Online Non-Additive Path Learning under Full and Partial Information

Holakou Rahmanian (Microsoft)

Joint work with **Corinna Cortes, Vitaly Kuznetsov** (Google Research NYC),
Mehryar Mohri (Courant Institute & Google Research NYC),
Manfred Warmuth (Google Inc., Zürich & UCSC)

ALT 2019

Structured Prediction

- Structured output:

$$Y = Y_1 \times Y_2 \times \cdots \times Y_\ell$$

- Examples:
  - Machine translation.
  - Automatic speech recognition.
  - Optical character recognition.
  - Computer vision.

Structured Prediction

- Structured output:

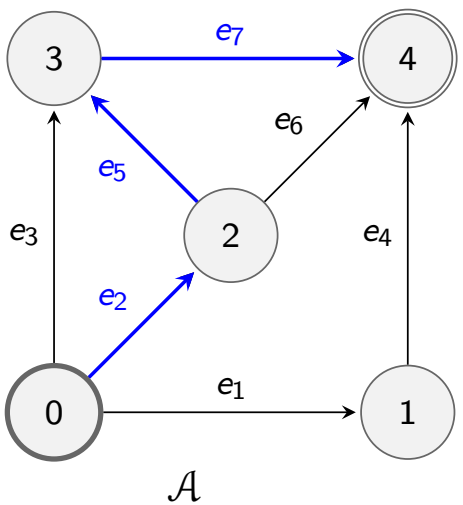
$$Y = Y_1 \times Y_2 \times \cdots \times Y_\ell$$

- Examples:
  - Machine translation.
  - Automatic speech recognition.
  - Optical character recognition.
  - Computer vision.

Online Path Learning

- Structured prediction can be usually represented by directed graph.
  - Each edge represents a different substructure.
- The most common and well-studied loss/gain is **additive**:
  - The gain of a given path is the sum of the gains of the edges along that path.
- Example: For path $\pi = e_2\, e_5\, e_7$

$$g(\pi) = g(e_2) + g(e_5) + g(e_7)$$



$\mathcal{A}$

Online Path Learning

- Extensive work on additive gains/losses:
  - Full information:
    - *Expanded Hedge* [Takimoto & Warmuth, 2003].
    - *Follow-the-Perturbed-Leader* [Kalai & Vempala, 2005].
    - *Component Hedge* [Koolen et al., 2010].
  - Different bandit settings:
    - Algorithm of [Győrgy et al., 2007].
    - ComBand [Cesa-Bianchi and Lugosi, 2012].

What if the loss/gain is **not additive**?

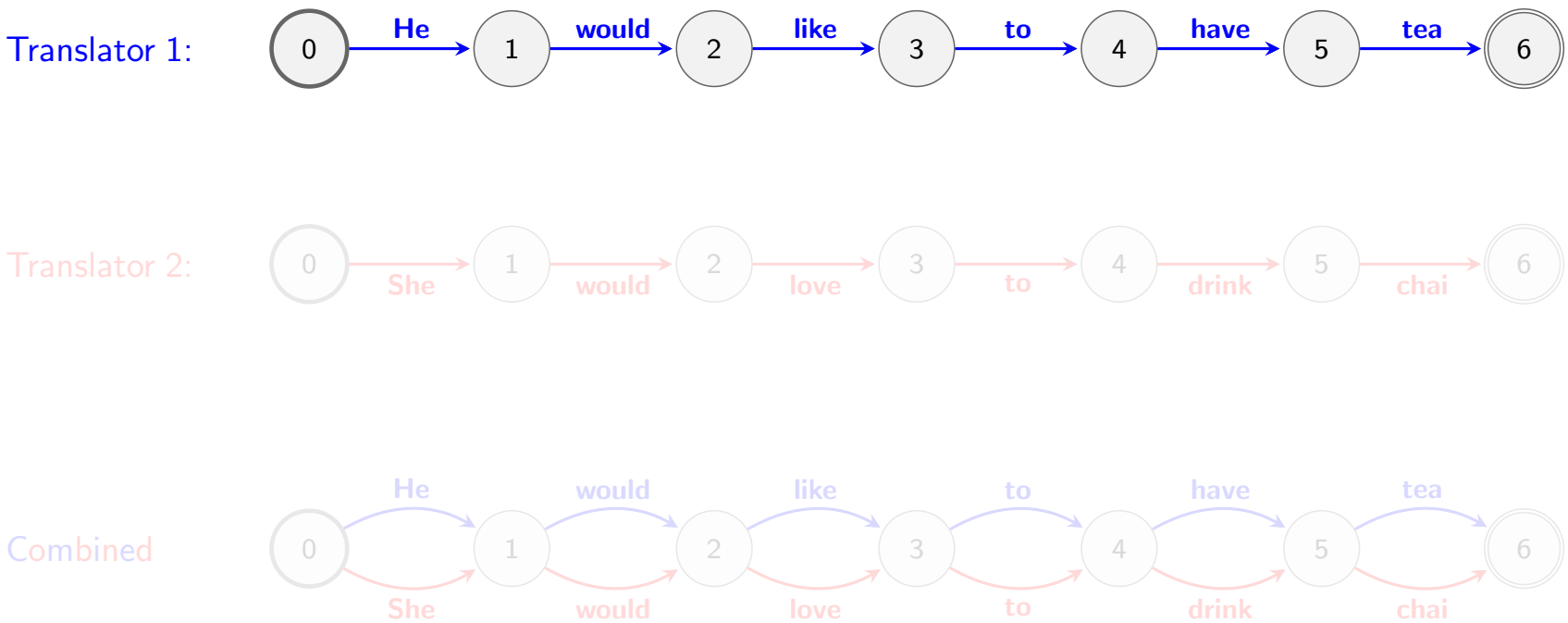
Online Path Learning

- Extensive work on additive gains/losses:
  - Full information:
    - *Expanded Hedge* [Takimoto & Warmuth, 2003].
    - *Follow-the-Perturbed-Leader* [Kalai & Vempala, 2005].
    - *Component Hedge* [Koolen et al., 2010].
  - Different bandit settings:
    - Algorithm of [Győrgy et al., 2007].
    - ComBand [Cesa-Bianchi and Lugosi, 2012].

What if the loss/gain is **not additive**?

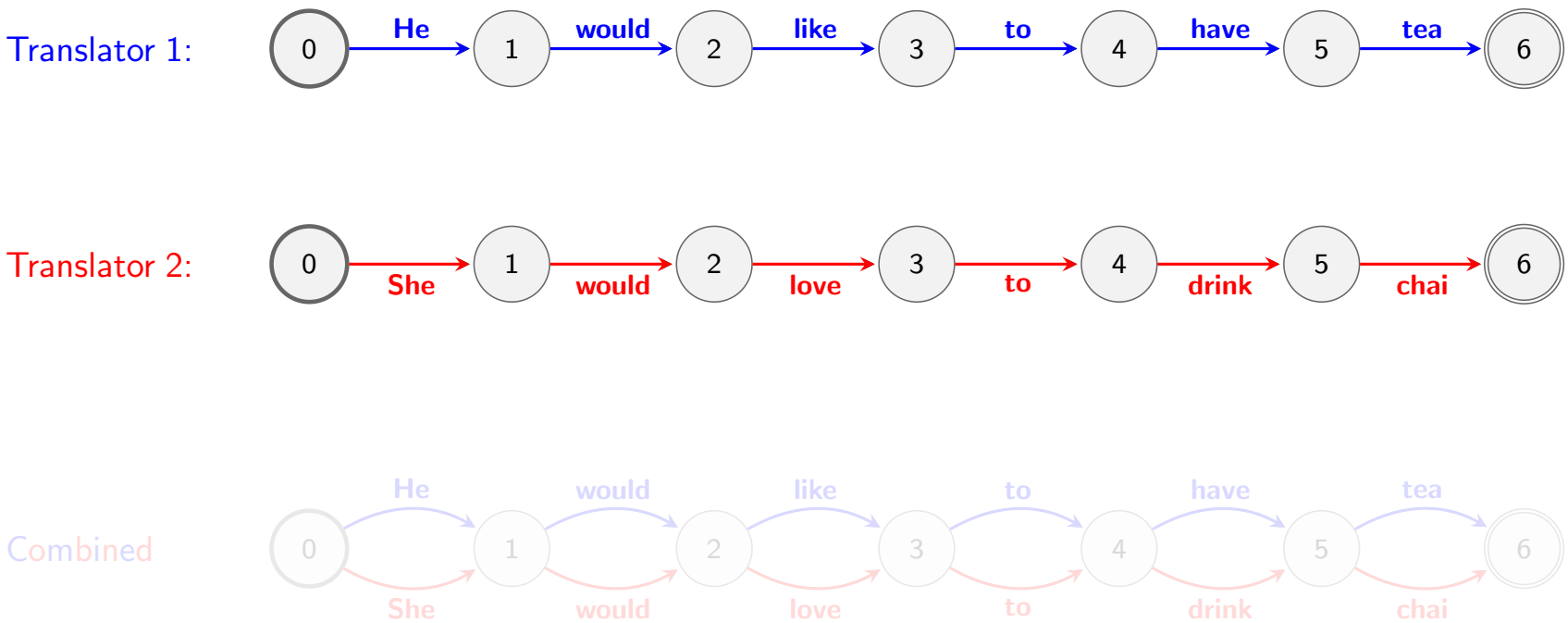
Motivation: *Ensemble Structured Prediction*

- Motivating example in the application of machine translation:



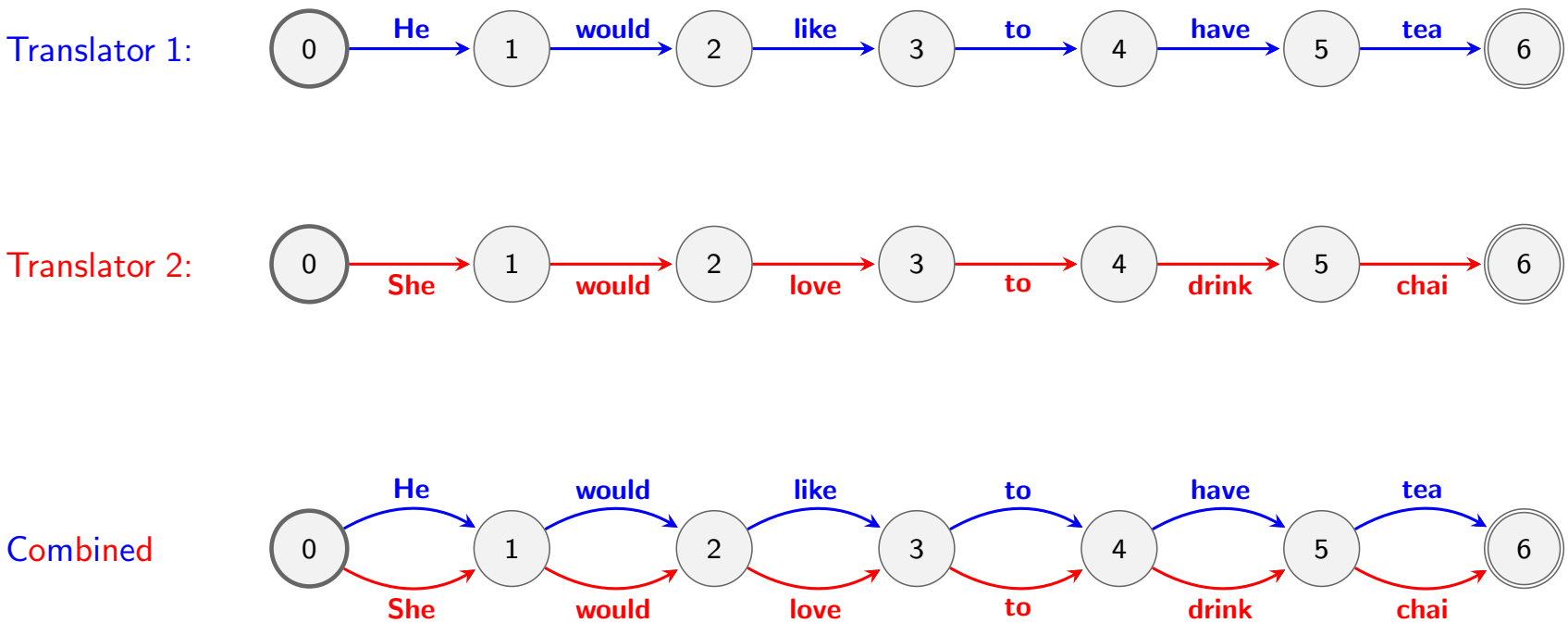
Motivation: *Ensemble Structured Prediction*

- Motivating example in the application of machine translation:



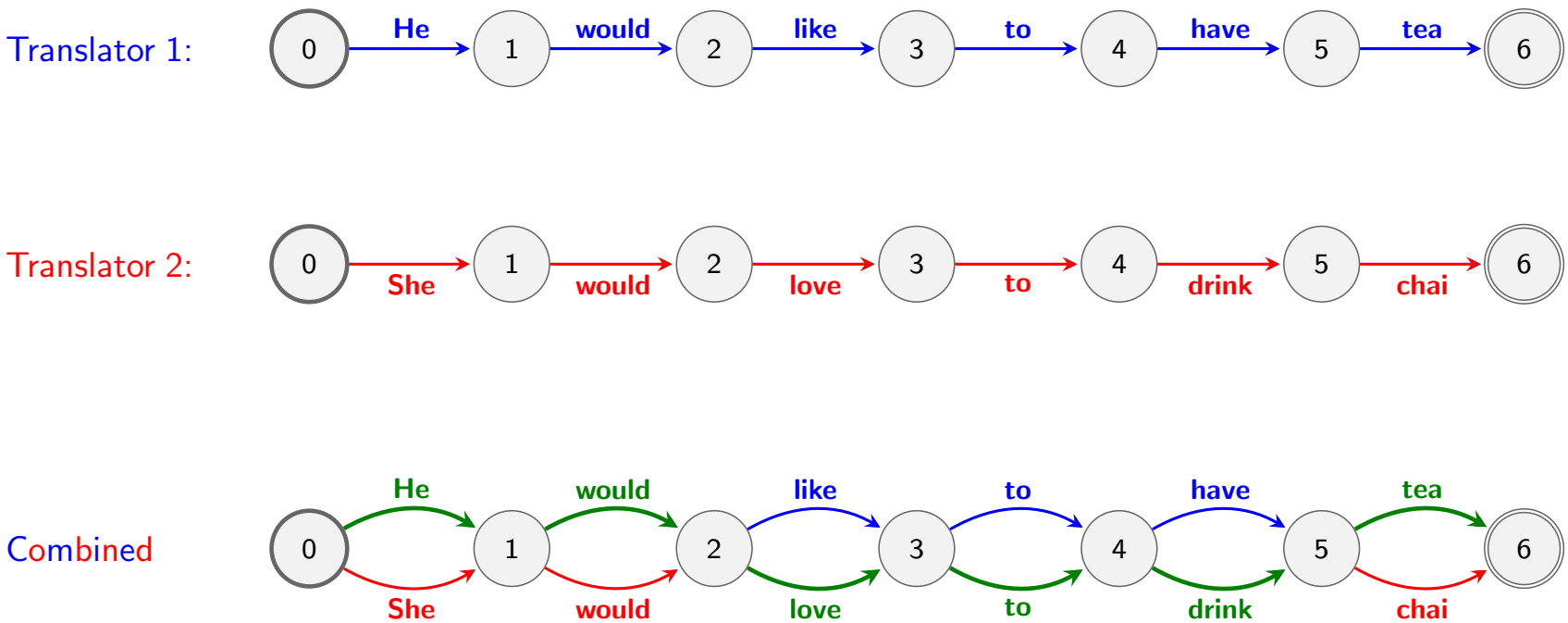
Motivation: *Ensemble Structured Prediction*

