Online Learning

- Online learning algorithms for huge data sets
  - Processing one sample at a time
    - computationally cheap
    - easy to implement
  - More efficient both in time and space comparing to batch learning algorithms
Online Learning

- Online learning algorithms for huge data sets
- Processing one sample at a time
  - computationally cheap
  - easy to implement
- More efficient both in time and space comparing to batch learning algorithms
Online Learning

- Online learning algorithms for huge data sets
- Processing one sample at a time
  - computationally cheap
  - easy to implement
- More efficient both in time and space comparing to batch learning algorithms
Online Learning

- Online learning algorithms for huge data sets
- Processing one sample at a time
  - computationally cheap
  - easy to implement
- More efficient both in time and space comparing to batch learning algorithms
Online Learning

- Online learning algorithms for huge data sets
- Processing one sample at a time
  - computationally cheap
  - easy to implement
- More efficient both in time and space comparing to batch learning algorithms
Online Learning of Combinatorial Object

- **Combinatorial objects** = structured concepts composed of components.
  - e.g. graphs, permutations, Huffman trees, binary search trees.
- Learning these concepts is a challenging task
  - Combinatorial nature: instances are exponentially many.
- **Idea:** Encoding the decisions of offline algorithms to obtain a “good” representation
  - sorting networks, dynamic programming

[Rahmanian et al., 2018, Rahmanian and Warmuth, 2017]
Combinatorial objects = structured concepts composed of components.
  ▶ e.g. graphs, permutations, Huffman trees, binary search trees.

Learning these concepts is a challenging task
  ▶ Combinatorial nature: instances are exponentially many.

Idea: Encoding the decisions of offline algorithms to obtain a “good” representation
  ▶ sorting networks, dynamic programming

[Rahmanian et al., 2018, Rahmanian and Warmuth, 2017]
Online Learning of Combinatorial Object

- **Combinatorial objects** = structured concepts composed of components.
  - e.g. graphs, permutations, Huffman trees, binary search trees.
- Learning these concepts is a challenging task
  - Combinatorial nature: instances are exponentially many.
- **Idea**: Encoding the decisions of offline algorithms to obtain a “good” representation
  - sorting networks, dynamic programming

[Rahmanian et al., 2018, Rahmanian and Warmuth, 2017]
Online Learning of Combinatorial Object

- **Combinatorial objects** = structured concepts composed of components.
  - e.g. graphs, permutations, Huffman trees, binary search trees.
- Learning these concepts is a challenging task
  - Combinatorial nature: instances are exponentially many.
- **Idea:** Encoding the decisions of offline algorithms to obtain a “good” representation
  - sorting networks, dynamic programming

[Rahmanian et al., 2018, Rahmanian and Warmuth, 2017]
Combinatorial objects = structured concepts composed of components.
  
  e.g. graphs, permutations, Huffman trees, binary search trees.

Learning these concepts is a challenging task
  
  Combinatorial nature: instances are exponentially many.

Idea: Encoding the decisions of offline algorithms to obtain a “good” representation
  
  sorting networks, dynamic programming

[Rahmanian et al., 2018, Rahmanian and Warmuth, 2017]
Online Learning of Combinatorial Object

- **Combinatorial objects** = structured concepts composed of components.
  - e.g. graphs, permutations, Huffman trees, binary search trees.
- Learning these concepts is a challenging task
  - Combinatorial nature: instances are exponentially many.
- **Idea**: Encoding the decisions of offline algorithms to obtain a “good” representation
  - sorting networks, dynamic programming

[Rahmanian et al., 2018, Rahmanian and Warmuth, 2017]
Outline

- Overview of dynamic programming with min/max-sum recurrences.
  - The representation in graph of subproblems.
- Describing the online learning scenario.
  - Using binary search tree as running example.
  - Our methods are general.
- Reducing the problem to learning subgraphs called multipaths in the graph of subproblems.
  - Expanded Hedge
  - Component Hedge
Outline

- Overview of dynamic programming with min/max-sum recurrences.
  - The representation in graph of subproblems.

- Describing the online learning scenario.
  - Using binary search tree as running example.
  - Our methods are general.

- Reducing the problem to learning subgraphs called multipaths in the graph of subproblems.
  - Expanded Hedge
  - Component Hedge
Overview of dynamic programming with min/max-sum recurrences.

- The representation in graph of subproblems.

Describing the online learning scenario.

- Using binary search tree as running example.

- Our methods are general.

Reducing the problem to learning subgraphs called multipaths in the graph of subproblems.

- Expanded Hedge
- Component Hedge
Outline

- Overview of dynamic programming with min/max-sum recurrences.
  - The representation in graph of subproblems.
- Describing the online learning scenario.
  - Using binary search tree as running example.
    - Our methods are general.
- Reducing the problem to learning subgraphs called **multipaths** in the graph of subproblems.
  - Expanded Hedge
  - Component Hedge
Overview of dynamic programming with min/max-sum recurrences.

- The representation in graph of subproblems.

Describing the online learning scenario.

- Using binary search tree as running example.
- Our methods are general.

Reducing the problem to learning subgraphs called multipaths in the graph of subproblems.

- Expanded Hedge
- Component Hedge
Overview of dynamic programming with min/max-sum recurrences.
  - The representation in graph of subproblems.

Describing the online learning scenario.
  - Using binary search tree as running example.
  - Our methods are general.

Reducing the problem to learning subgraphs called multipaths in the graph of subproblems.
  - Expanded Hedge
  - Component Hedge
Outline

- Overview of dynamic programming with min/max-sum recurrences.
  - The representation in graph of subproblems.
- Describing the online learning scenario.
  - Using binary search tree as running example.
  - Our methods are general.
- Reducing the problem to learning subgraphs called multipaths in the graph of subproblems.
  - Expanded Hedge
  - Component Hedge
Outline

- Overview of dynamic programming with min/max-sum recurrences.
  - The representation in graph of subproblems.
- Describing the online learning scenario.
  - Using binary search tree as running example.
  - Our methods are general.
- Reducing the problem to learning subgraphs called multipaths in the graph of subproblems.
  - Expanded Hedge
  - Component Hedge
Knapsack 0/1 – The Offline Problem

- **Given:** $n$ items with heavinesses $h \in \mathbb{N}^n$ and profits $p \in [0, 1]^n$.
- **Goal:** Find the optimal packing $\mathcal{I} \subseteq \{1..n\}$ maximizing

\[
\sum_{i \in \mathcal{I}} p_i
\]

subject to knapsack capacity $C \in \mathbb{N}$

\[
\sum_{i \in \mathcal{I}} h_i \leq C
\]
Knapsack 0/1 – Dynamic Programming

- **Subproblem** \((i, c)\): The first \(i\) items and capacity \(c\).
- **Base subproblems**: \(\mathcal{T} := \{(0, c) \mid 0 \leq c \leq C\}\)
- **Final subproblem**: \(s := (n, C)\).
- **Recurrence**:

\[
\text{OPT}(i, c) = \begin{cases} 
0 & i = 0 \\
\text{OPT}(i - 1, c) & c < h_i \\
\max\{\text{OPT}(i - 1, c), p_i + \text{OPT}(i - 1, c - h_i)\} & \text{else.}
\end{cases}
\]
◆ **Subproblem** \((i, c)\): The first \(i\) items and capacity \(c\).

◆ **Base subproblems:** \(\mathcal{T} := \{(0, c) \mid 0 \leq c \leq C\}\)

◆ **Final subproblem:** \(s := (n, C)\).

◆ **Recurrence:**

\[
\text{OPT}(i, c) = \begin{cases} 
0 & \text{if } i = 0 \\
\text{OPT}(i - 1, c) & \text{if } c < h_i \\
\max\{\text{OPT}(i - 1, c), p_i + \text{OPT}(i - 1, c - h_i)\} & \text{else.}
\end{cases}
\]
Knapsack 0/1 – Dynamic Programming

▶ **Subproblem** \((i, c)\): The first \(i\) items and capacity \(c\).

▶ **Base subproblems:** \(T := \{(0, c) \mid 0 \leq c \leq C\}\)

▶ **Final subproblem:** \(s := (n, C)\).

▶ **Recurrence:**

\[
\text{OPT}(i, c) = \begin{cases} 
0 & i = 0 \\
\text{OPT}(i - 1, c) & c < h_i \\
\max\{\text{OPT}(i - 1, c), p_i + \text{OPT}(i - 1, c - h_i)\} & \text{else.}
\end{cases}
\]
Knapsack 0/1 – Dynamic Programming

- **Subproblem** \((i, c)\): The first \(i\) items and capacity \(c\).
- **Base subproblems:** \(\mathcal{T} := \{(0, c) \mid 0 \leq c \leq C\}\)
- **Final subproblem:** \(s := (n, C)\).
- **Recurrence:**

\[
\text{OPT}(i, c) = \begin{cases} 
0 & i = 0 \\
\text{OPT}(i - 1, c) & c < h_i \\
\max\{\text{OPT}(i - 1, c), p_i + \text{OPT}(i - 1, c - h_i)\} & \text{else.}
\end{cases}
\]
The Knapsack DAG.

