (\mathbb{R} \cup \{+\infty\}, \text{min}, +, +\infty, 0)
  \end{align*}
  \]
Ordering

• idempotent
  A semiring \((A, \oplus, \otimes, 0, 1)\) is **idempotent** if for all \(a\) in \(A\), \(a \oplus a = a\).

• comparison
  \((a \leq b) \iff (a \oplus b = a)\) defines a partial ordering.

• examples: boolean, viterbi, tropical, real, ...
  \(((\{0, 1\}, \lor, \land, 0, 1)\) \quad (\mathbb{R}^+ \cup \{+\infty\}, \text{min}, +, +\infty, 0)
  \[[0, 1], \text{max}, \otimes, 0, 1\) \quad (\mathbb{R} \cup \{+\infty\}, \text{min}, +, +\infty, 0)

• total-order for optimization problems
  A semiring is **totally-ordered** if \(\oplus\) defines a total ordering.

• examples: all of the above
Monotonicity
Monotonicity

- monotonicity
Let $K = (A, \oplus, \otimes, 0, 1)$ be a semiring, and $\leq$ a partial ordering over $A$. We say $K$ is \textbf{monotonic} if for all $a, b, c \in A$

$$
(a \leq b) \Rightarrow (a \otimes c \leq b \otimes c) \quad (a \leq b) \Rightarrow (c \otimes a \leq c \otimes b)
$$
Monotonicity

- monotonicity

Let $K = (A, \oplus, \otimes, 0, 1)$ be a semiring, and $\leq$ a partial ordering over $A$. We say $K$ is **monotonic** if for all $a, b, c \in A$

$$(a \leq b) \Rightarrow (a \otimes c \leq b \otimes c) \quad (a \leq b) \Rightarrow (c \otimes a \leq c \otimes b)$$

- optimal substructure in dynamic programming
Monotonicity

- monotonicity

Let $K = (A, \oplus, \otimes, 0, 1)$ be a semiring, and $\leq$ a partial ordering over $A$. We say $K$ is monotonic if for all $a, b, c \in A$

$$(a \leq b) \implies (a \otimes c \leq b \otimes c) \quad (a \leq b) \implies (c \otimes a \leq c \otimes b)$$

- optimal substructure in dynamic programming
Monotonicity

- **monotonicity**

  Let $K = (A, \oplus, \otimes, 0, 1)$ be a semiring, and $\leq$ a partial ordering over $A$. We say $K$ is **monotonic** if for all $a, b, c \in A$

  $$(a \leq b) \Rightarrow (a \otimes c \leq b \otimes c) \quad (a \leq b) \Rightarrow (c \otimes a \leq c \otimes b)$$

- **optimal substructure** in dynamic programming

  ![Diagram]

  Let $K = (A, \oplus, \otimes, 0, 1)$ be a semiring, and $\leq$ a partial ordering over $A$. We say $K$ is **monotonic** if for all $a, b, c \in A$
Monotonicity

- **monotonicity**
  Let $K = (A, \oplus, \otimes, \bar{0}, \bar{1})$ be a semiring, and $\leq$ a partial ordering over $A$. We say $K$ is **monotonic** if for all $a, b, c \in A$

  $$(a \leq b) \Rightarrow (a \otimes c \leq b \otimes c) \quad (a \leq b) \Rightarrow (c \otimes a \leq c \otimes b)$$

- **optimal substructure** in dynamic programming

- **idempotent** $\Rightarrow$ **monotone** (from distributivity)
Monotonicity

- **monotonicity**
  
  Let $K = (A, \oplus, \otimes, 0, 1)$ be a semiring, and $\leq$ a partial ordering over $A$. We say $K$ is **monotonic** if for all $a, b, c \in A$
  
  $$ (a \leq b) \Rightarrow (a \otimes c \leq b \otimes c) \quad (a \leq b) \Rightarrow (c \otimes a \leq c \otimes b) $$

- **optimal substructure** in dynamic programming

  \[
  \begin{align*}
  A: b \otimes c &&& B: b \otimes c \\
  B: b &&& C: c \\
  \end{align*}
  \]

- **idempotent** $\Rightarrow$ **monotone** (from distributivity)
  - $(a+b) \otimes c = (a \otimes c) + (b \otimes c)$; if $a \leq b$, $(a \otimes c) = (a \otimes c) + (b \otimes c)$
Monotonicity

- **monotonicity**
  
  Let $K = (A, \oplus, \otimes, 0, 1)$ be a semiring, and $\leq$ a partial ordering over $A$. We say $K$ is **monotonic** if for all $a, b, c \in A$
  
  $$(a \leq b) \Rightarrow (a \otimes c \leq b \otimes c) \quad (a \leq b) \Rightarrow (c \otimes a \leq c \otimes b)$$

- **optimal substructure** in dynamic programming

- **idempotent** $\Rightarrow$ monotone (from distributivity)
  
  - $(a+b) \otimes c = (a \otimes c) + (b \otimes c)$; if $a \leq b$, $(a \otimes c) = (a \otimes c) + (b \otimes c)$
  
  - by def. of comparison, $a \otimes c \leq b \otimes c$
DP on Graphs

- optimization problems on graphs
  => generic shortest-path problem

- weighted directed graph $G=(V, E)$ with a function $w$ that assigns each edge a weight from a semiring

- compute the best weight of the target vertex $t$

- generic update along edge $(u, v)$

  $$d(v) \oplus = d(u) \otimes w(u, v)$$

  $d(v) \leftarrow d(v) \oplus (d(u) \otimes w(u, v))$

- how to avoid cyclic updates?
  - only update when $d(u)$ is fixed
# Two Dimensional Survey

<table>
<thead>
<tr>
<th>Search Space</th>
<th>Traversing Order</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Topological (acyclic)</strong></td>
<td>-</td>
</tr>
<tr>
<td><strong>Best-first (superior)</strong></td>
<td>-</td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Graphs with semirings (e.g., FSMs)</strong></td>
<td>Viterbi</td>
</tr>
<tr>
<td><strong>Hypergraphs with weight functions (e.g., CFGs)</strong></td>
<td>Generalized Viterbi</td>
</tr>
</tbody>
</table>
Viterbi Algorithm for DAGs

1. topological sort

2. visit each vertex $v$ in sorted order and do updates
   - for each incoming edge $(u, v)$ in $E$
   - use $d(u)$ to update $d(v)$: $d(v) \oplus = d(u) \otimes w(u, v)$
   - key observation: $d(u)$ is fixed to optimal at this time

   ![Diagram]

- time complexity: $O(V + E)$
Variant 1: forward-update

1. topological sort

2. visit each vertex $v$ in sorted order and do updates
   - for each outgoing edge $(v, u)$ in $E$
   - use $d(v)$ to update $d(u)$: $d(u) \oplus = d(v) \otimes w(v, u)$
   - key observation: $d(v)$ is fixed to optimal at this time

- time complexity: $O(V + E)$
Examples

- [Number of Paths in a DAG]
Examples

- [Number of Paths in a DAG]
  - just use the counting semiring \((\mathbb{N}, +, \times, 0, 1)\)
  - note: this is not an optimization problem!
Examples

- [Number of Paths in a DAG]
  - just use the counting semiring \((\mathbb{N}, +, \times, 0, 1)\)
  - note: this is not an optimization problem!

