Finding Matrix Multiplication Algorithms with Classical Planning

Fundamentos Lógicos da Inteligência Artificial

Autores

ESCRITORES DO ARTIGO

David Speck

Pesquisador de pós-doutorado na Universidade de Basel, Suíça

Paul Höft

Estudante de doutorado na Universidade de Linköping, Suécia

Daniel Gnad

Professor assistente na Universidade de Linköping, Suécia,

Jendrik Seipp

Professor associado na Universidade de Linköping, Suécia,

DAVID SPECK

Dr. David Speck é um pesquisador de pós-doutorado na Universidade de Basel, Suíça. Seu principal interesse de pesquisa está na área de inteligência artificial, com foco em planejamento automatizado. Essa área de estudo envolve a resolução de problemas para encontrar um curso de ação que permita a um agente inteligente mover-se de qualquer situação em que se encontre para uma que satisfaça seus objetivos.

Ele concluiu seu bacharelado em 2015 e seu mestrado em 2018 em ciência da computação na Universidade de Freiburg. Entre abril de 2018 e maio de 2022, atuou como funcionário científico na Universidade de Freiburg, na Cátedra de Fundamentos de Inteligência Artificial sob a supervisão do Prof. Dr. Bernhard Nebel. Dr. Speck obteve seu doutorado (Dr. rer. nat.) em fevereiro de 2022. De junho de 2022 a maio de 2024, fez parte do Laboratório de Raciocínio de Máquina como pesquisador de pós-doutorado na Universidade de Linköping, Suécia. Desde junho de 2024, integra a equipe do grupo de pesquisa em Inteligência Artificial na Universidade de Basel, Suíça.

Links

Publicações

Github

Últimas Conquistas

- Maio 2024 -> Venceu o prêmio de melhor dissertação no ICAPS 2024 pela dissertação Symbolic Search for Optimal Planning with Expressive Extensions
- Abril 2024 -> Prêmio de melhor Artigo no ICAPS 2024 com o trabalho Decoupled Search for the Masses: A Novel Task Transformation for Classical Planning
- Fevereiro 2024 -> Dois artigos aceitos no ICAPS 2024

PAUL HÖFT

Paul Höft é um estudante de doutorado na Universidade de Linköping, Suécia, onde trabalha na Divisão de Sistemas de Inteligência Artificial e Computação Integrada (AIICS) no Laboratório de Raciocínio de Máquina. Seus principais interesses de pesquisa são planejamento clássico ótimo e aprendizado de máquina. Ele se concentra em planejamento clássico ótimo e na combinação desta área com aprendizado de máquina, incluindo o desenvolvimento de heurísticas admissíveis e o avanço de algoritmos existentes através do aprendizado online e/ou da configuração dinâmica de algoritmos.

Paul concluiu seu bacharelado e mestrado em ciência da computação na Universidade de Basel em 2019 e 2021, respectivamente. Em setembro de 2021, ele se juntou ao Laboratório de Raciocínio de Máquina.

Links

Publicações

Github

Últimas Conquistas

- Vencedor da Trilha Determinística Ótima para o planejador Ragnarok na 10ª Competição Internacional de Planejamento IPC 2023 no ICAPS 2023.

DANIEL GNAD

Daniel Gnad é professor assistente na Divisão de Sistemas de Inteligência Artificial e Computação Integrada (AIICS) na Universidade de Linköping, Suécia. Ele trabalha com métodos de busca em espaço de estados, como a busca desacoplada, e no desenvolvimento de heurísticas independentes de domínio, como heurísticas red-black. Recentemente, ele se interessou pelo processo de grounding e desenvolve métodos de grounding parcial para mitigar a explosão exponencial, utilizando técnicas de aprendizado de máquina.

Daniel realizou seus estudos em ciência da computação na Universidade de Saarland. Após concluir seu mestrado, ele continuou como estudante de doutorado no grupo do Prof. Jörg Hoffmann. Em 2022, juntou-se ao RLPLab na Universidade de Linköping como pesquisador de pós-doutorado, onde se tornou professor assistente em 2023.