Figure: An example with $C = 7$ and $(h_1, h_2, h_3) = (2, 3, 4)$. 

▶ The Knapsack DAG.
Figure: An example with $C = 7$ and $(h_1, h_2, h_3) = (2, 3, 4)$.
Knapsack 0/1 – DAG

Figure: An example with $C = 7$ and $(h_1, h_2, h_3) = (2, 3, 4)$.

- Start; Item 3 is picked;
Figure: An example with $C = 7$ and $(h_1, h_2, h_3) = (2, 3, 4)$.

- Start; Item 3 is picked; Item 2 is not picked;
Figure: An example with $C = 7$ and $(h_1, h_2, h_3) = (2, 3, 4)$. 

- Start; Item 3 is picked; Item 2 is not picked; Item 1 is picked.
A new representation with dynamic programming:

1. Edges encode recursive calls.

2. Paths encode solutions.
A new representation with dynamic programming:

1. **Edges** encode **recursive calls**.

2. **Paths** encode **solutions**.
A new representation with dynamic programming:

1. **Edges** encode recursive calls.

2. **Paths** encode solutions.
Given: Keys $k_1 < k_2 < \ldots < k_n$ with search probabilities $p \in [0, 1]^n$ i.e. $\sum_{i=1}^n p_i = 1$.

Goal: Find the optimal binary search tree (BST) $\pi$ minimizing average search cost i.e.

$$\pi \cdot p = \sum_{i=1}^n \text{depth}_{\pi}(k_i) \cdot p_i$$

Example: $\pi = (2, 3, 4, 1, 2), p = (.1, .05, .05, .5, .3)$:

- $1 \times .5$
- $+2 \times .1 + 2 \times .3$
- $+3 \times .05$
- $+4 \times .05 = 1.65$ average search cost
Given: Keys $k_1 < k_2 < \ldots < k_n$ with search probabilities $p \in [0, 1]^n$ i.e. $\sum_{i=1}^n p_i = 1$.

Goal: Find the optimal binary search tree (BST) $\pi$ minimizing average search cost i.e.

$$\pi \cdot p = \sum_{i=1}^n \text{depth}_\pi(k_i) \cdot p_i$$

Example: $\pi = (2, 3, 4, 1, 2)$, $p = (0.1, 0.05, 0.05, 0.5, 0.3)$:

$$1 \times 0.5 + 2 \times 0.1 + 2 \times 0.3 + 3 \times 0.05 + 4 \times 0.05 = 1.65$$

average search cost
**Subproblem** \((i, j)\): The keys \(k_i < \ldots < k_j\).

- **Base subproblems:** \(T := \{(i, i - 1) \mid 1 \leq i \leq n\}\)
- **Final subproblem:** \(s := (1, n)\).
- **Recurrence:**

\[
\text{OPT}(i, j) = \begin{cases} 
0 & j = i - 1 \\
\min_{i \leq r \leq j} \{ \text{OPT}(i, r - 1) \\
+ \text{OPT}(r + 1, j) \\
+ \sum_{k=i}^{j} p_k \} & i \leq j.
\end{cases}
\]
Subproblem \((i, j)\): The keys \(k_i < \ldots < k_j\).

Base subproblems: \(\mathcal{T} := \{(i, i - 1) \mid 1 \leq i \leq n\}\)

Final subproblem: \(s := (1, n)\).

Recurrence:

\[
\text{OPT}(i, j) = \begin{cases} 
0 & j = i - 1 \\
\min_{i \leq r \leq j} \{\text{OPT}(i, r - 1) + \text{OPT}(r + 1, j) + \sum_{k=i}^{j} p_k\} & i \leq j.
\end{cases}
\]
Subproblem \((i, j)\): The keys \(k_i < \ldots < k_j\).

Base subproblems: \(T := \{(i, i - 1) \mid 1 \leq i \leq n\}\)

Final subproblem: \(s := (1, n)\).

Recurrence:

\[
\text{OPT}(i, j) = \begin{cases} 
0 & j = i - 1 \\
\min_{i \leq r \leq j} \{\text{OPT}(i, r - 1) \\
+ \text{OPT}(r + 1, j) \\
+ \sum_{k=i}^{j} p_k\} & i \leq j.
\end{cases}
\]
Subproblem \((i, j)\): The keys \(k_i < \ldots < k_j\).

Base subproblems: \(\mathcal{T} := \{(i, i - 1) \mid 1 \leq i \leq n\}\)

Final subproblem: \(s := (1, n)\).

Recurrence:

\[
\text{OPT}(i, j) = \begin{cases} 
0 & j = i - 1 \\
\min_{i \leq r \leq j} \{\text{OPT}(i, r - 1) + \text{OPT}(r + 1, j) + \sum_{k=i}^{j} p_k\} & i \leq j.
\end{cases}
\]
Figure: An example with $n = 5$. 

Binary Search Tree – DAG
Figure: An example with $n = 5$. 
Figure: An example with $n = 5$. 
Figure: An example with $n = 5$. 
Figure: An example with $n = 5$. 
Figure: An example with $n = 5$. 
Figure: An example with $n = 5$. 
A new representation with dynamic programming:

1. **Multiedges** encode recursive calls.
2. **Multipaths** encode solutions.

[Martin, Rardin, and Campbell, 1990]
A new representation with dynamic programming:

1. **Multiedges** encode recursive calls.

2. **Multipaths** encode solutions.

[Martin, Rardin, and Campbell, 1990]
A new representation with dynamic programming:

1. **Multiedges** encode **recursive calls**.

2. **Multipaths** encode **solutions**.

[Martin, Rardin, and Campbell, 1990]
From Graphs to Multigraphs

<table>
<thead>
<tr>
<th>Graph</th>
<th>Multigraph</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edge ((v, u))</td>
<td>Multiedge ((v, U))</td>
</tr>
<tr>
<td>(u, v \in V)</td>
<td>(v \in V, U \subset V)</td>
</tr>
</tbody>
</table>

Path  \(\implies\) Multipath

DAG  \(\implies\) Multi-DAG
The Binary Search Tree Game

The Algorithm VS The Adversary
The Binary Search Tree Game

- In each trial $t = 1 \ldots T$
  1. Predict (perhaps randomly) with a BST $\pi_t \in \mathbb{N}^n$
    - $\pi_t$ is in the form of depth sequence

\[
\pi_t = (2, 1, 4, 3, 2)
\]
The Binary Search Tree Game

- In each trial $t = 1 \ldots T$
  
  2. A **probability** vector $p_t \in [0, 1]^n$ is revealed

\[ \sum_i p_{t,i} = 1 \]

\[ p_t = (0.1, 0.2, 0.3, 0.15, 0.25) \]
In each trial $t = 1 \ldots T$

3. Incur the **average search cost** as the loss i.e. $\pi_t \cdot p_t$

\[
(2, 1, 4, 3, 2) \cdot (.1, .2, .3, .15, .25) = 2.55
\]
The Binary Search Tree Game

The goal is to minimize the regret

$$\sum_{t=1}^{T} \mathbb{E}[\pi_t \cdot p_t] - \min_{\pi \in \text{BST}_n} \sum_{t=1}^{T} \pi \cdot p_t$$
Challenges

⚠️ **Hard to maintain a distribution over all objects**

- Too many BSTs (\( C_n \) – nth Catalan number)
- Cannot keep one weight per permutation

⚠️ **Hard to even maintain a mean vector of a distribution over all objects**

- A mean vector \( \mathbf{f} \) lives in the convex hull of BSTs

\[ \mathcal{F} := \text{conv}(\text{BST}_n) \]

- \( \mathcal{F} \) has too many facets
  - In general, a description of \( \mathcal{F} \) in \( \mathbb{R}^n \) may not be even known
Challenges

⚠️ **Hard to maintain a distribution over all objects**

- Too many BSTs (\( C_n \) – \( n \)th Catalan number)
- Cannot keep one weight per permutation

⚠️ **Hard to even maintain a mean vector of a distribution over all objects**

- A mean vector \( \mathbf{f} \) lives in the convex hull of BSTs

\[
\mathcal{F} := \text{conv}(\text{BST}_n)
\]

- \( \mathcal{F} \) has too many facets
  - In general, a description of \( \mathcal{F} \) in \( \mathbb{R}^n \) may not be even known
Challenges

Graphical symbol

1. **Hard to maintain a distribution over all objects**
   - Too many BSTs (\( C_n \) – nth Catalan number)
   - Cannot keep one weight per permutation

Graphical symbol

2. **Hard to even maintain a mean vector of a distribution over all objects**
   - A mean vector \( \mathbf{f} \) lives in the convex hull of BSTs

\[ \mathcal{F} := \text{conv}(\text{BST}_n) \]

- \( \mathcal{F} \) has too many facets
  - In general, a description of \( \mathcal{F} \) in \( \mathbb{R}^n \) may not be even known
Challenges

.mailbox

Hard to maintain a distribution over all objects

▶ Too many BSTs (\(C_n – \) nth Catalan number)
▶ Cannot keep one weight per permutation

Hard to even maintain a mean vector of a distribution over all objects

▶ A mean vector \(f\) lives in the convex hull of BSTs

\[F := \text{conv}(\text{BST}_n)\]

▶ \(F\) has too many facets
  ▶ In general, a description of \(F\) in \(\mathbb{R}^n\) may not be even known
Dynamic Programming to the Rescue!