- [Longest Path in a DAG]
Examples

- [Number of Paths in a DAG]
  - just use the counting semiring \((\mathbb{N}, +, \times, 0, 1)\)
  - note: this is not an optimization problem!

- [Longest Path in a DAG]
  - just use the semiring \((\mathbb{R} \cup \{-\infty\}, \max, +, -\infty, 0)\)
Examples

- [Number of Paths in a DAG]
  - just use the counting semiring \((\mathbb{N}, +, \times, 0, 1)\)
  - note: this is not an optimization problem!

- [Longest Path in a DAG]
  - just use the semiring \((\mathbb{R} \cup \{-\infty\}, \max, +, -\infty, 0)\)

- [Part-of-Speech Tagging with a Hidden Markov Model]
Examples

- [Number of Paths in a DAG]
  - just use the counting semiring \((\mathbb{N}, +, \times, 0, 1)\)
  - note: this is **not** an optimization problem!

- [Longest Path in a DAG]
  - just use the semiring \((\mathbb{R} \cup \{-\infty\}, \max, +, -\infty, 0)\)

- [Part-of-Speech Tagging with a Hidden Markov Model]
Example: Speech Alignment

Time complexity: $O(n^2)$

also used in:
edit distance
biological sequence alignment
Example: Word Alignment

<table>
<thead>
<tr>
<th></th>
<th>I</th>
<th>love</th>
<th>you</th>
<th>.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Je</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t'</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>aime</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- key difference
- reorderings in translation!
- sequence/speech alignment is always monotonic
- complexity under HMM
- word alignment is \(O(n^3)\)
  - for every \((i, j)\)
    - enumerate all \((i-1, k)\)
- sequence alignment \(O(n^2)\)

I love you.

Je t'aime.
Chinese Word Segmentation

下雨天地面积水
xia yu tian di mian ji shui
Chinese Word Segmentation

民主
min-zhu
people-dominate

“democracy”
Chinese Word Segmentation

民主 min-zhu people-dominate "democracy"

江泽民 主席 jiang-ze-min zhu-xi "President Jiang Zemin"

下雨 天地面积 水 xia yu tian di mian ji shui
Chinese Word Segmentation

"democracy"

people-dominate

min-zhu

this was 5 years ago.

now Google is good at segmentation!

“President Jiang Zemin”

xia yu tian di mian ji shui

江泽民 主席

jiang-ze-min zhu-xi

... - ... - people dominate-podium

民主

people-dominate

min-zhu

Google

this was 5 years ago.

Liang Huang (Penn)

Dynamic Programming
Chinese Word Segmentation

“democracy”

people-dominate

min-zhu

this was 5 years ago.

now Google is good at segmentation!

“President Jiang Zemin”

“江泽民 主席”

jiang-ze-min  zhu-xi

... - ... - people  dominate-podium

people-dominate

min-zhu

now Google is good at segmentation!

“民主”

xia  yu  tian  di  mian  ji  shui

“下雨天气地面积水”

xia  yu  tian  di  mian  ji  shui

xia  yu  tian  di  mian  ji  shui

xia  yu  tian  di  mian  ji  shui
Chinese Word Segmentation

“democracy”

min-zhu people-dominate

Google

this was 5 years ago.

now Google is good at segmentation!

“President Jiang Zemin”

jiang-ze-min zhu-xi people-dominate

“..... people dominate-podium”

xia yu tian di mian ji shui

graph search
yu Shalong held a talk with Sharon

held a talk with Sharon

yu Shalong juxing le huitan

juxing le huitan held a talk with Sharon
Huang and Chiang

Phrase-based Decoding

与 沙龙 举行 了 会谈

yu Shalong juxing le huitan

held a talk with Sharon

with Sharon held talks

with SHalonl juxing le huitan

Yu Shalong juxing le huitan

Huang and Chiang
Huang and Chiang

Phrase-based Decoding

与 沙龙 举行 了 会谈
yu Shalong juxing le huitan

held a talk with Sharon

_ _ _ _ _

with Sharon held talks

with

yu Shalong juxing le huitan

held a talk
Huang and Chiang

Phrase-based Decoding

yu Shalong

held a talk with Sharon

juxing le huitan

-talks

with Sharon held

yu Shalong juxing le huitan

Forest Rescoring 20
Huang and Chiang

Phrase-based Decoding

yu Shalong juxing le huitan

held a talk with Sharon

with Sharon held talks

yu Shalong juxing le huitan
Huang and Chiang

Forest Rescoring

Phrase-based Decoding

与 沙龙 举行 了 会谈

yu Shalong juxing le huitan

held a talk with Sharon

held a talk with Sharon

yu Shalong juxing le huitan

with Sharon held talks
Phrase-based Decoding

 Held a talk with Sharon

source-side: coverage vector

held a talk

target-side: grow hypotheses

strictly left-to-right

Space: $O(2^n)$, time: $O(2^n n^2)$ -- cf. traveling salesman problem
Traveling Salesman Problem & MT

- a classical NP-hard problem
  - goal: visit each city once and only once
- exponential-time dynamic programming
  - state: cities visited so far (bit-vector)
  - search in this $O(2^n)$ transformed graph
- MT: each city is a source-language word
  - restrictions in reordering can reduce complexity => distortion limit
  - => syntax-based MT

(Held and Karp, 1962; Knight, 1999)
Traveling Salesman Problem & MT

- a classical NP-hard problem
- goal: visit each city once and only once
- exponential-time dynamic programming
  - state: cities visited so far (bit-vector)
  - search in this $O(2^n)$ transformed graph
- MT: each city is a source-language word
  - restrictions in reordering can reduce complexity $\Rightarrow$ distortion limit
  - $\Rightarrow$ syntax-based MT