- Motivating example in the application of machine translation:



Motivation: *Ensemble Structured Prediction*

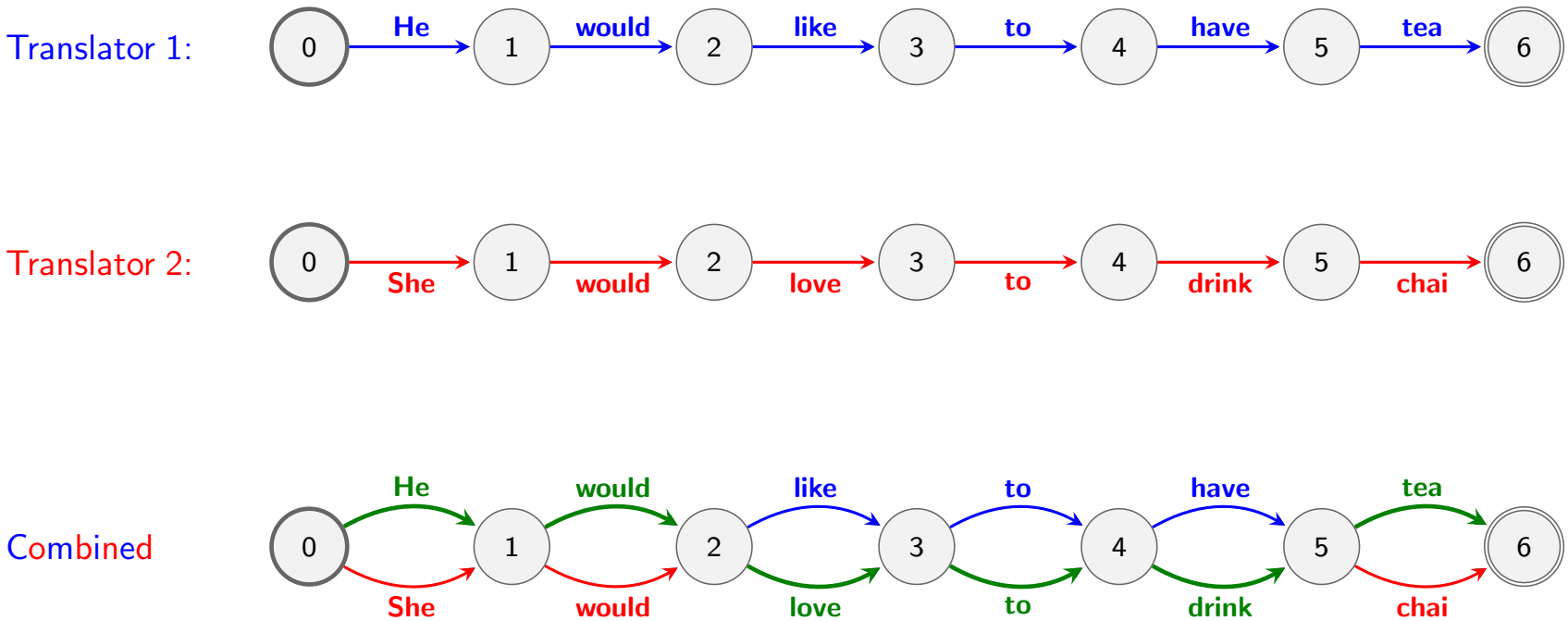
- Motivating example in the application of machine translation:



One particular translator may be better at predicting one specific word than the other translators.

Motivation: *Ensemble Structured Prediction*

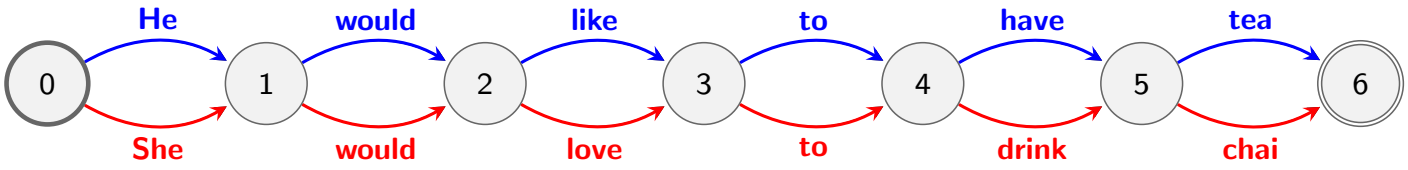
- Motivating example in the application of machine translation:



One particular translator may be better at predicting one specific word than the other translators.

Motivation: *Ensemble Structured Prediction*

Combined

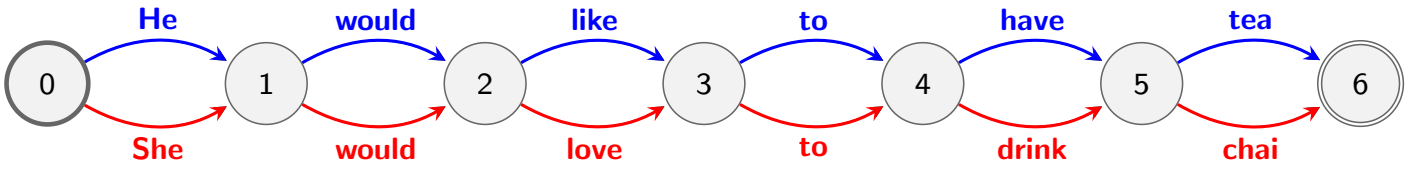


- In machine translation, the **BLEU score similarity** is used for evaluation.
- BLEU score $\approx$ the inner product of the count vectors of the n-gram occurrences in two sequences (typically $n = 4$)
  - e.g. a 4-gram is "like-to-drink-tea"
- BLEU score is not necessarily additive along the edges.
- We cannot directly apply the learning algorithms in the literature for additive gains.

Focus: Non-additive gain/loss based on counting patterns like n-grams.

Motivation: *Ensemble Structured Prediction*

Combined

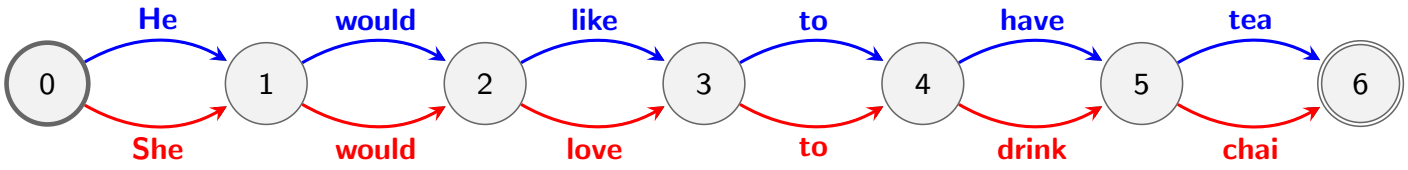


- In machine translation, the **BLEU score similarity** is used for evaluation.
- BLEU score $\approx$ the inner product of the count vectors of the n-gram occurrences in two sequences (typically $n = 4$)
  - e.g. a 4-gram is "like-to-drink-tea"
- BLEU score is not necessarily additive along the edges.
- We cannot directly apply the learning algorithms in the literature for additive gains.

Focus: Non-additive gain/loss based on counting patterns like n-grams.

Motivation: *Ensemble Structured Prediction*

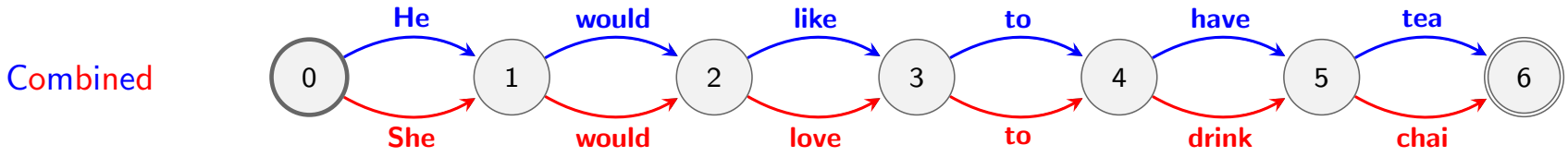
Combined



- In machine translation, the **BLEU score similarity** is used for evaluation.
- BLEU score $\approx$ the inner product of the count vectors of the n-gram occurrences in two sequences (typically $n = 4$)
  - e.g. a 4-gram is "like-to-drink-tea"
- BLEU score is not necessarily additive along the edges.
- We cannot directly apply the learning algorithms in the literature for additive gains.

Focus: Non-additive gain/loss based on counting patterns like n-grams.

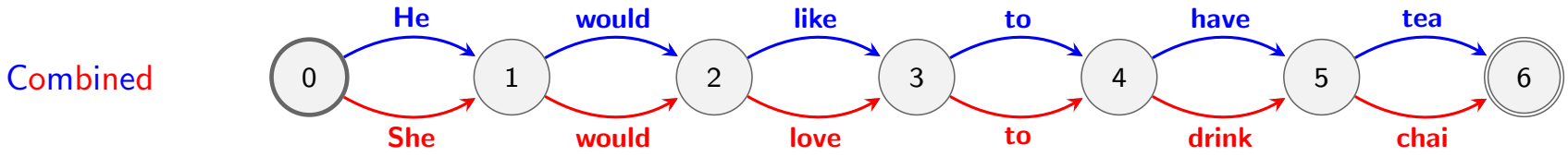
Motivation: *Ensemble Structured Prediction*



- In machine translation, the **BLEU score similarity** is used for evaluation.
- BLEU score $\approx$ the inner product of the count vectors of the n-gram occurrences in two sequences (typically $n = 4$)
  - e.g. a 4-gram is "like-to-drink-tea"
- BLEU score is not necessarily additive along the edges.
- We cannot directly apply the learning algorithms in the literature for additive gains.

Focus: Non-additive gain/loss based on counting patterns like n-grams.

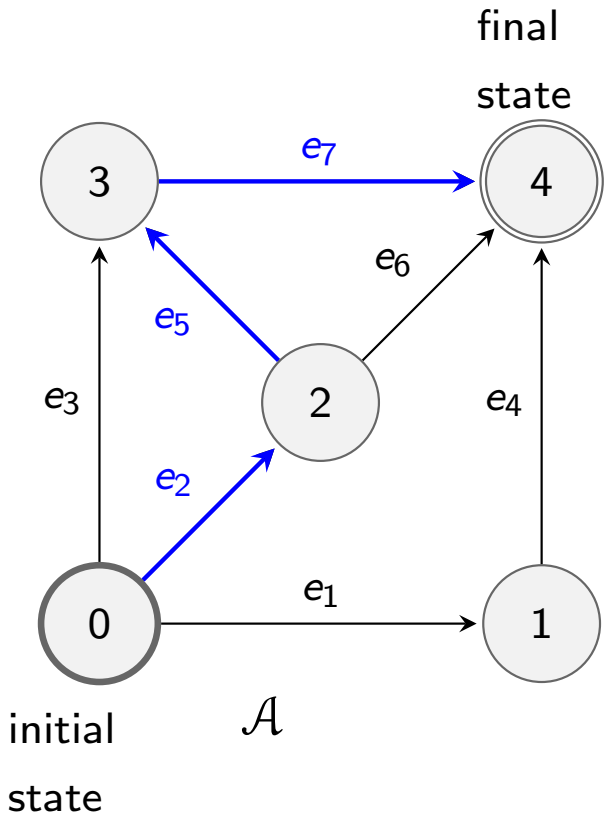
Motivation: *Ensemble Structured Prediction*



- In machine translation, the **BLEU score similarity** is used for evaluation.
- BLEU score $\approx$ the inner product of the count vectors of the n-gram occurrences in two sequences (typically $n = 4$)
  - e.g. a 4-gram is "like-to-drink-tea"
- BLEU score is not necessarily additive along the edges.
- We cannot directly apply the learning algorithms in the literature for additive gains.

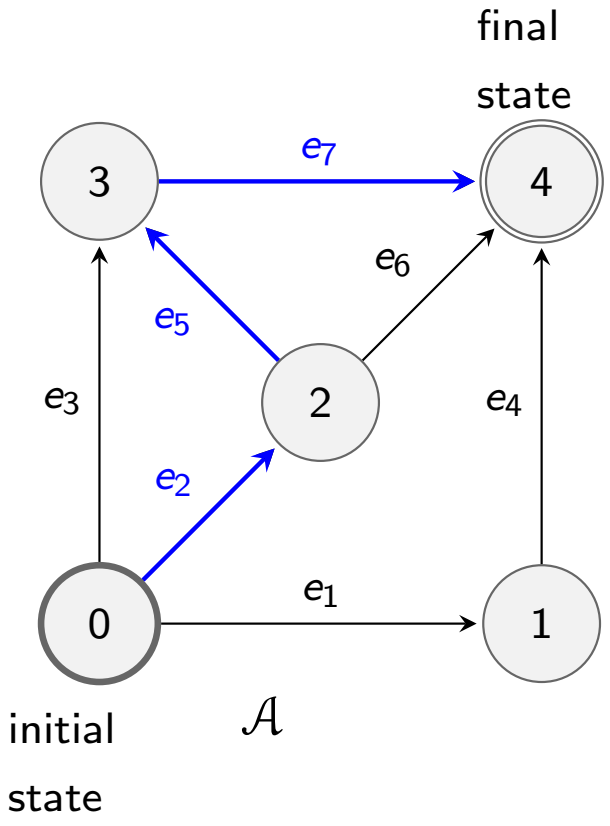
Focus: Non-additive gain/loss based on counting patterns like n-grams.