Links

Publicações

Github

Últimas Conquistas

- Prêmio Dr.-Eduard-Martin para a melhor dissertação da Faculdade de Matemática e Ciência da Computação em 2021, concedido pela Universidade de Saarland. Prêmio de Melhor Artigo SoCS 2022 no 15º Simpósio Anual sobre Busca Combinatória pelo artigo "Additive Pattern Databases for Decoupled Search".
- Prêmio de Melhor Dissertação ICAPS 2022 pela tese de doutorado "Star-Topology Decoupled State-Space Search in AI Planning and Model Checking" na 32ª Conferência Internacional sobre Planejamento e Agendamento Automatizados.

JENDRIK SEIPP

Jendrik Seipp é professor associado em Inteligência Artificial na Universidade de Linköping, Suécia, onde lidera o Laboratório de Raciocínio de Máquina na Divisão de Sistemas de Inteligência Artificial e Computação Integrada (AIICS). Seus principais interesses de pesquisa são o planejamento em IA e suas conexões com o aprendizado de máquina.

Jendrik recebeu seu mestrado em ciência da computação na Universidade de Freiburg, Alemanha, em dezembro de 2012. Em março de 2018, completou seu doutorado sob a supervisão do Prof. Malte Helmert com o grupo de Inteligência Artificial na Universidade de Basel, Suíça. Após isso, continuou como pós-doutorando até se tornar professor assistente na Universidade de Linköping, Suécia, em janeiro de 2021. Em setembro de 2023, foi promovido a professor associado.

Links

Publicações

Github

Últimas Conquistas

- 4x Primeiro Lugar, 2x Segundo Lugar no Segundo Desafio CoRe para o sistema PARIS 2023: Algoritmos de Planejamento para Reconfiguração de Conjuntos Independentes com Remo Christen, Salomé Eriksson, Michael Katz, Christian Muise, Florian Pommerening, Silvan Sievers e David Speck.
- Vencedor, Pista Ótima Determinística para o planejador Ragnarok com Dominik Drexler, Daniel Gnad, Paul Höft, David Speck e Simon Ståhlberg na 10ª Competição Internacional de Planejamento (IPC 2023) no ICAPS 2023.

QUAIS PROBLEMAS QUEREMOS RESOLVER COM MULTIPLICAÇÃO DE MATRIZ?

- Simulação temporal
- Física Quântica
- Computação Gráfica
- Aprendizado de Máquina
- Dentre outros...

Algoritmos de MM

Strassen, 1973
Multiplicando matrizes 2x2 com 7 operações de multiplicação.

Laderman, 1976

Blase, 2003

Heule, Kauers and Seidl, 2019

Fawzi, 2022

Algoritmo base de MM

$$\begin{bmatrix} A_1 & A_2 \\ A_3 & A_4 \end{bmatrix} \times \begin{bmatrix} B_1 & B_2 \\ B_3 & B_4 \end{bmatrix} = \begin{bmatrix} C_1 & C_2 \\ C_3 & C_4 \end{bmatrix}$$

- **8 operações de multiplicação no caso 2x2**
- **n^3 operações de multiplicação no caso nxn**

$$A_1 \times B_1 + A_2 \times B_3 = C_1$$
$$A_1 \times B_2 + A_2 \times B_4 = C_2$$
$$A_3 \times B_1 + A_4 \times B_3 = C_3$$
$$A_3 \times B_2 + A_4 \times B_4 = C_4$$

Algoritmo Strassen, 1973

$$\begin{bmatrix} A_1 & A_2 \\ A_3 & A_4 \end{bmatrix} \times \begin{bmatrix} B_1 & B_2 \\ B_3 & B_4 \end{bmatrix} = \begin{bmatrix} C_1 & C_2 \\ C_3 & C_4 \end{bmatrix}$$

- 7 operações de multiplicação no caso 2x2
- $n^{2.81}$ operações de multiplicação no caso nxn

$$(A_1 + A_4) \times (B_1 + B_4) = M_1$$
$$(A_3 + A_4) \times B_1 = M_2$$
$$A_1 \times (B_2 - B_4) = M_3$$
$$A_4 \times (B_3 - B_1) = M_4$$
$$(A_1 + A_2) \times B_4 = M_5$$
$$(A_3 - A_1) \times (B_1 + B_2) = M_6$$
$$(A_2 - A_4) \times (B_3 + B_4) = M_7$$

$$M_1 + M_4 - M_5 + M_7 = C_1$$
$$M_3 + M_5 = C_2$$
$$M_2 + M_4 = C_3$$
$$M_1 - M_2 + M_3 + M_6 = C_4$$

TENSORES

O que são tensores?

