Análise Computacional de Jogos Estocásticos Cooperativos e Não Cooperativos

Natan da Silveira Ferreira Nicolas da Silva Wischermann

Novembro de 2025

Parte 1 - Compreendendo o ambiente

Definição do Modelo e Matrizes de Payoff

A função `queuedefs()` define o modelo completo do jogo estocástico:

- **Parâmetros do sistema**: estados de canal (NS), tamanhos de buffer (NB), ações de potência (NA) e admissão (NB_adm)
- **Transições de canal**: Cadeia de Markov (PL) modelando qualidade do canal
- **Transições de fila**: Matriz P modelando dinâmica do buffer (chegadas/saídas)

Matrizes c1 e c2

As matrizes C_1 e C_2 representam, respectivamente, as recompensas dos jogadores 1 e 2, dadas pelo conjunto:

$$(\text{estado\_jogador\_1}, \text{ação\_jogador\_1}, \text{estado\_jogador\_2}, \text{ação\_jogador\_2})$$

E mudam conforme o modo:

- **Zero-Sum** (adversarial):

$$\begin{aligned} c1[i,a,j,b] &= \log_2(1+\text{SINR}_1) \\ c2[i,a,j,b] &= -c1 \quad \text{(ganho de um é perda do outro)} \end{aligned}$$

- **Non-Cooperative** (Nash):

$$\begin{aligned} c1[i,a,j,b] &= \log_2(1+\text{SINR}_1) \\ c2[i,a,j,b] &= \log_2(1+\text{SINR}_2) \quad \text{(cada um maximiza seu próprio throughput)} \end{aligned}$$

- **Cooperative** (Team):

$$c1 = c2 = \log_2(1+\text{SINR}_1) + \log_2(1+\text{SINR}_2) \quad \text{(maximizam throughput agregado)}$$

Solução via Programação Linear (Best-Response)

Para cada jogador $i \in \{1,2\}$, resolver o melhor-resposta contra a estratégia do oponente:

$$\max_{\rho_i} \mathbb{E}_{\rho_i,\rho_{-i}}[R_i] \quad \text{sujeito a:}$$

1. Equações de balanço de fluxo (distribuição estacionária):

$$\sum_{a,b} \rho_i(x,a,b) = \sum_{x'} P_L(x',x) \sum_{j',a',b'} \rho_i(x',a',b') P_Q(j',a',b',j)$$

2. Normalização (probabilidade):

$$\sum_{x,a,b} \rho_i(x,a,b) = 1$$

3. Limite de potência média:

$$\mathbb{E}_{\rho_i}[a] = \sum_{x,a,b} a \cdot \rho_i(x,a,b) \leq v_i$$

A função `queuestr_lp()` resolve o problema de um jogador contra a estratégia fixa do oponente:

- **Entrada**: estratégia (occupation measure) do oponente
- **Objetivo**: maximizar throughput esperado do jogador
- **Restrições**:
  - Equações de balanço de fluxo (1)
  - Limite de potência média (3)
  - Admissão válida apenas com buffer não-vazio
- **Saída**: occupation measure ótima ρ (probabilidade de estado-ação)

Algoritmo Iterativo de Best-Response

A função `queuesolvegame_py()` encontra o equilíbrio (Nash ou Team) iterativamente:

1. **Inicialização**: política heurística ou uniforme para ambos jogadores
2. **Iteração**: cada jogador calcula best-response contra estratégia atual do outro
3. **Convergência**: para quando valores (val1, val2) estabilizam ($\varepsilon < 10^{-10}$)
4. **Resultado**: equilíbrio de Nash (non-coop) ou solução de time (coop)

Nota: Corrigimos um bug de propagação do `CostFunction`; agora o best-response respeita corretamente os modos Zero-Sum, Non-Coop e Cooperative.

Experimentos: Comparação dos Três Modos de Jogo

Executamos o modelo com parâmetros idênticos em três configurações:

1. **Zero-Sum**: jogo adversarial (conflito total)
2. **Non-Cooperative**: equilíbrio de Nash (cada um por si)
3. **Cooperative**: problema de time (cooperação total)

Para cada modo, calculamos políticas ótimas de potência e admissão, e avaliamos o throughput físico agregado.