(Held and Karp, 1962; Knight, 1999)
Adding a Bigram Model

- “refined” graph: annotated with language model words
- still dynamic programming, just larger search space

[Huang and Chiang](#)
Adding a Bigram Model

- “refined” graph: annotated with language model words
- still dynamic programming, just larger search space
Adding a Bigram Model

- “refined” graph: annotated with language model words
- still dynamic programming, just larger search space

bigram

... meeting

... talk

with Sharon

... talks
Adding a Bigram Model

- "refined" graph: annotated with language model words
- still dynamic programming, just larger search space

![Diagram showing bigram model]

- space: $O(2^n)$, time: $O(2^n n^2)$
- $\Rightarrow$ space: $O(2^n V^{m-1})$, time: $O(2^n V^{m-1} n^2)$

for $m$-gram language models
**Two Dimensional Survey**

<table>
<thead>
<tr>
<th>Search Space</th>
<th>Traversing Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>graphs with semirings (e.g., FSMs)</td>
<td>topological (acyclic)</td>
</tr>
<tr>
<td>hypergraphs with weight functions (e.g., CFGs)</td>
<td>best-first (superior)</td>
</tr>
</tbody>
</table>

- **topological (acyclic)**: Viterbi
- **best-first (superior)**: Dijkstra
- **Generalized Viterbi**: Knuth
Dijkstra Algorithm

\[ d(u) \xrightarrow{w(e)} d(u) \otimes w(e) \]
Dijkstra Algorithm

- Dijkstra does not require acyclicity
  - instead of topological order, we use best-first order
- but this requires \textit{superiority} of the semiring

Let $K = (A, \oplus, \otimes, 0, 1)$ be a semiring, and $\leq$ a partial ordering over $A$. We say $K$ is \textit{superior} if for all $a, b \in A$

$$a \leq a \otimes b, \quad b \leq a \otimes b.$$ 

- intuition: combination always gets worse
Dijkstra Algorithm

- Dijkstra does not require acyclicity
  - instead of topological order, we use **best-first** order
- but this requires **superiority** of the semiring
  
  Let $K = (A, \oplus, \otimes, 0, 1)$ be a semiring, and $\leq$ a partial ordering over $A$. We say $K$ is **superior** if for all $a, b \in A$

  $$a \leq a \otimes b, \quad b \leq a \otimes b.$$ 

- intuition: combination always gets worse

- contrast: monotonicity: combination preserves order
  
  $$\left( a \leq b \right) \Rightarrow \left( a \otimes c \leq b \otimes c \right)$$

\[d(u) \xrightarrow{w(e)} d(u) \otimes w(e)\]
Dijkstra Algorithm

- Dijkstra does not require acyclicity
  - instead of topological order, we use best-first order
  - but this requires *superiority* of the semiring

Let $K = (A, \oplus, \otimes, \overline{0}, \overline{1})$ be a semiring, and $\leq$ a partial ordering over $A$.
We say $K$ is *superior* if for all $a, b \in A$

$$a \leq a \otimes b, \quad b \leq a \otimes b.$$ 

- intuition: combination always gets worse

- contrast: monotonicity: combination preserves order

$$(a \leq b) \Rightarrow (a \otimes c \leq b \otimes c)$$

$$d(u) \otimes w(e)$$

$d(u) \otimes w(e)$

Liang Huang (Penn) 26
Dijkstra Algorithm

- Dijkstra does not require acyclicity
  - instead of topological order, we use best-first order
- but this requires superiority of the semiring

Let $K = (A, \oplus, \otimes, \overline{0}, \overline{1})$ be a semiring, and $\leq$ a partial ordering over $A$. We say $K$ is superior if for all $a, b \in A$

$$a \leq a \otimes b, \quad b \leq a \otimes b.$$ 

- intuition: combination always gets worse
- contrast: monotonicity: combination preserves order

$$(a \leq b) \Rightarrow (a \otimes c \leq b \otimes c)$$

$$(\{0, 1\}, \lor, \land, 0, 1) \checkmark$$
$$(\overline{[0, 1]}, \max, \times, 0, 1) \checkmark$$
$$(\mathbb{R}^+ \cup \{+\infty\}, \min, +, +\infty, 0) \checkmark$$
$$(\mathbb{R} \cup \{+\infty\}, \min, +, +\infty, 0) \times$$
Dijkstra Algorithm

- keep a cut \((S : V - S)\) where \(S\) vertices are fixed
- maintain a priority queue \(Q\) of \(V - S\) vertices
- each iteration choose the best vertex \(v\) from \(Q\)
- move \(v\) to \(S\), and use \(d(v)\) to forward-update others

\[
d(u) \oplus = d(v) \otimes w(v, u)
\]

Time complexity:
- \(O((V+E) \lg V)\) (binary heap)
- \(O(V \lg V + E)\) (fib. heap)
Dijkstra Algorithm

- keep a cut \((S : V - S)\) where \(S\) vertices are fixed
- maintain a priority queue \(Q\) of \(V - S\) vertices
- each iteration choose the best vertex \(v\) from \(Q\)
- move \(v\) to \(S\), and use \(d(v)\) to forward-update others

\[ d(u) \oplus = d(v) \otimes w(v, u) \]

Time complexity:
- \(O((V+E) \lg V)\) (binary heap)
- \(O(V \lg V + E)\) (fib. heap)
Dijkstra Algorithm

- keep a cut \((S : V - S)\) where \(S\) vertices are fixed
- maintain a priority queue \(Q\) of \(V - S\) vertices
- each iteration choose the best vertex \(v\) from \(Q\)
- move \(v\) to \(S\), and use \(d(v)\) to forward-update others

\[
\begin{align*}
d(u) & = d(v) \odot w(v, u) \\
\text{time complexity:} & \quad \begin{cases} 
O((V+E) \lg V) \text{ (binary heap)} \\
O(V \lg V + E) \text{ (fib. heap)}
\end{cases}
\end{align*}
\]
Viterbi vs. Dijkstra

- structural vs. algebraic constraints
- Dijkstra only applicable to optimization problems

monotonic optimization problems
Viterbi vs. Dijkstra

- structural vs. algebraic constraints
- Dijkstra only applicable to optimization problems

monotonic optimization problems

acyclic:
- Viterbi
Viterbi vs. Dijkstra

- structural vs. algebraic constraints
- Dijkstra only applicable to optimization problems
Viterbi vs. Dijkstra

