BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ TP. HCM

ĐỒ ÁN CƠ SỞ

NGHIÊN CỨU THUẬT TOÁN MONTE CARLO TREE SEARCH VÀ ỨNG DỤNG

Ngành: **CÔNG NGHỆ THÔNG TIN**

Chuyên ngành: **TRÍ TUỆ NHÂN TẠO**

Giảng viên hướng dẫn: PGS.TS. Võ Đình Bảy

Sinh viên thực hiện	Nguyễn Phạm Anh Quốc	Trần Lê Huy Tín
MSSV	2180603614	2180605273
Lớp	21DTHB5	21DTHB5

TP. Hồ Chí Minh, 2024

Mục Lục

Giới thiệu

AlphaGo là một chương trình máy tính được phát triển bởi DeepMind Technologies, một công ty con của Google. Chương trình này được thiết kế để chơi cờ vây, một trò chơi cổ xưa của Trung Quốc nổi tiếng với độ phức tạp và đòi hỏi chiến thuật cao. AlphaGo đã tạo ra sự đột phá lớn trong lĩnh vực trí tuệ nhân tạo (AI) khi nó đánh bại Lee Sedol, một trong những kỳ thủ cờ vây hàng đầu thế giới, trong một loạt trận đấu vào năm 2016. Chiến thắng này không chỉ là một cột mốc quan trọng đối với AI, mà còn mở ra những tiềm năng mới trong việc ứng dụng AI vào các lĩnh vực khác nhau. AlphaGo sử dụng một sự kết hợp giữa các mạng nơ-ron sâu và thuật toán Monte Carlo Tree Search (MCTS), tạo nên một phương pháp tiếp cận sáng tạo và hiệu quả trong việc ra quyết định trong các môi trường phức tạp và không chắc chắn.

Thuật toán Monte Carlo Tree Search (MCTS) là một phương pháp tìm kiếm thông minh được sử dụng để giải quyết các vấn đề lựa chọn trong môi trường phức tạp, đặc biệt là trong các trò chơi trí tuệ nhân tạo (AI). MCTS hoạt động bằng cách xây dựng một "cây" mô phỏng các trạng thái có thể xảy ra trong trò chơi từ một trạng thái ban đầu. Cây này được mở rộng bằng cách mô phỏng các nước đi ngẫu nhiên và chọn nước đi dẫn đến kết quả tốt nhất cho người chơi. MCTS không chỉ giới hạn trong cờ vây mà còn được áp dụng thành công cho nhiều trò chơi trí tuệ khác như cờ tướng, Go Fish. Ngoài trò chơi, MCTS còn được ứng dụng rộng rãi trong các vấn đề thực tế như logistics, lập kế hoạch và hệ thống gợi ý.

MCTS có nguồn gốc từ thuật toán "Multi-Armed Bandit" (MBA), vốn được sử dụng để tối ưu hóa việc chọn lựa máy đánh bạc có tỷ lệ chiến thắng cao nhất. Vào cuối những năm 1970, hai nhà khoa học Richard Coulombe và Jean-Pascal Pépin đã ứng dụng MBA vào lĩnh vực cờ vây và phát triển thuật toán MCTS ban đầu, sử dụng các mô phỏng ngẫu nhiên để khám phá các nước đi tiềm năng và đưa ra quyết định tối ưu nhất.

MCTS bao gồm bốn bước chính trong mỗi vòng lặp: Lựa chọn (Selection), Mở rộng (Expansion), Mô phỏng (Simulation), và Lan truyền ngược (Backpropagation). Các bước này giúp thuật toán xây dựng và tối ưu hóa cây tìm kiếm để đưa ra quyết định tốt nhất. MCTS dựa trên các mô hình toán học như Markov Decision Process (MDP) và Partially Observable Markov Decision Process (POMDP). Các lý thuyết về trò chơi và phương pháp Monte Carlo được tích hợp để nâng cao hiệu quả của thuật toán. MCTS không yêu cầu kinh nghiệm hay kiến thức chuyên sâu về lĩnh vực cụ thể, khiến nó trở thành thuật toán linh hoạt và mạnh mẽ. Thuật toán có thể dừng bất cứ lúc nào mà vẫn cung cấp thông tin đủ để đưa ra quyết định tiềm năng nhất. Sự cân bằng giữa "khai thác" (exploitation) và "thăm dò" (exploration) giúp tối đa hóa hiệu quả tìm kiếm và ra quyết định.

Bằng cách cân bằng giữa khám phá các chiến lược mới và khai thác các chiến lược hiệu quả đã biết, MCTS cung cấp phương pháp tiếp cận hiệu quả cho việc ra quyết định theo trình tự trong điều kiện không chắc chắn. Ưu điểm của MCTS là không cần kiến thức chuyên sâu về lĩnh vực cụ thể, khiến nó trở thành thuật toán linh hoạt và mạnh mẽ.

