Advanced Dynamic Programming in AI/NLP
Theory, Algorithms, and Applications

We eat sushi with tuna

Liang Huang
University of Pennsylvania
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?
  - Richard Bellman (1957)
  - Andrew Viterbi (1967)
  - Edsger Dijkstra (1959)
  - Hart, Nilsson, and Raphael (1968)
    - Dijkstra => A* Algorithm
  - Donald Knuth (1977)
    - Dijkstra on 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
  - [optional] Sharing of Common Subproblems

- Structure of the Search Space
  - Incremental
    - => Graph Search
  - Branching
    - => Hypergraph Search
## Two Dimensional Survey

<table>
<thead>
<tr>
<th>Search Space</th>
<th>Traversing Order</th>
<th>Graphs with Semirings (e.g., FSMs)</th>
<th>Hypergraphs with Weight Functions (e.g., CFGs)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Viterbi</td>
<td>Generalized Viterbi</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Dijkstra</td>
<td>Knuth</td>
</tr>
</tbody>
</table>

**Traversing Order**
- **Topological (acyclic)**
- **Best-first (superior)**

**Search Space**
- **Graphs with Semirings (e.g., FSMs)**
- **Hypergraphs with Weight Functions (e.g., CFGs)**
Motivations for Semirings

- 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

\[
d(\pi_1) = \bigotimes_{e_i \in \pi_1} w(e_i) = w(e_1) \otimes w(e_2) \otimes w(e_3)
\]

\[
d(t) = \bigoplus_{\pi_i} w(\pi_i) = w(p_1) \oplus w(p_2) \oplus \cdots
\]
A **monoid** is a triple \((A, \otimes, \mathbf{1})\) where

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

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

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

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

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

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

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

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

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

2. \((A, \otimes, \overline{1})\) 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. \(\overline{0}\) is an **annihilator** for \(\otimes\): for all \(a\) in \(A\), \(\overline{0} \otimes a = a \otimes \overline{0} = \overline{0}\).
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 \(A\), \(0 \otimes a = a \otimes 0 = 0\).

\([0, 1], +, 0\)

\([0, 1], \times, 1\)

\([0, 1], \max, 0\)

\([0, 1], \max, \times, 0, 1\)

\([0, 1], +, \times, 0, 1\)
## 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>

*how about?*
A semiring \((A, \oplus, \otimes, 0, 1)\) is \textbf{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, ...**
  \(\{0, 1\}, \lor, \land, 0, 1\)  \((\mathbb{R}^+ \cup \{+\infty\}, \min, +, +\infty, 0)\)

  \([0, 1], \max, \otimes, 0, 1\)  \((\mathbb{R} \cup \{+\infty\}, \min, +, +\infty, 0)\)
• **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, ...**
  \(\{0, 1\}, \lor, \land, 0, 1\)  \(\mathbb{R}^+ \cup \{+\infty\}, \text{min}, +, +\infty, 0\)

  \([0, 1], \max, \otimes, 0, 1\)  \(\mathbb{R} \cup \{+\infty\}, \text{min}, +, +\infty, 0\)

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

• **examples: all of the above**
Monotonicity
Monotonicity

- monotonicity
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 $a, b, c \in A$

$$(a \leq b) \Rightarrow (a \otimes c \leq b \otimes c) \quad (a \leq b) \Rightarrow (c \otimes a \leq c \otimes b)$$
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 $a, b, c \in A$

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

- **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 $a, b, c \in A$

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

• optimal substructure in dynamic programming
Monotonicity

- **monotonicity**

  Let $K = (A, \oplus, \otimes, \bar{0}, \bar{1})$ 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**

  - $A: b \otimes c$
    - $B: b$
    - $C: c$
  
  - $A: b' \otimes c$
    - $B: b' \leq b$
    - $C: c$

- **[lemma]** idempotent $\Rightarrow$ monotone
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**

- **[lemma]** idempotent $\Rightarrow$ monotone

- Our focus, totally-ordered semirings, are monotone
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

• [lemma] idempotent $\Rightarrow$ monotone

• our focus, totally-ordered semirings, are monotone

free lunch!
## Two Dimensional Survey

<table>
<thead>
<tr>
<th>Search Space</th>
<th>Traversing Order</th>
<th>Topological (Acyclic)</th>
<th>Best-First (Superior)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graphs with semirings (e.g., FSMs)</td>
<td>Viterbi</td>
<td>Viterbi</td>
<td></td>
</tr>
<tr>
<td>Hypergraphs with weight functions (e.g., CFGs)</td>
<td>Generalized Viterbi</td>
<td>Dijkstra</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Knuth</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Topological (acyclic) traversal is used for graphs with semirings (e.g., FSMs).
- Best-first (superior) traversal is used for hypergraphs with weight functions (e.g., CFGs).