- structural vs. algebraic constraints
- Dijkstra only applicable to optimization problems

monotonic optimization problems

acyclic: Viterbi

many NLP problems

superior: Dijkstra
• structural vs. algebraic constraints

• Dijkstra only applicable to optimization problems

Viterbi vs. Dijkstra

monotonic optimization problems

acyclic: Viterbi

many NLP problems

superior: Dijkstra

forward-backward (Inside semiring)
Viterbi vs. Dijkstra

• structural vs. algebraic constraints

• Dijkstra only applicable to optimization problems

monotonic optimization problems

acyclic: Viterbi

many NLP problems

superior: Dijkstra

forward-backward (Inside semiring)

non-probabilistic models
Viterbi vs. Dijkstra

- structural vs. algebraic constraints
- Dijkstra only applicable to optimization problems

Monotonic optimization problems

Acyclic: Viterbi

Many NLP problems

Superior: Dijkstra

Non-probabilistic models

Cyclic FSMs/grammars

Forward-backward (Inside semiring)
What if both fail?

monotonic optimization problems

acyclic: Viterbi

many NLP problems
superior: Dijkstra

generalized Bellman-Ford
(CLR, 1990; Mohri, 2002)

or, first do strongly-connected components (SCC)
which gives a DAG; use Viterbi globally on this SCC-DAG;
use Bellman-Ford locally within each SCC
What if both work?

monotonic optimization problems

acyclic: Viterbi

many NLP problems

superior: Dijkstra

full Dijkstra is slower than Viterbi

\[ O((V + E) \lg V) \quad \text{vs.} \quad O(V + E) \]

but it can finish as early as the target vertex is popped

\[ a \times (V + E) \lg V \quad \text{vs.} \quad V + E \]

Q: how to (magically) reduce \(a\)?
A* Search: Intuition

- Dijkstra is “blind” about how far the target is
  - may get “trapped” by obstacles
- can we be more intelligent about the future?
- idea: prioritize by $s-v$ distance + $v-t$ estimate
A* Search: Intuition

- Dijkstra is “blind” about how far the target is
  - may get “trapped” by obstacles
  - can we be more intelligent about the future?
- idea: prioritize by $s-v$ distance + $v-t$ estimate
A* Search: Intuition

- Dijkstra is “blind” about how far the target is
  - may get “trapped” by obstacles
- can we be more intelligent about the future?
- idea: prioritize by $s-v$ distance + $v-t$ estimate
**A* Heuristic**

- $h(v)$: the distance from $v$ to target $t$
- $\hat{h}(v)$ must be an **optimistic** estimate of $h(v)$: $\hat{h}(v) \leq h(v)$
- Dijkstra is a special case where $\hat{h}(v) = \bar{1}$ (0 for dist.)
- now, prioritize the queue by $d(v) \otimes \hat{h}(v)$
- can stop when target gets popped -- why?
  - optimal subpaths should pop earlier than non-optimal
    - $d(v) \otimes \hat{h}(v) \leq d(v) \otimes h(v) \leq d(t) \leq \text{non-optimal paths of } t$
How to design a heuristic?

- more of an art than science
- basic idea: projection into coarser space
- cluster: \( w'(U,V) = \min \{ w(u, v) \mid u \in U, v \in V \} \)
- exact cost in coarser graph is estimate of finer graph
How to design a heuristic?

- more of an art than science
- basic idea: projection into coarser space
- cluster: \( w'(U, V) = \min \{ w(u, v) \mid u \in U, v \in V \} \)
- exact cost in coarser graph is estimate of finer graph

(Raphael, 2001)
Viterbi or A*?

- A* intuition: \( d(t) \otimes \hat{h}(t) \) ranks higher among \( d(v) \otimes \hat{h}(v) \)
  - can finish early if lucky
  - actually, \( d(t) \otimes \hat{h}(t) = d(t) \otimes h(t) = d(t) \otimes \hat{t} = d(t) \)
- with the price of maintaining priority queue - \( O(\log V) \)
- Q: how early? worth the price?
- if the rank is \( r \), then A* is better when \( r/V \log V < 1 \)
Viterbi or A*?

- A* intuition: $d(t) \otimes \hat{h}(t)$ ranks higher among $d(v) \otimes \hat{h}(v)$
- can finish early if lucky
- actually, $d(t) \otimes \hat{h}(t) = d(t) \otimes h(t) = d(t) \otimes \bar{I} = d(t)$
- with the price of maintaining priority queue - $O(\log V)$
- Q: how early? worth the price?
- if the rank is $r$, then A* is better when $r/V \log V < 1$

$r < V / \log V$
### Two Dimensional Survey

<table>
<thead>
<tr>
<th>Search Space</th>
<th>Traversing Order</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Graphs with Semirings</strong> (e.g., FSMs)</td>
<td>Topological (acyclic)</td>
</tr>
<tr>
<td></td>
<td>Viterbi</td>
</tr>
<tr>
<td><strong>Hypergraphs with Weight Functions</strong> (e.g., CFGs)</td>
<td>Generalized Viterbi</td>
</tr>
</tbody>
</table>
# Two Dimensional Survey

<table>
<thead>
<tr>
<th>Search Space</th>
<th>Traversing Order</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>graphs with semirings</strong></td>
<td>Viterbi</td>
</tr>
<tr>
<td>(e.g., FSMs)</td>
<td>Dijkstra</td>
</tr>
<tr>
<td><strong>hypergraphs with weight functions</strong></td>
<td>Generalized Viterbi</td>
</tr>
<tr>
<td>(e.g., CFGs)</td>
<td>Knuth</td>
</tr>
</tbody>
</table>

**traversing order**
- topological (acyclic)
- best-first (superior)
Background: CFG and Parsing

- For each diff ($\leq n$)
  - For each $i$ ($\leq n$)
    - For each rule $X \rightarrow Y Z$
      - For each split point $k$
        $$\text{score}[X][i][j] = \max \text{score}[X][i][j],$$
        $$\text{score}[X\rightarrow YZ] \times$$
        $$\text{score}[Y][i][k] \times$$
        $$\text{score}[Z][k][j]$$
For each diff (<= n)
  For each i (<= n)
    For each rule X → Y Z
      For each split point k
        \[
        \text{score}[X][i][j] = \max \text{score}[X][i][j],
        \text{score}(X→YZ) \times
        \text{score}[Y][i][k] \times
        \text{score}[Z][k][j]
        \]