Các lý thuyết nền tảng tạo nên thuật toán Monte Carlo Tree Search

I. Markov Decision Process (MDPs)

- Markov Decision Process là một mô hình toán học dùng để giải quyết các vấn đề yêu cầu, đưa ra các quyết định tuần tự trong một môi trường mà chủ thể có thể quan sát được đầy đủ.
- Mô hình toán học MDP gồm 4 thành phần:
    - S: Một tập hợp các trạng thái với s0 là trạng thái ban đầu.
    - A: Một tập hợp các hành động.
    - T(s, a, s'): Là một mô hình chuyển đổi sẽ đưa ra xác suất để đạt được trạng thái s' nếu áp dụng hành động a cho trạng thái s.
    - R(s): là một hàm tính điểm.
- Mục tiêu của MDP là dựa trên mô hình nói trên để tìm ra một quy luật hay chiến lược đem lại phần thưởng cao nhất cho mỗi quyết định được đưa ra từ đó có thể đạt được mục đích mong muốn.

II. Partially Observation Markov Decision Process (POMDP)

- Khi chủ thể không thể quan sát đầy đủ môi trường xung quanh thì mô hình Partially Observable Markov Decision Process sẽ được áp dụng thay thế cho MDP.Tương tự MDP, POMDP cũng hướng đến việc tìm ra một chiến lược hay quy tắc giúp cho chủ thể đạt được mục đích mong muốn trong môi trường mà chủ thể không thể quan sát đầy đủ. Do thiếu thông tin về môi trường xung quanh nên quá trình đưa ra chiến lược tối ưu sẽ phức tạp hơn so với MDP.

III. Lý thuyết trò chơi

- Lý thuyết game được kế thừa từ lý thuyết quyết định trong trường hợp có nhiều tác nhân tương tác với nhau. Một trò chơi được định nghĩa là một tập hợp các quy tắc được thiết lập trước và một hay nhiều tác nhân sẽ thực hiện các quy tắc trên để tạo ra một kết quả. Một trò chơi bao gồm các thành phần sau:
    - S: là tập hợp các trạng thái của trò chơi, s0 là trạng thái bắt đầu.

- St C S: là một tập các trạng thái dẫn đến kết thúc trò chơi.
- N thuộc N: là số lượng người chơi.
- A: là tập hợp các hành động.
- F: S x A => S: là hàm chuyển đổi trạng thái.
- R: S => R^k: là hàm cho biết giá trị lợi ích (hiểu đơn giản là điểm) của từng người chơi tại trạng thái S.
- P: S => (0, 1, 2, …): là hàm xác định người chơi nào sẽ được hành động ở trạng thái S.

- Mỗi game sẽ bắt đầu từ trạng thái S0 và thay đổi trạng thái theo thời gian t = 1, 2, … cho đến khi đạt được một trạng thái St. Mỗi người chơi sẽ thực hiện một hành động, hành động này thông qua hàm f dẫn chuyển đổi từ trạng thái St thành St + 1. Sau mỗi lượt hành động, người chơi sẽ nhận được 1 điểm số được tính toán bằng hàm R. Giá trị của điểm số thông thường sẽ là 0 cho các trạng thái không phải St, 1 cho các trạng thái đưa chủ thể đến chiến thắng, -1 với trạng thái khiến chủ thể thua cuộc và 0 đối với các trạng thái có kết quả hòa .

IV. Multi Armed Bandit–Based Method

- Bài toán Multi Armed Bandit (MBA) đặt ra tình huống khi bạn có nhiều hành động khác nhau, mỗi hành động sẽ đem lại 1 phần thưởng có giá trị giao động trong một khoảng cố định. Bài toán yêu cầu bạn tối đa hóa phần thưởng thông qua việc chọn ra hành động mang lại phần thưởng cao nhất. Tuy nhiên bạn không biết về phần thưởng mà một hành động trực tiếp mang lại mà phải thông qua việc thống kê từ các lần mà hành động đó đã được thực hiện trong quá khứ.
- Để tối đa được phần thưởng ta cần cân bằng giữa 2 khái niệm “Khai thác” (Exploration) và “Thăm dò” (Exploitation).
  - Khai thác đề cập đến việc khi ta biết được đâu là hành động mang lại phần thưởng cao nhất, ta sẽ tiếp tục thực hiện lại hành động đó để tối đa phần thưởng đạt được. Khai thác giúp ta tận dụng hành động mang lại phần thưởng cao để có thể tối đa tổng phần thưởng đạt được.
  - Thăm dò đề cập đến việc ta sẽ thực hiện tất cả các hành động một số lần cho đến khi ta có đủ dữ kiện để tính toán xem đâu là hành động sẽ mang lại phần thưởng cao nhất. Thăm dò giúp tìm ra các hành động có tiềm năng sẽ mang lại phần thưởng cao hơn hành động đang khai thác.
- UCB1 là một trong các chiến lược giúp cân bằng “Khai thác” và “Thăm dò” một cách hiệu quả và nó cũng được áp dụng vào thuật toán MCTS. Trong chiến lược UCB, tại lần thứ t, ta sẽ chọn hành động có r có giá trị lớn nhất được tính theo công thức sau:

$$UCB(r) = \mu_r + \sqrt{\frac{2ln(t)}{N_t(r)}}$$

- μ_r là phần thưởng trung bình của hành động r.
- t là tổng số lượt chọn hành động.
- $N_t(r)$ là số lần mà hành động r đã được chọn cho đến thời điểm t.