- Dynamic programming representation encodes the solutions.
- Each multipath encodes each object as a series of successive decisions (i.e., multiedges) over which the loss is linear.
- To learn these objects, one can equivalently learn multipaths with additive loss over trials.

(a) Knapsack 0/1

(b) Binary Search Tree
Dynamic Programming to the Rescue!

- Dynamic programming representation encodes the solutions.
- Each multipath encode each object as a series of successive decisions (i.e. multiedges) over with the loss is linear.
- To learn these objects, one can equivalently learn multipaths with additive loss over trials.
Dynamic Programming to the Rescue!

- Dynamic programming representation encodes the solutions.
- Each multipath encode each object as a series of successive decisions (i.e. multiedges) over with the loss is linear.
- **To learn these objects, one can equivalently learn multipaths with additive loss over trials.**
Dynamic Programming to the Rescue!

- Dynamic programming representation encodes the solutions.
- Each multipath encode each object as a series of successive decisions (i.e. multiedges) over with the loss is linear.
- To learn these objects, one can equivalently learn multipaths with additive loss over trials.

(g) Knapsack 0/1

(h) Binary Search Tree
Correspondence between the objects and multipaths.

- \( V := \) the set of vertices.
- \( M := \) the set of multiedges.
- \( \mathcal{P} := \) the set of multipaths.
  - Each multipath \( \pi \) is a count vector in \(|M|\)-dimensional space.
- A loss \( \ell_m \) is associated with each multiedge \( m \in M \).
V := the set of vertices.

M := the set of multiedges.

P := the set of multipaths.

Each multipath $\pi$ is a count vector in $|M|$-dimensional space.

A loss $\ell_m$ is associated with each multiedge $m \in M$. 

Correspondence between the objects and multipaths.
Dynamic Programming Multi-DAG

- $V := \text{the set of vertices.}$
- $M := \text{the set of multiedges.}$
- $P := \text{the set of multipaths.}$
  - Each multipath $\pi$ is a count vector in $|M|$-dimensional space.
- A loss $\ell_m$ is associated with each multiedge $m \in M$.

Correspondence between the objects and multipaths.
Dynamic Programming Multi-DAG

- $V := \text{the set of vertices.}$
- $M := \text{the set of multiedges.}$
- $P := \text{the set of multipaths.}$
  - Each multipath $\pi$ is a count vector in $|M|$-dimensional space.
  - A loss $\ell_m$ is associated with each multiedge $m \in M$.

Correspondence between the objects and multipaths.
Dynamic Programming Multi-DAG

- $V :=$ the set of vertices.
- $M :=$ the set of multiedges.
- $P :=$ the set of multipaths.
  - Each multipath $\pi$ is a count vector in $|M|$-dimensional space.
- A loss $\ell_m$ is associated with each multiedge $m \in M$.

Correspondence between the objects and multipaths.
V := the set of vertices.
M := the set of multiedges.
P := the set of multipaths.
Each multipath $\pi$ is a count vector in $|M|$-dimensional space.
A loss $\ell_m$ is associated with each multiedge $m \in M$. 

Correspondence between the objects and multipaths.
Dynamic Programming Multi-DAG

- $V :=$ the set of vertices.
- $M :=$ the set of multiedges.
- $\mathcal{P} :=$ the set of multipaths.
  - Each multipath $\pi$ is a count vector in $|M|$-dimensional space.
  - A loss $\ell_m$ is associated with each multiedge $m \in M$.

Correspondence between the objects and multipaths.
Dynamic Programming Multi-DAG

- $V :=$ the set of vertices.
- $M :=$ the set of multiedges.
- $\mathcal{P} :=$ the set of multipaths.
  - Each multipath $\pi$ is a count vector in $|M|$-dimensional space.
  - A loss $\ell_m$ is associated with each multiedge $m \in M$.

Correspondence between the objects and multipaths.
Current Tools in Our Toolbox

There are two main algorithms:

1. **Expanded Hedge (EH)** [Takimoto and Warmuth 2003]
   - Applying the **Hedge Algorithm** [Freund and Shapire, 1997] and **Randomized Weighted Majority** [Littlestone and Warmuth, 1994] to combinatorial objects.

2. **Component Hedge (CH)** [Koolen et. al. 2010]
   - There are also similar algorithms to CH e.g. [Suheiro et. al. 2012]
Current Tools in Our Toolbox

There are two main algorithms:

1. **Expanded Hedge (EH)** [Takimoto and Warmuth 2003]
   - Applying the **Hedge Algorithm** [Freund and Shapire, 1997] and **Randomized Weighted Majority** [Littlestone and Warmuth, 1994] to combinatorial objects.

2. **Component Hedge (CH)** [Koolen et. al. 2010]
   - There are also similar algorithms to CH e.g. [Suheiro et. al. 2012]
There are two main algorithms:

1. **Expanded Hedge (EH)** [Takimoto and Warmuth 2003]
   - Applying the **Hedge Algorithm** [Freund and Shapire, 1997] and **Randomized Weighted Majority** [Littlestone and Warmuth, 1994] to combinatorial objects.

2. **Component Hedge (CH)** [Koolen et. al. 2010]
   - There are also similar algorithms to CH
     e.g. [Suheiro et. al. 2012]
Hedge – Overview

▶ Maintains a distribution over all $N$ objects at trial $t$:

$$W_t = (w_{t,1}, \ldots, w_{t,N})$$

▶ Initialize $W_1$ to the uniform distribution.

▶ In trial $t = 1, 2, \ldots, T$

  ▶ **Sample** an object $\pi_t$ with probability $w_{t,\pi_t}$.
  ▶ Receive the loss of all objects $L_t = (L_{t,1}, \ldots, L_{t,N})$.
  ▶ Incur an expected loss $\mathbb{E}_{W_t}[L_{t,\pi_t}] = W_t \cdot L_t$
  ▶ **Update** $\hat{w}_{t,i} = w_{t,i} e^{-\eta \ell_{t,i}}$ for all $i \in \{1..N\}$
  ▶ **Normalize** the weights to sum up to obtain $W_{t+1}$.

**Problem:** $N$ is huge for combinatorial objects!
Hedge – Overview

🔹 Maintains a distribution over all $N$ objects at trial $t$:

$$W_t = (w_{t,1}, \ldots, w_{t,N})$$

🔹 Initialize $W_1$ to the uniform distribution.

🔹 In trial $t = 1, 2, \ldots, T$

   ▪ **Sample** an object $\pi_t$ with probability $w_{t,\pi_t}$.
   ▪ Receive the loss of all objects $L_t = (L_{t,1}, \ldots, L_{t,N})$.
   ▪ Incur an expected loss $\mathbb{E}_{W_t}[L_{t,\pi_t}] = W_t \cdot L_t$
   ▪ **Update** $\hat{w}_{t,i} = w_{t,i} e^{-\eta \ell_{t,i}}$ for all $i \in \{1..N\}$
   ▪ **Normalize** the weights to sum up to obtain $W_{t+1}$.

**Problem:** $N$ is huge for combinatorial objects!
Hedge – Overview

- Maintains a distribution over all $N$ objects at trial $t$:
  \[ W_t = (w_{t,1}, \ldots, w_{t,N}) \]

- Initialize $W_1$ to the uniform distribution.

- In trial $t = 1, 2, \ldots T$
  - Sample an object $\pi_t$ with probability $w_{t,\pi_t}$.
  - Receive the loss of all objects $L_t = (L_{t,1}, \ldots, L_{t,N})$.
  - Incur an expected loss $\mathbb{E}_{W_t}[L_{t,\pi_t}] = W_t \cdot L_t$
  - Update $\hat{w}_{t,i} = w_{t,i} e^{-\eta \ell_{t,i}}$ for all $i \in \{1..N\}$
  - Normalize the weights to sum up to obtain $W_{t+1}$.

Problem: $N$ is huge for combinatorial objects!
Hedge – Overview

- Maintains a distribution over all $N$ objects at trial $t$:
  \[ W_t = (w_{t,1}, \ldots, w_{t,N}) \]

- Initialize $W_1$ to the uniform distribution.

- In trial $t = 1, 2, \ldots T$
  - Sample an object $\pi_t$ with probability $w_{t,\pi_t}$.
  - Receive the loss of all objects $L_t = (L_{t,1}, \ldots, L_{t,N})$.
  - Incur an expected loss $\mathbb{E}_{W_t}[L_{t,\pi_t}] = W_t \cdot L_t$
  - Update $\hat{w}_{t,i} = w_{t,i} e^{-\eta \ell_{t,i}}$ for all $i \in \{1..N\}$
  - Normalize the weights to sum up to obtain $W_{t+1}$.

