Aprendizado, minimização do arrependimento e equilíbrio

(Learning, Regret Minimization, and Equilibria)

Victor Alberto Romero

Instituto de Matemática e Estatística
Universidade de São Paulo

Teoria dos Jogos e Algorítmica

Introdução

- O problema principal é *tomar decisões em ambientes desconhecidos.*
- A aproximação usada neste capítulo é desenhar algoritmos adaptativos que podem aprender entendendo a dinâmica do sistema.
- A estratégia usada é a análise do arrependimento (*Regret*), ou seja, o cálculo da diferença entre a melhor utilidade que poderia ter sido e a utilidade real obtida. Temos três análises possíveis do arrependimento

Introdução

- O problema principal é *tomar decisões em ambientes desconhecidos.*
- A aproximação usada neste capítulo é desenhar algoritmos adaptativos que podem aprender entendendo a dinâmica do sistema.
- A estratégia usada é a análise do arrependimento (*Regret*), ou seja, o cálculo da diferença entre a melhor utilidade que poderia ter sido e a utilidade real obtida. Temos três análises possíveis do arrependimento

Introdução

- O problema principal é *tomar decisões em ambientes desconhecidos.*
- A aproximação usada neste capítulo é desenhar algoritmos adaptativos que podem aprender entendendo a dinâmica do sistema.
- A estratégia usada é a análise do arrependimento (*Regret*), ou seja, o cálculo da diferença entre a melhor utilidade que poderia ter sido e a utilidade real obtida. Temos três análises possíveis do arrependimento

- Também conhecido como *Combining expert advice*
- Compara com a melhor opção individual que poderia ter sido (a melhor em retrospectiva)
- Dá uma metodologia geral para o desenvolvimento de algoritmos online.

- Também conhecido como *Combining expert advice*
- Compara com a melhor opção individual que poderia ter sido (a melhor em retrospectiva)
- Dá uma metodologia geral para o desenvolvimento de algoritmos online.

- Também conhecido como *Combining expert advice*
- Compara com a melhor opção individual que poderia ter sido (a melhor em retrospectiva)
- Dá uma metodologia geral para o desenvolvimento de algoritmos online.

- No capítulo, o autor faz um desenvolvimento muito interessante: começa como um algoritmo guloso simples, e sobre ele vai construindo até chegar num algoritmo no qual a perda é, no máximo, $O\left(\sqrt{T\log N}\right)$ a mais do que a melhor escolha, onde T é o número de passos dados até o momento.

- Ou seja, o máximo afastamento do ótimo por lance é:

$$O\left(\sqrt{\frac{\log N}{T}}\right)$$

- No capítulo, o autor faz um desenvolvimento muito interessante: começa como um algoritmo guloso simples, e sobre ele vai construindo até chegar num algoritmo no qual a perda é, no máximo, $O\left(\sqrt{T\log N}\right)$ a mais do que a melhor escolha, onde T é o número de passos dados até o momento.

- Ou seja, o máximo afastamento do ótimo por lance é:

$$O\left(\sqrt{\frac{\log N}{T}}\right)$$

- Arrependimento interno (*Internal Regret*)
  - Considera a historia
  - Permite que **uma** ação passada seja substituída por outra
- Arrependimento de troca (*Swap Regret*)
  - Considera a historia
  - Permite qualquer mapeamento de $\{1,...,N\}$ para $\{1,...,N\}$ e pode ser aumentado num fator de N

- Arrependimento interno (*Internal Regret*)
  - Considera a historia
  - Permite que **uma** ação passada seja substituída por outra
- Arrependimento de troca (*Swap Regret*)
  - Considera a historia
  - Permite qualquer mapeamento de $\{1,...,N\}$ para $\{1,...,N\}$ e pode ser aumentado num fator de N

Introdução
Modelo de informação parcial (*partial information model*)

- Também conhecido como *Multiarmed bandit - MAB*
- Só pode ver as perdas de sua escolha atual, não as perdas das outras escolhas (e por isso, não pode saber diretamente se é a melhor escolha ou não!!)