- Phần μ_r của công thức đại diện cho sự "Khai thác". Khi ta chọn hành động có giá trị cao nhất được tính theo công thức trên đồng nghĩa với chọn ra hành động r có phần thưởng trung bình cao**.** Tuy nhiên giá trị trung bình không phải là yếu tố duy nhất quyết định hành động r có được chọn hay không.

- Phần $\sqrt{\frac{2ln(t)}{N_t(r)}}$ của công thức đại diện cho sự "Thăm dò" hiểu cách khác là mức khuyến cáo nên chọn hay không của hành động r. Trong bài toán MAB ta cần tránh việc chỉ chọn hành động có phần thưởng trung bình cao mà bỏ qua việc thăm dò các hành động khác. Nếu một hành động hiện đang có phần thưởng trung bình ở mức thấp được chọn đủ nhiều để có thêm dữ liệu cho việc ước lượng phần thưởng thì nó có thể trở thành hành động mang lại phần thưởng trung bình cao nhất.

  Theo công thức $\sqrt{\frac{2ln(t)}{N_t(r)}}$ khi một hành động r được chọn nhiều lần tức $N_t(r)$ tăng kéo theo giá trị của phần công thức trên giảm từ đó làm giảm giá trị của toàn bộ công thức cho dù phần thưởng trung bình μ_r đang cao. Điều này đồng nghĩa với độ khuyến cáo chọn hành động r thấp hơn so với các hành động khác vì các hành động khác được chọn ít hơn tức $N_t(r)$ nhỏ dẫn đến mức độ khuyến cáo chọn cao hơn. Từ đó chuyển từ trạng thái khai thác hành động r sang khám phá các hành động khác để tránh được tình huống nêu trên trong bài toán MAB. Một góc nhìn khác là nếu như một hành động chưa từng được lựa chọn hay $N_t(r) = 0$ sẽ dẫn đến giá trị của phần công thứcc trên là ∞ nên đảm bảo các node chưa được chọn sẽ được chọn ít nhất một lần tránh bỏ qua các node có tiềm năng mang lại phần thưởng cao.

V. Monte Carlo method

- Phương pháp Monte Carlo được sử dụng để tính toán các bài toán phức tạp mà các phép tính thông thường rất khó để tính toán và thường là các bài toán có liên quan đến nhiều biến số. Phương pháp Monte Carlo sử dụng máy tính để tạo ra một số lượng lớn các mô phỏng ngẫu nhiên (giá trị đầu vào ngẫu nhiên) của bài toán và dựa vào kết quả của các lần mô phỏng để ước lượng kết quả của bài toán. Độ chính xác của kết quả thu được phụ thuộc vào số lượng mô phỏng được thực hiện và độ biến thiên của các mô phỏng. Số lượng và độ biến thiên càng lớn thì kết quả sẽ càng chính xác.

Monte Carlo Tree Search

- Trong phạm vi bài viết, ta sẽ dùng thuật toán MCTS để đưa ra quyết định trong một trò chơi. Mục tiêu của thuật toán MCTS tính toán và đưa ra nước đi mang lại giá trị lớn nhất từ một trạng thái của trò chơi được cung cấp.

- Dựa trên mô hình MDP và lý thuyết trò chơi, thuật toán MCTS tạo ra một môi trường mô phỏng các trạng thái có thể xảy ra từ một trạng thái gốc được cung cấp. Ý tưởng của thuật toán dựa trên 2 quá trình.

  - Tính toán giá trị mà những hành động có thể thực hiện từ trạng thái gốc mang lại thông qua phương pháp mô phỏng Monte Carlo.
  - Dùng kết quả của mỗi hành động để tối ưu cũng như điều chỉnh quá trình tìm ra hành động thông qua chiến lược UBC1 đem lại giá trị lớn nhất cho trạng thái gốc được cung cấp.

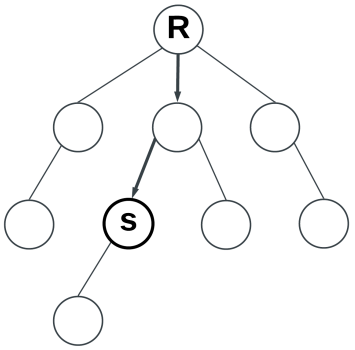
- Môi trường mô phỏng được biểu diễn bằng cấu trúc dữ liệu cây. Cây tìm kiếm Monte Carlo sẽ được xây dựng bằng việc lặp lại 2 quá trình nêu trên. Mỗi node trong cây biểu thị một trạng thái của trò chơi, các node con là các trạng thái khi thực hiện các hành động hợp lệ từ node cha.