**Problem:** $N$ is huge for combinatorial objects!
Hedge – Overview

- Maintains a distribution over all $N$ objects at trial $t$:
  \[ W_t = (w_{t,1}, \ldots, w_{t,N}) \]

- Initialize $W_1$ to the uniform distribution.
- In trial $t = 1, 2, \ldots T$
  - **Sample** an object $\pi_t$ with probability $w_{t,\pi_t}$.
  - Receive the loss of all objects $L_t = (L_{t,1}, \ldots, L_{t,N})$.
  - Incur an expected loss $E_{W_t}[L_t, \pi_t] = W_t \cdot L_t$
  - **Update** $\hat{w}_{t,i} = w_{t,i} e^{-\eta \ell_{t,i}}$ for all $i \in \{1..N\}$
  - **Normalize** the weights to sum up to obtain $W_{t+1}$.

**Problem:** $N$ is huge for combinatorial objects!
Hedge – Overview

- Maintains a distribution over all $N$ objects at trial $t$:

$$W_t = (w_{t,1}, \ldots, w_{t,N})$$

- Initialize $W_1$ to the uniform distribution.

- In trial $t = 1, 2, \ldots T$
  - **Sample** an object $\pi_t$ with probability $w_{t,\pi_t}$.
  - Receive the loss of all objects $L_t = (L_{t,1}, \ldots, L_{t,N})$.
  - Incur an expected loss $\mathbb{E}_{W_t}[L_{t,\pi_t}] = W_t \cdot L_t$
  - **Update** $\hat{w}_{t,i} = w_{t,i} e^{-\eta \ell_{t,i}}$ for all $i \in \{1..N\}$
  - **Normalize** the weights to sum up to obtain $W_{t+1}$.

**Problem:** $N$ is huge for combinatorial objects!
Hedge – Overview

- Maintains a distribution over all $N$ objects at trial $t$:
  
  $W_t = (w_{t,1}, \ldots, w_{t,N})$

- Initialize $W_1$ to the uniform distribution.

- In trial $t = 1, 2, \ldots T$
  
  - **Sample** an object $\pi_t$ with probability $w_{t,\pi_t}$.
  - Receive the loss of all objects $L_t = (L_{t,1}, \ldots, L_{t,N})$.
  - Incur an expected loss $\mathbb{E}_{W_t}[L_{t,\pi_t}] = W_t \cdot L_t$
  - **Update** $\hat{w}_{t,i} = w_{t,i} e^{-\eta \ell_t,i}$ for all $i \in \{1..N\}$
  - Normalize the weights to sum up to obtain $W_{t+1}$.

**Problem:** $N$ is huge for combinatorial objects!
Hedge – Overview

- Maintains a distribution over all $N$ objects at trial $t$:
  
  $$W_t = (w_{t,1}, \ldots, w_{t,N})$$

- Initialize $W_1$ to the uniform distribution.

- In trial $t = 1, 2, \ldots, T$
  
  - **Sample** an object $\pi_t$ with probability $w_{t,\pi_t}$.
  
  - Receive the loss of all objects $L_t = (L_{t,1}, \ldots, L_{t,N})$.
  
  - Incur an expected loss $\mathbb{E}_{W_t}[L_{t,\pi_t}] = W_t \cdot L_t$

  - **Update** $\hat{w}_{t,i} = w_{t,i} e^{-\eta \ell_{t,i}}$ for all $i \in \{1..N\}$

  - **Normalize** the weights to sum up to obtain $W_{t+1}$.

**Problem:** $N$ is huge for combinatorial objects!
Hedge – Overview

- Maintains a distribution over all $N$ objects at trial $t$:
  \[ W_t = (w_{t,1}, \ldots, w_{t,N}) \]

- Initialize $W_1$ to the uniform distribution.

- In trial $t = 1, 2, \ldots T$
  - Sample an object $\pi_t$ with probability $w_{t,\pi_t}$.
  - Receive the loss of all objects $L_t = (L_{t,1}, \ldots, L_{t,N})$.
  - Incur an expected loss $\mathbb{E}_{W_t}[L_{t,\pi_t}] = W_t \cdot L_t$
  - Update $\hat{w}_{t,i} = w_{t,i} e^{-\eta \ell_{t,i}}$ for all $i \in \{1..N\}$
  - Normalize the weights to sum up to obtain $W_{t+1}$.

Problem: $N$ is huge for combinatorial objects!
Hedge – Overview

- Maintains a distribution over all $N$ objects at trial $t$:

$$W_t = (w_{t,1}, \ldots, w_{t,N})$$

- Initialize $W_1$ to the uniform distribution.

- In trial $t = 1, 2, \ldots T$
  - **Sample** an object $\pi_t$ with probability $w_{t,\pi_t}$.
  - Receive the loss of all objects $L_t = (L_{t,1}, \ldots, L_{t,N})$.
  - Incur an expected loss $\mathbb{E}_{W_t}[L_{t,\pi_t}] = W_t \cdot L_t$
  - **Update** $\hat{w}_{t,i} = w_{t,i} e^{-\eta \ell_{t,i}}$ for all $i \in \{1..N\}$
  - **Normalize** the weights to sum up to obtain $W_{t+1}$.

**Problem:** $N$ is huge for combinatorial objects!
Expanded Hedge (EH) – Idea

- Maintain a structured distribution $W_t$ on all multipaths s.t.
  - Easy to sample from $W_t$.
  - Easy to update $W_t$.
  - Easy to normalize $W_t$. 
Expanded Hedge (EH) – Idea

- Maintain a structured distribution $W_t$ on all multipaths s.t.
  - Easy to sample from $W_t$.
  - Easy to update $W_t$.
  - Easy to normalize $W_t$. 
Expanded Hedge (EH) – Idea

- Maintain a structured distribution $W_t$ on all multipaths s.t.
  - Easy to sample from $W_t$.
  - Easy to update $W_t$.
  - Easy to normalize $W_t$. 
Maintain a structured distribution $W_t$ on all multipaths s.t.
- Easy to sample from $W_t$.
- Easy to update $W_t$.
- Easy to normalize $W_t$. 
With each **multiedge** $m \in M$, associate a weight $w_{t,m}$.

Let $W_t$ be in *Stochastic Product Form*

1. The weights are in *product form*, i.e.

   $$W_t(\pi) = \prod_{m \in M} (w_{t,m})^{\pi_m}.$$

2. The weights are *stochastic*, i.e.

   $$\forall v \in V - T : \sum_{m \in M_v} w_{t,m} = 1.$$

3. The total multipath weight is one, i.e.

   $$\sum_{\pi} W_t(\pi) = 1.$$

**Sampling** time complexity $O(|M|)$

**Sampling** is easy.
With each multiedge \( m \in M \), associate a weight \( w_{t,m} \).

Let \( W_t \) be in *Stochastic Product Form*

1. The weights are in *product form*, i.e.
\[
W_t(\pi) = \prod_{m \in M} (w_{t,m})^{\pi_m}.
\]

2. The weights are *stochastic*, i.e.
\[
\forall v \in V - T : \sum_{m \in M_v} w_{t,m} = 1.
\]

3. The total multipath weight is one, i.e.
\[
\sum_{\pi} W_t(\pi) = 1.
\]

Sampling time complexity \( O(|M|) \)

Sampling is easy.
With each multiedge $m \in M$, associate a weight $w_{t,m}$.

Let $W_t$ be in Stochastic Product Form

1. The weights are in product form, i.e.
   \[
   W_t(\pi) = \prod_{m \in M} (w_{t,m})^{\pi_m}.
   \]

2. The weights are stochastic, i.e.
   \[
   \forall v \in V - T : \sum_{m \in M_v} w_{t,m} = 1.
   \]

3. The total multipath weight is one, i.e.
   \[
   \sum_{\pi} W_t(\pi) = 1.
   \]

Sampling time complexity $O(|M|)$

Sampling is easy.
With each multiedge $m \in M$, associate a weight $w_{t,m}$.

Let $W_t$ be in *Stochastic Product Form*

1. The weights are in *product form*, i.e.

$$W_t(\pi) = \prod_{m \in M} (w_{t,m})^{\pi_m}.$$

2. The weights are *stochastic*, i.e.

$$\forall v \in V - T : \sum_{m \in M_v} w_{t,m} = 1.$$

3. The total multipath weight is one, i.e.

$$\sum_{\pi} W_t(\pi) = 1.$$

**Sampling** time complexity $O(|M|)$

**Sampling** is easy.
With each multiedge $m \in M$, associate a weight $w_{t,m}$.

Let $W_t$ be in Stochastic Product Form

1. The weights are in product form, i.e.

$$W_t(\pi) = \prod_{m \in M} (w_{t,m})^{\pi_m}.$$ 

2. The weights are stochastic, i.e.

$$\forall v \in V - T : \sum_{m \in M_v} w_{t,m} = 1.$$ 

3. The total multipath weight is one, i.e.

$$\sum_{\pi} W_t(\pi) = 1.$$ 

Sampling time complexity $O(|M|)$

Sampling is easy.
With each multiedge \( m \in M \), associate a weight \( w_{t,m} \).