(S, 0, n)
Background: CFG and Parsing

- For each diff ($\leq n$)
  - For each $i$ ($\leq n$)
    - For each rule $X \rightarrow Y Z$
      - For each split point $k$
        \[
        \text{score}[X][i][j] = \max \text{score}[X][i][j],
        \text{score}(X\rightarrowYZ) \times
        \text{score}[Y][i][k] \times
        \text{score}[Z][k][j]
        \]
For each diff (\(<= n\))
  - For each i (\(<= n\))
    - For each rule \(X \rightarrow YZ\)
      - For each split point \(k\)
        \[
        \text{score}[X][i][j] = \max \text{score}[X][i][j], \text{score}(X \rightarrow YZ) \times \text{score}[Y][i][k] \times \text{score}[Z][k][j]
        \]

\(w_0 \ w_1 \ ... \ w_{n-1}\)
(Directed) Hypergraphs

- a generalization of graphs
  - edge => hyperedge: several vertices to one vertex
  - $e = (T(e), h(e), f_e)$. arity $|e| = |T(e)|$
  - a totally-ordered weight set $R$
    - we borrow the $\oplus$ operator to be the comparison
  - weight function $f_e : R^{|e|} \rightarrow R$
    - generalizes the $\otimes$ operator in semirings
  - simple case: $f_e(a, b) = a \otimes b \otimes w(e)$

$d(v) \oplus = f_e(d(u_1), d(u_2))$
Hypergraphs and Deduction

\[(B, i, k) \times (C, k, j) \times \Pr(A \rightarrow B C)\]

(Nederhof, 2003)
Hypergraphs and Deduction

\[(A, i, j)\]

\[
\begin{align*}
(B, i, k) & : a \\
(C, k, j) & : b \\
\text{fe} & : a \times b \times \Pr(A \rightarrow B C)
\end{align*}
\]

\[(B, i, k) \quad (C, k, j) \Rightarrow (A, i, j) \quad A \rightarrow B C\]

\[(Nederhof, 2003)\]

\[\text{tails} \quad \text{head} \]

\[
\begin{align*}
\text{u}_1 & : a \\
\text{u}_2 & : b \\
\text{v} & : \text{fe} (a,b)
\end{align*}
\]

\[\text{antecedents} \quad \text{consequent} \]

\[
\begin{align*}
\text{u}_1 & : a \\
\text{u}_2 & : b \\
\text{v} & : \text{fe} (a,b)
\end{align*}
\]
## Related Formalisms

<table>
<thead>
<tr>
<th>hypergraph</th>
<th>AND/OR graph</th>
<th>context-free grammar</th>
<th>deductive system</th>
</tr>
</thead>
<tbody>
<tr>
<td>vertex</td>
<td>OR-node</td>
<td>symbol</td>
<td>item</td>
</tr>
<tr>
<td>source-vertex</td>
<td>leaf OR-node</td>
<td>terminal</td>
<td>axiom</td>
</tr>
<tr>
<td>target-vertex</td>
<td>root OR-node</td>
<td>start symbol</td>
<td>goal item</td>
</tr>
<tr>
<td>hyperedge</td>
<td>AND-node</td>
<td>production</td>
<td>instantiated deduction</td>
</tr>
<tr>
<td>({u_1, u_2}, v, f)</td>
<td></td>
<td>(v \xrightarrow{f} u_1 u_2)</td>
<td>(v : f(a, b))</td>
</tr>
</tbody>
</table>

**Diagram:**

- OR-node
- AND-node
- OR-nodes
Packed Forests

- a compact representation of many parses
- by sharing common sub-derivations
- polynomial-space encoding of exponentially large set

![Diagram of Packed Forests]

0 I saw 2 him 3 with 4 a mirror 6

(Klein and Manning, 2001; Huang and Chiang, 2005)
Packed Forests

- a compact representation of many parses
- by sharing common sub-derivations
- polynomial-space encoding of exponentially large set

0 I saw 2 him 3 with 4 a mirror 6

(Klein and Manning, 2001; Huang and Chiang, 2005)
Weight Functions and Semirings

\[ f_e(u_1, ..., u_k) \]

\[ f_e(a_1, ..., a_k) \]

tails

head

\[ v \]
Weight Functions and Semirings

\[ f_e(a_1, ..., a_k) = a_1 \otimes ... \otimes a_k \otimes w(e) \]

special case
Weight Functions and Semirings

\[ d(u) \xrightarrow{w(e)} d(u) \otimes w(e) \]

\[ f_e(a) = a \otimes w(e) \]

\[ f_e(d(u)) \]

\[ f_e(a_1, \ldots, a_k) = a_1 \otimes \ldots \otimes a_k \otimes w(e) \]

special case
Weight Functions and Semirings

\[ d(u) \xrightarrow{w(e)} d(u) \otimes w(e) \]

\[ d(u) \xrightarrow{f_e} f_e(d(u)) \]

\[ f_e(a) = a \otimes w(e) \]

\[ f_e(a_1, ..., a_k) = a_1 \otimes ... \otimes a_k \otimes w(e) \]

special case

semiring-composed

Liang Huang (Penn)
Weight Functions and Semirings

\[ d(u) \rightarrow_{w(e)} d(u) \otimes w(e) \]

\[ d(u) \rightarrow_{f_e} f_e(d(u)) \]

\[ f_e(a) = a \otimes w(e) \]

\[ f_e(a_1, \ldots, a_k) = a_1 \otimes \ldots \otimes a_k \otimes w(e) \]

Can also extend \textit{monotonicity} and \textit{superiority} to general weight functions.