- Viterbi: Used for graphs with semirings (e.g., FSMs).
- Dijkstra: Used for hypergraphs with weight functions (e.g., CFGs).
- Generalized Viterbi: Used for hypergraphs with weight functions (e.g., CFGs).
- Knuth: Used for hypergraphs with weight functions (e.g., CFGs).
**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 $(u, v)$

\[
\begin{align*}
\forall u, v \in V, \quad d(v) \oplus = d(u) \otimes w(u, v) \\
\text{how to avoid cyclic updates?} \\
\rightarrow \quad d(v) \leftarrow d(v) \oplus (d(u) \otimes w(u, v))
\end{align*}
\]

- Only update when $d(u)$ is fixed
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

\[ u \quad w(u, v) \]

\[ v \]

• 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

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

- 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 \textit{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: Word Alignment
Dijkstra Algorithm
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 $a, b \in A$

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

• basically, combination always gets worse

• or, no negative edge in a graph
Dijkstra Algorithm

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

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

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

- basically, combination always gets worse
- or, no negative edge in a graph

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

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

- Basically, combination always gets worse
- Or, no negative edge in a graph

\[
\begin{array}{c}
d(u) \xrightarrow{w(e)} d(u) \otimes w(e)
\end{array}
\]
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 $a, b \in A$

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

- basically, combination always gets worse

- or, no negative edge in a graph

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

\[
\begin{align*}
\{0, 1\}, & \lor, \land, 0, 1 \checkmark \\
[0, 1], & \max, \times, 0, 1 \\
\mathbb{R}^+ \cup \{+\infty\}, & \min, +, +\infty, 0 \\
\mathbb{R} \cup \{+\infty\}, & \min, +, +\infty, 0
\end{align*}
\]
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 $a, b \in A$

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

- Basically, combination always gets worse
- Or, no negative edge in a graph

$$(\{0, 1\}, \lor, \land, 0, 1) \checkmark$$
$$(\{0, 1\}, \max, \times, 0, 1) \checkmark$$
$$(\mathbb{R}^+ \cup \{+\infty\}, \min, +, +\infty, 0)$$
$$(\mathbb{R} \cup \{+\infty\}, \min, +, +\infty, 0)$$
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 $a, b \in A$

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

- basically, combination always gets worse
- or, no negative edge in a graph

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

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

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

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

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

- basically, combination always gets worse
- or, no negative edge in a graph

Let $d(u) \otimes w(e)$ be the weight of the edge from $u$ to $v$.

$Liang Huang$  
Dynamic Programming  
18
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:
- \(O((V+E) \log V)\) (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)
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
Viterbi vs. Dijkstra

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

\[
\begin{align*}
\text{acyclic:} & \quad \text{Viterbi} \\
\text{superior:} & \quad \text{Dijkstra}
\end{align*}
\]
Viterbi vs. Dijkstra

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

Monotonic optimization problems:
- Acyclic: Viterbi
- Many NLP problems: Dijkstra

Superior: Dijkstra
Viterbi vs. Dijkstra

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

Venn diagram:

- acyclic: Viterbi
- superior: Dijkstra
- many NLP problems

Monotonic optimization problems:

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)

max-margin models
Viterbi vs. Dijkstra

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

- acyclic: Viterbi
- superior: Dijkstra

- monotonic optimization problems
- forward-backward (Inside semiring)
- max-margin models
- many NLP problems
- cyclic FSMs/grammars

Liang Huang
What if both fail?

monotonic optimization problems

acyclic: Viterbi

many NLP problems

superior: Dijkstra

generalized Bellman-Ford
(CLIR, 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**

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

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

\[ a \cdot (V + E) \log V \quad \text{vs.} \quad V + E \]