I. Các bước trong 1 vòng lặp để xây dựng cây tìm kiếm Monte Carlo

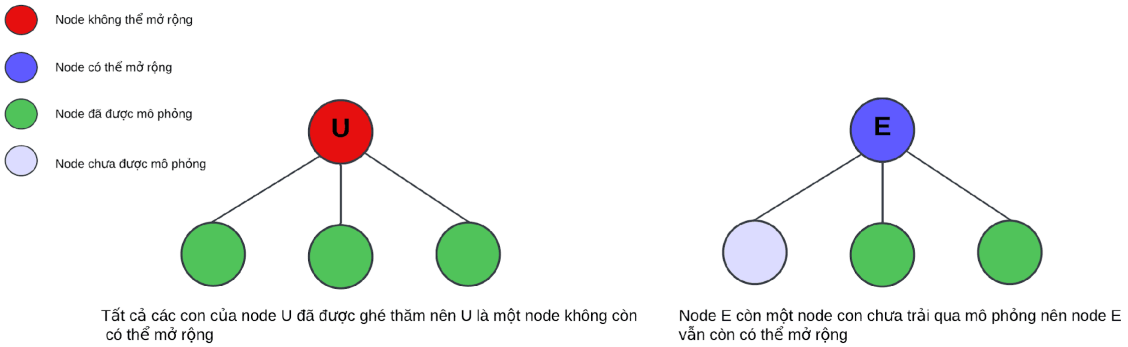
- Mỗi vòng lặp của thuật toán sẽ trải qua 4 bước:

1. Lựa chọn (Selection)

- Bắt đầu từ Node gốc R, duyệt qua toàn bộ các node trong cây theo chiến lược **UCT**(sẽ được đề cập sau) thể cho đến khi gặp được một node **S có thể mở rộng** hoặc node **T biểu diễn trạng thái kết thúc**, nếu node T được chọn tiến thẳng đến bước lan truyền ngược bỏ qua hai bước bên dưới.



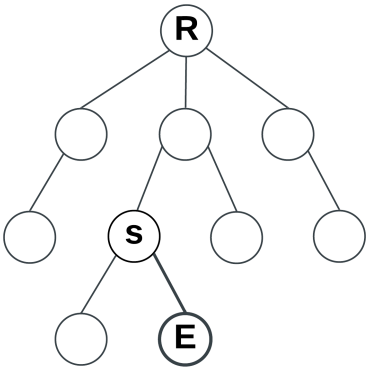
- Node biểu diễn trạng thái kết thúc (terminated node) là node mà nó biểu diễn trạng thái trạng thái đã phân định thắng thua hoặc trạng thái mà game không thể tiếp tục được chơi.
- Một node được coi là **có thể mở rộng** khi có một trong các node con của nó **chưa được ghé thăm.** Một node được coi là **chưa được ghé thăm** khi chưa có mô phỏng nào được thực hiện từ node đó hay nói cách khác là giá trị của node đó chưa được tính toán.



- Lưu ý rằng node lá (node chưa có con hay chưa được mở rộng thêm các trạng thái) cũng được xem là một node có thể mở rộng và cũng có thể được chọn trong quá trình lựa chọn.

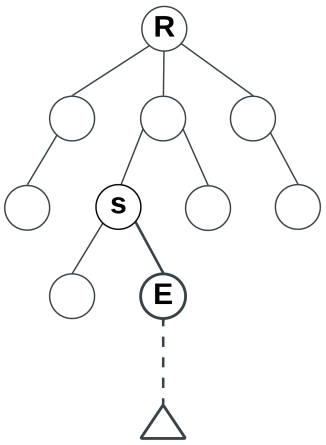
2. Mở rộng (Expand)

- Nếu node được chọn ở bước lựa chọn không phải là node biểu diễn trạng thái kết thúc thì tiến hành thêm một (hoặc nhiều) node con **E** dựa trên các nước đi hợp lệ từ node S.



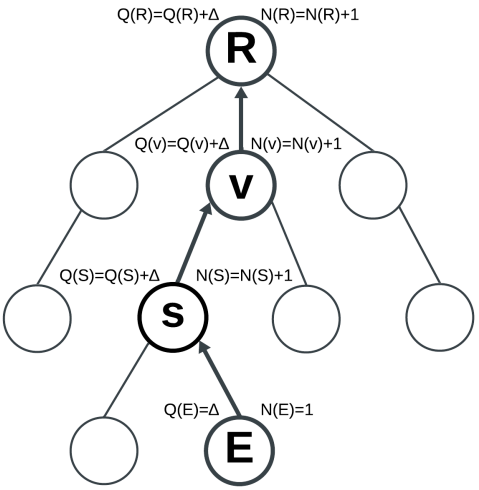
3. Mô phỏng (simulation hay rollout hoặc playout)

- Từ node **E** (nếu ở bước mở rộng thêm nhiều con thì chọn 1 node bất kì) được thêm vào ở bước mở rộng tiến hành mô phỏng Monte Carlo để ước lượng giá trị cho node đó. Mô phỏng trong ngữ cảnh này nghĩa là thực hiện một loạt các nước đi ngẫu nhiên hợp lệ cho phía bản thân và đối thủ bắt đầu từ node **E** cho đến khi trò chơi đạt trạng thái kết thúc và thu về kết quả **Δ.** Trong thực tế quá trình mô phỏng sẽ được thiết kế sao cho có thể hoàn tất trong thời gian ngắn nhất mà vẫn giữ được tính ngẫu nhiên trong các lần lựa chọn. Sau khi hoàn thành mô phỏng từ node **E** thì **E** sẽ trở thành node đã được ghé thăm.