Resumo resultados iniciais

- **Zero-Sum** limita o throughput conjunto (valores simétricos e menores), como esperado pelo conflito total
- **Non-Coop** entrega throughput intermediário; cada jogador otimiza o próprio ganho
- **Cooperative** maximiza o throughput agregado, superando os outros modos

1. Descreva qualitativamente o significado das matrizes C_1 e C_2 e como elas mudam entre os modos.

Parte 2

O equilíbrio de Nash é formulado como um Problema de Complementaridade Linear (LCP): encontrar vetor $x \geq 0$ tal que $Mx + q \geq 0$ e $x^T(Mx + q) = 0$.

Definição de x, M e q

O vetor x concatena as estratégias (ρ) e os multiplicadores de Lagrange de igualdade (λ) e desigualdade (μ) para ambos os jogadores:

$$x = [\rho_1^T, \rho_2^T, \lambda_1^T, \lambda_2^T, \mu_1^T, \mu_2^T]^T$$

A matriz M e o vetor q unificam as condições de otimalidade e viabilidade:

$$M = \begin{pmatrix} 0 & -C_1 & E_1^T & 0 & K_1^T & 0 \\ -C_2 & 0 & 0 & E_2^T & 0 & K_2^T \\ -E_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -E_2 & 0 & 0 & 0 & 0 \\ -K_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -K_2 & 0 & 0 & 0 & 0 \end{pmatrix}, \quad q = \begin{pmatrix} \mathbf{0} \\ \mathbf{0} \\ \beta_1 \\ \beta_2 \\ L_1 \\ L_2 \end{pmatrix}$$

Onde os componentes representam:

- C_1, C_2: Matrizes de payoff (versão achatada dos tensores).
- E, β: E contém as equações de fluxo ($\delta - P$) e uma última linha preenchida com 1s para somar as probabilidades. $\beta = [0, \ldots, 0, 1]^T$ define os alvos: fluxo líquido nulo e probabilidade total unitária.
- K, L: K é a matriz de custos composta por duas linhas: custos de potência $d(x, a)$ e tamanhos de fila j. $L = [v, B_{\max}]^T$ contém os limites superiores.
- $\mathbf{0}$: Vetor nulo com dimensão igual ao número de variáveis, resultante da derivada parcial do Lagrangiano em relação a ρ.

Implementação Numérica

Nós tentamos resolver o LCP usando tanto a biblioteca `lemkelcp` quanto a `quantecon`, mas o algoritmo não convergiu em nenhum dos dois casos.

Parte 3

A inclusão da restrição $\mathbb{E}[\text{Backlog}] \leq B_{\max}$ impactou principalmente três componentes do código:

- **Classe `Settings`**: Adicionamos o atributo `BMAX: int` à classe `Settings`, podendo ser inicializado como por exemplo:

  ```
  BMAX: int = 10
  ```

  Este valor define o limite máximo permitido para o backlog médio.

- **Função `queuedefs(s)`**: A linha `BMAX = s.BMAX` foi adicionada ao dicionário `defs` retornado, garantindo que o valor seja repassado ao processo de otimização.

- **Função `queuestr_lp(for_player, rho_other, defs)`**: Esta foi a função mais impactada, pois formula e resolve o problema de Programação Linear. As modificações incluem:

  1. Recuperação de $B_{\max}$ através de `BMAX = defs['BMAX']`;
  2. Criação de um novo conjunto de coeficientes para A_{ub} e b_{ub}, que implementam a restrição linear:
     $$\sum j \cdot \rho[i, j, a, b] \leq B_{\max};$$
  3. Combinação dessa restrição com as já existentes utilizando `np.vstack` e `np.concatenate`;
  4. Impressão do valor real obtido do backlog médio (`actual_backlog`), possibilitando verificar a proximidade com o limite $B_{\max}$.

Análise comparativa em função de $B_{\max}$

Observa-se que, na ausência de restrição sobre o número médio de pacotes no buffer, as políticas de admissão tendem a sempre aceitar novos pacotes. Isso ocorre porque o eixo horizontal dos gráficos — que representa o estado do buffer — varia de 0 a $N_{\text{BufferState}} - 1$, o que corresponde a um recorte que não exibe explicitamente as rejeições que ocorreriam quando o buffer ultrapassa seu limite físico.