The Basic Setup – Expert Automaton



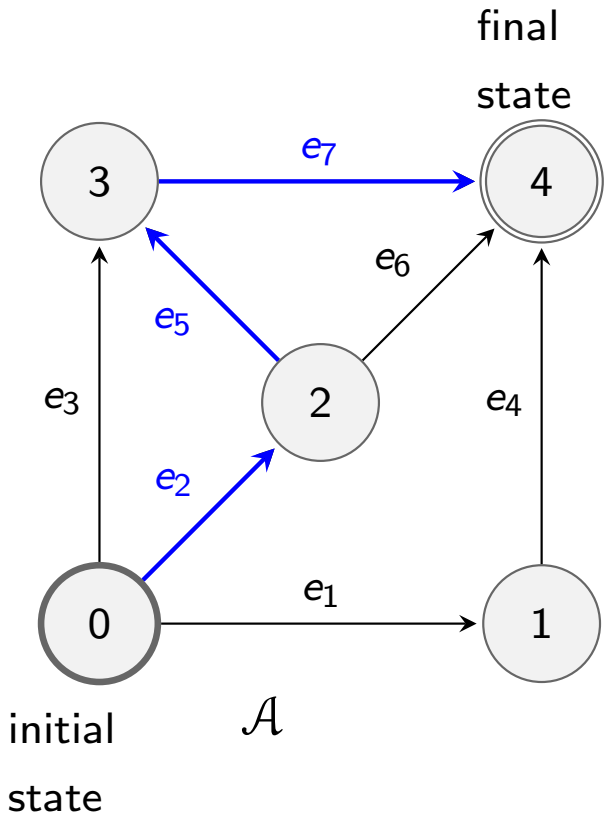
- An **acyclic automaton** $\mathcal{A}$ is given.
- Each **transition** is labeled with a unique **name**.
- $E :=$ the set of all transition names.
- Each **path expert** is a sequence of transitions from the initial state to a final state
  - e.g. $\pi = e_2\, e_5\, e_7$
- $\mathcal{A}$ can be viewed as an indicator function:
  - e.g. $\mathcal{A}(e_2\, e_5\, e_7) = 1, \quad \mathcal{A}(e_1\, e_2) = 0.$

The Basic Setup – Expert Automaton



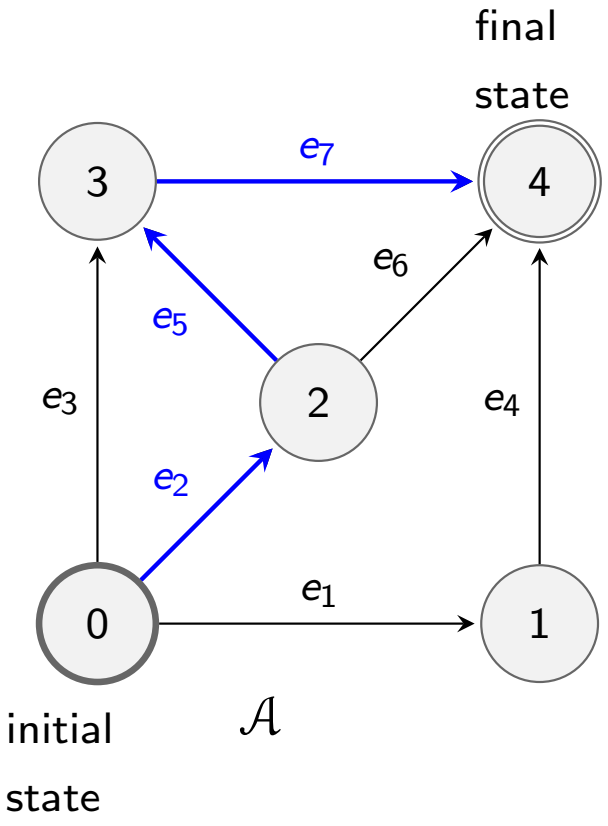
- An **acyclic automaton** $\mathcal{A}$ is given.
- Each **transition** is labeled with a unique **name**.
- $E :=$ the set of all transition names.
- Each **path expert** is a sequence of transitions from the initial state to a final state
  - e.g. $\pi = e_2\ e_5\ e_7$
- $\mathcal{A}$ can be viewed as an indicator function:
  - e.g. $\mathcal{A}(e_2\ e_5\ e_7) = 1, \quad \mathcal{A}(e_1\ e_2) = 0.$

The Basic Setup – Expert Automaton



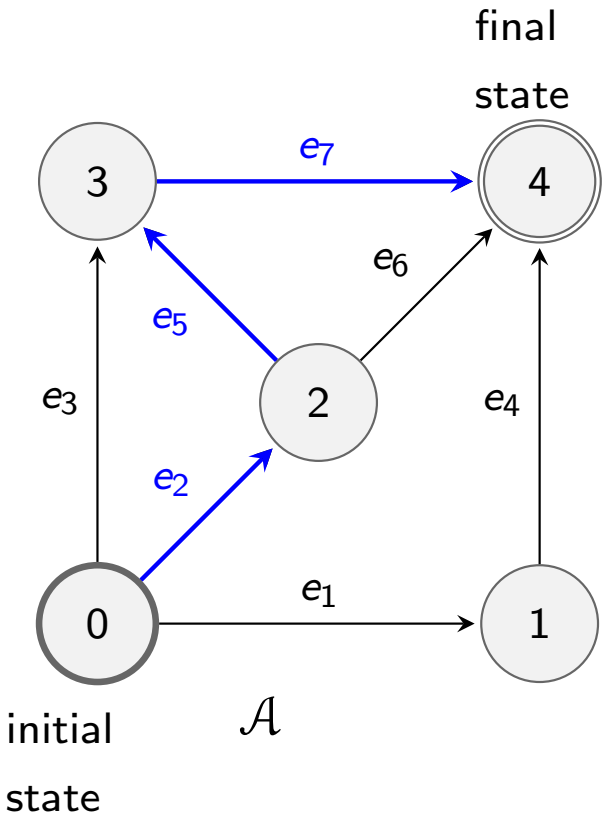
- An **acyclic automaton** $\mathcal{A}$ is given.
- Each **transition** is labeled with a unique **name**.
- $E :=$ the set of all transition names.
- Each **path expert** is a sequence of transitions from the initial state to a final state
  - e.g. $\pi = e_2\, e_5\, e_7$
- $\mathcal{A}$ can be viewed as an indicator function:
  - e.g. $\mathcal{A}(e_2\, e_5\, e_7) = 1, \quad \mathcal{A}(e_1\, e_2) = 0.$

The Basic Setup – Expert Automaton



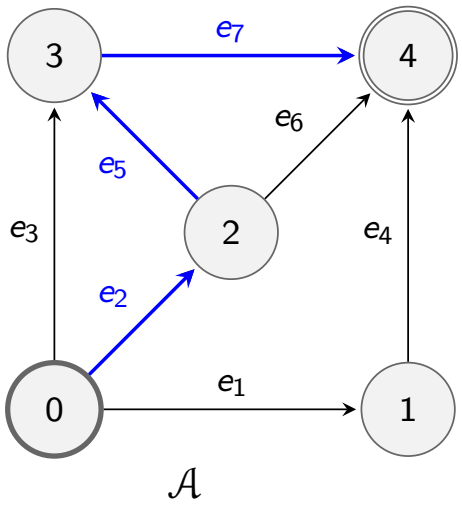
- An **acyclic automaton** $\mathcal{A}$ is given.
- Each **transition** is labeled with a unique **name**.
- $E :=$ the set of all transition names.
- Each **path expert** is a sequence of transitions from the initial state to a final state
  - e.g. $\pi = e_2\, e_5\, e_7$
- $\mathcal{A}$ can be viewed as an indicator function:
  - e.g. $\mathcal{A}(e_2\, e_5\, e_7) = 1, \quad \mathcal{A}(e_1\, e_2) = 0.$

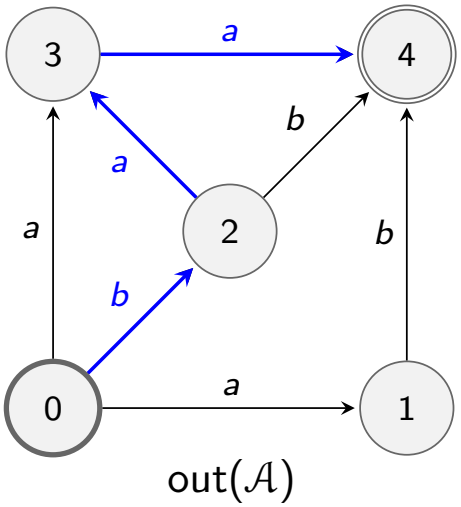
The Basic Setup – Expert Automaton



- An **acyclic automaton** $\mathcal{A}$ is given.
- Each **transition** is labeled with a unique **name**.
- $E :=$ the set of all transition names.
- Each **path expert** is a sequence of transitions from the initial state to a final state
  - e.g. $\pi = e_2\ e_5\ e_7$
- $\mathcal{A}$ can be viewed as an indicator function:
  - e.g. $\mathcal{A}(e_2\ e_5\ e_7) = 1, \quad \mathcal{A}(e_1\ e_2) = 0.$

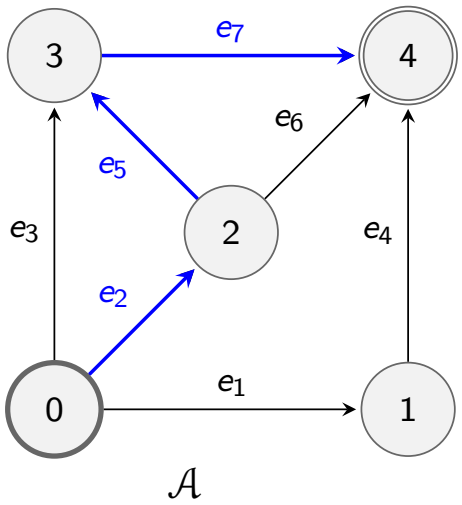
The Basic Setup – Expert Predictions



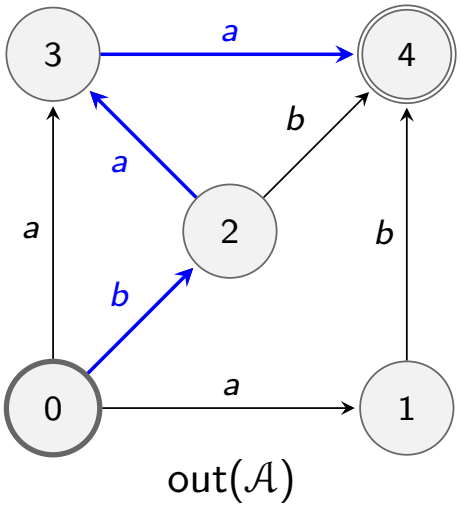


- In each round $t = 1, \ldots, T$, each transition $e \in E$ outputs a **symbol** $\text{out}_t(e)$ from a given alphabet Σ.
- The **prediction** of each path expert $\pi \in E^*$ is the sequence of symbols $\text{out}_t(\pi) \in \Sigma^*$ along its transitions.
  - e.g. $\pi = e_2\, e_5\, e_7 \longrightarrow \text{out}_t(\pi) = \text{out}_t(e_2)\,\text{out}_t(e_5)\,\text{out}_t(e_7) = baa$.
- $\text{out}_t(\mathcal{A}) :=$ the automaton with the same topology as $\mathcal{A}$ where each transition e is labeled with $\text{out}_t(e)$.

The Basic Setup – Expert Predictions



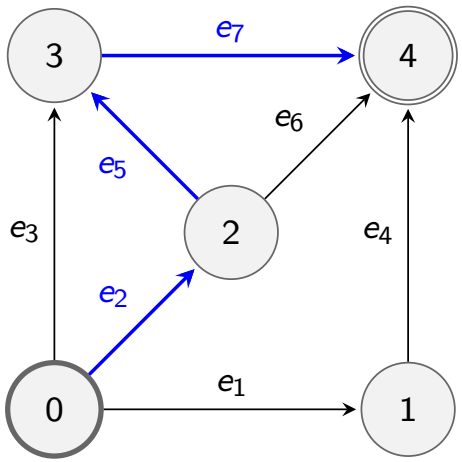
$\mathcal{A}$



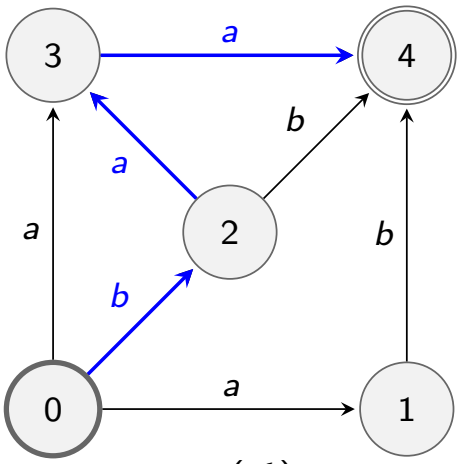
$\text{out}(\mathcal{A})$

- In each round $t = 1, \ldots, T$, each transition $e \in E$ outputs a **symbol** $\text{out}_t(e)$ from a given alphabet Σ.
- The **prediction** of each path expert $\pi \in E^*$ is the sequence of symbols $\text{out}_t(\pi) \in \Sigma^*$ along its transitions.
  - e.g. $\pi = e_2\, e_5\, e_7 \longrightarrow \text{out}_t(\pi) = \text{out}_t(e_2)\,\text{out}_t(e_5)\,\text{out}_t(e_7) = baa$.
- $\text{out}_t(\mathcal{A}) :=$ the automaton with the same topology as $\mathcal{A}$ where each transition e is labeled with $\text{out}_t(e)$.

The Basic Setup – Expert Predictions



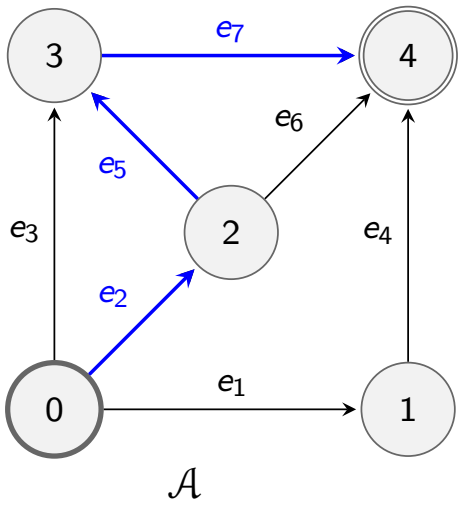
$\mathcal{A}$



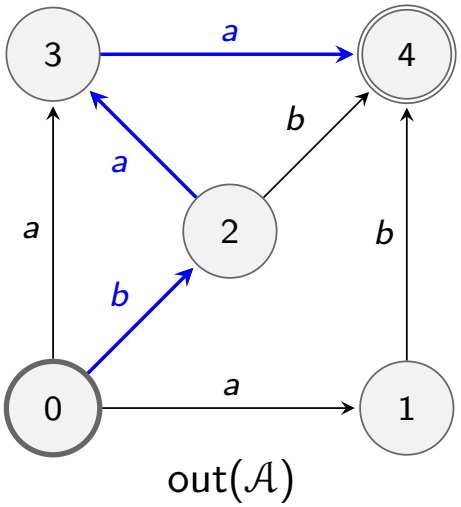
$\text{out}(\mathcal{A})$

- In each round $t = 1, \ldots, T$, each transition $e \in E$ outputs a **symbol** $\text{out}_t(e)$ from a given alphabet Σ.
- The **prediction** of each path expert $\pi \in E^*$ is the sequence of symbols $\text{out}_t(\pi) \in \Sigma^*$ along its transitions.
  - e.g. $\pi = e_2\, e_5\, e_7 \longrightarrow \text{out}_t(\pi) = \text{out}_t(e_2)\,\text{out}_t(e_5)\,\text{out}_t(e_7) = baa$.
- $\text{out}_t(\mathcal{A})$:= the automaton with the same topology as $\mathcal{A}$ where each transition e is labeled with $\text{out}_t(e)$.