O autor deixa claro que os resultados obtidos nesse capítulo não são os melhores que se conhecem. Nas notas do capítulo tem uma lista de trabalhos que apresentam melhores algoritmos para este problema.

Introdução
Modelo de informação parcial (*partial information model*)

- Também conhecido como *Multiarmed bandit - MAB*
- Só pode ver as perdas de sua escolha atual, não as perdas das outras escolhas (e por isso, não pode saber diretamente se é a melhor escolha ou não!!)

O autor deixa claro que os resultados obtidos nesse capítulo não são os melhores que se conhecem. Nas notas do capítulo tem uma lista de trabalhos que apresentam melhores algoritmos para este problema.

- Também conhecido como *Multiarmed bandit - MAB*
- Só pode ver as perdas de sua escolha atual, não as perdas das outras escolhas (e por isso, não pode saber diretamente se é a melhor escolha ou não!!)

O autor deixa claro que os resultados obtidos nesse capítulo não são os melhores que se conhecem. Nas notas do capítulo tem uma lista de trabalhos que apresentam melhores algoritmos para este problema.

Modelo e preliminares

- Jogos de adversários **online** (processa só a entrada, sem memória de estados anteriores) como $X = \{1, ..., N\}$ opções para cada jogador
- Em cada passo t o algoritmo H escolhe uma distribuição p^t sobre o conjunto de N ações.
- Depois disso **o adversário** escolhe um vetor de perdas $\ell^t \in [0,1]^N$ (um valor de perda para cada escolha), onde $\ell_i^t \in [0,1]$ é a perda se o jogador escolhe a opção i no momento t

Modelo e preliminares

- Jogos de adversários **online** (processa só a entrada, sem memória de estados anteriores) como $X = \{1, ..., N\}$ opções para cada jogador
- Em cada passo t o algoritmo H escolhe uma distribuição p^t sobre o conjunto de N ações.
- Depois disso **o adversário** escolhe um vetor de perdas $\ell^t \in [0,1]^N$ (um valor de perda para cada escolha), onde $\ell_i^t \in [0,1]$ é a perda se o jogador escolhe a opção i no momento t

Modelo e preliminares

- Jogos de adversários **online** (processa só a entrada, sem memória de estados anteriores) como $X = \{1, ..., N\}$ opções para cada jogador

- Em cada passo t o algoritmo H escolhe uma distribuição p^t sobre o conjunto de N ações.

- Depois disso **o adversário** escolhe um vetor de perdas $\ell^t \in [0,1]^N$ (um valor de perda para cada escolha), onde $\ell_i^t \in [0,1]$ é a perda se o jogador escolhe a opção i no momento t

No modelo com informação completa (*full information model*) o algoritmo H recebe o vetor de perdas ℓ^t e tem um valor de perda:

$$\ell_H^t = \sum_{i=1}^{N} p_i^t \ell_i^t$$

que é a perda esperada quando o algoritmo escolhe a opção $i \in X$ com probabilidade p_i^t.

No modelo de informação parcial (*partial information model*) o algoritmo H recebe $(\ell^t_{k^t}, k^t)$, onde k^t segue a distribuição p^t, e $\ell^t_H = \ell^t_{k^t}$ é sua perda. A perda da i-ésima ação durante os primeiros T passos é:

$$L^T_i = \sum_{t=1}^{T} \ell^t_i$$

e a perda de H é:

$$L^T_H = \sum_{t=1}^{T} \ell^t_H$$

A razão para calcular o arrependimento externo (*external regret*) é poder criar um algoritmo online que possa se aproximar ao melhor algoritmo de uma classe específica de algoritmos $\mathscr{G}$. Isto quer dizer, que queremos uma perda próxima a:

$$L^T_{\mathscr{G},\text{min}} = \text{min}_{g \in \mathscr{G}} L^T_g$$

Formalmente nós queremos minimizar o arrependimento externo:

$$R_{\mathscr{G}} = L^T_H - L^T_{\mathscr{G},\text{min}}$$

Aqui $\mathscr{G}$ é chamada a **classe de comparação.** Em geral, usa-se uma classe de comparação $\mathscr{G} = X$ (ou seja, todas as possíveis escolhas) e procura-se um algoritmo online com uma perda perto de

$$L^T_{\min} = \min_i L^T_i$$

sendo o arrependimento externo:

$$R = L^T_H - L^T_{\min}$$

- Normalmente o arrependimento externo usa uma classe de comparação fixa $\mathscr{G}$, mas poderia ter-se uma classe que dependa das ações escolhidas pelo algoritmo online. Neste caso podemos considerar regras de modificação que mudem as ações escolhidas pelo algoritmo online e produzam uma estratégia alternativa.

- Uma **regra de modificação** F recebe na entrada o histórico de ações e a ação atual escolhidas por H, e devolve uma ação (que **pode** ser diferente ou não). Chama-se F^t à função F no momento t

- Normalmente o arrependimento externo usa uma classe de comparação fixa $\mathscr{G}$, mas poderia ter-se uma classe que dependa das ações escolhidas pelo algoritmo online. Neste caso podemos considerar regras de modificação que mudem as ações escolhidas pelo algoritmo online e produzam uma estratégia alternativa.

- Uma **regra de modificação** F recebe na entrada o histórico de ações e a ação atual escolhidas por H, e devolve uma ação (que **pode** ser diferente ou não). Chama-se F^t à função F no momento t

Dada uma sequência de distribuições de probabilidade p^t usada pelo algoritmo H e uma regra de modificação F, pode-se definir uma nova sequência de distribuições de probabilidade $f^t = F^t(p^t)$, onde

$$f_i^t = \sum_{j:F^t(j)=i} p_j^t$$

A perda desta nova sequência é:

$$L_{H,F}^T = \sum_t \sum_i f_i^t \ell_i^t$$

no livro é: $L_{H,F} = \sum_t \sum_i f_i^t \ell_i^t$

Nota-se que no momento t, F muda a probabilidade de H associando a ação j a ação $F^t(j)$. Isto implica que a regra de modificação F gera uma distribuição diferente, que é função da distribuição de H, p^t.

Dada uma sequência de distribuições de probabilidade p^t usada pelo algoritmo H e uma regra de modificação F, pode-se definir uma nova sequência de distribuições de probabilidade $f^t = F^t(p^t)$, onde

$$f_i^t = \sum_{j:F^t(j)=i} p_j^t$$

A perda desta nova sequência é:

$$L_{H,F}^T = \sum_t \sum_i f_i^t \ell_i^t$$

no livro é: $L_{H,F} = \sum_t \sum_i f_i^t \ell_i^t$

Nota-se que no momento t, F muda a probabilidade de H associando a ação j a ação $F^t(j)$. Isto implica que a regra de modificação F gera uma distribuição diferente, que é função da distribuição de H, p^t.

Dada uma sequência de distribuições de probabilidade p^t usada pelo algoritmo H e uma regra de modificação F, pode-se definir uma nova sequência de distribuições de probabilidade $f^t = F^t(p^t)$, onde

$$f_i^t = \sum_{j:F^t(j)=i} p_j^t$$

A perda desta nova sequência é:

$$L_{H,F}^T = \sum_t \sum_i f_i^t \ell_i^t$$

no livro é: $L_{H,F} = \sum_t \sum_i f_i^t \ell_i^t$

Nota-se que no momento t, F muda a probabilidade de H associando a ação j a ação $F^t(j)$. Isto implica que a regra de modificação F gera uma distribuição diferente, que é função da distribuição de H, p^t.