Liang Huang (Penn)

Dynamic Programming
Generalizing Semiring Properties

• monotonicity

• semiring:  \( a \leq b \Rightarrow a \times c \leq b \times c \)

• for all weight function \( f \), for all \( a_1 \ldots a_k \), for all \( i \), if \( a'_i \leq a_i \) then \( f(a_1 \ldots a'_i \ldots a_k) \leq f(a_1 \ldots a_i \ldots a_k) \)

• superiority

• semiring:  \( a \leq a \times b, \ b \leq a \times b \)

• for all \( f \), for all \( a_1 \ldots a_k \), for all \( i \), \( a_i \leq f(a_1, \ldots, a_k) \)

• acyclicity

• degenerate a hypergraph back into a graph
Two Dimensional Survey

<table>
<thead>
<tr>
<th>Search Space</th>
<th>Traversing Order</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Graphs with semirings</strong> (e.g., FSMs)</td>
<td><strong>Topological (acyclic)</strong></td>
</tr>
<tr>
<td><strong>Hypergraphs with weight functions</strong> (e.g., CFGs)</td>
<td>Viterbi</td>
</tr>
<tr>
<td></td>
<td>Generalized Viterbi</td>
</tr>
</tbody>
</table>
1. topological sort

2. visit each vertex v in sorted order and do updates
   - for each incoming edge \((u, v)\) in \(E\)
   - use \(d(u)\) to update \(d(v)\):
     - key observation: \(d(u)\) is fixed to optimal at this time
     - time complexity: \(O(V + E)\)
Viterbi Algorithm for DAHs

1. topological sort

2. visit each vertex $v$ in sorted order and do updates
   - for each incoming hyperedge $e = ((u_1, \ldots, u_{|e|}), v, f_e)$
   - use $d(u_i)$’s to update $d(v)$
   - key observation: $d(u_i)$’s are fixed to optimal at this time

\[ d(v) \oplus = f_e(d(u_1), \ldots, d(u_{|e|})) \]

- time complexity: $O(V + E)$ (assuming constant arity)
Example: CKY Parsing

- parsing with CFGs in Chomsky Normal Form (CNF)
- typical instance of the generalized Viterbi for DAHs
- many variants of CKY ~ various topological ordering

\[ O(n^3|P|) \]

(\(S, 0, n\))

\[
\text{For each diff (}\leq n\text{)}
\]
\[
\quad \text{For each } i (\leq n)
\]
\[
\quad \quad \text{For each rule } X \rightarrow Y Z
\]
\[
\quad \quad \quad \text{For each split point } k
\]
\[
\quad \quad \quad \text{score}[X][i][j] = \max
\]
Example: CKY Parsing

- parsing with CFGs in Chomsky Normal Form (CNF)
- typical instance of the generalized Viterbi for DAHs
- many variants of CKY ~ various topological ordering

\(O(n^3|P|)\)

Bottom-up
Example: CKY Parsing

- parsing with CFGs in Chomsky Normal Form (CNF)
- typical instance of the generalized Viterbi for DAHs
- many variants of CKY ~ various topological ordering

\[ O(n^3 |P|) \]
Example: CKY Parsing

- parsing with CFGs in Chomsky Normal Form (CNF)
- typical instance of the generalized Viterbi for DAHs
- many variants of CKY ~ various topological ordering

\[ O(n^3|P|) \]

(S, 0, n)

bottom-up

(S, 0, n)

left-to-right

(S, 0, n)
Example: CKY Parsing

- parsing with CFGs in Chomsky Normal Form (CNF)
- typical instance of the generalized Viterbi for DAHs
- many variants of CKY ~ various topological ordering

\(O(n^3 |P|)\)

bottom-up left-to-right right-to-left
Example: Syntax-based MT

- synchronous context-free grammars (SCFGs)
- context-free grammar in two dimensions
- generating pairs of strings/trees simultaneously
- co-indexed nonterminal further rewritten as a unit

\[
\begin{align*}
\text{VP} & \rightarrow \text{PP}^{(1)} \text{VP}^{(2)}, \quad \text{VP}^{(2)} \text{PP}^{(1)} \\
\text{VP} & \rightarrow \text{\textit{juxing le huitan}}, \quad \text{held a meeting} \\
\text{PP} & \rightarrow \text{\textit{yu Shalong}}, \quad \text{with Sharon}
\end{align*}
\]
Translation as Parsing

• translation with SCFGs => monolingual parsing
• parse the source input with the source projection
  • build the corresponding target sub-strings in parallel

\[
\begin{align*}
VP & \rightarrow PP^{(1)} \ VP^{(2)}, \\
VP & \rightarrow juxing \ le \ huitan, \\
PP & \rightarrow yu \ Shalong,
\end{align*}
\]
Translation as Parsing

- translation with SCFGs => monolingual parsing
- parse the source input with the source projection
- build the corresponding target sub-strings in parallel

\[
\begin{align*}
VP & \rightarrow PP^{(1)} VP^{(2)}, & VP^{(2)} PP^{(1)} \\
VP & \rightarrow juxing le huitan, & \text{held a meeting} \\
PP & \rightarrow yu Shalong, & \text{with Sharon}
\end{align*}
\]
Translation as Parsing

- translation with SCFGs => monolingual parsing
- parse the source input with the source projection
- build the corresponding target sub-strings in parallel

\[
\begin{align*}
VP & \rightarrow PP^{(1)} VP^{(2)}, \\
VP & \rightarrow juxing le huitan, \\
PP & \rightarrow yu Shalong,
\end{align*}
\]

\[
\begin{align*}
VP^{(2)} PP^{(1)} & \quad \text{held a meeting with Sharon} \\
VP^{(2)} PP^{(1)} & \quad \text{held a talk with Sharon} \\
PP^{(1), 3} & \quad \text{with Sharon} \\
VP^{(3), 6} & \quad \text{held a talk} \\
yu Shalong & \quad juxing le huitan
\end{align*}
\]
Translation as Parsing

- translation with SCFGs => monolingual parsing
- parse the source input with the source projection
- build the corresponding target sub-strings in parallel

\[
\begin{align*}
\text{VP} & \rightarrow \text{PP}^{(1)} \text{ VP}^{(2)}, \\
\text{VP} & \rightarrow juxing le huitan, \\
\text{PP} & \rightarrow yu Shalong, \\
\text{VP}^{(2)} & \rightarrow \text{PP}^{(1)}
\end{align*}
\]

complexity: same as CKY parsing -- \( O(n^3) \)

- held a talk with Sharon
- held a meeting with Sharon
- with Sharon
- held a talk
- yu Shalong
- juxing le huitan
Adding a Bigram Model

Liang Huang (Penn)

Dynamic Programming

Adding a Bigram Model

VP₁, 6

PP₁, 3

VP₃, 6

held ... talk
with ... Sharon
PP₁, 3

with ... Sharon
along ... Sharon

held ... meeting

hold ... talks

Dynamic Programming
Adding a Bigram Model

Liang Huang (Penn)