The Basic Setup – Expert Predictions



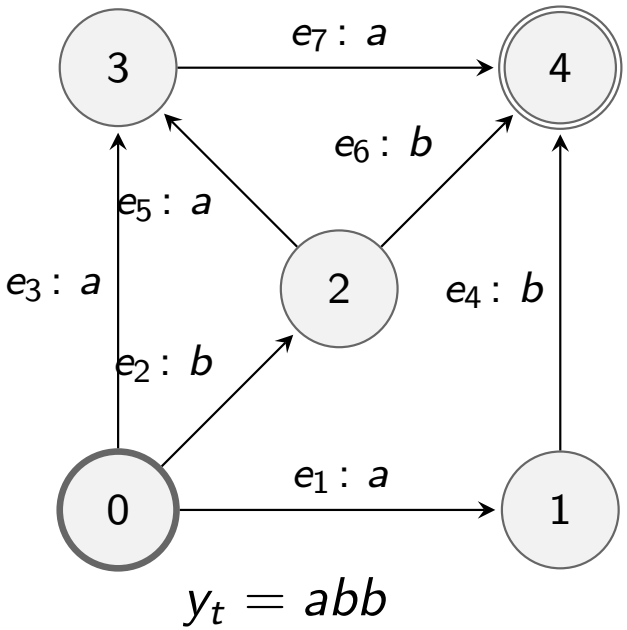
$\mathcal{A}$



$\text{out}(\mathcal{A})$

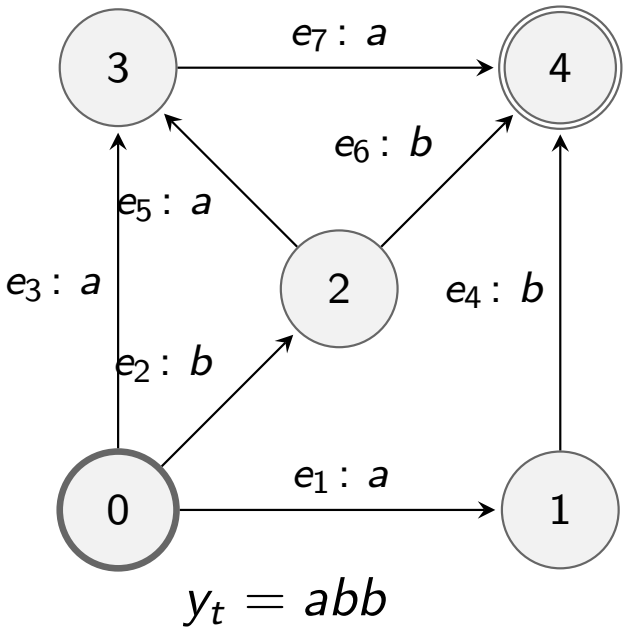
- In each round $t = 1, \ldots, T$, each transition $e \in E$ outputs a **symbol** $\text{out}_t(e)$ from a given alphabet Σ.
- The **prediction** of each path expert $\pi \in E^*$ is the sequence of symbols $\text{out}_t(\pi) \in \Sigma^*$ along its transitions.
  - e.g. $\pi = e_2\, e_5\, e_7 \longrightarrow \text{out}_t(\pi) = \text{out}_t(e_2)\,\text{out}_t(e_5)\,\text{out}_t(e_7) = baa$.
- $\text{out}_t(\mathcal{A}) :=$ the automaton with the same topology as $\mathcal{A}$ where each transition e is labeled with $\text{out}_t(e)$.

The Basic Setup – Experts Gains



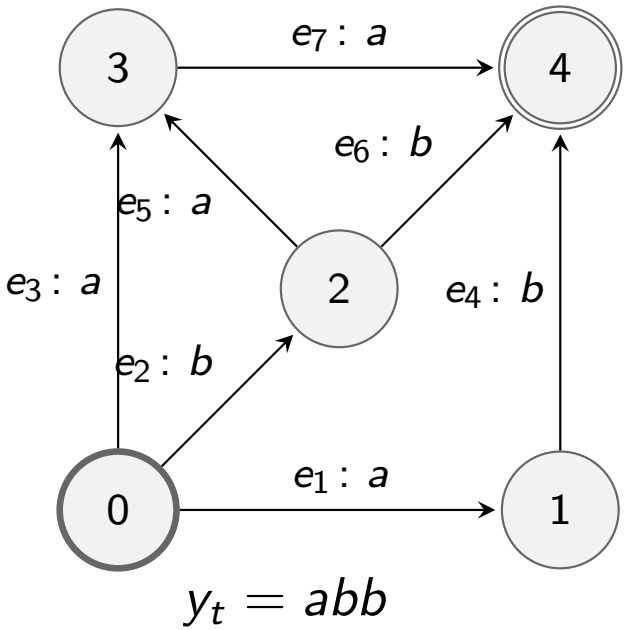
- At each round t, there is a **target sequence** y_t.
- In round t, the **gain/loss** of each path expert π is $\mathcal{U}(\text{out}_t(\pi), y_t)$ where $\mathcal{U} : \Sigma^* \times \Sigma^* \longrightarrow \mathbb{R}_{\geq 0}$.
  - May not be additive along the transitions in $\mathcal{A}$.
  - Examples of $\mathcal{U}$:
    1. **distance function** like edit distance.
    2. **similarity function** like n-gram gains ($n \geq 2$).

The Basic Setup – Experts Gains



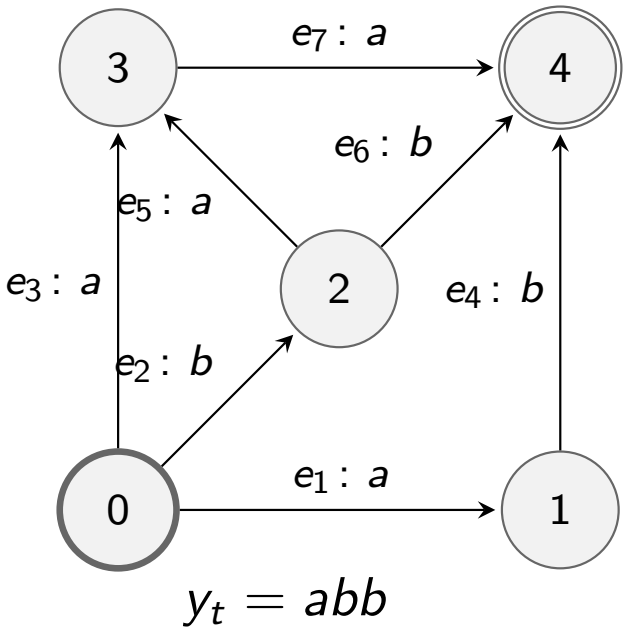
- At each round t, there is a **target sequence** y_t.
- In round t, the **gain/loss** of each path expert π is $\mathcal{U}(\text{out}_t(\pi), y_t)$ where $\mathcal{U} : \Sigma^* \times \Sigma^* \longrightarrow \mathbb{R}_{\geq 0}$.
  - May not be additive along the transitions in $\mathcal{A}$.
  - Examples of $\mathcal{U}$:
    1. **distance function** like edit distance.
    2. **similarity function** like n-gram gains ($n \geq 2$).

The Basic Setup – Experts Gains



- At each round t, there is a **target sequence** y_t.
- In round t, the **gain/loss** of each path expert π is $\mathcal{U}(\text{out}_t(\pi), y_t)$ where $\mathcal{U} : \Sigma^* \times \Sigma^* \longrightarrow \mathbb{R}_{\geq 0}$.
  - May not be additive along the transitions in $\mathcal{A}$.
  - Examples of $\mathcal{U}$:
    1. **distance function** like edit distance.
    2. **similarity function** like n-gram gains ($n \geq 2$).

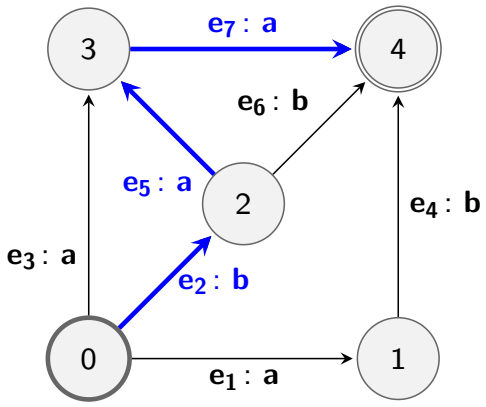
The Basic Setup – Experts Gains



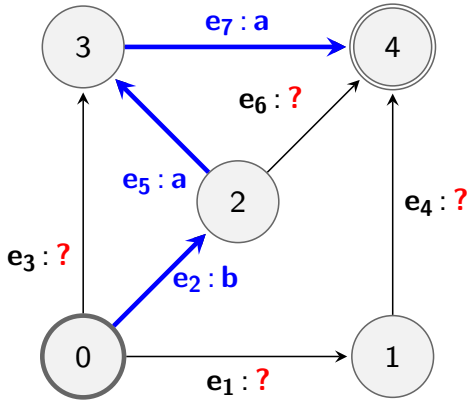
- At each round t, there is a **target sequence** y_t.
- In round t, the **gain/loss** of each path expert π is $\mathcal{U}(\text{out}_t(\pi), y_t)$ where $\mathcal{U} : \Sigma^* \times \Sigma^* \longrightarrow \mathbb{R}_{\geq 0}$.
  - May not be additive along the transitions in $\mathcal{A}$.
  - Examples of $\mathcal{U}$:
    1. **distance function** like edit distance.
    2. **similarity function** like n-gram gains ($n \geq 2$).

Online Learning Scenario

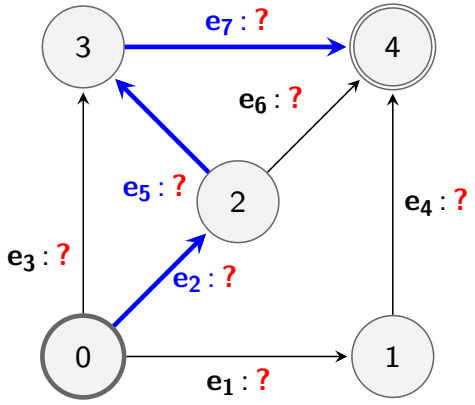
Full Information



Semi-bandit



Full Bandit

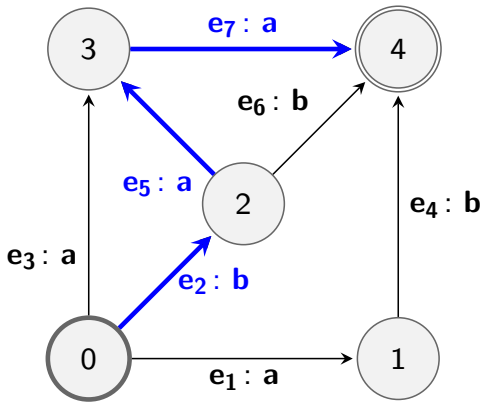


- At each round $t = 1 \dots T$
  1. The **learner** picks a path expert π_t and predicts with its prediction $\text{out}_t(\pi_t)$.
  2. The **adversary** reveals:
     - y_t and $\text{out}_t(e)$ for all $e \in E$ in **full information** setting.
     - y_t and $\text{out}_t(e)$ for all $e \in \pi_t$ in **semi-bandit** setting.
     - only the gain of $\mathcal{U}(\text{out}_t(\pi_t), y_t)$ in **full bandit** setting.
  3. The **learner** observes and receives the gain of $\mathcal{U}(\text{out}_t(\pi_t), y_t)$.
- The goal is to minimize the **regret**

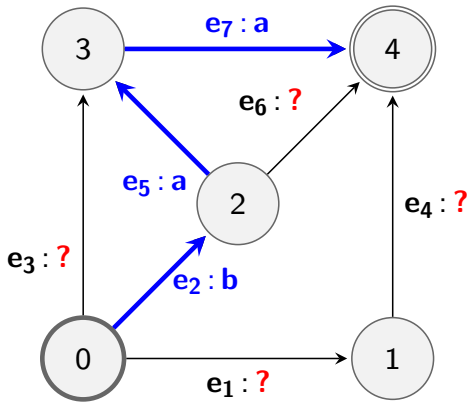
$$\underbrace{\min_{\pi^*} \left[\sum_{t=1}^{T} \mathcal{U}(\text{out}_t(\pi^*), y_t) \right]}_{\text{gain of the best path expert}} - \underbrace{\mathbb{E}\left[\sum_{t=1}^{T} \mathcal{U}(\text{out}_t(\pi_t), y_t) \right]}_{\text{expected gain of the learner}}$$

Online Learning Scenario

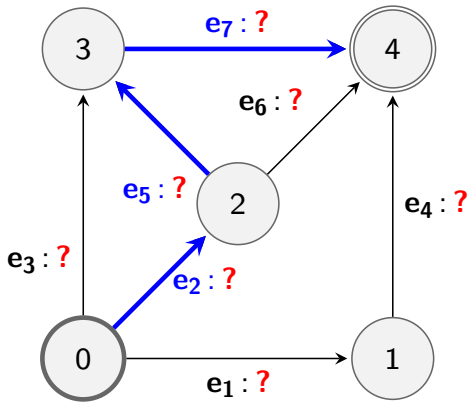
Full Information



Semi-bandit



Full Bandit

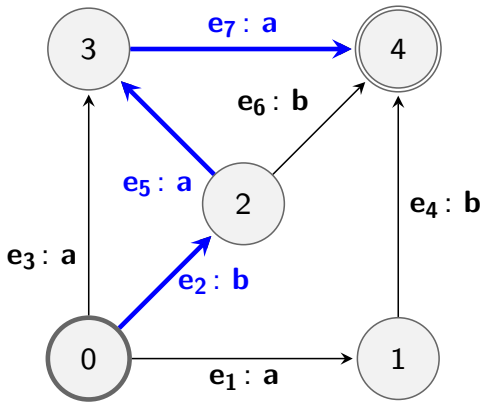


- At each round $t = 1 \dots T$
  1. The **learner** picks a path expert π_t and predicts with its prediction $\text{out}_t(\pi_t)$.
  2. The **adversary** reveals:
     - y_t and $\text{out}_t(e)$ for all $e \in E$ in **full information** setting.
     - y_t and $\text{out}_t(e)$ for all $e \in \pi_t$ in **semi-bandit** setting.
     - only the gain of $\mathcal{U}(\text{out}_t(\pi_t), y_t)$ in **full bandit** setting.
  3. The **learner** observes and receives the gain of $\mathcal{U}(\text{out}_t(\pi_t), y_t)$.
- The goal is to minimize the **regret**