Ao introduzir a restrição sobre o número médio de pacotes no buffer por meio de $B_{\max}$, torna-se evidente o comportamento mencionado: o agente passa a rejeitar novos pacotes quando o backlog excede a média permitida por certo limiar.

Nas figuras a seguir, apresentamos as políticas de admissão considerando $B_{\max} = 1$ e $B_{\max} = 4$. Embora existam pequenas variações decorrentes das diferentes funções de perda utilizadas, observa-se claramente que a restrição sobre o número médio de pacotes impõe um limite inferior, em relação a $N_{\text{BufferState}}$, para a decisão de aceitar ou rejeitar novos pacotes, conforme esperado.

latex

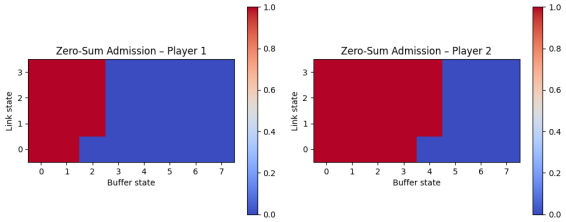

Figure 1: Zero-Sum com $B_{\max} = 1$

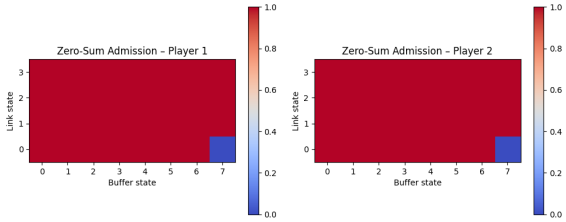

Figure 2: Zero-Sum com $B_{\max} = 4$

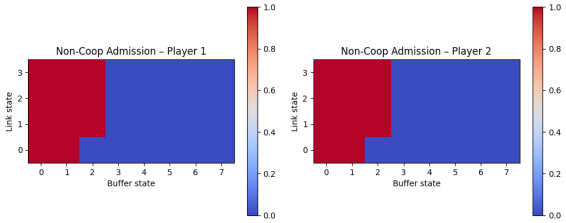

Figure 3: Non-Cooperative com $B_{\max} = 1$

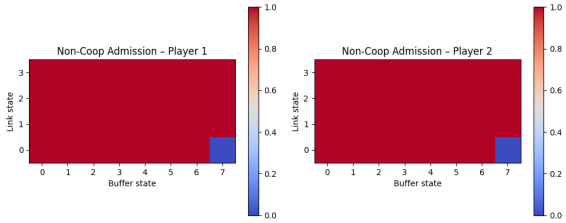

Figure 4: Non-Cooperative com $B_{\max} = 4$

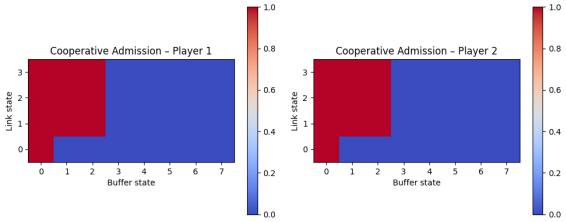

Figure 5: Cooperative com $B_{\max} = 1$

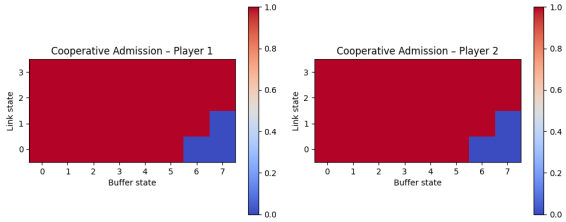

Figure 6: Cooperative com $B_{\max} = 4$

Conforme observado e comprovado por meio do notebook experimental, quando $B_{\max} > N_{\text{BufferState}}$, a solução degenera para o caso sem a restrição

$$\mathbb{E}[\text{Backlog}] \leq B_{\max}.$$

Esse comportamento é esperado, pois uma das variáveis passa a atuar como um limite inferior para o número de pacotes no buffer.

Em outras palavras, quando $B_{\max} < N_{\text{BufferState}}$, o valor de $B_{\max}$ impõe uma restrição efetiva sobre $N_{\text{BufferState}}$; de forma análoga, quando $B_{\max} > N_{\text{BufferState}}$, a restrição deixa de produzir efeito prático. Como o gráfico está expresso em função de $N_{\text{BufferState}}$, é possível observar diretamente as variações decorrentes da alteração de $B_{\max}$ nos resultados apresentados acima.