Dynamic Programming
Adding a Bigram Model

complexity: $O(n^3 V^{4(m-1)})$

Liang Huang (Penn)
### Two Dimensional Survey

<table>
<thead>
<tr>
<th>Search Space</th>
<th>Traversing Order</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>topological (acyclic)</strong></td>
<td></td>
</tr>
<tr>
<td>Graphs with semirings (e.g., FSMs)</td>
<td>Viterbi</td>
</tr>
<tr>
<td>Hypergraphs with weight functions (e.g., CFGs)</td>
<td>Generalized Viterbi</td>
</tr>
<tr>
<td><strong>best-first (superior)</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Dijkstra</td>
</tr>
<tr>
<td></td>
<td>Knuth</td>
</tr>
</tbody>
</table>
Viterbi Algorithm for DAHs

1. topological sort

2. visit each vertex \( v \) in sorted order and do updates
   - for each incoming **hyperedge** \( e = ((u_1, ..., u_{|e|}), v, f_e) \)
   - use \( d(u_i) \)'s to update \( d(v) \)
   - key observation: \( d(u_i) \)'s are fixed to optimal at this time

\[
d(v) \oplus = f_e(d(u_1), \cdots, d(u_{|e|}))
\]

- time complexity: \( O(V + E) \) (assuming constant arity)
Forward Variant for DAHs

1. topological sort

2. visit each vertex $v$ in sorted order and do updates
   - for each outgoing hyperedge $e = ((u_1, .., u_{|e|}), h(e), f_e)$
   - if $d(u_i)$'s have all been fixed to optimal
     - use $d(u_i)$'s to update $d(h(e))$

- time complexity: $O(V + E)$
Forward Variant for DAHs

1. topological sort

2. visit each vertex v in sorted order and do updates
   - for each outgoing hyperedge \( e = ((u_1, \ldots, u_{|e|}), h(e), f_e) \)
   - if \( d(u_i) \)'s have all been fixed to optimal
     - use \( d(u_i) \)'s to update \( d(h(e)) \)

- time complexity: \( O(V + E) \)
Forward Variant for DAHs

1. topological sort

2. visit each vertex v in sorted order and do updates
   - for each outgoing hyperedge e = ((u₁, .., u|e|), h(e), fₑ)
   - if d(uᵢ)’s have all been fixed to optimal
     - use d(uᵢ)’s to update d(h(e))

Q: how to avoid repeated checking?
   maintain a counter r[e] for each e:
   how many tails yet to be fixed?
   fire this hyperedge only if r[e]=0

- time complexity: O(V + E)
Dijkstra Algorithm

- keep a cut \((S : V - S)\) where \(S\) vertices are fixed
- maintain a priority queue \(Q\) of \(V - S\) vertices
- each iteration choose the best vertex \(v\) from \(Q\)
- move \(v\) to \(S\), and use \(d(v)\) to forward-update others

\[
d(u) \oplus = d(v) \odot w(v, u)
\]

time complexity:
- \(O((V+E) \lg V)\) (binary heap)
- \(O(V \lg V + E)\) (fib. heap)
Dijkstra Algorithm

- keep a cut \((S : V - S)\) where \(S\) vertices are fixed
- maintain a priority queue \(Q\) of \(V - S\) vertices
- each iteration choose the best vertex \(v\) from \(Q\)
- move \(v\) to \(S\), and use \(d(v)\) to forward-update others

\[
d(u) \oplus = d(v) \otimes w(v, u)
\]

Time complexity:
- \(O((V+E) \log V)\) (binary heap)
- \(O(V \log V + E)\) (fib. heap)
Dijkstra Algorithm

- Keep a cut \((S : V - S)\) where \(S\) vertices are fixed
- Maintain a priority queue \(Q\) of \(V - S\) vertices
- Each iteration choose the best vertex \(v\) from \(Q\)
- Move \(v\) to \(S\), and use \(d(v)\) to forward-update others

\[
d(u) \oplus = d(v) \otimes w(v, u)
\]

Time complexity:
- \(O((V+E) \lg V)\) (binary heap)
- \(O(V \lg V + E)\) (fib. heap)
Knuth (1977) Algorithm

- keep a cut \((S : V - S)\) where \(S\) vertices are fixed
- maintain a priority queue \(Q\) of \(V - S\) vertices
- each iteration choose the best vertex \(v\) from \(Q\)
- move \(v\) to \(S\), and use \(d(v)\) to forward-update others

Time complexity:
- \(O((V+E) \log V)\) (binary heap)
- \(O(V \log V + E)\) (fib. heap)
Knuth (1977) Algorithm

- Keep a cut \((S: V - S)\) where \(S\) vertices are fixed
- Maintain a priority queue \(Q\) of \(V - S\) vertices
- Each iteration choose the best vertex \(v\) from \(Q\)
- Move \(v\) to \(S\), and use \(d(v)\) to forward-update others

Time complexity:
- \(O((V+E) \lg V)\) (binary heap)
- \(O(V \lg V + E)\) (fib. heap)
Knuth (1977) Algorithm

- keep a cut $(S : V - S)$ where $S$ vertices are fixed
- maintain a priority queue $Q$ of $V - S$ vertices
- each iteration choose the best vertex $v$ from $Q$
- move $v$ to $S$, and use $d(v)$ to forward-update others

Time complexity:
- $O((V+E) \log V)$ (binary heap)
- $O(V \log V + E)$ (fib. heap)
Knuth (1977) Algorithm

- keep a cut \((S : V - S)\) where \(S\) vertices are fixed
- maintain a priority queue \(Q\) of \(V - S\) vertices
- each iteration choose the best vertex \(v\) from \(Q\)
- move \(v\) to \(S\), and use \(d(v)\) to forward-update others

\[ s \quad \ldots \quad u_1 \quad v \quad \ldots \quad h(e) \]

\[ S \quad V - S \]

- time complexity:
  - \(O((V+E) \lg V)\) (binary heap)
  - \(O(V \lg V + E)\) (fib. heap)
Example: Best-First/A* Parsing

- Knuth for parsing: best-first  (Caraballo & Charniak, 1998)
- further speed-up: use A* heuristics
  - showed significant speed up with carefully designed heuristic functions  (Klein and Manning, 2003)
- heuristic function: an estimate of outside cost
Example: Best-First/A* Parsing

- Knuth for parsing: best-first (Caraballo & Charniak, 1998)
- further speed-up: use A* heuristics
  - showed significant speed up with carefully designed heuristic functions (Klein and Manning, 2003)
- heuristic function: an estimate of outside cost