Let \( W_t \) be in *Stochastic Product Form*

1. The weights are in *product form*, i.e.

\[
W_t(\pi) = \prod_{m \in M} (w_{t,m})^{\pi_m}.
\]

2. The weights are *stochastic*, i.e.

\[
\forall v \in V - \mathcal{T} : \sum_{m \in M_v} w_{t,m} = 1.
\]

3. The total multipath weight is one, i.e.

\[
\sum_{\pi} W_t(\pi) = 1.
\]

**Sampling** time complexity \( O(|M|) \)

**Sampling** is easy.
With each multiedge \( m \in M \), associate a weight \( w_{t,m} \).

Let \( W_t \) be in *Stochastic Product Form*

1. The weights are in *product form*, i.e.

\[
W_t(\pi) = \prod_{m \in M} (w_{t,m})^{\pi_m}.
\]

2. The weights are *stochastic*, i.e.

\[
\forall v \in V - T : \sum_{m \in M_v} w_{t,m} = 1.
\]

3. The total multipath weight is one, i.e.

\[
\sum_{\pi} W_t(\pi) = 1.
\]

**Sampling** time complexity \( O(|M|) \)

Sampling is easy.
With each **multiedge** $m \in M$, associate a weight $w_{t,m}$.

Let $W_t$ be in *Stochastic Product Form*

1. The weights are in *product form*, i.e.

   $$W_t(\pi) = \prod_{m \in M} (w_{t,m})^{\pi_m}.$$

2. The weights are *stochastic*, i.e.

   $$\forall v \in V - T : \sum_{m \in M_v} w_{t,m} = 1.$$

3. The total multipath weight is one, i.e.

   $$\sum_{\pi} W_t(\pi) = 1.$$

**Sampling** time complexity $O(|M|)$

**Sampling** is easy.
Updated Hedge (EH) – Update

**Update** exploits the **additive loss**:

\[
W_{t+1}(\pi) = \frac{1}{Z} W_t(\pi) \exp(-\eta \pi \cdot \ell_t)
\]

\[
= \frac{1}{Z} \left( \prod_{m \in M} (w_{t,m}^{\pi_m}) \right) \exp \left[ -\eta \sum_{m \in M} \pi_m \ell_{t,m} \right]
\]

\[
= \frac{1}{Z} \prod_{m \in M} \left( w_{t,m} \exp \left[ -\eta \ell_{t,m} \right] \right)^{\pi_m} := \hat{w}_{t,m}
\]

**Time complexity** $O(|M|)$

**Multipath updates** break down to multiedge updates.
**Expanded Hedge (EH) – Update**

*Update* exploits the **additive loss**:

\[
W_{t+1}(\pi) = \frac{1}{Z} W_t(\pi) \exp(-\eta \pi \cdot \ell_t)
\]

\[
= \frac{1}{Z} \left( \prod_{m \in M} (w_{t,m})^{\pi_m} \right) \exp \left[ -\eta \sum_{m \in M} \pi_m \ell_{t,m} \right]
\]

\[
= \frac{1}{Z} \prod_{m \in M} \left( w_{t,m} \exp \left[ -\eta \ell_{t,m} \right]\right)^{\pi_m}.
\]

Time complexity \(O(|M|)\)

---

**Multipath updates break down to multiedge updates.**
**Update** exploits the **additive loss:**

\[
W_{t+1}(\pi) = \frac{1}{Z} W_t(\pi) \exp(-\eta \pi \cdot \ell_t)
\]

\[
= \frac{1}{Z} \left( \prod_{m \in M} (w_{t,m}^{\pi_m}) \right) \exp \left[ -\eta \sum_{m \in M} \pi_m \ell_{t,m} \right]
\]

\[
= \frac{1}{Z} \prod_{m \in M} \left( w_{t,m} \exp \left[ -\eta \ell_{t,m} \right] \right)^{\pi_m}.
\]

Time complexity \(O(|M|)\)

*Multipath updates break down to multiedge updates.*
Update exploits the additive loss:

\[
W_{t+1}(\pi) = \frac{1}{Z} W_t(\pi) \exp(-\eta \pi \cdot \ell_t)
\]

\[
= \frac{1}{Z} \left( \prod_{m \in M} (w_{t,m})^{\pi_m} \right) \exp \left[ -\eta \sum_{m \in M} \pi_m \ell_{t,m} \right]
\]

\[
= \frac{1}{Z} \prod_{m \in M} \left( w_{t,m} \exp \left[ -\eta \ell_{t,m} \right] \right)^{\pi_m}.
\]

Time complexity \( O(|M|) \)

Multipath updates break down to multiedge updates.
Normalization is done via Generalized Weight Pushing:

1. For sinks $v \in \mathcal{T}$,

$$Z_v := 1.$$ 

2. Recursing backwards in the DAG, for all non-sinks $v$:

$$Z_v := \sum_{m \in M_v} \hat{w}_{t,m} \prod_{u : (v,u) \in m} Z_u.$$ 

3. For each multiedge $m$ from $v$ to $u_1, \ldots, u_k$:

$$w_{t+1,m} := \hat{w}_{t,m} \left( \prod_{i=1}^{k} Z_{u_i} \right) / Z_v.$$ 

Time complexity $O(|M| + |V|)$

Efficient normalization.
Expanded Hedge (EH) – Normalization

Normalization is done via Generalized Weight Pushing:

1. For sinks $v \in \mathcal{T}$,

$$Z_v := 1.$$

2. Recursing backwards in the DAG, for all non-sinks $v$:

$$Z_v := \sum_{m \in M_v} \hat{w}_{t,m} \prod_{u:(v,u) \in m} Z_u.$$

3. For each multiedge $m$ from $v$ to $u_1, \ldots, u_k$:

$$w_{t+1,m} := \hat{w}_{t,m} \left( \prod_{i=1}^{k} Z_{u_i} \right)/Z_v.$$

Time complexity $O(|M| + |V|)$

Efficient normalization.
**Normalization** is done via **Generalized Weight Pushing**:

1. For sinks \( v \in \mathcal{T} \),

   \[
   Z_v := 1.
   \]

2. Recursing backwards in the DAG, for all non-sinks \( v \):

   \[
   Z_v := \sum_{m \in M_v} \frac{\prod_{u: (v, u) \in m} Z_u}{Z_v} \hat{w}_{t,m}.
   \]

3. For each multiedge \( m \) from \( v \) to \( u_1, \ldots, u_k \):

   \[
   w_{t+1,m} := \frac{\hat{w}_{t,m} \left( \prod_{i=1}^{k} Z_{u_i} \right)}{Z_v}.
   \]

Time complexity \( O(\|M\| + |V|) \)

**Efficient** normalization.
Normalization is done via **Generalized Weight Pushing**:

1. For sinks \( v \in \mathcal{T} \),

   \[
   Z_v := 1.
   \]

2. Recursing backwards in the DAG, for all non-sinks \( v \):

   \[
   Z_v := \sum_{m \in \mathcal{M}_v} \hat{w}_{t,m} \prod_{u : (v, u) \in m} Z_u.
   \]

3. For each multiedge \( m \) from \( v \) to \( u_1, \ldots, u_k \):

   \[
   w_{t+1, m} := \hat{w}_{t,m} \left( \prod_{i=1}^{k} Z_{u_i} \right) / Z_v.
   \]

Time complexity \( O(|M| + |V|) \)

Efficient normalization.
Normalization is done via Generalized Weight Pushing:

1. For sinks $v \in \mathcal{T}$,
   \[ Z_v := 1. \]

2. Recursing backwards in the DAG, for all non-sinks $v$:
   \[ Z_v := \sum_{m \in M_v} \hat{w}_{t,m} \prod_{u: (v,u) \in m} Z_u. \]

3. For each multiedge $m$ from $v$ to $u_1, \ldots, u_k$:
   \[ w_{t+1,m} := \hat{w}_{t,m} \left( \prod_{i=1}^{k} Z_{u_i} \right) / Z_v. \]

Time complexity $O(|M| + |V|)$

Efficient normalization.
**Expanded Hedge (EH) – Normalization**

**Normalization** is done via **Generalized Weight Pushing**:

1. For sinks \( v \in \mathcal{T} \),
   
   \[ Z_v := 1. \]

2. Recursing backwards in the DAG, for all non-sinks \( v \):
   
   \[ Z_v := \sum_{m \in M_v} \hat{w}_{t,m} \prod_{u: (v,u) \in m} Z_u. \]

3. For each multiedge \( m \) from \( v \) to \( u_1, \ldots, u_k \):
   
   \[ w_{t+1,m} := \hat{w}_{t,m} \left( \prod_{i=1}^{k} Z_{u_i} \right)/Z_v. \]

**Time complexity** \( O(|M| + |V|) \)

**Efficient normalization.**
Normalization is done via Generalized Weight Pushing:

1. For sinks $v \in \mathcal{T}$,
   $$Z_v := 1.$$

2. Recursing backwards in the DAG, for all non-sinks $v$:
   $$Z_v := \sum_{m \in M_v} \hat{w}_{t,m} \prod_{u:(v,u) \in m} Z_u.$$

3. For each multiedge $m$ from $v$ to $u_1, \ldots, u_k$:
   $$w_{t+1,m} := \hat{w}_{t,m} \left( \prod_{i=1}^{k} Z_{u_i} \right) / Z_v.$$

Time complexity $O(|M| + |V|)$

Efficient normalization.
Example of (generalized) weight pushing on regular DAG.