4. Lan truyền ngược (Backpropagation)

- Kết quả Δ của node **E** sau khi được mô phỏng hoặc lấy từ node đạt trạng thái kết thúc trạng thái **T** sẽ được dùng để cập nhật lại số liệu cho các node trên đường đi từ node **E** ngược về node gốc **R**.
- Số liệu của một node bao gồm:
    - **Q(v)**: là tổng phần thưởng của của các mô phỏng có đi qua node v.
    - **N(v)**: là tổng số lần mà node v được duyệt qua.
- Với mỗi node **v** nằm trên đường lan truyền ngược từ **E** về node gốc **S**, cộng phần thưởng của Δ vào **Q(v),** cộng một cho **N(v).**



II. UCT (Upper Confidence Bound of Tree or UCB apply to Tree)

- Việc xây dựng cây tìm kiếm phụ thuộc phần lớn vào quá trình lựa chọn. Ta có thể xem việc mỗi lần thực hiện lựa chọn node là một bài toán Mulit Armed Bandit độc lập với giá trị của mỗi hành động là chính giá trị đang được ước tính của một node mang lại thông qua mô phỏng Monte Carlo. Để cân bằng sự "khai thác" và "thăm dò" việc lựa chọn node ta sẽ áp dụng chiến lược UCB1 đã được đề cập ở bài toán MAB. Node S được lựa chọn ở mỗi lần duyệt cây sẽ là node có giá trị lớn nhất được tính theo công thức:

$$UCT(S) = \frac{Q(S)}{N(S)} + 2C_p\sqrt{\frac{2\ln N(parent_S)}{N(S)}}$$

- Công thức trên được phát triển từ công thức UCB1 và có thêm hằng số C_p với $C_p > 0$. Cp có thể dùng để điều chỉnh mức độ "Thăm dò". Theo lý thuyết $C_p = 1/\sqrt{2}$ sẽ thõa mãn bất đẳng thức Hoffding với giá trị trong khoảng [0,1] (với 0 là thua và 1 là thắng). Đối với các phần thưởng nằm ngoài khoảng này thì cần điều chỉnh C_p để chiến lược UCT có thể hoạt động tốt hơn.

III. Chọn ra nước đi tốt nhất

- Bốn bước trên sẽ được lặp lại để mở rộng cây tìm kiếm cho đến khi đạt đến giới hạn. Giới hạn ở đây có thể là thời gian thực hiện, dung lượng bộ nhớ, ... hoặc đạt trạng thái kết thúc trong quá trình mô phỏng. Sau khi kết thúc việc xây dựng cây tìm kiếm, nước đi có số lần được duyệt nhiều nhất thường sẽ được chọn làm nước đi tốt nhất cho trạng thái gốc **R**.

IV. Mã giả của thuật toán MCTS

```
def tìm_kiếm_MCTS(trạng_thái_gốc):
    while còn_tài_nguyên():
        node_được_chọn = duyệt_cây(trạng_thái_gốc);
        node_mở_rộng = mở_rộng(node_được_chọn);
        kết_quả_mô_phỏng = mô_phỏng(node_mở_rộng);
        lan_truyền_ngược(node_mở_rộng, kết_quả_mô_phỏng);
    return tìm_con_tốt_nhất(trạng_thái_gốc)

def duyệt_cây(node):
    while có_thể_mở_rộng(node):
        node = max_UCT(node);
    return node;

def có_thể_mở_rộng(node):
    if not là_node_kết_thúc(node) or len(node.danh_sách_node_con_chưa_ghé_thăm) == 0:
        return False;
    return True:

def mở_rộng(node):
    node_mở_rộng = node.danh_sách_node_con_chưa_ghé_thăm.pop();
    node.danh_sách_node_con.append(node_mở_rộng);
    return node_mở_rộng;

def mô_phỏng(node):
    while not là_node_kết_thúc(node):
        node = chọn_ngẫu_nhiên(node.danh_sách_node_con_chưa_ghé_thăm);
    return kết_quả(node);

def lan_truyền_ngược(node, kết_quả):
    while node != None:
        node.kết_quả += kết_quả;
    node.số_lần_ghé_thăm += 1;
    node = node.node_cha
```

def tìm_con_tốt_nhất(node):

- Tìm node con được ghé thăm nhiều nhất (có thể có tìm theo các tiêu chí khác).

V. Một vài tính chất quan trọng của thuật toán MCTS

1. Không cần kinh nghiệm (Aheuristic)

- Khác với các thuật toán đưa ra quyết định trước đây chẳng hạn minimax alpha-beta, MCTS chỉ cần luật chơi hay cách thức hoạt động của vấn đề và không yêu cầu thêm bất cứ kiến thức cũng như kinh nghiệm nào khác về vấn đề nó đang giải quyết. Tuy nhiên thuật toán MCTS vẫn có thể tận dụng các kiến thức và kinh nghiệm về vấn đề cần giải quyết để cải thiện và hoạt động hiệu quả hơn. Đây là một trong các tính chất quan trọng khiến MCTS có thể áp dụng vào rất nhiều lĩnh vực và không chỉ dừng lại ở trò chơi.