Tensores são objetos algébricos que descrevem relações multi lineares entre outros objetos algébricos e um espaço vetorial.

Definição 1: Podem ser vistos como arrays multi dimensionais.

Definição 2: É um elemento de um produto tensorial entre campos vetoriais.

Exemplos de Tensores

Escalar:

$$x = 2$$

Vetores:

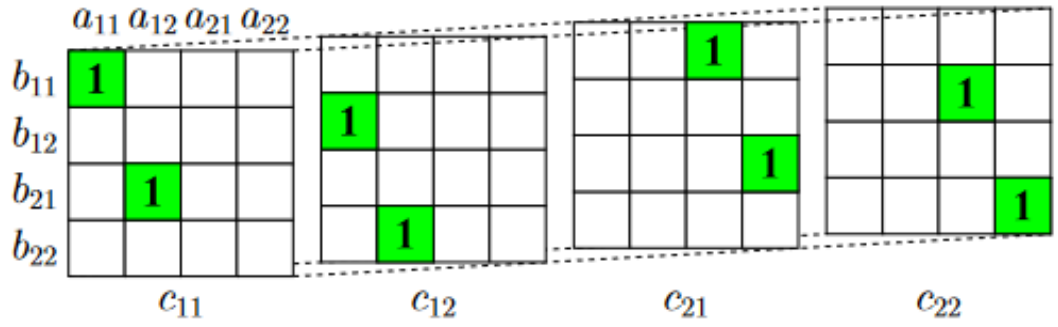
$$\vec{v} = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

Matrizes:

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

Tensor Tridimensional:

$$T_{2,2,2} =$$



Produto Tensorial

O produto tensorial $x \otimes y$ mapeia um par $(x, y), x \in V, y \in W$ para um elemento de $V \otimes W$.

Sejam $\vec{x}, \vec{y}$ vetores, e B_V, B_W as bases vetoriais de V, W respectivamente, o produto tensorial $\vec{x} \otimes \vec{y}$ será:

$$\vec{x} \otimes \vec{y} = \sum_{\vec{v} \in B_V} \sum_{\vec{w} \in B_W} x_v y_w (\vec{v} \otimes \vec{w})$$

Produto Kronecker

O produto Kronecker $A \otimes B$ é uma operação entre matrizes que resulta em uma nova matriz de dimensão $dim(A) \times dim(B)$.

Na prática, distribuímos a matriz B multiplicando cada elemento da matriz A.

$$\begin{bmatrix} a_{11} \\ a_{21} \end{bmatrix} \otimes \begin{bmatrix} b_{11} \\ b_{21} \end{bmatrix} = \begin{bmatrix} a_{11} \begin{bmatrix} b_{11} \\ b_{21} \end{bmatrix} \\ a_{21} \begin{bmatrix} b_{11} \\ b_{21} \end{bmatrix} \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} a_{11}b_{11} \\ a_{11}b_{21} \end{bmatrix} \\ \begin{bmatrix} a_{21}b_{11} \\ a_{21}b_{21} \end{bmatrix} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} \\ a_{11}b_{21} \\ a_{21}b_{11} \\ a_{21}b_{21} \end{bmatrix}$$

Tensorial x Kronecker

Considerando as bases:

$$B_V = \begin{bmatrix}1\\0\end{bmatrix}, \begin{bmatrix}0\\1\end{bmatrix} \qquad B_W = \begin{bmatrix}2\\3\end{bmatrix}, \begin{bmatrix}3\\2\end{bmatrix}$$

Temos os produtos Kronecker $\vec{v} \otimes \vec{w}$:

$$\begin{bmatrix}1\\0\end{bmatrix} \otimes \begin{bmatrix}2\\3\end{bmatrix} = \begin{bmatrix}2\\3\\0\\0\end{bmatrix} \qquad \begin{bmatrix}0\\1\end{bmatrix} \otimes \begin{bmatrix}2\\3\end{bmatrix} = \begin{bmatrix}0\\0\\2\\3\end{bmatrix}$$

$$\begin{bmatrix}1\\0\end{bmatrix} \otimes \begin{bmatrix}3\\2\end{bmatrix} = \begin{bmatrix}3\\2\\0\\0\end{bmatrix} \qquad \begin{bmatrix}0\\1\end{bmatrix} \otimes \begin{bmatrix}3\\2\end{bmatrix} = \begin{bmatrix}0\\0\\3\\2\end{bmatrix}$$