Dada uma sequência de distribuições de probabilidade p^t usada pelo algoritmo H e uma regra de modificação F, pode-se definir uma nova sequência de distribuições de probabilidade $f^t = F^t(p^t)$, onde

$$f_i^t = \sum_{j:F^t(j)=i} p_j^t$$

A perda desta nova sequência é:

$$L_{H,F}^T = \sum_t \sum_i f_i^t \ell_i^t$$

no livro é: $L_{H,F} = \sum_t \sum_i f_i^t \ell_i^t$

Nota-se que no momento t, F muda a probabilidade de H associando a ação j a ação $F^t(j)$. Isto implica que a regra de modificação F gera uma distribuição diferente, que é função da distribuição de H, p^t.

- O artigo foca-se num conjunto finito $\mathscr{F}$ de regras de modificação que não requer memória.

- Dada uma sequência de vetores de perda, o arrependimento de usar o algoritmo online H com relação às regras de modificação $\mathscr{F}$ é:

$$R_{\mathscr{F}} = \max_{F \in \mathscr{F}} \left\{ L_H^T - L_{H,F}^T \right\}$$

- O artigo foca-se num conjunto finito $\mathscr{F}$ de regras de modificação que não requer memória.

- Dada uma sequência de vetores de perda, o arrependimento de usar o algoritmo online H com relação às regras de modificação $\mathscr{F}$ é:

$$R_{\mathscr{F}} = \max_{F \in \mathscr{F}} \left\{ L_H^T - L_{H,F}^T \right\}$$

- Pode-se ver que o arrependimento externo é equivalente a ter um conjunto de $\mathscr{F}^{ex}$ de N regras de modificação F_i, onde cada F_i tem como saída a ação i. Para o **arrependimento interno** (*internal regret*), o conjunto $\mathscr{F}^{in}$ consiste em $N(N-1)$ regras de modificação $F_{i,j}$, onde $F_{i,j}(i) = j$ e $F_{i,j}(i') = i'$, para $i' \neq i$. Então o arrependimento interno de H é:

$$\max_{F \in \mathscr{F}^{in}} \left\{ L_H^T - L_{H,F}^T \right\} = \max_{i,j \in X} \left\{ \sum_{t=1}^{T} p_i^t \left(\ell_i^t - \ell_j^t \right) \right\}$$

- Um caso mais geral das regras de modificação sem memória é o arrependimento de troca (*swap regret*), definido pela classe $\mathscr{F}^{sw}$. $\mathscr{F}^{sw}$ inclui as N^N funções $F : \{1,...,N\} \to \{1,...,N\}$, onde a função F troca a ação atual de H, de i para $F(i)$. O arrependimento de troca para H é:

$$\max_{F \in \mathscr{F}^{sw}} \left\{ L_H^T - L_{H,F}^T \right\} = \sum_{i=1}^{N} \max_{j \in X} \left\{ \sum_{t=1}^{T} p_i^t \left(\ell_i^t - \ell_j^t \right) \right\}$$

- É importante notar que:

$$\mathscr{F}^{ex} \subseteq \mathscr{F}^{sw}$$
$$\mathscr{F}^{in} \subseteq \mathscr{F}^{sw}$$

- Um caso mais geral das regras de modificação sem memória é o arrependimento de troca (*swap regret*), definido pela classe $\mathscr{F}^{sw}$. $\mathscr{F}^{sw}$ inclui as N^N funções $F : \{1,...,N\} \to \{1,...,N\}$, onde a função F troca a ação atual de H, de i para $F(i)$. O arrependimento de troca para H é:

$$\max_{F \in \mathscr{F}^{sw}} \left\{ L_H^T - L_{H,F}^T \right\} = \sum_{i=1}^{N} \max_{j \in X} \left\{ \sum_{t=1}^{T} p_i^t \left(\ell_i^t - \ell_j^t \right) \right\}$$

- É importante notar que:

$$\begin{aligned} \mathscr{F}^{ex} &\subseteq \mathscr{F}^{sw} \\ \mathscr{F}^{in} &\subseteq \mathscr{F}^{sw} \end{aligned}$$