2. Có thể dừng bất cứ lúc nào

- Ngay khi mỗi vòng lặp kết thúc dữ liệu của toàn bộ cây sẽ được cập nhật ngay lập tức thông qua quá trình lan truyền ngược. Điều này có nghĩa là thuật toán có thể dừng lại bất cứ khi nào muốn mà vẫn có đầy đủ thông tin để đưa ra được quyết định tiềm năng nhất. Tuy nhiên độ chính xác sẽ tỉ lệ thuận với số lần lặp, nếu dừng quá sớm thì kết quả cho ra có thể không được chính xác.

3. Hạn chế của MCTS

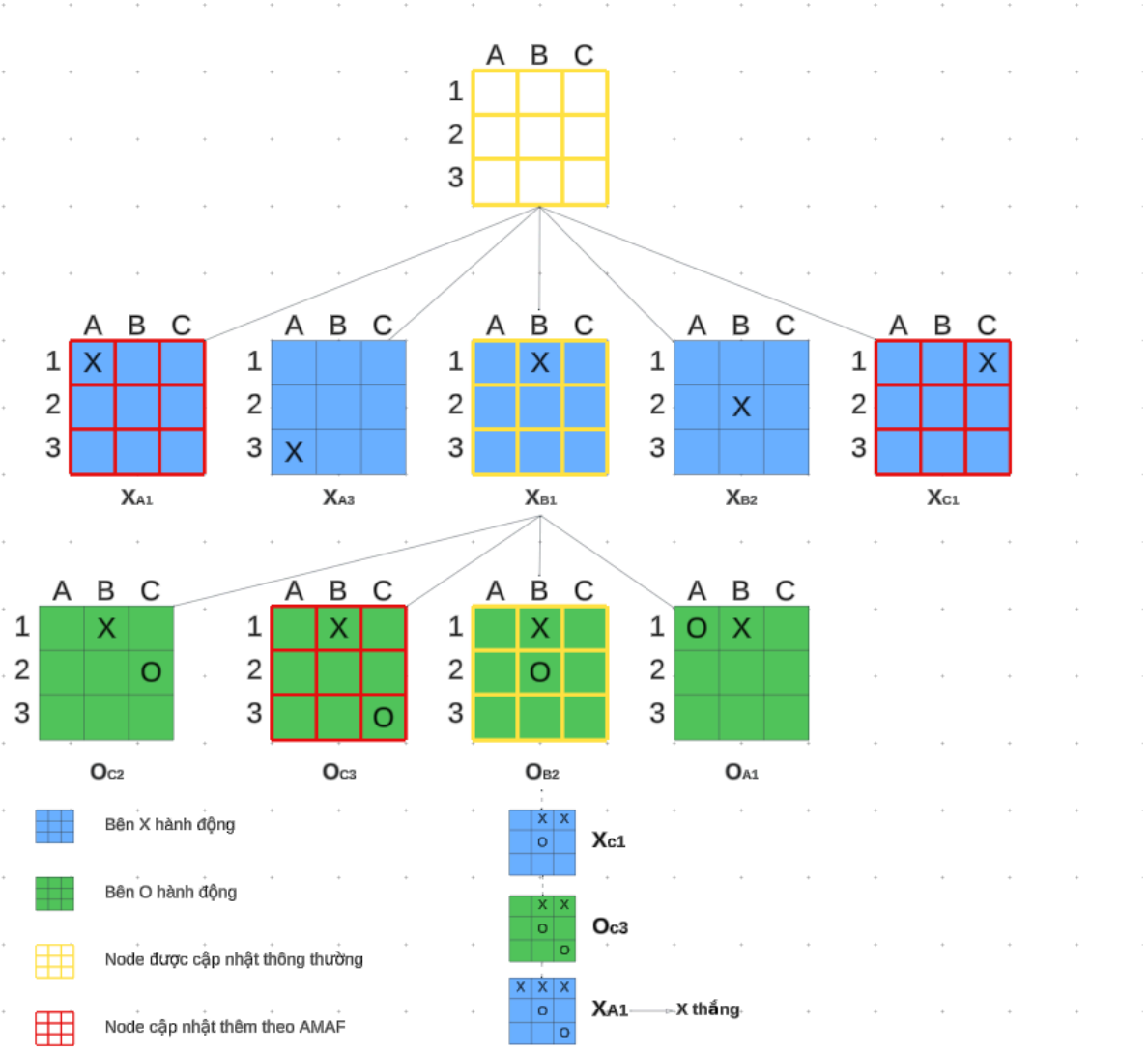
- Một điểm yếu của MCTS là với một vị trị nhất định, có thể sẽ có những nước đi trông rất tiềm năng nhưng thực sự nước đi "bẫy" đã được đối thủ tính toán trước để dành chiến thắng. MCTS chỉ chọn nước đi tiềm năng nhất trước mắt chứ không phân tích sâu các nước phía sau. Dựa trên tính chất ưu tiên nước đi tiềm năng nhất như nước "bẫy" nói trên vẫn sẽ được lựa chọn và đó là tiền đề cho một chuỗi các nước đi dẫn đến thua cuộc đã được đối thủ tính toán trước.