Tensorial x Kronecker

Formando a nova base:

$$B_{VW} = \begin{bmatrix} 2 \\ 3 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 3 \\ 2 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 2 \\ 3 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 3 \\ 2 \end{bmatrix}$$

Agora considerando os seguintes vetores:

$$\vec{x} = \begin{bmatrix} 5 \\ 1 \end{bmatrix}_{B_V} \qquad \vec{y} = \begin{bmatrix} 4 \\ 3 \end{bmatrix}_{B_W}$$

Tensorial x Kronecker

O produto tensorial $\vec{x} \otimes \vec{y}$ será:

$$\vec{x} \otimes \vec{y} = \begin{bmatrix} 5 \\ 1 \end{bmatrix}_{B_V} \otimes \begin{bmatrix} 4 \\ 3 \end{bmatrix}_{B_W}$$

Aplicando o produto Kronecker:

$$\vec{x} \otimes \vec{y} = \begin{bmatrix} 20 \\ 15 \\ 4 \\ 3 \end{bmatrix}_{B_{VW}}$$

DISCOVERING FASTER MATRIX MULTIPLICATION ALGORITHMS WITH REINFORCEMENT LEARNING

Alhussein Fawzi, 2022

- **Google Deep Mind**
- **Alpha Go -> Alpha Zero -> Alpha Tensor**
- **MM como uma decomposição de tensores**
- **Tensor Game**
- **Tensor Game com planejamento**

ALPHA GO

Programa de computador desenvolvido pela Google Deep Mind para jogar o jogo Go

Em outubro de 2015, em uma partida contra Fan Hui, o AlphaGo se tornou o primeiro programa de computador de Go a vencer um jogador profissional humano de Go

ALPHA GO ZERO

~ ALPHA ZERO

Aprendia sozinho sem ajuda de jogos com humanos

Versão generalizada para xadrez e shogi

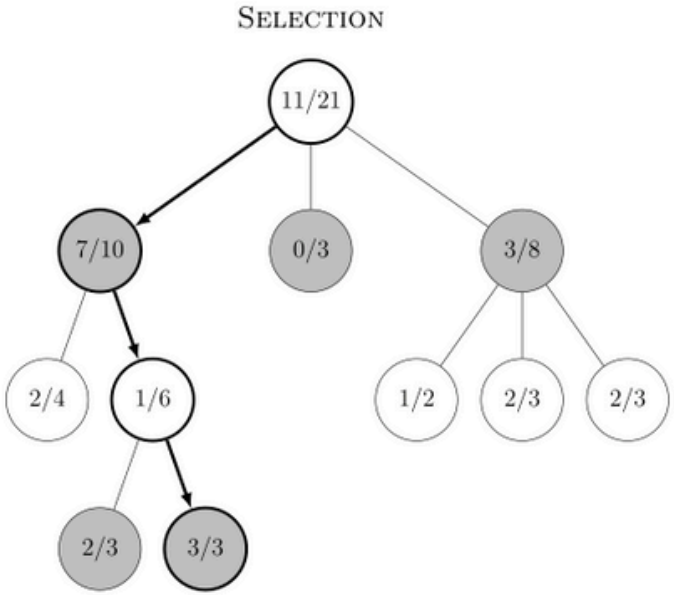
MONTE CARLO TREE SEARCH

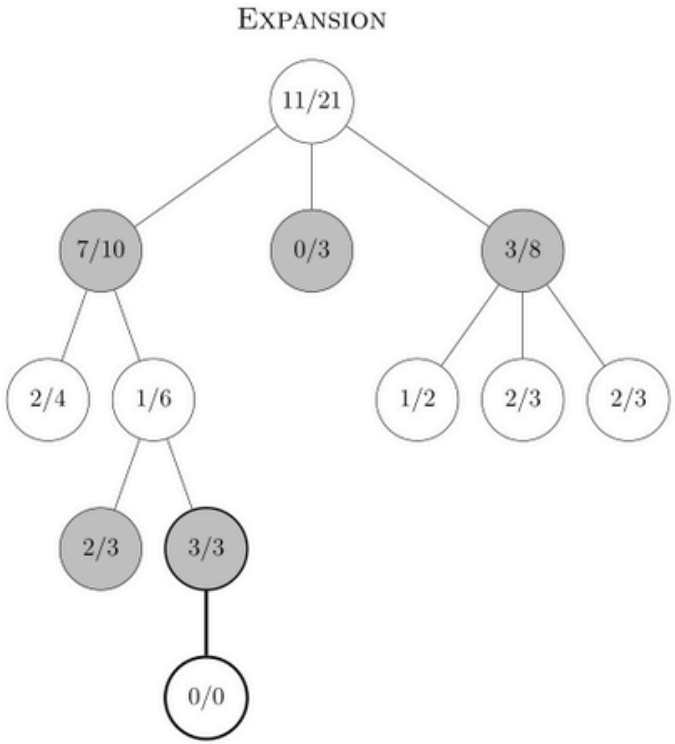
AlphaGo e seus sucessores usam um algoritmo de busca em árvore Monte Carlo para encontrar suas jogadas com base no conhecimento previamente adquirido por aprendizado de máquina, especificamente por uma rede neural artificial (um método de aprendizado profundo) através de treinamento extenso, tanto de jogos humanos quanto de computador.