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

- $d(v)$: the distance from source $s$ to $v$
- $h(v)$: the distance from $v$ to target $t$
  - $\hat{h}(v)$ must be an **optimistic** estimate of $h(v)$: $\hat{h}(v) \leq h(v)$
- now, prioritize the queue by $d(v) \otimes \hat{h}(v)$
  - Dijkstra is a special case where $\hat{h}(v) = \bar{1}$
  - also requires $d(v) \otimes \hat{h}(v)$ never improves (superior)
  - hope: $d(t) \otimes \hat{h}(t) = d(t)$ can be popped sooner
More on A*
<table>
<thead>
<tr>
<th>Search Space</th>
<th>Traversing Order</th>
<th>Topological (Acyclic)</th>
<th>Best-First (Superior)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hypergraphs with weight functions (e.g., CFGs)</td>
<td>Generalized Viterbi</td>
<td>Dijkstra</td>
<td></td>
</tr>
<tr>
<td>Graphs with semirings (e.g., FSMs)</td>
<td>Viterbi</td>
<td>Knuth</td>
<td></td>
</tr>
</tbody>
</table>

**Two Dimensional Survey**

Dynamic Programming

Liang Huang
<table>
<thead>
<tr>
<th>search space</th>
<th>topological (acyclic)</th>
<th>best-first (superior)</th>
</tr>
</thead>
<tbody>
<tr>
<td>graphs with semirings (e.g., FSMs)</td>
<td>Viterbi</td>
<td>Dijkstra</td>
</tr>
<tr>
<td>hypergraphs with weight functions (e.g., CFGs)</td>
<td>Generalized Viterbi</td>
<td>Knuth</td>
</tr>
</tbody>
</table>
Background: CFG and Parsing
(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

\[
d(v) \oplus = f_e(d(u_1), d(u_2))
\]
Packed Forests

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

(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)
## 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>
</tbody>
</table>

\[
\begin{align*}
\{u_1, u_2\}, v, f \\
v & \rightarrow u_1 \ u_2 \\
\end{align*}
\]

\[
\begin{align*}
\frac{u_1 : a \quad u_2 : b}{v : f(a, b)}
\end{align*}
\]
Hypergraphs and Deduction

tails $u_1 : a$

$fe$

$v : fe(a,b)$

head

$u_2 : b$

antecedents

$v : fe(a,b)$

consequent

(Nederhof, 2003)
Hypergraphs and Deduction

\[ \text{heads: } v \text{ : } f_e(a,b) \]

\[ \text{tails: } u_1 : a \quad u_2 : b \]

\[ \text{antecedents: } u_1 : a \quad u_2 : b \]

\[ \text{consequent: } v \text{ : } f_e(a,b) \]

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

\[ v \text{ : } a \times b \times \Pr(A \rightarrow B C) \]

(Nederhof, 2003)
Hypergraphs and Deduction

- **tails**: \( u_1 : a \) \( \quad u_2 : b \)
- **head**: \( v : f_e(a, b) \)
- **antecedents**: \( u_1 : a \) \( u_2 : b \)
- **consequent**: \( v : f_e(a, b) \)

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

\[A \rightarrow B \ C\]

(Nederhof, 2003)
Weight Functions and Semirings
Weight Functions and Semirings

\[ d(u) \xrightarrow{w(e)} d(u) \otimes w(e) \]
Weight Functions and Semirings

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

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

\[ f_e(a) = a \otimes w(e) \]
Weight Functions and Semirings

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

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

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

tails

\[ u_1 \]

\[ u_2 \]

\[ \ldots \]

\[ u_k \]

head

\[ v \]
Weight Functions and Semirings

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

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

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

tails

\[ \text{u}_1 \xrightarrow{f_e} \text{u}_2 \xrightarrow{f_e} \ldots \xrightarrow{f_e} \text{u}_k \xrightarrow{f_e} \text{v} \]

head

\[ f_e(a_1, \ldots, a_k) \]
Weight Functions and Semirings

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

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

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

\[ f_e(a_1, ..., a_k) = a_1 \otimes ... \otimes a_k \otimes w(e) \]
Weight Functions and Semirings