(S, 0, n)
Outside Cost in Hypergraph

- outside cost: yet to pay to reach goal
- let’s only consider semiring-composed case
  - and only acyclic hypergraphs
- after computing $d(v)$ for all $v$ from bottom-up
- backwards Viterbi from top-down (outside-in)

$$h(S_{0,n}) = \bar{1}$$
$$h(v) + = h(u) \otimes w(e) \otimes d(v')$$
Outside Cost in Hypergraph

- outside cost: yet to pay to reach goal
- let’s only consider semiring-composed case
  - and only acyclic hypergraphs
- after computing $d(v)$ for all $v$ from bottom-up
  - backwards Viterbi from top-down (outside-in)

$$h(S_{0,n}) = \mathbf{1}$$
$$h(v) \oplus = h(u) \otimes w(e) \otimes d(v')$$

Q: $d(v) \otimes h(v) =$?
Projection-based Heuristics

- how to guess? project onto a coarser-grained space
- and parse with the coarser grammar
  - outside cost of of the coarser item as heuristics

(Klein and Manning, 2003)
Projection-based Heuristics

- how to guess? project onto a coarser-grained space
- and parse with the coarser grammar
- outside cost of of the coarser item as heuristics

(Klein and Manning, 2003)
Projection-based Heuristics

- how to guess? project onto a coarser-grained space
- and parse with the coarser grammar
- outside cost of of the coarser item as heuristics

(Klein and Manning, 2003)
Projection-based Heuristics

- how to guess? project onto a coarser-grained space
- and parse with the coarser grammar
- outside cost of of the coarser item as heuristics

(Klein and Manning, 2003)
Projection-based Heuristics

- how to guess? project onto a coarser-grained space
- and parse with the coarser grammar
- outside cost of the coarser item as heuristics

(Klein and Manning, 2003)
Projection-based Heuristics

- how to guess? project onto a coarser-grained space
- and parse with the coarser grammar
- outside cost of the coarser item as heuristics

\[
\hat{h} (VBD_{2,3}) = h' (V_{2,3}) \quad \text{(Klein and Manning, 2003)}
\]
Analogy with Graphs

S
   NP       VP
   |         |
  NN | NNS  | VBD | PP
  Faculty | payrolls | fell  | IN | NNP
          |         |       | in | Sept.
Analogy with Graphs

- S
  - NP
    - NN: Faculty
    - NNS: payrolls
  - VP
    - VBD: fell
    - IN: in
    - PP: NNP
- X
  - N
    - N: Faculty
  - V
    - X: payrolls
  - X
    - P: fell
    - N: in
    - X: Sept.
More on Coarse-to-Fine

- multilevel coarse-to-fine A*
  - heuristic = exact outside cost in previous stage
  - \( \hat{h}_i(v) = h_{i-1}(\text{proj}_{i-1}(v)) \)
- VBD>V>X. \( \hat{h}_i(VBD_{1,5}) = h_{i-1}(V_{1,5}); \hat{h}_{i-1}(V_{1,5}) = h_{i-2}(X_{1,5}) \)
- multilevel coarse-to-fine Viterbi w/ beam-search
  - Viterbi + beam pruning in each stage
  - prune according to merit: \( d(v) \otimes h(v) \odot d(\text{TOP}) \)
  - hard to derive a provably correct threshold
  - in practice: use a preset threshold (but works well!)
More on Coarse-to-Fine

- multilevel coarse-to-fine A*
  - heuristic = exact outside cost in
  - \( \hat{h}_i(v) = h_{i-1}(\text{proj}_{i-1}(v)) \)
  - VBD>V>X. \( \hat{h}_i(\text{VBD}_{1,5}) = h_{i-1}(V_{1,5}); \hat{h}_{i-1}(V_{1,5}) = h_{i-2}(X_{1,5}) \)
- multilevel coarse-to-fine Viterbi w/ beam-search
  - Viterbi + beam pruning in each stage
  - prune according to merit: \( d(v) \odot h(v) \odot d(\text{TOP}) \)
  - hard to derive a provably correct threshold
  - in practice: use a preset threshold (but works well!)
More on Coarse-to-Fine

- multilevel coarse-to-fine A*
  - heuristic = exact outside cost in
  - $\hat{h}_i(v) = h_{i-1}(\text{proj}_{i-1}(v))$
- $\text{VBD} > \text{V} > \text{X}$. $\hat{h}_i(\text{VBD}_{1,5}) = h_{i-1}(\text{V}_{1,5}); \hat{h}_{i-1}(\text{V}_{1,5}) = h_{i-2}(\text{X}_{1,5})$
- multilevel coarse-to-fine Viterbi w/ beam-search
  - Viterbi + beam pruning in each stage
  - prune according to merit: $d(v) \otimes h(v) \otimes d(\text{TOP})$
  - hard to derive a provably correct threshold
  - in practice: use a preset threshold (but works well!)
monotonic optimization problems

acyclic: Viterbi

many NLP problems

superior: Knuth
monotonic optimization problems

acyclic: Viterbi

superior: Knuth

many NLP problems

PCFG parsing with CNF
monotonic optimization problems

acyclic:
  Viterbi

many NLP problems
superior:
  Knuth

Inside-Outside Alg. (Inside semiring)

PCFG parsing with CNF
monotonic optimization problems

acyclic: Viterbi

many NLP problems

superior: Knuth

Inside-Outside Alg. (Inside semiring) (discriminative) parsing

non-prob. parsing

PCFG parsing with CNF
monotonic optimization problems

acyclic: Viterbi

many NLP problems

superior: Knuth

Inside-Outside Alg. (Inside semiring)

non-prob. (discriminative) parsing

PCFG parsing with CNF

cyclic grammars
monotonic optimization problems

acyclic: Viterbi

many NLP problems

superior: Knuth

PCFG parsing with CNF

cyclic grammars

generalized Bellman-Ford (open)

non-prob. (discriminative) parsing

Inside-Outside Alg. (Inside semiring)
Take Home Message

- Dynamic Programming is cool, easy, and universal!
- two frameworks and two types of algorithms
  - monotonicity; acyclicity and/or superiority
  - topological (Viterbi) vs. best-first style (Dijkstra/Knuth/A*)
    - when to choose which: A* can finish early if lucky
  - graph (lattice) vs. hypergraph (forest)
    - incremental, finite-state vs. branching, context-free
- covered many typical NLP applications
- a better understanding of theory helps in practice
Thanks!

Questions?
Comments?

final slides will be available on my website.