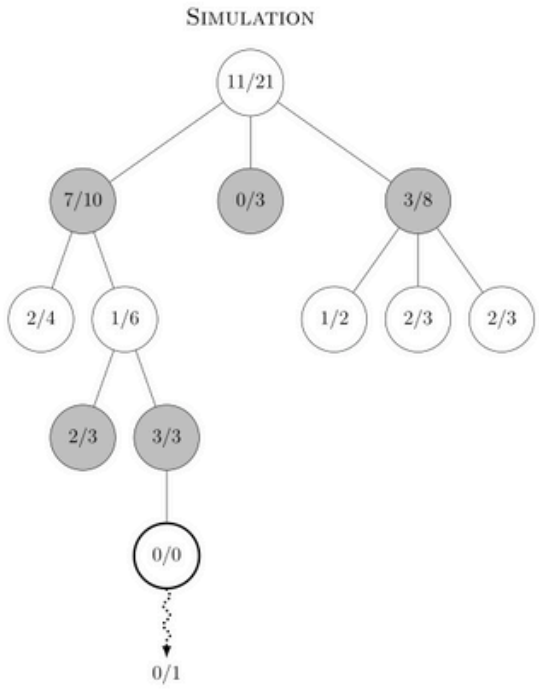
Heurística UCB1 = Upper Confidence Bound 1

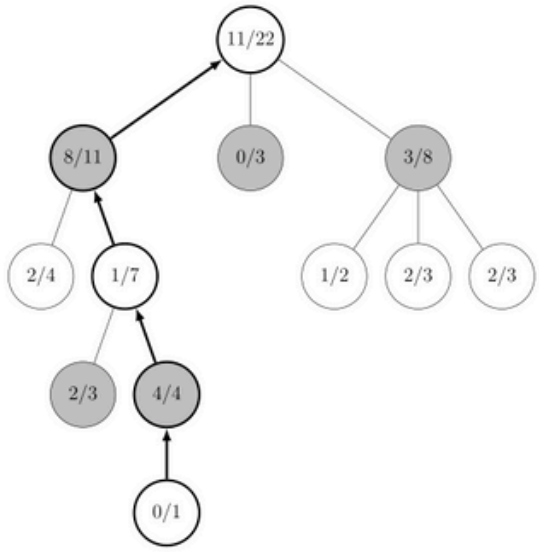
$$UCB1 = \left(\frac{w_i}{n_i}\right) + C\sqrt{\frac{\ln N}{n_i}}, C = 2$$

Rollout: Expande de forma pseudo-aleatória (quando temos mais de uma transição a partir do estado atual)