Minimização do arrependimento externo

- Más noticias: Não é possível garantir um baixo arrependimento em relação à sequência ótima de decisões em retrospetiva. (Borodin e El-Yaniv, 1998; Sleator e Tarjan, 1985)

- Como alternativa, o autor propõe focar em classes de comparação mais restritas. Vamos usar a classe $\mathscr{G}_{\text{all}}$, que é o conjunto de funções que fazem o mapeamento de instantes de tempo $\{1,...,T\}$ para escolhas $X = \{1,...,N\}$

Minimização do arrependimento externo

- Más noticias: Não é possível garantir um baixo arrependimento em relação à sequência ótima de decisões em retrospetiva. (Borodin e El-Yaniv, 1998; Sleator e Tarjan, 1985)

- Como alternativa, o autor propõe focar em classes de comparação mais restritas. Vamos usar a classe $\mathscr{G}_{\mathsf{all}}$, que é o conjunto de funções que fazem o mapeamento de instantes de tempo $\{1,...,T\}$ para escolhas $X = \{1,...,N\}$

Teorema

Para todo algoritmo online H existe uma sequência de vetores de perda T como a qual o arrependimento $R_{\mathscr{G}_{all}}$ é pelo menos $T\left(1-\frac{1}{N}\right)$.

Demonstração.

Para cada instante t, a escolha i^t de mais baixa probabilidade p_i^t toma um valor de perda de 0 e todas as outras escolhas tomam o valor de 1. Como:

$$\min_i\{p_i^t\} \leq \frac{1}{N}$$

Então, a perda de H ao longo de T passos é pelo menos $T\left(1-\frac{1}{N}\right)$. Também pode-se definir um $g \in \mathscr{G}_{\text{all}}$, tal que $g(t) = i_t$ e tem um perda total de 0. □

Teorema

Para todo algoritmo online H existe uma sequência de vetores de perda T como a qual o arrependimento $R_{\mathscr{G}_{all}}$ é pelo menos $T\left(1-\frac{1}{N}\right)$.

Demonstração.

Para cada instante t, a escolha i^t de mais baixa probabilidade p_i^t toma um valor de perda de 0 e todas as outras escolhas tomam o valor de 1. Como:

$$\min_i\{p_i^t\} \leq \frac{1}{N}$$

Então, a perda de H ao longo de T passos é pelo menos $T\left(1-\frac{1}{N}\right)$. Também pode-se definir um $g \in \mathscr{G}_{\text{all}}$, tal que $g(t) = i_t$ e tem um perda total de 0. □

Minimização do arrependimento externo

- Para simplificar, vamos supôr que todas as perdas são ou 0 ou 1.
- Vamos chamar $L_i^t = \sum_{\tau=1}^{t} \ell_i^\tau$ a perda acumulada até o momento t da ação i
- Um algoritmo guloso vai escolher para cada momento t uma ação:

$$x^t = \arg\min_{i \in X} L_i^{t-1}$$

Minimização do arrependimento externo

- Para simplificar, vamos supôr que todas as perdas são ou 0 ou 1.
- Vamos chamar $L_i^t = \sum_{\tau=1}^{t} \ell_i^\tau$ a perda acumulada até o momento t da ação i
- Um algoritmo guloso vai escolher para cada momento t uma ação:

$$x^t = \arg\min_{i \in X} L_i^{t-1}$$

Minimização do arrependimento externo

- Para simplificar, vamos supôr que todas as perdas são ou 0 ou 1.
- Vamos chamar $L_i^t = \sum_{\tau=1}^{t} \ell_i^\tau$ a perda acumulada até o momento t da ação i
- Um algoritmo guloso vai escolher para cada momento t uma ação:

$$x^t \quad = \quad \text{arg min}_{i \in X} L_i^{t-1}$$

Algoritmo Guloso

Inicializa: $x^1 = 1$

Em cada momento t:

- $L_{\min}^{t-1} = \min_{i \in X} L_i^{t-1}$
- $S^{t-1} = \left\{ i : L_i^{t-1} = L_{\min}^{t-1} \right\}$
- $x^t = \min S^{t-1}$

Teorema

O algoritmo guloso tem uma perda acumulada
$L_G^T \leq N \cdot L_{min}^T + (N-1)$ *para qualquer sequência de perdas*

Algoritmo Guloso

Inicializa: $x^1 = 1$

Em cada momento t:

- $L_{\min}^{t-1} = \min_{i \in X} L_i^{t-1}$
- $S^{t-1} = \left\{ i : L_i^{t-1} = L_{\min}^{t-1} \right\}$
- $x^t = \min S^{t-1}$

Teorema

O algoritmo guloso tem uma perda acumulada
$L_G^T \leq N \cdot L_{min}^T + (N-1)$ *para qualquer sequência de perdas*

Demonstração.

Em cada momento t no qual o algortimo perda 1 e o L^t_{min} não mude, pelo menos uma ação e removida do S^t. Isto ocorre no máximo N vezes até que L^t_{min} incremente em 1. Ou seja, o algoritmo tem uma perda de máximo N entre incrementos de L^t_{min}:

$$L^t_G \leq N - |S^t| + N \cdot L^t_{\text{min}}$$

□

Pode-se ver que o algoritmo é $O(N)$ vezes pior que o ótimo. Vamos ver que qualquer algoritmo determinístico tem pelo menos o mesmo fator.

Demonstração.

Em cada momento t no qual o algortimo perda 1 e o L^t_{min} não mude, pelo menos uma ação e removida do S^t. Isto ocorre no máximo N vezes até que L^t_{min} incremente em 1. Ou seja, o algoritmo tem uma perda de máximo N entre incrementos de L^t_{min}:

$$L^t_G \leq N - |S^t| + N \cdot L^t_{\text{min}}$$

□

Pode-se ver que o algoritmo é $O(N)$ vezes pior que o ótimo. Vamos ver que qualquer algoritmo determinístico tem pelo menos o mesmo fator.

Minimização do arrependimento externo
Algoritmo Guloso

Teorema

Para qualquer algoritmo determinístico D existe uma sequência de perdas para a qual $L_D^T = T$ e $L_{min}^T = \lfloor \frac{T}{N} \rfloor$

Nota-se que este teorema implica que $L_D^T \geq N \cdot L_{\text{min}}^T + (T \bmod N)$ o que é um limite superior para o algoritmo guloso.

Teorema

Para qualquer algoritmo determinístico D existe uma sequência de perdas para a qual $L_D^T = T$ e $L_{min}^T = \lfloor \frac{T}{N} \rfloor$

Nota-se que este teorema implica que $L_D^T \geq N \cdot L_{\text{min}}^T + (T \bmod N)$ o que é um limite superior para o algoritmo guloso.

Demonstração.

Para a prova, vamos fixar um algoritmo online D e vamos dizer que x^t é sua saída no momento t. Vamos então a gerar uma sequência assim: No momento t vamos definir a perda para x^t de 1 e para qualquer outra ação de 0. Isto garante que o algoritmo D tem uma perda de 1 em cada passo, então $L_D^T = T$.
Como existem N ações diferentes, tem que existir uma ação que o algoritmo D tenha escolhido pelo menos $\lfloor \frac{T}{N} \rfloor$ vezes. Então, por construção, como só as ações escolhidas por D tem perdas, implica que $L_{\min}^T \leq \lfloor \frac{T}{N} \rfloor$ □

Dá para pensar então que pode acontecer se o algoritmo guloso tem como saída uma distribuição de probabilidade. Vamos chamar esse algoritmo *Gusloso Aleatório*

Algoritmo Guloso Aleátorio

Inicializa: $p_i^1 = 1/N$ para $i \in X$

Em cada momento t:

- $L_{\min}^{t-1} = \min_{i \in X} L_i^{t-1}$
- $S^{t-1} = \left\{ i : L_i^{t-1} = L_{\min}^{t-1} \right\}$
- $p_i^t = \frac{1}{|S^{t-1}|}$ para $i \in S^{t-1}$ e $p_i^t = 0$ em outro contrario

Dá para pensar então que pode acontecer se o algoritmo guloso tem como saída uma distribuição de probabilidade. Vamos chamar esse algoritmo *Gusloso Aleatório*

Algoritmo Guloso Aleátorio

Inicializa: $p_i^1 = 1/N$ para $i \in X$

Em cada momento t:

- $L_{\text{min}}^{t-1} = \text{min}_{\ i \in X} L_i^{t-1}$
- $S^{t-1} = \left\{ i : L_i^{t-1} = L_{\text{min}}^{t-1} \right\}$
- $p_i^t = \frac{1}{|S^{t-1}|}$ para $i \in S^{t-1}$ e $p_i^t = 0$ em outro contrario

Teorema

O algoritmo guloso aleatório tem, para qualquer sequência de perdas, uma perda de $L^T_{RG} \leq (ln\ N) + (1 + ln\ N)L^T_{min}$

Demonstração.

Como no caso anterior, vai se provar que entre incrementos sucessivos de L^T_{min} o algoritmo incorre numa perda pelo menos $1 + \ln N$. Seja t_j o momento onde L^t_{min} atinge por primeira vez uma perda j. Vamos ver que acontece entre os instantes t_j e t_{j+1}. Para qualquer instante de tempo t temos que $1 \leq |S^t| \leq N$.

(continua) □

Teorema

O algoritmo guloso aleatório tem, para qualquer sequência de perdas, uma perda de $L^T_{RG} \leq (ln\ N) + (1 + ln\ N)L^T_{min}$

Demonstração.

Como no caso anterior, vai se provar que entre incrementos sucessivos de L^T_{min} o algoritmo incorre numa perda pelo menos $1 + \ln N$. Seja t_j o momento onde L^t_{min} atinge por primeira vez uma perda j. Vamos ver que acontece entre os instantes t_j e t_{j+1}. Para qualquer instante de tempo t temos que $1 \leq |S^t| \leq N$.

(continua) □

Demonstração.

(continuação) Agora, se num instante $t \in (t_j, t_{j+1}]$ o comprimento de S^t encolhe de n' para $n' - k$, a perda do algoritmo vai ser $\frac{k}{n'}$ (dado que toda ação tem peso de $\frac{1}{n'}$. Por último, pode-se ver que podemos fazer uma aproximação de $\frac{k}{n'}$ por $\frac{1}{n'} + \frac{1}{(n'-1)} + \cdots + \frac{1}{(n'-k+1)}$. Isto gera uma perda para o intervalo $(t_j, t_{j+1}]$ de pelo menos:

$$\frac{1}{N} + \frac{1}{(N-1)} + \frac{1}{(N-2)} + \cdots + \frac{1}{1} \leq 1 + \ln N$$

Por indução pode-se ver que

$$L^t_{RG} \leq \left(\frac{1}{N} + \frac{1}{(N-1)} + \frac{1}{(N-2)} + \cdots + \frac{1}{(|S^t|+1)}\right) + (1 + \ln N) L^t_{\min}$$

□

Minimização do arrependimento externo

Algoritmo aleatório de peso maioritário

Uma ideia que surge naturalmente quando se pensa no algoritmo guloso aleatório é mudar as probabilidades para que não todas sejam iguais. Vamos associar para cada escolha i, como uma perda total L_i, um peso $w_i = (i-\eta)^{L_i}$, e uma probabilidade $p_i = \frac{w_i}{\sum_{j=1}^{N} w_j}$.