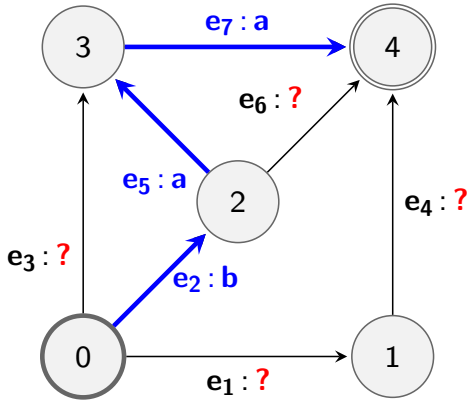
$$\underbrace{\min_{\pi^*} \left[\sum_{t=1}^{T} \mathcal{U}(\text{out}_t(\pi^*), y_t) \right]}_{\text{gain of the best path expert}} - \underbrace{\mathbb{E}\left[\sum_{t=1}^{T} \mathcal{U}(\text{out}_t(\pi_t), y_t) \right]}_{\text{expected gain of the learner}}$$

Online Learning Scenario

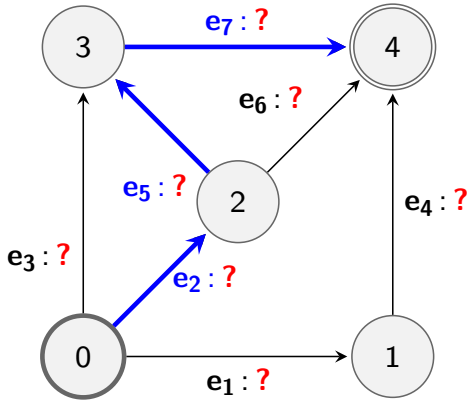
Full Information



Semi-bandit



Full Bandit

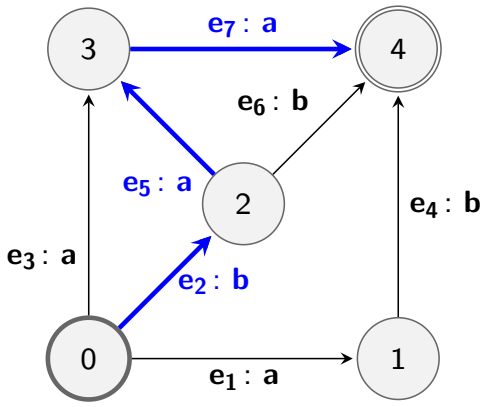


- At each round $t = 1 \dots T$
  1. The **learner** picks a path expert π_t and predicts with its prediction $\text{out}_t(\pi_t)$.
  2. The **adversary** reveals:
     - y_t and $\text{out}_t(e)$ for all $e \in E$ in **full information** setting.
     - y_t and $\text{out}_t(e)$ for all $e \in \pi_t$ in **semi-bandit** setting.
     - only the gain of $\mathcal{U}(\text{out}_t(\pi_t), y_t)$ in **full bandit** setting.
  3. The **learner** observes and receives the gain of $\mathcal{U}(\text{out}_t(\pi_t), y_t)$.
- The goal is to minimize the **regret**

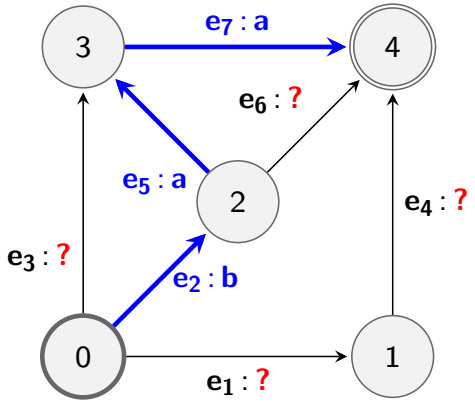
$$\underbrace{\min_{\pi^*} \left[\sum_{t=1}^{T} \mathcal{U}(\text{out}_t(\pi^*), y_t) \right]}_{\text{gain of the best path expert}} - \underbrace{\mathbb{E}\left[\sum_{t=1}^{T} \mathcal{U}(\text{out}_t(\pi_t), y_t) \right]}_{\text{expected gain of the learner}}$$

Online Learning Scenario

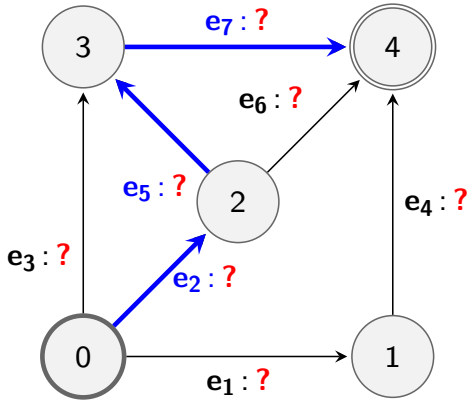
Full Information



Semi-bandit



Full Bandit

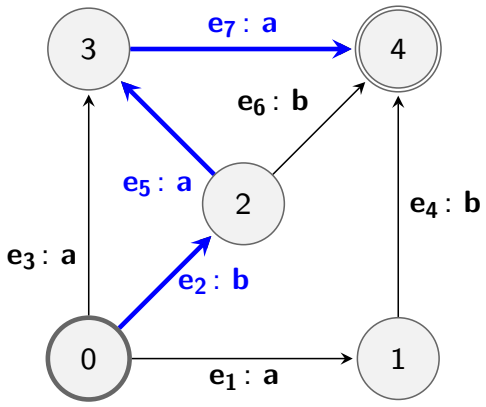


- At each round $t = 1 \dots T$
  1. The **learner** picks a path expert π_t and predicts with its prediction $\text{out}_t(\pi_t)$.
  2. The **adversary** reveals:
     - y_t and $\text{out}_t(e)$ for all $e \in E$ in **full information** setting.
     - y_t and $\text{out}_t(e)$ for all $e \in \pi_t$ in **semi-bandit** setting.
     - only the gain of $\mathcal{U}(\text{out}_t(\pi_t), y_t)$ in **full bandit** setting.
  3. The **learner** observes and receives the gain of $\mathcal{U}(\text{out}_t(\pi_t), y_t)$.
- The goal is to minimize the **regret**

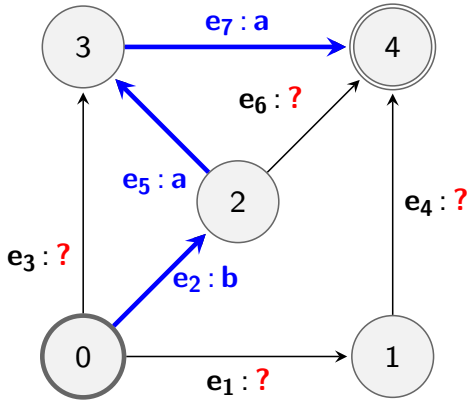
$$\underbrace{\min_{\pi^*} \left[\sum_{t=1}^{T} \mathcal{U}(\text{out}_t(\pi^*), y_t) \right]}_{\text{gain of the best path expert}} - \underbrace{\mathbb{E}\left[\sum_{t=1}^{T} \mathcal{U}(\text{out}_t(\pi_t), y_t) \right]}_{\text{expected gain of the learner}}$$

Online Learning Scenario

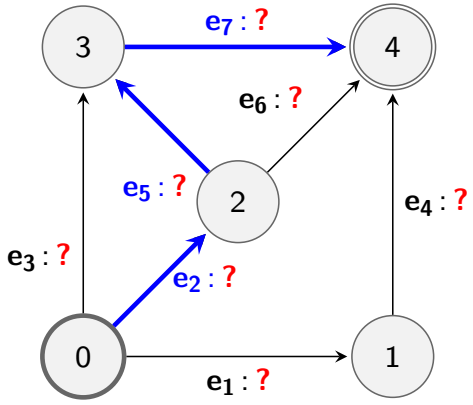
Full Information



Semi-bandit



Full Bandit

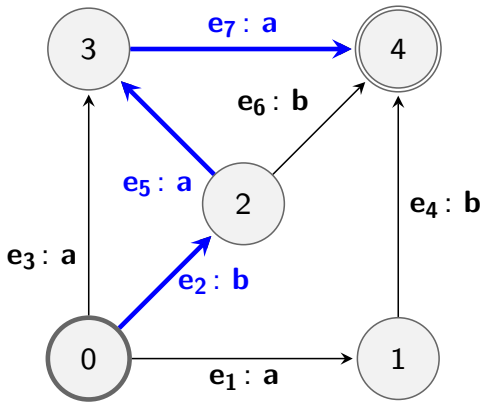


- At each round $t = 1 \dots T$
  1. The **learner** picks a path expert π_t and predicts with its prediction $\text{out}_t(\pi_t)$.
  2. The **adversary** reveals:
     - y_t and $\text{out}_t(e)$ for all $e \in E$ in **full information** setting.
     - y_t and $\text{out}_t(e)$ for all $e \in \pi_t$ in **semi-bandit** setting.
     - only the gain of $\mathcal{U}(\text{out}_t(\pi_t), y_t)$ in **full bandit** setting.
  3. The **learner** observes and receives the gain of $\mathcal{U}(\text{out}_t(\pi_t), y_t)$.
- The goal is to minimize the **regret**

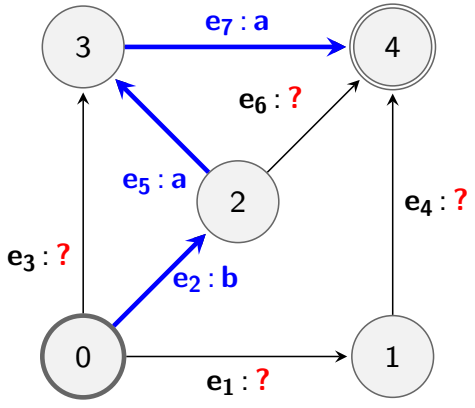
$$\underbrace{\min_{\pi^*} \left[\sum_{t=1}^{T} \mathcal{U}(\text{out}_t(\pi^*), y_t) \right]}_{\text{gain of the best path expert}} - \underbrace{\mathbb{E}\left[\sum_{t=1}^{T} \mathcal{U}(\text{out}_t(\pi_t), y_t) \right]}_{\text{expected gain of the learner}}$$

Online Learning Scenario

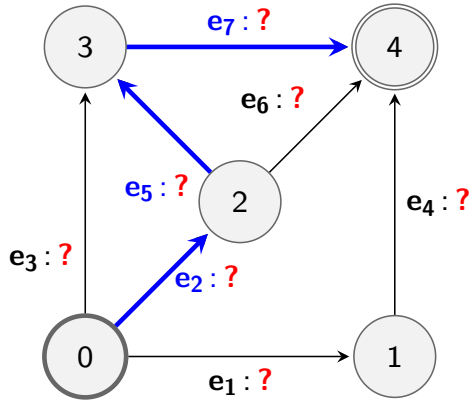
Full Information



Semi-bandit



Full Bandit

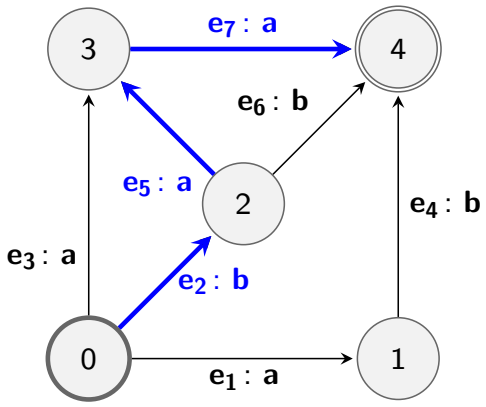


- At each round $t = 1 \dots T$
  1. The **learner** picks a path expert π_t and predicts with its prediction $\text{out}_t(\pi_t)$.
  2. The **adversary** reveals:
     - y_t and $\text{out}_t(e)$ for all $e \in E$ in **full information** setting.
     - y_t and $\text{out}_t(e)$ for all $e \in \pi_t$ in **semi-bandit** setting.
     - only the gain of $\mathcal{U}(\text{out}_t(\pi_t), y_t)$ in **full bandit** setting.
  3. The **learner** observes and receives the gain of $\mathcal{U}(\text{out}_t(\pi_t), y_t)$.
- The goal is to minimize the **regret**

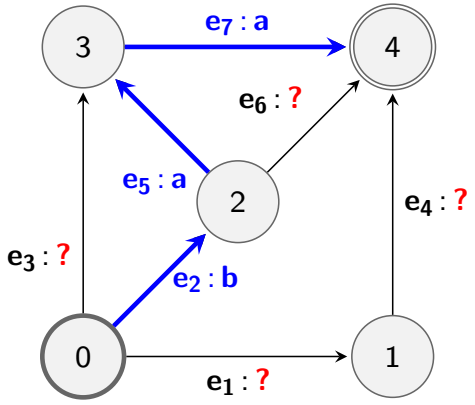
$$\underbrace{\min_{\pi^*} \left[\sum_{t=1}^{T} \mathcal{U}(\text{out}_t(\pi^*), y_t) \right]}_{\text{gain of the best path expert}} - \underbrace{\mathbb{E}\left[\sum_{t=1}^{T} \mathcal{U}(\text{out}_t(\pi_t), y_t) \right]}_{\text{expected gain of the learner}}$$

Online Learning Scenario

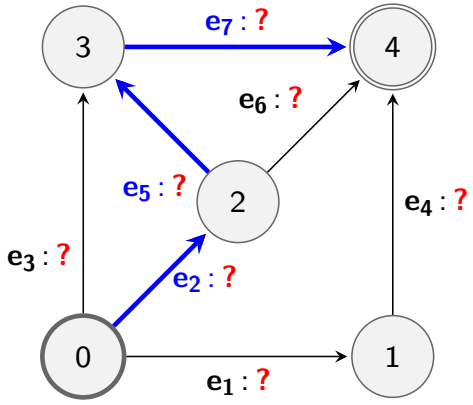
Full Information



Semi-bandit



Full Bandit

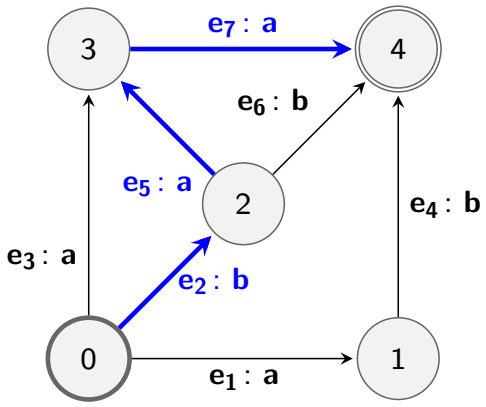


- At each round $t = 1 \dots T$
  1. The **learner** picks a path expert π_t and predicts with its prediction $\text{out}_t(\pi_t)$.
  2. The **adversary** reveals:
     - y_t and $\text{out}_t(e)$ for all $e \in E$ in **full information** setting.
     - y_t and $\text{out}_t(e)$ for all $e \in \pi_t$ in **semi-bandit** setting.
     - only the gain of $\mathcal{U}(\text{out}_t(\pi_t), y_t)$ in **full bandit** setting.
  3. The **learner** observes and receives the gain of $\mathcal{U}(\text{out}_t(\pi_t), y_t)$.
- The goal is to minimize the **regret**