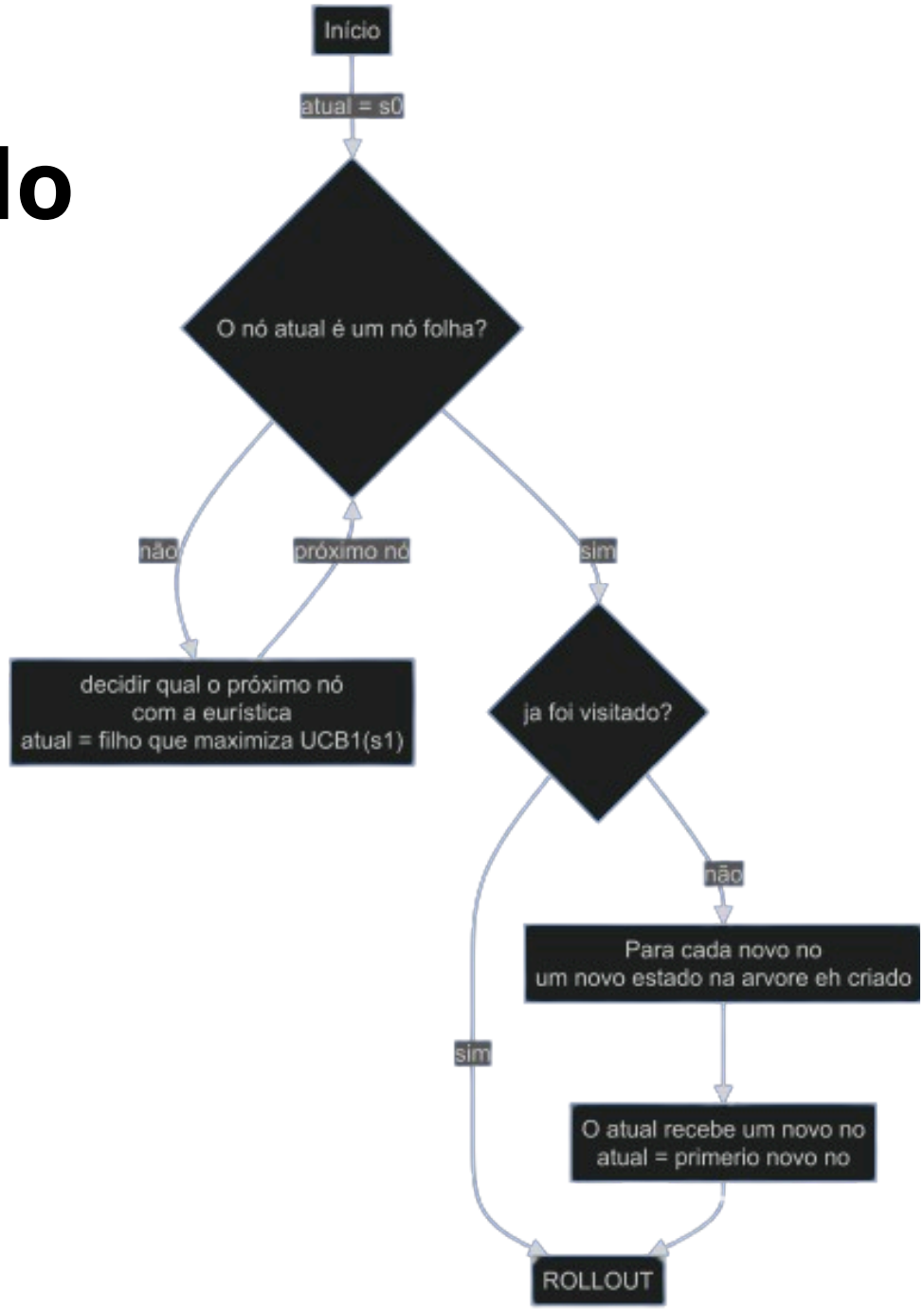

Selection
11/21
7/10
0/3
3/8
2/4
1/6
1/2
2/3
2/3
2/3
3/3


Expansion
11/21
7/10
0/3
3/8
2/4
1/6
1/2
2/3
2/3
2/3
3/3
0/0


Simulation
11/21
7/10
0/3
3/8
2/4
1/6
1/2
2/3
2/3
2/3
3/3
0/0
0/1


Backpropagation
11/22
8/11
0/3
3/8
2/4
1/7
1/2
2/3
2/3
2/3
4/4
0/1

Expansão do MCTS



ALPHA TENSOR

Responsável por descobrir novos algoritmos de MM

Aproveita as ferramentas que o ***AlphaZero*** usou para jogar Xadrez e Go para explorar o espaço de busca dos algoritmos.

O Alpha Tensor pode ser generalizado para busca de outros algoritmos além de MM

AlphaTensor se “auto treina” em um jogo chamado TensorGame, onde o jogador manipula um tensor de entrada de forma a resultar em um conjunto de instruções que representam um novo algoritmo de multiplicação.

MATRIX COMO UM TENSOR

Podemos representar uma matrix como um tensor de múltiplas dimensões

Exemplo: Seja **A** uma matrix ***i*x*j***, então

$$A[a_{ij}]$$

E T um tensor 2D, então podemos dizer que T é uma resesentação da matrix A como um Tensor.