Aqui η é uma constante pequena e ajustável. No capítulo usam $\eta = 0.01$

Algoritmo aleatório de peso maioritário

Inicializa:

- $w_i^1 = 1$
- $p_i^1 = \frac{1}{N}$ para $i \in X$

Em cada momento t:

- se $\ell_i^{t-1} = 1$ então $w_i^t = w_i^{t-1}(1-\eta)$; se no então $w_i^t = w_i^{t-1}$
- $p_i^t = \frac{w_i^t}{W^t}$, onde $W^t = \sum_{i \in X} w_i^t$

Minimização do arrependimento externo
Algoritmo aleatório de peso maioritário

Uma ideia que surge naturalmente quando se pensa no algoritmo guloso aleatório é mudar as probabilidades para que não todas sejam iguais. Vamos associar para cada escolha i, como uma perda total L_i, um peso $w_i = (i-\eta)^{L_i}$, e uma probabilidade $p_i = \frac{w_i}{\sum_{j=1}^{N} w_j}$.

Aqui η é uma constante pequena e ajustável. No capítulo usam $\eta = 0.01$

Algoritmo aleatório de peso maioritário

Inicializa:

- $w_i^1 = 1$
- $p_i^1 = \frac{1}{N}$ para $i \in X$

Em cada momento t:

- se $\ell_i^{t-1} = 1$ então $w_i^t = w_i^{t-1}(1-\eta)$; se no então $w_i^t = w_i^{t-1}$
- $p_i^t = \frac{w_i^t}{W^t}$, onde $W^t = \sum_{i \in X} w_i^t$

Teorema

Para qualquer $\eta \leq \frac{1}{2}$, a perda do algoritmo aleatório de peso maioritário como uma sequência binária $\{0,1\}$ de perdas é:

$$L_{RWM}^{T} \leq (1+\eta) L_{min}^{T} + \frac{ln\ N}{\eta}$$

Definindo $\eta = min\{\sqrt{\frac{(ln\ N)}{T}}, \frac{1}{2}\}$, temos que

$$L_{RWM}^{T} \leq L_{min}^{T} + 2\sqrt{T\ ln\ N}$$

Limite Inferior

- Mas, quanto podemos baixar?

Teorema

Vamos assumir $T < log_2 N$. Então existe uma geração aleatória de perdas que, para qualquer algoritmo $R1$, gera uma perda esperada $E[L^T_{R1}] = \frac{T}{2}$, e ainda assim $L^T_{min} = 0$

Limite Inferior

- Mas, quanto podemos baixar?

Teorema

Vamos assumir $T < log_2 N$. *Então existe uma geração aleatória de perdas que, para qualquer algoritmo* $R1$, *gera uma perda esperada* $E[L_{R1}^T] = \frac{T}{2}$, *e ainda assim* $L_{min}^T = 0$

Limite Inferior

Demonstração.

Vamos considerar a seguinte sequência de perdas. No instante $t = 1$, vamos ter um subconjunto de $N/2$ ações como uma perda de 0, e o resto vai ter uma perda de 1. No instante $t = 2$, vamos escolher um subconjunto aleatório de $N/4$ ações (dos que no $t = 1$ tiveram uma perda de 0), que vão ter uma perda de 0 no instante $t = 2$ e ao resto damos uma perda de 1. Se repetimos o proceso, em cada passo qualquer algoritmo online vai ter uma perda esperada de $1/2$. Mas ainda assim, pode-se ver que pelo menos para $T < \log_2 N$, sempre existe uma ação como uma perda total de 0. □

Rota para pesquisas futuras

- O limite imposto para o arrependimento que foi presentado depende únicamente do número de passos dados T, e não depende do resultado da melhor ação. Isto é conhecido como limite de ordem zero. Pode-se encontrar outros limites:
  - *Primeiro Ordem:* Depende da perda da melhor escolha
  - *Segundo Ordem:* Depende da soma dos quadrados das perdas.
- Ter um arrependimento externo que seja proporcional à variança da melhor escolha.
- Problemas como um número de ações possíveis muito grande. Por exemplo, um problema onde as escolhas possíveis sejam todos os caminhos simples entre dois nós (combinatório)
- Analizar a dinâmica dos algoritmos de minimização de arrependimento.