[Mohri, 2009]
Expanded Hedge (EH) – Normalization

Example of (generalized) weight pushing on regular DAG.

[Diagram showing weight pushing process]

[Mohri, 2009]
Example of (generalized) weight pushing on regular DAG.

\[ \text{Expanded Hedge (EH) – Normalization} \]

\[ [\text{Mohri, 2009}] \]
Example of (generalized) weight pushing on regular DAG.

[Mohri, 2009]
To initialize $W_1$ to uniform distribution:

1. Set all the weights to 1.
2. Apply Generalized Weight Pushing.
To initialize $W_1$ to uniform distribution:

1. Set all the weights to 1.
2. Apply Generalized Weight Pushing.
To initialize $W_1$ to uniform distribution:

1. Set all the weights to 1.
2. Apply Generalized Weight Pushing.
Expanded Hedge (EH) – Initialization

- To initialize $W_1$ to uniform distribution:
  1. Set all the weights to 1.
  2. Apply Generalized Weight Pushing.
To initialize $W_1$ to uniform distribution:

1. Set all the weights to 1.
2. Apply Generalized Weight Pushing.
To initialize $W_1$ to uniform distribution:

1. Set all the weights to 1.
2. Apply Generalized Weight Pushing.
With proper tuning of the learning rate $\eta$

$$\text{Regret}_{EH} \leq \sqrt{2L^* D \log N} + D \log N.$$ 

where

- $L^* :=$ total loss of the best solution
- $D :=$ upperbound on the number of multiedges in the solution

Binary Search Tree

$$\text{Regret}_{EH} = O(n\sqrt{L^*})$$
With proper tuning of the learning rate $\eta$

\[
\text{Regret}_{\text{EH}} \leq \sqrt{2L^* D \log N} + D \log N.
\]

where

- $L^*$ := total loss of the best solution
- $D$ := upperbound on the number of multiedges in the solution

Binary Search Tree

\[
\text{Regret}_{\text{EH}} = O(n\sqrt{L^*})
\]
Component Hedge (CH) – Overview

- With each multiedge $m \in M$, associate a flow weight $f_m$.
- Having an additive loss, it is sufficient to maintain a mean vector $f \in \mathcal{F}$ of multipaths:

$$\mathbb{E}[\pi \cdot \ell] = \mathbb{E}[\pi] \cdot \ell = f \cdot \ell$$
For $t = 1 \ldots T$

- **Prediction**
  Sample $\pi_t$ from a $D$ s.t. $\mathbb{E}_D[\pi_t] = f_t$

- **Receive loss** $\ell_t \in [0,1]|M|$.
  Incur expected loss $\mathbb{E}[\pi_t \cdot \ell_t] = f_t \cdot \ell_t$

- **Update**
  $\forall m \in M, \quad \hat{f}_{t,m} = f_{t,m} \exp(-\eta \ell_{t,m})$

- **Projection**
  Find $f_{t+1} := \arg \min_{f \in \mathcal{F}} \Delta(f \| \hat{f}_t)$.
Component Hedge (CH) – Overview

For $t = 1 \ldots T$

- **Prediction**
  
  Sample $\pi_t$ from a $\mathcal{D}$ s.t. $E_D[\pi_t] = f_t$

- **Receive loss** $\ell_t \in [0,1]|M|$
  
  Incur expected loss $E[\pi_t \cdot \ell_t] = f_t \cdot \ell_t$

- **Update**
  
  $\forall m \in M$, $\hat{f}_{t,m} = f_{t,m} \exp(-\eta \ell_{t,m})$

- **Projection**
  
  Find $f_{t+1} := \arg \min_{f \in \mathcal{F}} \Delta(f \| \hat{f}_t)$. 

\[ f_{t+1} := \arg \min_{f \in \mathcal{F}} \Delta(f \| \hat{f}_t). \]
Component Hedge (CH) – Overview

For $t = 1 \ldots T$

- **Prediction**
  
  Sample $\pi_t$ from a $\mathcal{D}$ s.t. $\mathbb{E}_\mathcal{D}[\pi_t] = f_t$

- Receive loss $\ell_t \in [0,1]^{|M|}$.
  
  Incur expected loss $\mathbb{E} [\pi_t \cdot \ell_t] = f_t \cdot \ell_t$

- **Update**
  
  $\forall m \in M$, $\hat{f}_{t,m} = f_{t,m} \exp(-\eta \ell_{t,m})$

- **Projection**
  
  Find $f_{t+1} := \arg \min_{f \in \mathcal{F}} \Delta(f || \hat{f}_t)$. 
For $t = 1 \ldots T$

- **Prediction**
  
  Sample $\pi_t$ from a $\mathcal{D}$ s.t. $\mathbb{E}_D[\pi_t] = f_t$

- **Receive loss** $\ell_t \in [0,1]^{\vert M \vert}$.
  
  Incur expected loss $\mathbb{E}[\pi_t \cdot \ell_t] = f_t \cdot \ell_t$

- **Update**
  
  $\forall m \in M, \quad \hat{f}_{t,m} = f_{t,m} \exp(-\eta \ell_{t,m})$

- **Projection**
  
  Find $f_{t+1} := \arg \min_{f \in \mathcal{F}} \Delta(f \| \hat{f}_t).$
Component Hedge (CH) – Overview

For \( t = 1 \ldots T \)

- **Prediction**
  
  Sample \( \pi_t \) from a \( \mathcal{D} \) s.t. \( \mathbb{E}_\mathcal{D}[\pi_t] = f_t \)

- **Receive loss** \( \ell_t \in [0,1]^{|M|} \).
  
  Incur expected loss \( \mathbb{E}[\pi_t \cdot \ell_t] = f_t \cdot \ell_t \)

- **Update**
  
  \( \forall m \in M, \quad \hat{f}_{t,m} = f_{t,m} \exp(-\eta \ell_{t,m}) \)

- **Projection**
  
  Find \( f_{t+1} := \arg \min_{f \in \mathcal{F}} \Delta(f \| \hat{f}_t) \).
Component Hedge (CH) – Unit-Flow Polytope $\mathcal{F}$

- With each **multiedge** $m \in M$, associate a flow weight $f_m$.
- Maintain a mean vector $f$ on all multipaths in the polytope $\mathcal{F}$ below:

1. Unit outflow from the source $s$:

\[
\sum_{m \in M_s^{(out)}} f_m = 1
\]

2. Flow conservation at each internal node $v \in V - \mathcal{T} - \{s\}$:

\[
\sum_{m \in M_v^{(out)}} f_m = \sum_{m \in M_v^{(in)}} f_m
\]
With each **multiedge** \( m \in M \), associate a **flow weight** \( f_m \).

Maintain a **mean vector** \( f \) on all multipaths in the polytope \( \mathcal{F} \) below:

1. **Unit outflow from the source** \( s \):\n
   \[
   \sum_{m \in M_s^{(out)}} f_m = 1
   \]

2. **Flow conservation at each internal node** \( v \in V - T - \{s\} \):\n
   \[
   \sum_{m \in M_v^{(out)}} f_m = \sum_{m \in M_v^{(in)}} f_m
   \]
With each **multiedge** $m \in M$, associate a **flow weight** $f_m$.

Maintain a **mean vector** $f$ on all multipaths in the polytope $\mathcal{F}$ below:

1. **Unit outflow from the source** $s$:

   $$\sum_{m \in M_s^{(out)}} f_m = 1$$

2. **Flow conservation** at each internal node $v \in V - T - \{s\}$:

   $$\sum_{m \in M_v^{(out)}} f_m = \sum_{m \in M_v^{(in)}} f_m$$
With each **multiedge** \( m \in M \), associate a **flow** weight \( f_m \).