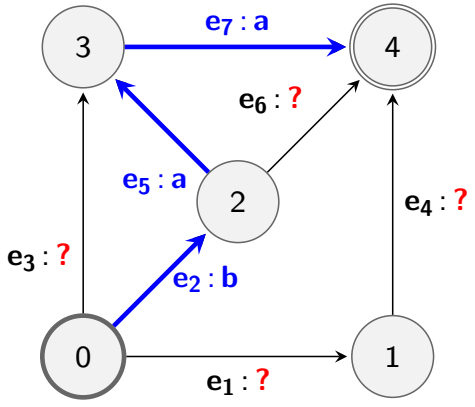
$$\underbrace{\min_{\pi^*} \left[\sum_{t=1}^{T} \mathcal{U}(\text{out}_t(\pi^*), y_t) \right]}_{\text{gain of the best path expert}} - \underbrace{\mathbb{E}\left[\sum_{t=1}^{T} \mathcal{U}(\text{out}_t(\pi_t), y_t) \right]}_{\text{expected gain of the learner}}$$

Online Learning Scenario

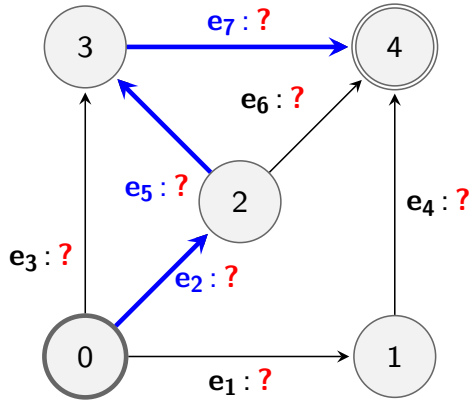
Full Information



Semi-bandit



Full Bandit



- At each round $t = 1 \dots T$
  1. The **learner** picks a path expert π_t and predicts with its prediction $\text{out}_t(\pi_t)$.
  2. The **adversary** reveals:
     - y_t and $\text{out}_t(e)$ for all $e \in E$ in **full information** setting.
     - y_t and $\text{out}_t(e)$ for all $e \in \pi_t$ in **semi-bandit** setting.
     - only the gain of $\mathcal{U}(\text{out}_t(\pi_t), y_t)$ in **full bandit** setting.
  3. The **learner** observes and receives the gain of $\mathcal{U}(\text{out}_t(\pi_t), y_t)$.
- The goal is to minimize the **regret**

$$\underbrace{\min_{\pi^*} \left[\sum_{t=1}^{T} \mathcal{U}(\text{out}_t(\pi^*), y_t) \right]}_{\text{gain of the best path expert}} - \underbrace{\mathbb{E}\left[\sum_{t=1}^{T} \mathcal{U}(\text{out}_t(\pi_t), y_t) \right]}_{\text{expected gain of the learner}}$$

Count-Based Gains

- Many of the most commonly used non-additive gains in applications belong to the broad family of **count-based gains**.

> The gain of each path π depends on the counts of occurrences[1] of the members of a finite language of output symbols $\mathcal{L} = \{\theta_1, \ldots, \theta_p\} \subset \Sigma^*$ in the output $\text{out}(\pi)$ of that path.

- $\theta_1, \theta_2, \ldots, \theta_p$ are called **patterns**.
- Example: ***n*-grams gains.**
  - $p = |\Sigma|^n$.
  - Given $\Sigma = \{a, b\}$, for $n = 2$, we count $\theta_1 = aa, \theta_2 = ab, \theta_3 = ba, \theta_4 = bb$.

[1] This type of gain will be generalized into **discounted** counts of **gappy** occurrences of θ_is.

Count-Based Gains

- Many of the most commonly used non-additive gains in applications belong to the broad family of **count-based gains**.

> The gain of each path π depends on the counts of occurrences[1] of the members of a finite language of output symbols $\mathcal{L} = \{\theta_1, \ldots, \theta_p\} \subset \Sigma^*$ in the output $\text{out}(\pi)$ of that path.

- $\theta_1, \theta_2, \ldots, \theta_p$ are called **patterns**.
- Example: ***n*-grams gains.**
  - $p = |\Sigma|^n$.
  - Given $\Sigma = \{a, b\}$, for $n = 2$, we count $\theta_1 = aa, \theta_2 = ab, \theta_3 = ba, \theta_4 = bb$.

[1] This type of gain will be generalized into **discounted** counts of **gappy** occurrences of θ_is.

Count-Based Gains

- Many of the most commonly used non-additive gains in applications belong to the broad family of **count-based gains**.

> The gain of each path π depends on the counts of occurrences[1] of the members of a finite language of output symbols $\mathcal{L} = \{\theta_1, \ldots, \theta_p\} \subset \Sigma^*$ in the output $\text{out}(\pi)$ of that path.

- $\theta_1, \theta_2, \ldots, \theta_p$ are called **patterns**.
- Example: ***n*-grams gains.**
  - $p = |\Sigma|^n$.
  - Given $\Sigma = \{a, b\}$, for $n = 2$, we count $\theta_1 = aa, \theta_2 = ab, \theta_3 = ba, \theta_4 = bb$.

[1] This type of gain will be generalized into **discounted** counts of **gappy** occurrences of θ_is.

Count-Based Gains

- Each sequence of symbols $y \in \Sigma^*$ can represented[2] as a count vector $\Theta(y) \in \mathbb{R}^p$ of the patterns $\theta_1, \ldots, \theta_p \in \Sigma^*$:

$$\Theta(y) := \begin{bmatrix} \text{number of occurrences of } \theta_1 \text{ in } y \\ \vdots \\ \text{number of occurrences of } \theta_p \text{ in } y \end{bmatrix}.$$

- The count-based gain function $\mathcal{U}$ at round t for a path π and target sequence y_t is defined as below:

$$\mathcal{U}(y_t, \text{out}_t(\pi)) := \Theta(y_t) \cdot \Theta(\text{out}_t(\pi)) \geq 0.$$

- These gains are **not additive** along the path π in $\mathcal{A}$.

How can we design algorithms for the online path learning problem with such non-additive gains?

[2] This representation can also be extended to **weighted** counts.

Count-Based Gains

- Each sequence of symbols $y \in \Sigma^*$ can represented[2] as a count vector $\Theta(y) \in \mathbb{R}^p$ of the patterns $\theta_1, \ldots, \theta_p \in \Sigma^*$:

$$\Theta(y) := \begin{bmatrix} \text{number of occurrences of } \theta_1 \text{ in } y \\ \vdots \\ \text{number of occurrences of } \theta_p \text{ in } y \end{bmatrix}.$$

- The count-based gain function $\mathcal{U}$ at round t for a path π and target sequence y_t is defined as below:

$$\mathcal{U}(y_t, \text{out}_t(\pi)) := \Theta(y_t) \cdot \Theta(\text{out}_t(\pi)) \geq 0.$$

- These gains are **not additive** along the path π in $\mathcal{A}$.

How can we design algorithms for the online path learning problem with such non-additive gains?

[2] This representation can also be extended to **weighted** counts.

Count-Based Gains

- Each sequence of symbols $y \in \Sigma^*$ can represented[2] as a count vector $\Theta(y) \in \mathbb{R}^p$ of the patterns $\theta_1, \ldots, \theta_p \in \Sigma^*$:

$$\Theta(y) := \begin{bmatrix} \text{number of occurrences of } \theta_1 \text{ in } y \\ \vdots \\ \text{number of occurrences of } \theta_p \text{ in } y \end{bmatrix}.$$

- The count-based gain function $\mathcal{U}$ at round t for a path π and target sequence y_t is defined as below:

$$\mathcal{U}(y_t, \text{out}_t(\pi)) := \Theta(y_t) \cdot \Theta(\text{out}_t(\pi)) \geq 0.$$

- These gains are **not additive** along the path π in $\mathcal{A}$.

How can we design algorithms for the online path learning problem with such non-additive gains?

[2] This representation can also be extended to **weighted** counts.

Count-Based Gains

- Each sequence of symbols $y \in \Sigma^*$ can represented[2] as a count vector $\Theta(y) \in \mathbb{R}^p$ of the patterns $\theta_1, \ldots, \theta_p \in \Sigma^*$:

$$\Theta(y) := \begin{bmatrix} \text{number of occurrences of } \theta_1 \text{ in } y \\ \vdots \\ \text{number of occurrences of } \theta_p \text{ in } y \end{bmatrix}.$$

- The count-based gain function $\mathcal{U}$ at round t for a path π and target sequence y_t is defined as below:

$$\mathcal{U}(y_t, \text{out}_t(\pi)) := \Theta(y_t) \cdot \Theta(\text{out}_t(\pi)) \geq 0.$$

- These gains are **not additive** along the path π in $\mathcal{A}$.

How can we design algorithms for the online path learning problem with such non-additive gains?

[2] This representation can also be extended to **weighted** counts.

Idea: The Alternative Additive Automaton $\mathcal{A}'$

- We will define an automaton $\mathcal{A}'$ where the gains will be **additive** along the paths.
  - $\mathcal{A}'$ has a **fixed topology** over rounds.
  - In each round, the patterns θ_is appear as the labels of the transitions.
  - For each path π in $\mathcal{A}$, there is a corresponding path in π' in $\mathcal{A}'$ which outputs all patterns occurring on the original path π.

Question: How can we create $\mathcal{A}'$ and map $\mathcal{A}$ to it?

- **Answer:** via **context-dependent rewrite rules**.
- We call $\mathcal{A}'$ the **context-dependent automaton**.

Idea: The Alternative Additive Automaton $\mathcal{A}'$

- We will define an automaton $\mathcal{A}'$ where the gains will be **additive** along the paths.
  - $\mathcal{A}'$ has a **fixed topology** over rounds.
  - In each round, the patterns θ_is appear as the labels of the transitions.
  - For each path π in $\mathcal{A}$, there is a corresponding path in π' in $\mathcal{A}'$ which outputs all patterns occurring on the original path π.

Question: How can we create $\mathcal{A}'$ and map $\mathcal{A}$ to it?

- **Answer:** via **context-dependent rewrite rules**.
- We call $\mathcal{A}'$ the **context-dependent automaton**.

Idea: The Alternative Additive Automaton $\mathcal{A}'$

- We will define an automaton $\mathcal{A}'$ where the gains will be **additive** along the paths.
  - $\mathcal{A}'$ has a **fixed topology** over rounds.
  - In each round, the patterns θ_is appear as the labels of the transitions.
  - For each path π in $\mathcal{A}$, there is a corresponding path in π' in $\mathcal{A}'$ which outputs all patterns occurring on the original path π.

Question: How can we create $\mathcal{A}'$ and map $\mathcal{A}$ to it?

- **Answer:** via **context-dependent rewrite rules**.
- We call $\mathcal{A}'$ the **context-dependent automaton**.

Idea: The Alternative Additive Automaton $\mathcal{A}'$

- We will define an automaton $\mathcal{A}'$ where the gains will be **additive** along the paths.
  - $\mathcal{A}'$ has a **fixed topology** over rounds.
  - In each round, the patterns θ_is appear as the labels of the transitions.
  - For each path π in $\mathcal{A}$, there is a corresponding path in π' in $\mathcal{A}'$ which outputs all patterns occurring on the original path π.

Question: How can we create $\mathcal{A}'$ and map $\mathcal{A}$ to it?

- **Answer:** via **context-dependent rewrite rules**.
- We call $\mathcal{A}'$ the **context-dependent automaton**.

Idea: The Alternative Additive Automaton $\mathcal{A}'$

- We will define an automaton $\mathcal{A}'$ where the gains will be **additive** along the paths.
  - $\mathcal{A}'$ has a **fixed topology** over rounds.
  - In each round, the patterns θ_is appear as the labels of the transitions.
  - For each path π in $\mathcal{A}$, there is a corresponding path in π' in $\mathcal{A}'$ which outputs all patterns occurring on the original path π.

Question: How can we create $\mathcal{A}'$ and map $\mathcal{A}$ to it?

- **Answer:** via **context-dependent rewrite rules**.
- We call $\mathcal{A}'$ the **context-dependent automaton**.

Idea: The Alternative Additive Automaton $\mathcal{A}'$

- We will define an automaton $\mathcal{A}'$ where the gains will be **additive** along the paths.
  - $\mathcal{A}'$ has a **fixed topology** over rounds.
  - In each round, the patterns θ_is appear as the labels of the transitions.
  - For each path π in $\mathcal{A}$, there is a corresponding path in π' in $\mathcal{A}'$ which outputs all patterns occurring on the original path π.

Question: How can we create $\mathcal{A}'$ and map $\mathcal{A}$ to it?

- **Answer:** via **context-dependent rewrite rules**.
- We call $\mathcal{A}'$ the **context-dependent automaton**.

Idea: The Alternative Additive Automaton $\mathcal{A}'$

- We will define an automaton $\mathcal{A}'$ where the gains will be **additive** along the paths.
  - $\mathcal{A}'$ has a **fixed topology** over rounds.
  - In each round, the patterns θ_is appear as the labels of the transitions.
  - For each path π in $\mathcal{A}$, there is a corresponding path in π' in $\mathcal{A}'$ which outputs all patterns occurring on the original path π.

Question: How can we create $\mathcal{A}'$ and map $\mathcal{A}$ to it?

- **Answer:** via **context-dependent rewrite rules**.
- We call $\mathcal{A}'$ the **context-dependent automaton**.

Context-dependent Rewrite Rules

- We will use **context-dependent rewrite rules** to map $\mathcal{A}$ to $\mathcal{A}'$:

$$\phi \rightarrow \psi / \lambda \_\_\_ \rho,$$

  where ϕ, ψ, λ, and ρ are regular expressions over the alphabet of the rules.

- Interpretation:

  ϕ is to be replaced by ψ whenever it is preceded by λ and followed by ρ.

We want to "rewrite" the **symbols** as **patterns**.

[Kaplan & Kay, 1994]

Context-dependent Rewrite Rules

- We will use **context-dependent rewrite rules** to map $\mathcal{A}$ to $\mathcal{A}'$:

$$\phi \rightarrow \psi / \lambda \_\_\_ \rho,$$

  where ϕ, ψ, λ, and ρ are regular expressions over the alphabet of the rules.

- Interpretation:

  ϕ is to be replaced by ψ whenever it is preceded by λ and followed by ρ.

We want to "rewrite" the **symbols** as **patterns**.

[Kaplan & Kay, 1994]

Context-dependent Rewrite Rules

- We will use **context-dependent rewrite rules** to map $\mathcal{A}$ to $\mathcal{A}'$:

$$\phi \rightarrow \psi / \lambda \_\_\_ \rho,$$

  where ϕ, ψ, λ, and ρ are regular expressions over the alphabet of the rules.

- Interpretation:

  ϕ is to be replaced by ψ whenever it is preceded by λ and followed by ρ.