VI. Các cải tiến cho thuật toán MCTS

1. All Move As First (AMAF)

- AMAF là cơ chế dùng để nâng cấp độ chính xác của thuật toán MCTS sử dụng các dữ liệu trong quá khứ.
- Trong thuật toán MCTS, giá trị của một node chỉ đáng tin cậy nếu nó đã trãi qua mô phỏng đủ nhiều. Ở các vòng lặp đầu khi chưa có nhiều mô phỏng được thực hiện, việc chọn ra nước đi tiềm năng có thể nói gần như là ngẫu nhiên vì các thông tin của các node tại thời điểm này quá ít. Khi này cần một giải pháp để cung cấp thêm nhiều thông tin hơn cho các node và AMAF ra đời để giải quyết vấn đề trên.

- Ý tưởng của AMAF là mọi nước đi được thực hiện trong quá trình mô phỏng đều có thể được dùng để cung cấp thêm thông tin cho các node đang tồn tại trong cây tìm kiếm, nói cách khác AMAF thay đổi cách thức hoạt động trong bước lan truyền ngược của MCTS. Cụ thể, thay vì chỉ cập nhật dữ liệu cho các node trên đường từ node gốc R đến node có thể mở rộng E (được chọn thông qua bước lựa chọn), các node con C_i của các node N nằm trên đường đi từ node R đến E cũng sẽ được cập nhật dữ liệu nếu như hành động A tạo ra trạng thái con C_i từ node N được thực hiện trong quá trình mô phỏng node E.



- Khi áp dụng AMAF, mỗi node sẽ có thêm 2 thông tin là là Q_{AMAF} và N_{AMAF}. Q_{AMAF} là tổng phần thưởng nhận được thông qua quá trình cập nhật theo AMAF và N_{AMAF} là tổng số lần được cập nhật theo AMAF hay có thể hiểu N_{AMAF} là số lần mà hành động đó xuất hiện trong những lần mô phỏng. Khi này mỗi node sẽ có tổng 4 thông tin là Q_{UCT}, N_{UCT} cho quá trình cập nhật thông thường và Q_{AMAF}, N_{AMAF} cho qua trình cập nhật theo AMAF.

2. RAVE (Rapid Action Value Estimates)

- Sau khi có được thêm các thông tin từ quá trình cập nhật theo AMAF, các node được chọn trong bước lựa chọn phải đạt giá trị cao nhất được tính theo công thức RAVE:

$$RAVE = \alpha.\ AMAF + (1 - \alpha).\ UCT$$

- Giá trị RAVE khi này sẽ được dùng để quyết định node được lựa chọn thay vì UCT. Giá trị RAVE được tính toán bằng cách kết hợp giá trị UCT và giá trị AMAF. Giá trị UCT được tính toán theo công thức UCT. Giá trị AMAF được tính như sau:

$$AMAF = \frac{Q_{AMAF}}{N_{AMAF}}$$

- Theo công thức RAVE, một node có được lựa chọn hay không phụ thuộc vào giá trị UCT và giá trị AMAF. Số α ở đây thể hiện mức độ ảnh hưởng của hai giá trị UCT và AMAF lên giá trị RAVE hay nói cách khác là mức độ ảnh hưởng của hai giá trị UCT và AMAF lên quyết định liệu node có được lựa chọn hay không. α được tính như sau:

$$\alpha = \left\{0,\ \frac{C_{AMAF} - N_{UCT}}{C_{AMAF}}\right\}$$

- Theo công thức trên giá trị của α sẽ nằm trong khoảng từ 0 đến 1. C_{AMAF} cho biết số lần mà một node phải được ghé thăm theo chiến lược UCT (không tính các lần cập nhật AMAF) cho đến khi giá trị AMAF không còn ảnh hưởng đến giá trị cuối cùng của node đó. Giá trị C_{AMAF} thường nằm trong khoảng từ 200 đến 700 và có thể được tinh chỉnh tùy theo yêu cầu bài toán.
- Giả sử cho $C_{AMAF} = 200$ tức một node trước khi được lựa chọn lần thứ 200 thì giá trị của nó sẽ bị ảnh hưởng bởi giá trị AMAF. Tiếp tục giả sử như node hiện tại đã được ghé thăm 20 lần thì giá trị α lúc này sẽ là:

$$\alpha = max\left\{0,\ \frac{200-20}{200}\right\} = 0,9$$

- Theo công thức RAVE với α = 0.9, giá trị AMAF chiếm 90% giá trị của node, giá trị UCT sẽ chiếm 10% giá trị còn lại của node. Có thể thấy nếu như số lần được ghé thăm theo chiến lược UCT của một node tăng lên thì giá trị α càng gần về 0, giá trị α càng giảm tức mức độ ảnh hưởng của giá trị AMAF lên giá trị của node càng giảm, đến khi bằng α = 0 thì giá trị của một node hoàn toàn tính theo giá trị UCT đồng nghĩa với node đã được ghé thăm đủ nhiều và giá trị của nó đã đáng tin cậy, không còn cần thêm thông tin từ quá trình cập nhật AMAF.