Maintain a **mean vector** \( f \) on all multipaths in the polytope \( \mathcal{F} \) below:

1. Unit outflow from the source \( s \):

   \[
   \sum_{m \in M_s^{(out)}} f_m = 1
   \]

2. Flow conservation at each internal node \( v \in V - \mathcal{T} - \{s\} \):

   \[
   \sum_{m \in M_v^{(out)}} f_m = \sum_{m \in M_v^{(in)}} f_m
   \]
For prediction, find a $\mathcal{D}$ with efficient sampling s.t. $\mathbb{E}_{\mathcal{D}}[\pi_t] = f_t$

- We introduce a $\mathcal{D}$ in Stochastic Product Form.
- For each multiedge $m \in M$, find weight $w_{t,m}$:
  1. For each vertex $v \in V$, find the incoming flow
     $$f_{\text{in}}(v) := \sum_{m \in M_v^{\text{out}}} f_{t,m}$$
     with initialization $f_{\text{in}}(s) := 1$
  2. For each multiedge $m$ from node $v$, $w_{t,m} = \frac{f_{t,m}}{f_{\text{in}}(v)}$
- Time complexity $O(|V| + |M|)$
Component Hedge (CH) – Prediction

For **prediction**, find a $\mathcal{D}$ with efficient sampling s.t. $\mathbb{E}_\mathcal{D}[\pi_t] = f_t$

- We introduce a $\mathcal{D}$ in **Stochastic Product Form**.
  - For each multiedge $m \in M$, find weight $w_{t,m}$:
    1. For each vertex $v \in V$, find the incoming flow $f_{\text{in}}(v) := \sum_{m \in M^{(\text{out})}_v} f_{t,m}$ with initialization $f_{\text{in}}(s) := 1$
    2. For each multiedge $m$ from node $v$, $w_{t,m} = \frac{f_{t,m}}{f_{\text{in}}(v)}$
  - Time complexity $O(|V| + |M|)$
For **prediction**, find a $D$ with efficient sampling s.t. $\mathbb{E}_D[\pi_t] = f_t$

- We introduce a $D$ in *Stochastic Product Form*.
- For each multiedge $m \in M$, find weight $w_{t,m}$:
  1. For each vertex $v \in V$, find the incoming flow
     $$f_{\text{in}}(v) := \sum_{m \in M_{v(\text{out})}} f_{t,m}$$
     with initialization $f_{\text{in}}(s) := 1$
  2. For each multiedge $m$ from node $v$, $w_{t,m} = \frac{f_{t,m}}{f_{\text{in}}(v)}$
- Time complexity $O(|V| + |M|)$
Component Hedge (CH) – Prediction

For prediction, find a $\mathcal{D}$ with efficient sampling s.t. $\mathbb{E}_\mathcal{D}[\pi_t] = f_t$

- We introduce a $\mathcal{D}$ in Stochastic Product Form.
- For each multiedge $m \in M$, find weight $w_{t,m}$:
  1. For each vertex $v \in V$, find the incoming flow $f_{in}(v) := \sum_{m \in M_v^{out}} f_{t,m}$ with initialization $f_{in}(s) := 1$
  2. For each multiedge $m$ from node $v$, $w_{t,m} = \frac{f_{t,m}}{f_{in}(v)}$

- Time complexity $O(|V| + |M|)$
Component Hedge (CH) – Prediction

For **prediction**, find a $\mathcal{D}$ with efficient sampling s.t. $\mathbb{E}_{\mathcal{D}}[\pi_t] = f_t$

- We introduce a $\mathcal{D}$ in *Stochastic Product Form*.

- For each multiedge $m \in M$, find weight $w_{t,m}$:
  1. For each vertex $v \in V$, find the incoming flow $f_{\text{in}}(v) := \sum_{m \in M^{(\text{out})}_v} f_{t,m}$ with initialization $f_{\text{in}}(s) := 1$
  2. For each multiedge $m$ from node $v$, $w_{t,m} = \frac{f_{t,m}}{f_{\text{in}}(v)}$

- Time complexity $O(|V| + |M|)$
For **prediction**, find a $D$ with efficient sampling s.t. $\mathbb{E}_D[\pi_t] = f_t$

- We introduce a $D$ in *Stochastic Product Form*.
- For each multiedge $m \in M$, find weight $w_{t,m}$:
  1. For each vertex $v \in V$, find the incoming flow $f_{in}(v) := \sum_{m \in M_{v}^{out}} f_{t,m}$ with initialization $f_{in}(s) := 1$
  2. For each multiedge $m$ from node $v$, $w_{t,m} = \frac{f_{t,m}}{f_{in}(v)}$
- Time complexity $O(|V| + |M|)$
Example of finding weights $w$ from flows $f$ using in-flows on regular DAG.
Example of finding weights $w$ from flows $f$ using in-flows on regular DAG.
Example of finding weights $w$ from flows $f$ using in-flows on regular DAG.
Component Hedge (CH) – Update

**Update**

- **Motivation**
  \[ \hat{f}_t = \arg \min_f \Delta(f||f_t) + \eta f \cdot \ell \]

- For each multiedge \( m \in M \)
  \[ \hat{f}_{t,m} := f_{t,m} \cdot e^{-\eta \ell_{t,m}} \]

- Time complexity \( O(|M|) \)
Component Hedge (CH) – Update

**Update**

- **Motivation**
  
  $$\hat{f}_t = \arg \min_f \Delta(f||f_t) + \eta f \cdot \ell$$

- For each multiedge $m \in M$
  
  $$\hat{f}_{t,m} := f_{t,m} \cdot e^{-\eta \ell_{t,m}}$$

- Time complexity $O(|M|)$
Update

Motivation

\[ \hat{f}_t = \arg \min_f \Delta(f\|f_t) + \eta f \cdot \ell \]

For each multiedge \( m \in M \)

\[ \hat{f}_{t,m} := f_{t,m} \cdot e^{-\eta \ell_{t,m}} \]

Time complexity \( O(|M|) \)
Component Hedge (CH) – Update

**Update**

- **Motivation**

\[ \hat{f}_t = \arg \min_f \Delta(f || f_t) + \eta f \cdot \ell \]

- For each multiedge \( m \in M \)

\[ \hat{f}_{t,m} := f_{t,m} \cdot e^{-\eta \ell_{t,m}} \]

- Time complexity \( O(|M|) \)
**Projection** is done via **Iterative Bregman Projections**: 

- Cycle through the flow constraints and enforce the equality:
  1. Normalize the source outflow to 1.
  2. Given a vertex $v \in V - T - \{s\}$, scale the adjacent multiedges of $v$ s.t.
    
    $$f_{\text{out}}(v) \leftarrow f_{\text{in}}(v) \leftarrow \sqrt{f_{\text{out}}(v) \cdot f_{\text{in}}(v)}$$

- Time complexity $O(|V||M| \log(1/\epsilon))$

* Projection is only needed if the model must make a prediction.
** believed to have linear convergence.
**Projection** is done via **Iterative Bregman Projections:**

- Cycle through the flow constraints and enforce the equality:
  1. Normalize the source outflow to 1.
  2. Given a vertex $v \in V - T - \{s\}$, scale the adjacent multiedges of $v$ s.t.

\[
\hat{f}_t \leftarrow f_{\text{out}}(v) \leftarrow \sqrt{f_{\text{out}}(v) \cdot f_{\text{in}}(v)}
\]

- Time complexity $O(|V| |M| \log(1/\epsilon))^{**}$

---

* Projection is only needed if the model must make a prediction.
** believed to have linear convergence.
**Projection** is done via **Iterative Bregman Projections:**

- Cycle through the flow constraints and enforce the equality:
  1. Normalize the source outflow to 1.
  2. Given a vertex $v \in V - T - \{s\}$, scale the adjacent multiedges of $v$ s.t.

\[
\hat{f}_t \leftarrow f_{\text{in}}(v) \leftarrow \sqrt{f_{\text{out}}(v) \cdot f_{\text{in}}(v)}
\]

- Time complexity $O(|V||M| \log(1/\epsilon))$**

* Projection is only needed if the model must make a prediction.
** believed to have linear convergence.
**Projection** is done via **Iterative Bregman Projections**:

- **Cycle through the flow constraints and enforce the equality:**
  1. Normalize the source outflow to 1.
  2. Given a vertex $v \in V - \mathcal{T} - \{s\}$, scale the adjacent multiedges of $v$ s.t.

     $$f_{\text{out}}(v) \leftarrow f_{\text{in}}(v) \leftarrow \sqrt{f_{\text{out}}(v) \cdot f_{\text{in}}(v)}$$

- **Time complexity** $O(|V||M| \log(1/\epsilon))**

---

* Projection is only needed if the model must make a prediction.
** believed to have linear convergence.
**Projection** is done via **Iterative Bregman Projections**:

- Cycle through the flow constraints and enforce the equality:
  1. Normalize the source outflow to 1.
  2. Given a vertex $v \in V - T - \{s\}$, scale the adjacent multiedges of $v$ s.t.

\[
    f_{\text{out}}(v) \leftarrow f_{\text{in}}(v) \leftarrow \sqrt{f_{\text{out}}(v) \cdot f_{\text{in}}(v)}
\]

- Time complexity $O(|V| |M| \log(1/\epsilon))^{**}$

---

* Projection is only needed if the model must make a prediction.
** believed to have linear convergence.
Implicit initialization:

1. Set $\hat{f}_0 := \frac{1}{|V|^2} \mathbf{1}$.

2. Initialize $f_1 := \arg \min_{f \in \mathcal{F}} \Delta(f || \hat{f}_0)$.  

[Rahmanian, Helmbold, Vishwanathan, 2018]
Component Hedge (CH) – Regret Bounds

- With proper tuning of the learning rate $\eta$
  - General:
    $$ R_{CH} \leq \sqrt{2L^* D (2 \log |V| + \log D)} + 2D \log |V| + D \log D. $$
  - Bit-vectors:
    $$ R_{CH} \leq \sqrt{4L^* D \log |V|} + 2D \log |V|. $$
  - Binary Search Tree
    $$ \text{Regret}_{CH} = \mathcal{O}(n^{\frac{1}{2}} (\log n)^{\frac{1}{2}} \sqrt{L^*}). $$
With proper tuning of the learning rate $\eta$