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

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

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

\[ f_e(a_1, ..., a_k) = a_1 \otimes ... \otimes a_k \otimes w(e) \]
Weight Functions and Semirings

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

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

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

\[ f_e(a_1, ..., a_k) = a_1 \otimes ... \otimes a_k \otimes w(e) \]
Weight Functions and Semirings

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

\[ f_e(d(u)) \]

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

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

also extend \textit{monotonicity} and \textit{superiority} to weight functions
Monotonicity and Superiority
# 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</td>
<td>best-first (superior)</td>
</tr>
<tr>
<td>(e.g., CFGs)</td>
<td></td>
</tr>
</tbody>
</table>

- **Topological (acyclic)**: Viterbi
- **Best-first (superior)**: Dijkstra

- **Generalized Viterbi**: Knuth

---

Liang Huang

Dynamic Programming
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 \( d(v) \):
     - key observation: \( d(u) \) is fixed to optimal at this time

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

- 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

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

with CKY pseudo-code

$O(n^3|P|)$
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, 1, n)  (S, 1, n)  (S, 1, n)

bottom-up

with CKY pseudo-code
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

\[ \text{bottom-up} \]

with CKY pseudo-code

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

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

(S, l, n)

(bottom-up)

O(n^3|P|)

left-to-right

with CKY pseudo-code
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

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

with CKY pseudo-code
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 $e = ((u_1, ..., u_{|e|}), h(e), f_e)$
   - 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 \( d(h(e)) \)

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: \( O(V + E) \)
Example: Treebank Parsers

- State-of-the-art statistical parsers
  - (Collins, 1999; Charniak, 2000)
- no fixed grammar (every production is possible)
- can’t do backward updates
  - don’t know how to decompose a big item
- forward update from vertex \((X, i, j)\)
  - check all vertices like \((Y, j, k)\) or \((Y, k, i)\) in the chart (fixed)
  - try combine them to form bigger item \((Z, i, k)\) or \((Z, k, j)\)
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

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

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

![Diagram of graph with nodes and edges]

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

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

\(f_e\)

\(u_1\)

\(S\)

\(V - S\)

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:
- \(O((V+E) \lg V)\) (binary heap)
- \(O(V \lg V + E)\) (fib. heap)
Example: A* Parsing

- Use A* search on top of the Knuth Algorithm
- Showed significant speed up with carefully designed heuristic functions (Klein and Manning, 2003)

[open problem] can you still define heuristic function if weight functions are not semiring-composed?
Example: A* Parsing

- Use A* search on top of the Knuth Algorithm
- Showed significant speed up with carefully designed heuristic functions (Klein and Manning, 2003)

[open problem] can you still define heuristic function if weight functions are not semiring-composed?
monotonic optimization problems

acyclic: Viterbi

many NLP problems

superior: Knuth
monotonic optimization problems

acyclic: Viterbi

many NLP problems

superior: Knuth

PCFG parsing with CNF

Liang Huang

Dynamic Programming
same picture again

monotonic optimization problems

acyclic: Viterbi

many NLP problems

superior: Knuth

Inside-Outside Alg. (Inside semiring)

PCFG parsing with CNF
Same Picture Again

monotonic optimization problems

acyclic: Viterbi

many NLP problems

superior: Knuth

Inside-Outside Alg. (Inside semiring)

max-margin parsing

PCFG parsing with CNF
Same Picture Again

monotonic optimization problems

acyclic: Viterbi

many NLP problems

superior: Knuth

Inside-Outside Alg. (Inside semiring)

max-margin parsing

PCFG parsing with CNF

cyclic grammars

Dynamic Programming
monotonic optimization problems

acyclic:

Viterbi

many NLP problems

superior:

Knuth

Inside-Outside Alg. (Inside semiring)

max-margin parsing

PCFG parsing with CNF

cyclic grammars

generalized generalized Bellman-Ford (open)
Take Home Message

• Dynamic Programming is cool, easy, and universal!

• two frameworks and two types of algorithms
  • monotonicity; acyclicity and/or superiority
  • topological style (Viterbi) vs. best-first style (Dijkstra)
    • when to choose which
  • graph vs. hypergraph

• covered many typical NLP applications
  • a better understanding of theory helps in practice
Thanks!

Questions?

Comments?
Thanks!

Questions?
Comments?