We want to "rewrite" the **symbols** as **patterns**.

[Kaplan & Kay, 1994]

Our Rewrite Rules

- We introduce the alphabet E' as the set of transition names for the target automaton $\mathcal{A}'$:

  $$E' = \Big\{ \#e_1 \cdots e_r \mid e_1 \cdots e_r \text{ is a path segment of length } r \text{ in } \mathcal{A}, r \in \{|\theta_1|, \ldots, |\theta_p|\} \Big\}.$$

- One rewrite rule per element $\#e_1 \cdots e_r \in E'$:

  $$\underbrace{e_1 \cdots e_r}_{r \text{ symbols in } E} \rightarrow \underbrace{\#e_1 \cdots e_r}_{\text{one element in } E'} / \underbrace{\epsilon\_\_\_\epsilon}_{\text{no pre- or post-contexts}} .$$

Our Rewrite Rules

- We introduce the alphabet E' as the set of transition names for the target automaton $\mathcal{A}'$:

  $$E' = \left\{ \#e_1 \cdots e_r \mid e_1 \cdots e_r \text{ is a path segment of length } r \text{ in } \mathcal{A}, r \in \{|\theta_1|, \ldots, |\theta_p|\} \right\}.$$

- One rewrite rule per element $\#e_1 \cdots e_r \in E'$:

  $$\underbrace{e_1 \cdots e_r}_{r \text{ symbols in } E} \rightarrow \underbrace{\#e_1 \cdots e_r}_{\text{one element in } E'} / \underbrace{\epsilon\_\_\_\epsilon}_{\text{no pre- or post-contexts}} .$$

Compiling the Rewrite Rules

- Such context-dependent rewrite rules can be efficiently compiled into a **finite-state transducer (FST)** $\mathcal{T}$.
- An FST is a finite automaton whose transitions are augmented with an output label, in addition to the input label and can be viewed as an indicator function:

$$\mathcal{T} : E^* \times E'^* \to \{0, 1\}$$

- Given $x \in E^*$ and $y \in E'^*$, we have $\mathcal{T}(x, y) = 1$ iff there exists a path from an initial state to a final state with input label x and output label y.
- Example: Compiling the rewrite rule "$e_1, e_3 \to \#e_1e_3/\epsilon\_\_\_\epsilon$" for bigram gains .

[Mohri & Sproat, 1996; Mohri, 2009]

Compiling the Rewrite Rules

- Such context-dependent rewrite rules can be efficiently compiled into a **finite-state transducer (FST)** $\mathcal{T}$.
- An FST is a finite automaton whose transitions are augmented with an output label, in addition to the input label and can be viewed as an indicator function:

$$\mathcal{T} : E^* \times E'^* \to \{0, 1\}$$

- Given $x \in E^*$ and $y \in E'^*$, we have $\mathcal{T}(x, y) = 1$ iff there exists a path from an initial state to a final state with input label x and output label y.
- Example: Compiling the rewrite rule "$e_1, e_3 \to \#e_1e_3 / \epsilon \_\_\_ \epsilon$" for bigram gains .

[Mohri & Sproat, 1996; Mohri, 2009]

Compiling the Rewrite Rules

- Such context-dependent rewrite rules can be efficiently compiled into a **finite-state transducer (FST)** $\mathcal{T}$.
- An FST is a finite automaton whose transitions are augmented with an output label, in addition to the input label and can be viewed as an indicator function:

$$\mathcal{T} : E^* \times E'^* \to \{0, 1\}$$

- Given $x \in E^*$ and $y \in E'^*$, we have $\mathcal{T}(x, y) = 1$ iff there exists a path from an initial state to a final state with input label x and output label y.
- Example: Compiling the rewrite rule "$e_1, e_3 \to \#e_1e_3 / \epsilon \_\_\_ \epsilon$" for bigram gains .

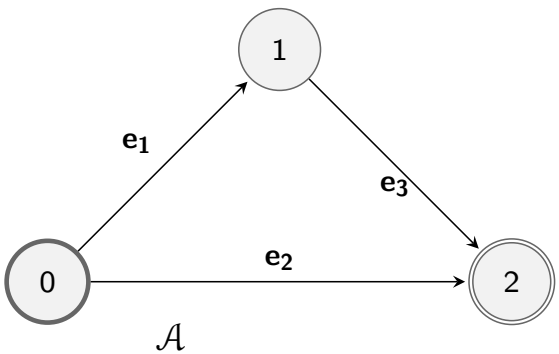
[Mohri & Sproat, 1996; Mohri, 2009]

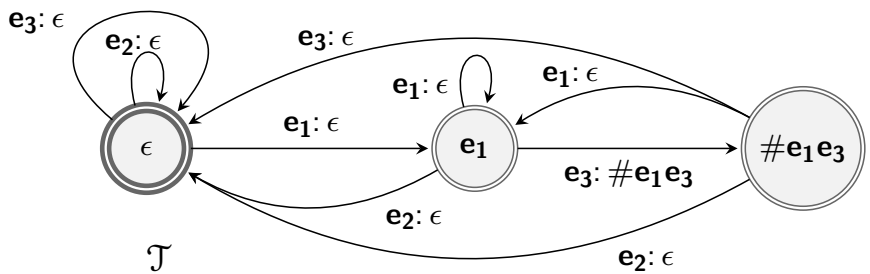
Compiling the Rewrite Rules

- Such context-dependent rewrite rules can be efficiently compiled into a **finite-state transducer (FST)** $\mathcal{T}$.
- An FST is a finite automaton whose transitions are augmented with an output label, in addition to the input label and can be viewed as an indicator function:

$$\mathcal{T} : E^* \times E'^* \to \{0, 1\}$$

- Given $x \in E^*$ and $y \in E'^*$, we have $\mathcal{T}(x, y) = 1$ iff there exists a path from an initial state to a final state with input label x and output label y.
- Example: Compiling the rewrite rule "$e_1, e_3 \to \#e_1e_3/\epsilon\_\_\_\epsilon$" for bigram gains .





[Mohri & Sproat, 1996; Mohri, 2009]

Context-Dependent Automaton $\mathcal{A}'$

- To construct the context-dependent automaton $\mathcal{A}'$, we will use the **composition** operation.
- The composition of $\mathcal{A}$ and $\mathcal{T}$ is an FST denoted by $\mathcal{A} \circ \mathcal{T}$ and defined as:

$$\forall x \in E^*,\ \forall y \in E'^* : \quad (\mathcal{A} \circ \mathcal{T})(x, y) := \mathcal{A}(x) \cdot \mathcal{T}(x, y).$$

- This composition can be found efficiently in $O(|\mathcal{A}|\,|\mathcal{T}|)$.
- To obtain $\mathcal{A}'$ from the FST $(\mathcal{A} \circ \mathcal{T})$, we use **projection**.
- Denoted by $\Pi(\cdot)$, projection is simply omitting the input label of each transition and keeping only the output label.
- Thus $\mathcal{A}'$ is defined as:

$$\mathcal{A}' := \Pi(\mathcal{A} \circ \mathcal{T}).$$

[Pereira & Riley1997; Mohri et al., 1996; Mohri, 2009]

Context-Dependent Automaton $\mathcal{A}'$

- To construct the context-dependent automaton $\mathcal{A}'$, we will use the **composition** operation.
- The composition of $\mathcal{A}$ and $\mathcal{T}$ is an FST denoted by $\mathcal{A} \circ \mathcal{T}$ and defined as:

$$\forall x \in E^*,\ \forall y \in E'^*: \quad (\mathcal{A} \circ \mathcal{T})(x, y) := \mathcal{A}(x) \cdot \mathcal{T}(x, y).$$

- This composition can be found efficiently in $O(|\mathcal{A}|\,|\mathcal{T}|)$.
- To obtain $\mathcal{A}'$ from the FST $(\mathcal{A} \circ \mathcal{T})$, we use **projection**.
- Denoted by $\Pi(\cdot)$, projection is simply omitting the input label of each transition and keeping only the output label.
- Thus $\mathcal{A}'$ is defined as:

$$\mathcal{A}' := \Pi(\mathcal{A} \circ \mathcal{T}).$$

[Pereira & Riley1997; Mohri et al., 1996; Mohri, 2009]

Context-Dependent Automaton $\mathcal{A}'$

- To construct the context-dependent automaton $\mathcal{A}'$, we will use the **composition** operation.
- The composition of $\mathcal{A}$ and $\mathcal{T}$ is an FST denoted by $\mathcal{A} \circ \mathcal{T}$ and defined as:

$$\forall x \in E^*,\ \forall y \in E'^*: \quad (\mathcal{A} \circ \mathcal{T})(x, y) := \mathcal{A}(x) \cdot \mathcal{T}(x, y).$$

- This composition can be found efficiently in $O(|\mathcal{A}|\,|\mathcal{T}|)$.
- To obtain $\mathcal{A}'$ from the FST $(\mathcal{A} \circ \mathcal{T})$, we use **projection**.
- Denoted by $\Pi(\cdot)$, projection is simply omitting the input label of each transition and keeping only the output label.
- Thus $\mathcal{A}'$ is defined as:

$$\mathcal{A}' := \Pi(\mathcal{A} \circ \mathcal{T}).$$

[Pereira & Riley1997; Mohri et al., 1996; Mohri, 2009]

Context-Dependent Automaton $\mathcal{A}'$

- To construct the context-dependent automaton $\mathcal{A}'$, we will use the **composition** operation.
- The composition of $\mathcal{A}$ and $\mathcal{T}$ is an FST denoted by $\mathcal{A} \circ \mathcal{T}$ and defined as:

$$\forall x \in E^*,\ \forall y \in E'^*: \quad (\mathcal{A} \circ \mathcal{T})(x, y) := \mathcal{A}(x) \cdot \mathcal{T}(x, y).$$

- This composition can be found efficiently in $O(|\mathcal{A}|\,|\mathcal{T}|)$.
- To obtain $\mathcal{A}'$ from the FST $(\mathcal{A} \circ \mathcal{T})$, we use **projection**.
- Denoted by $\Pi(\cdot)$, projection is simply omitting the input label of each transition and keeping only the output label.
- Thus $\mathcal{A}'$ is defined as:

$$\mathcal{A}' := \Pi(\mathcal{A} \circ \mathcal{T}).$$

[Pereira & Riley1997; Mohri et al., 1996; Mohri, 2009]

Context-Dependent Automaton $\mathcal{A}'$

- To construct the context-dependent automaton $\mathcal{A}'$, we will use the **composition** operation.
- The composition of $\mathcal{A}$ and $\mathcal{T}$ is an FST denoted by $\mathcal{A} \circ \mathcal{T}$ and defined as:

$$\forall x \in E^*,\ \forall y \in E'^*: \quad (\mathcal{A} \circ \mathcal{T})(x, y) := \mathcal{A}(x) \cdot \mathcal{T}(x, y).$$

- This composition can be found efficiently in $O(|\mathcal{A}|\,|\mathcal{T}|)$.
- To obtain $\mathcal{A}'$ from the FST $(\mathcal{A} \circ \mathcal{T})$, we use **projection**.
- Denoted by $\Pi(\cdot)$, projection is simply omitting the input label of each transition and keeping only the output label.
- Thus $\mathcal{A}'$ is defined as:

$$\mathcal{A}' := \Pi(\mathcal{A} \circ \mathcal{T}).$$

[Pereira & Riley1997; Mohri et al., 1996; Mohri, 2009]

Context-Dependent Automaton $\mathcal{A}'$

- To construct the context-dependent automaton $\mathcal{A}'$, we will use the **composition** operation.
- The composition of $\mathcal{A}$ and $\mathcal{T}$ is an FST denoted by $\mathcal{A} \circ \mathcal{T}$ and defined as:

$$\forall x \in E^*,\ \forall y \in E'^*: \quad (\mathcal{A} \circ \mathcal{T})(x, y) := \mathcal{A}(x) \cdot \mathcal{T}(x, y).$$

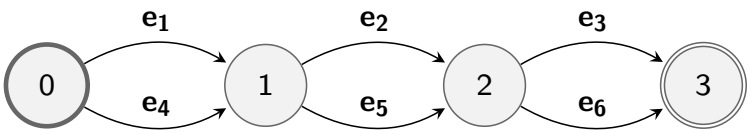
- This composition can be found efficiently in $O(|\mathcal{A}|\,|\mathcal{T}|)$.
- To obtain $\mathcal{A}'$ from the FST $(\mathcal{A} \circ \mathcal{T})$, we use **projection**.
- Denoted by $\Pi(\cdot)$, projection is simply omitting the input label of each transition and keeping only the output label.
- Thus $\mathcal{A}'$ is defined as:

$$\mathcal{A}' := \Pi(\mathcal{A} \circ \mathcal{T}).$$

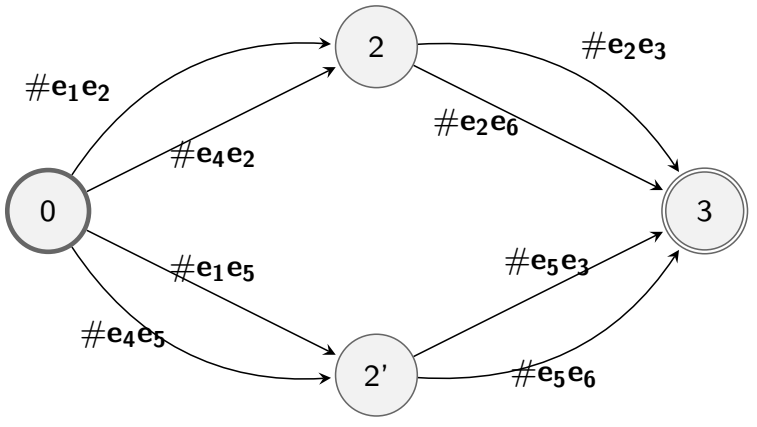
[Pereira & Riley1997; Mohri et al., 1996; Mohri, 2009]

Context-Dependent Automaton $\mathcal{A}'$

- Example: The expert and context-dependent automata for bigram gains:



$\mathcal{A}$

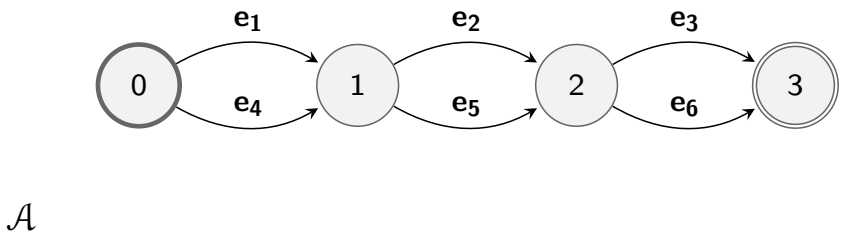


$\mathcal{A}'$

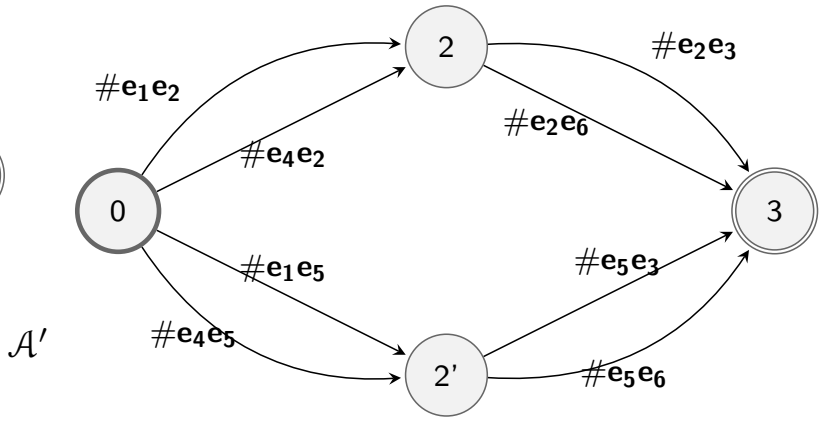
- Properties:
  1. $\mathcal{A}'$ is data-independent and can be constructed as a pre-processing step.
  2. The size of $\mathcal{A}'$ depends on
     - The expert automaton $\mathcal{A}$.
     - Length of the patterns $|\theta_1|, \ldots, |\theta_p|$.
  3. The size of $\mathcal{A}'$ is independent of
     - Size of the alphabet $|\Sigma|$.
     - Number of patterns p.