- **General:**
  \[
  R_{CH} \leq \sqrt{2 L^* D (2 \log |V| + \log D)} + 2D \log |V| + D \log D.
  \]

- **Bit-vectors:**
  \[
  R_{CH} \leq \sqrt{4 L^* D \log |V|} + 2D \log |V|.
  \]

- **Binary Search Tree**

  \[
  \text{Regret}_{CH} = O(n^{\frac{1}{2}} (\log n)^{\frac{1}{2}} \sqrt{L^*})
  \]
With proper tuning of the learning rate $\eta$

- General:

\[ R_{\text{CH}} \leq \sqrt{2 L^* D (2 \log |V| + \log D)} + 2D \log |V| + D \log D. \]

- Bit-vectors:

\[ R_{\text{CH}} \leq \sqrt{4 L^* D \log |V|} + 2D \log |V|. \]

- Binary Search Tree

\[ \text{Regret}_{\text{CH}} = \mathcal{O}(n^{\frac{1}{2}} (\log n)^{\frac{1}{2}} \sqrt{L^*}) \]
### Performance Comparison

<table>
<thead>
<tr>
<th>Problem</th>
<th>FPL</th>
<th>EH</th>
<th>CH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimal Binary Search Trees</td>
<td>$\mathcal{O}(n (\log n)^{\frac{1}{2}} \sqrt{L^*})$</td>
<td>$\mathcal{O}(n \sqrt{L^*})$</td>
<td>$\mathcal{O}(n^{\frac{1}{2}} (\log n)^{\frac{1}{2}} \sqrt{L^*})$</td>
</tr>
<tr>
<td>Matrix-Chain Multiplications</td>
<td>$\mathcal{O}(n (\log n)^{\frac{1}{2}} \sqrt{L^*})$</td>
<td>$\mathcal{O}(n (d_{\text{max}})^{\frac{3}{2}} \sqrt{L^*})$</td>
<td>$\mathcal{O}(n^{\frac{1}{2}} (\log n)^{\frac{1}{2}} (d_{\text{max}})^{\frac{3}{2}} \sqrt{L^*})$</td>
</tr>
<tr>
<td>Knapsack</td>
<td></td>
<td>$\mathcal{O}(n \sqrt{L^*})$</td>
<td>$\mathcal{O}(n^{\frac{1}{2}} (\log n)^{\frac{1}{2}} \sqrt{L^*})$</td>
</tr>
<tr>
<td>Rod Cutting</td>
<td>$\mathcal{O}(n (\log n)^{\frac{1}{2}} \sqrt{L^*})$</td>
<td>$\mathcal{O}(n \sqrt{L^*})$</td>
<td>$\mathcal{O}(n^{\frac{1}{2}} (\log n)^{\frac{1}{2}} \sqrt{L^*})$</td>
</tr>
<tr>
<td>Weighted Interval Scheduling</td>
<td>$\mathcal{O}(n (\log n)^{\frac{1}{2}} \sqrt{L^*})$</td>
<td>$\mathcal{O}(n \sqrt{L^*})$</td>
<td>$\mathcal{O}(n^{\frac{1}{2}} (\log n)^{\frac{1}{2}} \sqrt{L^*})$</td>
</tr>
</tbody>
</table>

**Notes:**

- FPL = Follow the Perturbed Leader
- Using “losses” instead of “gains”.
Expanded Hedge

\[ w \in \mathbb{R}^{M} \]

Stochastic Product Form

- Stochastic
- Multiplicative Updates
- Weight Pushing

Dynamic Programming

starting from \( s \in V \)

\[ f_m := w_m f_{in}(v), \ m \in M_v^{(out)} \]

Preserving Mean

“Conditional Outflow”

in parallel

\[ w_m := \frac{f_m}{f_{in}(v)}, \ m \in M_v^{(out)} \]

Component Hedge

\[ f \in \mathbb{R}^{M} \]

Mean Form

- Unit-Flow Polytope
- Multiplicative Updates
- Projection
Conclusions and Future Work

Online learning of combinatorial objects is hard:
1. Exponentially many objects
2. Unknown or ill-behaved polytope

Parameterizing the decisions of the dynamic programming algorithm to obtain a novel representation:
1. Distribution with efficient sampling
2. Well-behaved polytope (*Extended Formulation*)
Conclusions and Future Work

Online learning of combinatorial objects is hard:
1. Exponentially many objects
2. Unknown or ill-behaved polytope

Parameterizing the decisions of the dynamic programming algorithm to obtain a novel representation:
1. Distribution with efficient sampling
2. Well-behaved polytope (Extended Formulation)
Conclusions and Future Work

Online learning of combinatorial objects is hard:

1. Exponentially many objects
2. Unknown or ill-behaved polytope

Parameterizing the decisions of the dynamic programming algorithm to obtain a novel representation:

1. Distribution with efficient sampling
2. Well-behaved polytope (Extended Formulation)
Conclusions and Future Work

- Online learning of combinatorial objects is hard:
  1. Exponentially many objects
  2. Unknown or ill-behaved polytope

- Parameterizing the decisions of the dynamic programming algorithm to obtain a novel representation:
  1. Distribution with efficient sampling
  2. Well-behaved polytope (*Extended Formulation*)
Online learning of combinatorial objects is hard:

1. Exponentially many objects
2. Unknown or ill-behaved polytope

Parameterizing the decisions of the dynamic programming algorithm to obtain a novel representation:

1. Distribution with efficient sampling
2. Well-behaved polytope (*Extended Formulation*)
Conclusions and Future Work

Online learning of combinatorial objects is hard:

1. Exponentially many objects
2. Unknown or ill-behaved polytope

Parameterizing the decisions of the dynamic programming algorithm to obtain a novel representation:

1. Distribution with efficient sampling
2. Well-behaved polytope (*Extended Formulation*)
Conclusions and Future Work

Online learning of combinatorial objects is hard:
1. Exponentially many objects
2. Unknown or ill-behaved polytope

Parameterizing the decisions of the dynamic programming algorithm to obtain a novel representation:
1. Distribution with efficient sampling
2. Well-behaved polytope (Extended Formulation)
Conclusions and Future Work

- Online Dynamic Programming reduces to Online Multipath Learning
  1. Expanded Hedge
  2. Component Hedge

- **Application:** Extending EH and CH to multi-armed bandit settings.
  - Both semi-bandit and full bandit settings
  - Using ComBand [Cesa-Bianchi and Lugosi, 2012], Online Shortest Path [György et al., 2007], and other general techniques [Audibert et al., 2013]
Conclusions and Future Work

- **Online Dynamic Programming reduces to Online Multipath Learning**
  1. Expanded Hedge
  2. Component Hedge

- **Application:** Extending EH and CH to multi-armed bandit settings.
  - Both semi-bandit and full bandit settings
  - Using ComBand [Cesa-Bianchi and Lugosi, 2012], Online Shortest Path [Győrgy et al., 2007], and other general techniques [Audibert et al., 2013]
Online Dynamic Programming reduces to Online Multipath Learning

1. Expanded Hedge
2. Component Hedge

Application: Extending EH and CH to multi-armed bandit settings.

- Both semi-bandit and full bandit settings
- Using ComBand [Cesa-Bianchi and Lugosi, 2012], Online Shortest Path [György et al., 2007], and other general techniques [Audibert et al., 2013]
Conclusions and Future Work

- Online Dynamic Programming reduces to Online Multipath Learning
  1. Expanded Hedge
  2. Component Hedge

- **Application:** Extending EH and CH to multi-armed bandit settings.
  - Both semi-bandit and full bandit settings
  - Using ComBand [Cesa-Bianchi and Lugosi, 2012], Online Shortest Path [György et al., 2007], and other general techniques [Audibert et al., 2013]
Conclusions and Future Work

- Online Dynamic Programming reduces to Online Multipath Learning
  1. Expanded Hedge
  2. Component Hedge

- **Application**: Extending EH and CH to multi-armed bandit settings.
  - Both semi-bandit and full bandit settings
  - Using ComBand [Cesa-Bianchi and Lugosi, 2012], Online Shortest Path [Győrgy et al., 2007], and other general techniques [Audibert et al., 2013]
Conclusions and Future Work

- **Online Dynamic Programming reduces to Online Multipath Learning**
  1. Expanded Hedge
  2. Component Hedge

- **Application:** Extending EH and CH to multi-armed bandit settings.
  - Both semi-bandit and full bandit settings
  - Using ComBand [Cesa-Bianchi and Lugosi, 2012], Online Shortest Path [György et al., 2007], and other general techniques [Audibert et al., 2013]
Thanks!

Questions?