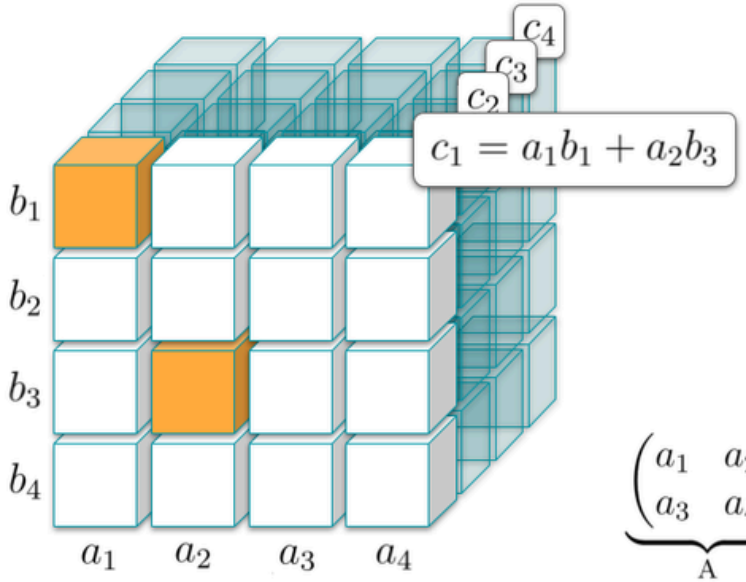
$$\tau[t_{ij}]$$

COMO REPRESENTAR UMA MM COMO UMA DECOMPOSIÇÃO DE TENSOR?

Vamos ilustrar o caso de uma *MM* matrix 2x2.

$$AB = C$$

***Os quadrados em laranja fazem referência a linha(Bi) e a coluna(Aj) que geram o coeficiente (Ck)**

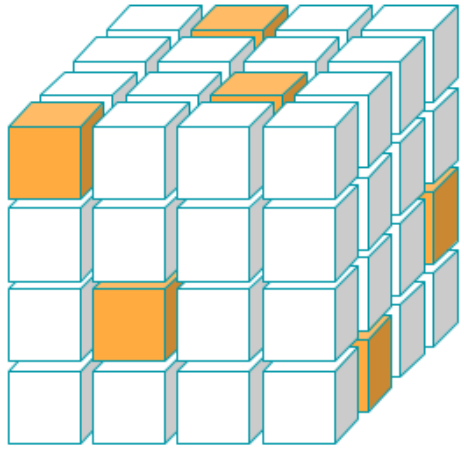


$$\underbrace{\begin{pmatrix} a_1 & a_2 \\ a_3 & a_4 \end{pmatrix}}_{\text{A}} \cdot \underbrace{\begin{pmatrix} b_1 & b_2 \\ b_3 & b_4 \end{pmatrix}}_{\text{B}} = \underbrace{\begin{pmatrix} c_1 & c_2 \\ c_3 & c_4 \end{pmatrix}}_{\text{C}}$$

DECOMPOSIÇÃO DE TENSORES

$$\mathcal{T} = \sum_{t=1}^{R} \mathbf{u}^{(t)} \otimes \mathbf{v}^{(t)} \otimes \mathbf{w}^{(t)}$$

R = Rank da decomposição



$$\mathcal{T} = \mathbf{u}^{(1)} \otimes \mathbf{v}^{(1)} \otimes \mathbf{w}^{(1)} + \cdots + \mathbf{u}^{(7)} \otimes \mathbf{v}^{(7)} \otimes \mathbf{w}^{(7)}$$

Rank-1 tensor

Cada fator **u⊗v⊗w** representa um passo de multiplicação do algoritmo.

DECOMPOSIÇÃO DE TENSORES

$$m_1 = (a_1 + a_4)(b_1 + b_4)$$
$$m_2 = (a_3 + a_4)\, b_1$$
$$m_3 = a_1\,(b_2 - b_4)$$
$$m_4 = a_4\,(b_3 - b_1)$$
$$m_5 = (a_1 + a_2)\, b_4$$
$$m_6 = (a_3 - a_1)(b_1 + b_2)$$
$$m_7 = (a_2 - a_4)(b_3 + b_4)$$

$$c_1 = m_1 + m_4 - m_5 + m_7$$
$$c_2 = m_3 + m_5$$
$$c_3 = m_2 + m_4$$
$$c_4 = m_1 - m_2 + m_3 + m_6$$

$$\mathbf{U} = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & -1 \end{pmatrix}$$

$$\mathbf{V} = \begin{pmatrix} 1 & 1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & -1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

$$\mathbf{W} = \begin{pmatrix} 1 & 0 & 0 & 1 & -1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & -1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Multiplicação de duas matrizes 2x2