Context-Dependent Automaton $\mathcal{A}'$

- Example: The expert and context-dependent automata for bigram gains:



$\mathcal{A}$

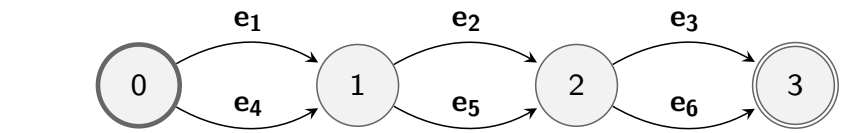


$\mathcal{A}'$

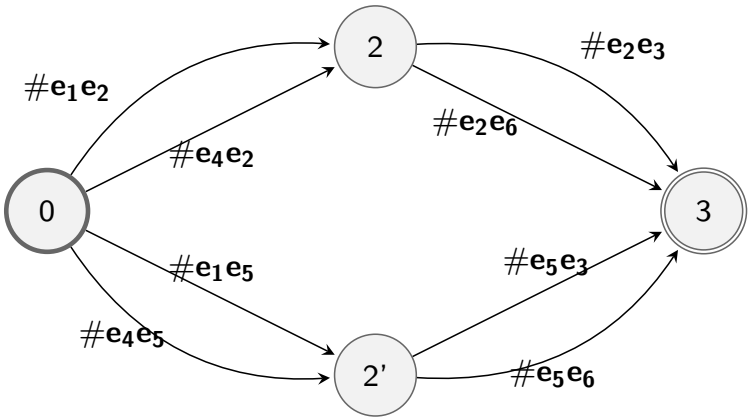
- Properties:
  1. $\mathcal{A}'$ is data-independent and can be constructed as a pre-processing step.
  2. The size of $\mathcal{A}'$ depends on
     - The expert automaton $\mathcal{A}$.
     - Length of the patterns $|\theta_1|, \ldots, |\theta_p|$.
  3. The size of $\mathcal{A}'$ is independent of
     - Size of the alphabet $|\Sigma|$.
     - Number of patterns p.

Context-Dependent Automaton $\mathcal{A}'$

- Example: The expert and context-dependent automata for bigram gains:



$\mathcal{A}$

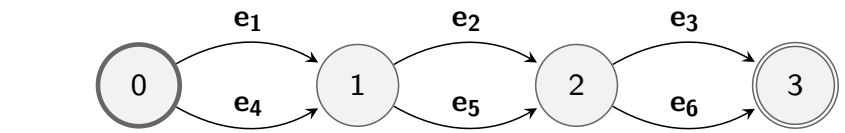


$\mathcal{A}'$

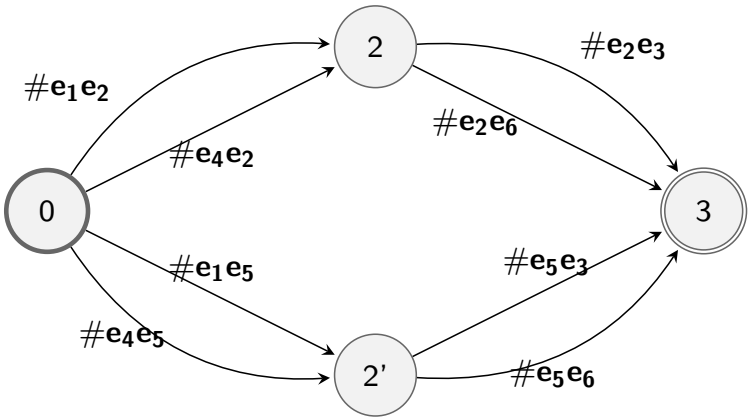
- Properties:
  1. $\mathcal{A}'$ is data-independent and can be constructed as a pre-processing step.
  2. The size of $\mathcal{A}'$ depends on
     - The expert automaton $\mathcal{A}$.
     - Length of the patterns $|\theta_1|, \ldots, |\theta_p|$.
  3. The size of $\mathcal{A}'$ is independent of
     - Size of the alphabet $|\Sigma|$.
     - Number of patterns p.

Context-Dependent Automaton $\mathcal{A}'$

- Example: The expert and context-dependent automata for bigram gains:



$\mathcal{A}$



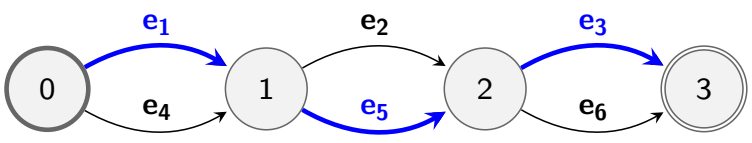
$\mathcal{A}'$

- Properties:
  1. $\mathcal{A}'$ is data-independent and can be constructed as a pre-processing step.
  2. The size of $\mathcal{A}'$ depends on
     - The expert automaton $\mathcal{A}$.
     - Length of the patterns $|\theta_1|, \ldots, |\theta_p|$.
  3. The size of $\mathcal{A}'$ is independent of
     - Size of the alphabet $|\Sigma|$.
     - Number of patterns p.

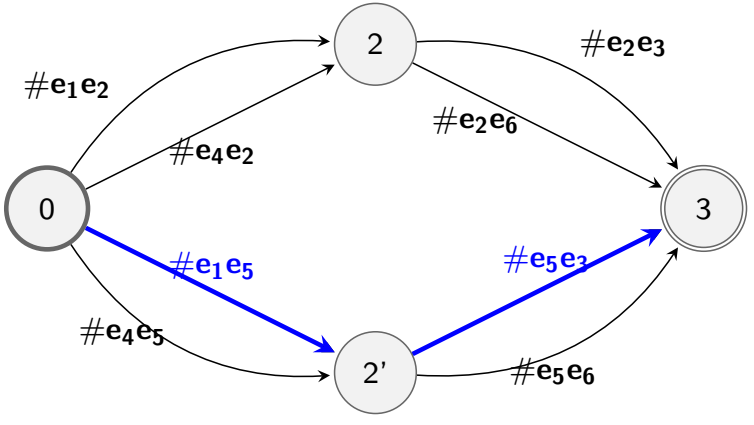
Context-Dependent Automaton $\mathcal{A}'$ – Path Mapping

Proposition

Let $\mathcal{A}$ be an expert automaton and let $\mathcal{T}$ be a deterministic transducer representing the rewrite rules. Then, for each path π in $\mathcal{A}$, there exists a unique corresponding path π' in $\mathcal{A}' = \Pi(\mathcal{A} \circ T_{\mathcal{A}})$.



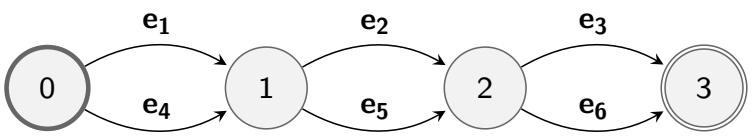
$\mathcal{A}$



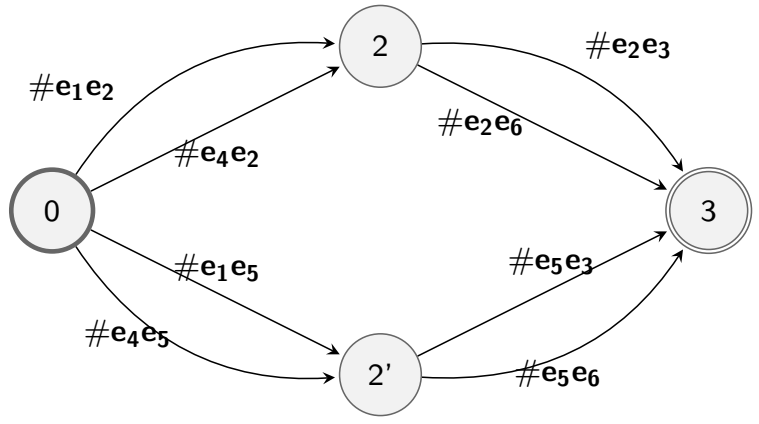
$\mathcal{A}'$

Context-Dependent Automaton $\mathcal{A}'$ – Path Outputs and Additive Gains

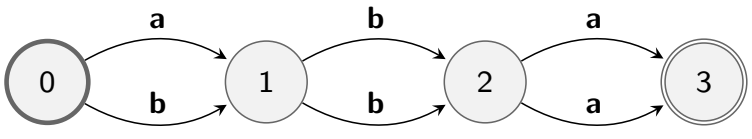
- Example:
  $\Sigma = \{a, b\}, y_t = aba \longrightarrow (\theta_1, \theta_2, \theta_3, \theta_4) = (aa, ab, ba, bb), \Theta(y_t) = [0, 1, 1, 0]^T$.



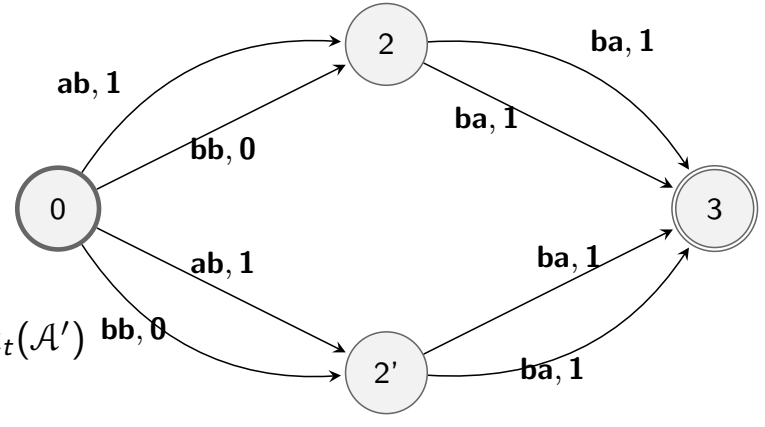
$\mathcal{A}$



$\mathcal{A}'$



$\text{out}_t(\mathcal{A})$



$\text{out}_t(\mathcal{A}')$

Context-Dependent Automaton $\mathcal{A}'$ – Path Outputs and Additive Gains

Theorem

At any round $t \in \{1..T\}$, define the gain $g_{e',t}$ of the transition $e' \in E'$ in $\mathcal{A}'$:

$$g_{e',t} := [\Theta(y_t)]_k \,,$$

if $out_t(e') = \theta_k$ for some $k \in \{1..p\}$ and $g_{e',t} := 0$ if no such k exists. Then, the gain of each path π in $\mathcal{A}$ at round t can be expressed as an additive gain of the corresponding unique path π' in $\mathcal{A}'$:

$$\mathcal{U}(out_t(\pi), y_t) = \sum_{e' \in \pi'} g_{e',t} \,.$$

Applying Algorithms for Additive Gains

- Having established the additivity of gains in $\mathcal{A}'$, we can apply the well-known algorithms for additive losses/gains on top.
- **Full information**: *Component Hedge* [Koolen, Warmuth, Kivinen, 2010].
  - Better bounds w.r.t. [Cortes et al., 2015].
  - No additional assumptions are required for efficiency despite [Cortes et al., 2015].
- **Semi-Bandit:** Algorithm of [Győrgy et al., 2007]
  - First efficient algorithm with favorable regret bounds for non-additive gains.
- **Full Bandit:** ComBand [Cesa-Bianchi and Lugosi, 2011]
  - First efficient algorithm with favorable regret bounds for non-additive gains.

Applying Algorithms for Additive Gains

- Having established the additivity of gains in $\mathcal{A}'$, we can apply the well-known algorithms for additive losses/gains on top.
- **Full information**: *Component Hedge* [Koolen, Warmuth, Kivinen, 2010].
  - Better bounds w.r.t. [Cortes et al., 2015].
  - No additional assumptions are required for efficiency despite [Cortes et al., 2015].
- **Semi-Bandit:** Algorithm of [Győrgy et al., 2007]
  - First efficient algorithm with favorable regret bounds for non-additive gains.
- **Full Bandit:** ComBand [Cesa-Bianchi and Lugosi, 2011]
  - First efficient algorithm with favorable regret bounds for non-additive gains.

Applying Algorithms for Additive Gains

- Having established the additivity of gains in $\mathcal{A}'$, we can apply the well-known algorithms for additive losses/gains on top.
- **Full information**: *Component Hedge* [Koolen, Warmuth, Kivinen, 2010].
  - Better bounds w.r.t. [Cortes et al., 2015].
  - No additional assumptions are required for efficiency despite [Cortes et al., 2015].
- **Semi-Bandit:** Algorithm of [Győrgy et al., 2007]
  - First efficient algorithm with favorable regret bounds for non-additive gains.
- **Full Bandit:** ComBand [Cesa-Bianchi and Lugosi, 2011]
  - First efficient algorithm with favorable regret bounds for non-additive gains.

Applying Algorithms for Additive Gains

- Having established the additivity of gains in $\mathcal{A}'$, we can apply the well-known algorithms for additive losses/gains on top.
- **Full information**: *Component Hedge* [Koolen, Warmuth, Kivinen, 2010].
  - Better bounds w.r.t. [Cortes et al., 2015].
  - No additional assumptions are required for efficiency despite [Cortes et al., 2015].
- **Semi-Bandit:** Algorithm of [Győrgy et al., 2007]
  - First efficient algorithm with favorable regret bounds for non-additive gains.
- **Full Bandit:** ComBand [Cesa-Bianchi and Lugosi, 2011]
  - First efficient algorithm with favorable regret bounds for non-additive gains.

Future Work and Open Problems

1. Data-dependent on-the-fly construction?
2. Cyclic automaton $\mathcal{A}$ and infinite regular language $\mathcal{L}$?

Future Work and Open Problems

1. Data-dependent on-the-fly construction?
2. Cyclic automaton $\mathcal{A}$ and infinite regular language $\mathcal{L}$?

Thanks!