Já em relação ao throughput agregado x backlog médio, temos:

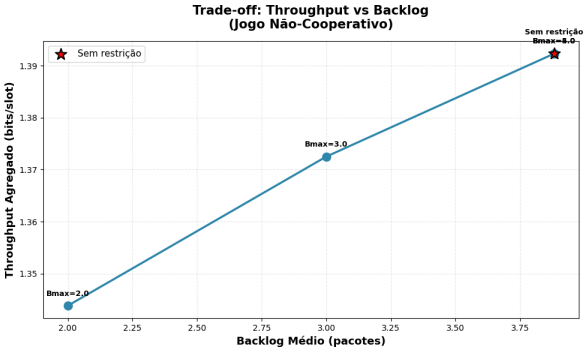

Figure 7: Trade-off: throughput x backlog médio

Parte 4 - Resumo e Conclusões da Análise Paramétrica

Investigamos como variações em parâmetros chave do sistema afetam o desempenho dos modos Cooperativo (Team) e Não-Cooperativo (Nash) em termos de throughput agregado e backlog médio, sob uma restrição de $B_{max} = 3.0$.

Probabilidade de Chegada (`ArrProb`)

- **Throughput:** Aumenta consistentemente com o `ArrProb` (de 1.11 para 1.42), indicando maior aproveitamento do sistema com mais demanda.
- **Backlog:** Permanece próximo ao limite $B_{max} = 3.0$, mostrando que a restrição é ativa.
- **Ganhos da Cooperação:** O modo cooperativo apresenta um throughput ligeiramente superior (ganhos modestos de 0.006 a 0.019) em comparação ao não-cooperativo.

Limites de Potência (`v1`, `v2`)

- **Throughput:** Aumentar os limites de potência eleva o throughput (de 1.20 para 1.47), pois os jogadores podem transmitir com mais energia.
- **Backlog:** Mantém-se no limite $B_{max} = 3.0$.
- **Ganhos da Cooperação:** Consistentes, mas pequenos (cerca de 0.008 a 0.01) sobre o caso não-cooperativo.

Número de Estados de Canal (`NLinkStates`)

- **Throughput:** Contrariando a intuição, um aumento nos `NLinkStates` (de 3 para 6) **reduziu** o throughput agregado (de 1.42 para 1.31).
- **Backlog:** Permanece em 3.0.
- **Interpretação:** Uma maior granularidade de estados não resultou em melhor desempenho sob as condições atuais (Bmax e potência), talvez devido à complexidade da política ou à forma como a capacidade é distribuída.
- **Ganhos da Cooperação:** Minúsculos, na ordem de milésimos.

4. Tamanho do Buffer (`NBufferStates`)

- **Backlog:** Para `NBufferStates`=6, o backlog ficou ligeiramente abaixo do limite (2.873), mas para tamanhos maiores (8 e 10), ele saturou em 3.0.
- **Throughput:** Variou pouco (aproximadamente 1.372 a 1.379).

- **Interpretação:** Quando a restrição de backlog é dominante, aumentar a capacidade do buffer tem impacto limitado, pois o sistema já opera para manter o atraso sob controle.

A restrição de backlog médio ($B_{max} = 3.0$) emerge como um fator crítico que domina o comportamento do sistema. O backlog real quase sempre satura nesse limite, o que leva a políticas de admissão mais conservadoras (rejeitando pacotes mais cedo) para evitar o acúmulo excessivo. Os ganhos de throughput são impulsionados principalmente por parâmetros que aumentam a capacidade intrínseca do sistema (como `ArrProb` e limites de potência).

O modo cooperativo é consistentemente superior ao não-cooperativo, mas a magnitude desses ganhos é reduzida quando a restrição de backlog é muito apertada. Isso ocorre porque a prioridade de controlar a fila limita a agressividade com que ambos os jogadores podem otimizar o throughput, independentemente de estarem cooperando ou não. Para observar diferenças mais acentuadas entre os modos, seria necessário relaxar a restrição de B_{max} ou explorar cenários com diferentes custos de potência e interferência.