TENSOR GAME

OBJETIVO DO JOGO

Dado qualquer tensor T, queremos encontrar uma decomposição de T como uma soma de R produtos tensoriais (como na slide anterior), onde R corresponde ao número de multiplicações no algoritmo, sendo o menor possível.

Como o Alpha Tensor obtém a menor decomposição possível?

TENSOR GAME

Como o Alpha Tensor obtém a menor decomposição possível?

- Rank-1 tensor
- Custo por decomposição
- Penalidades por decomposições ruins

TENSOR GAME COM PLANEJAMENTO

- Representamos um **tensor como um estado** e as ações representam como vamos mudar o tensor

PARSER

O parser, para solucionar as multiplicações, cria os domínios dinamicamente de acordo com a ordem das matrizes multiplicadas.

O número de elementos do tensor resultante será igual ao produto da quantidade de elementos das matrizes A, B e C, tal que:

$$AB = C$$

Em, resumo, se a ordem de A é n x m, e a ordem de B é m x p, por consequência, a ordem de C será n x p. Então, a quantidade de elementos x do tensor será:

$$x = (n \cdot m)(m \cdot p)(n \cdot p)$$

Logo, o nosso arquivo de problema terá que alocar x elementos para o tensor.

ENTRA E SAÍDA DO PLANO :D

```
a222021817@chococino:~/trabalho$ python3 create-pddl-task.py 2 2 2
Creating a (2x2)X(2x2) matrix multiplication tensor space of size 64
Name: mm2x2X2x2
Creating operators:
Created 3375 operators
```

criação de uma Matrix $(MxN)X(NxP) sendo, M = N = P = 2$

```
swiss-luidbook :: speck-et-al-icaps2023-code/benchmarks/generators » python3 create-pddl-task.py -h          2 ↵
usage: create-pddl-task.py [-h] [--choices CHOICES] [--combine-same-uv-to-block] [--avoid-conditional-effects]
                           [--zip-output] [-o OUTPUT]
                           sizes [sizes ...]

Create matrix multiplication pddl task

positional arguments:
  sizes                 matrix sizes, quadratic (1 argument) or (m x n) x (n x p) (3 arguments)
```

Explicação do uso

Bibliografia

[1] Wikipedia. Computational complexity of matrix multiplication. Disponível em: https://en.wikipedia.org/wiki/Computational_complexity_of_matrix_multiplication

[2] Wikipedia. Strassen Algorithm. Disponível em: https://en.wikipedia.org/wiki/Strassen_algorithm

[3] Laderman, Julian. A Noncomutative Algorithm for Multiplying 3 x 3 Matrices Using 23 Multiplications. 9 de outubro. 1975. Disponível em: https://www.ams.org/journals/bull/1976-82-01/S0002-9904-1976-13988-2/S0002-9904-1976-13988-2.pdf

[4] Wikipedia. Go game. Disponível em: https://en.wikipedia.org/wiki/Go_(game)

[5] Quanta Magazine. How AI Discovered a Faster Matrix Multiplication Algorithm. Disponível em: https://www.youtube.com/watch?v=fDAPJ7rvcUw&t=213s

[6] Ramponi, Marco. DeepMind's AlphaTensor Explained. 22 de novembro. 2022. Disponível em: https://www.assemblyai.com/blog/deepminds-alphatensor-explained/

Bibliografia

[7] Blaser, Markus. On the complexity of the multiplication of matrices of small formats. 2002. Disponível em: https://www.sciencedirect.com/science/article/pii/S0885064X02000079?ref=pdf_download&fr=RR-2&rr=89cab555da3d629b

[8] Fawzi, Alhussein. Discovering faster matrix multiplication algorithms. 14 de fevereiro. 2023 Disponível em: https://www.youtube.com/watch?v=WHuH4qSqoRs

[9] Eigenchris. Tensors for begginers. 2017-2018. Disponível em: https://www.youtube.com/playlist?list=PLJHszsWbB6hrkmmq57lX8BV-o-YIOFsiG