3. Last Good Reply Policy

- LGRP là một chiến lược dành cho người chơi trong quá trình mô phỏng, diễn ra bên trong thuật toán Monte Carlo Tree Search (MCTS). Trong quá trình mô phỏng, LGRP "ghi nhớ" các nước đi thành công dẫn đến kết quả tốt cho người chơi. LGRP giúp MCTS khám phá các phần hứa hẹn hơn của cây tìm kiếm và đưa ra quyết định tốt hơn cho người chơi
- LGRP (Last Good Reply Policy) đóng vai trò quan trọng trong việc **cải thiện hiệu quả tìm kiếm** các nước đi tốt nhất trong trò chơi. Cụ thể, LGRP mang lại những lợi ích sau:

- Tăng tốc độ tìm kiếm: LGRP giúp MCTS tập trung vào những vị trí có khả năng dẫn đến chiến thắng cao hơn, dựa trên các nước đi tốt nhất đã được tìm thấy trước đây.
- Nâng cao độ chính xác: LGRP giúp MCTS tránh lặp lại những sai lầm đã mắc phải trong quá khứ, từ đó đưa ra những quyết định chính xác hơn.
- Cân bằng giữa khám phá và khai thác: LGRP giúp MCTS cân bằng giữa việc khám phá những nước đi mới tiềm năng và khai thác những nước đi đã được chứng minh là hiệu quả.

Cách hoạt động:

- Học hỏi từ thành công: LGRP theo dõi các nước đi dẫn đến phần thưởng cao (thắng hoặc vị trí tốt) trong các lần mô phỏng trước đó.
- Ưu tiên thành công trước đó: Khi cần thực hiện một nước đi trong mô phỏng, LGRP kiểm tra trí nhớ của nó để xem liệu có một nước đáp trả (nước đi tốt) nào cho nước đi gần đây của đối thủ không.
- Thực hiện nước đáp trả tốt: Nếu tìm thấy nước đáp trả phù hợp và hợp lệ thì sẽ ưu tiên thực hiện nước đi đó.
- Chiến lược dự phòng: Nếu không tìm thấy nước đáp trả nào trong bộ nhớ, LGRP sử dụng một chiến lược riêng biệt (ví dụ: chọn ngẫu nhiên) để chọn nước đi.

Xác định vị trí lưu trữ nước đi hồi đáp tốt nhất:

- Để xác định vị trí lưu trữ nước đi hồi đáp tốt nhất, LGRP sử dụng số thứ tự vai trò của người chơi (0, 1, 2, ..., n-1) và tổng số người chơi (n).

Hai bảng cho mỗi người chơi:

- LGRP tạo ra hai bảng riêng biệt cho mỗi người chơi (ký hiệu là i).
- Bảng thứ nhất: Lưu trữ nước đi hồi đáp tốt nhất cho nước đi trước đó của người chơi có số thứ tự vai trò (i + n - 1) mod n.
- Bảng thứ hai: Lưu trữ nước đi hồi đáp tốt nhất cho **hai** nước đi trước đó.
    - Dãy hai nước đi này bắt đầu bằng nước đi của người chơi có số thứ tự vai trò (i + n - 2) mod n và kết thúc bằng nước đi của người chơi có số thứ tự vai trò (i + n - 1) mod n. Với điều kiện là hai nước đi này phải diễn ra **liền tiếp** nhau.
    - Ví dụ:

- Nếu có 3 người chơi, bảng thứ hai sẽ lưu trữ nước đi hồi đáp tốt nhất cho cặp nước đi trước đó của người chơi 0 và 1, và cặp nước đi trước đó của người chơi 1 và 2.
- Điều này có nghĩa là:
  - Cặp nước đi thứ nhất sẽ bắt đầu bằng nước đi của người chơi 0 và kết thúc bằng nước đi của người chơi 1.
  - Cặp nước đi thứ hai sẽ bắt đầu bằng nước đi của người chơi 1 và kết thúc bằng nước đi của người chơi 2.

Cập nhật bảng sau mỗi lần mô phỏng:

- Bảng LGRP được cập nhật sau mỗi lần mô phỏng một ván cờ.
- Quy tắc cập nhật:
  - Nếu người chơi i thắng ván cờ, nước đi của người chơi i được lưu trữ trong bảng như là nước đi hồi đáp tốt nhất cho nước đi trước đó (hoặc hai nước đi trước đó).
  - Nếu người chơi i thua, *bất kỳ* nước đi nào của người chơi i đã được lưu trữ trước đó sẽ bị xóa khỏi bảng.

Ví dụ mô phỏng:

- **Bảng 1a:** Lưu trữ nước đi hồi đáp tốt nhất cho nước đi trước đó của người chơi có số thứ tự vai trò (i + n - 1) mod n. Nếu người chơi X đi nước đi trước đó là X3, thì nước đi hồi đáp tốt nhất cho người chơi O là O1

Nước đi trước đó của X	Nước đi phản hồi tốt nhất O
X3	O1
X4	O2

- **Bảng 1b:** Nếu người chơi O đi nước đi trước đó là O2, thì nước đi hồi đáp tốt nhất cho người chơi X là X2.

Nước đi trước đó của O	Nước đi phản hồi tốt nhất của X
O2	X2
O1	X4

- **Bảng 2a:** Lưu trữ nước đi hồi đáp tốt nhất cho hai nước đi trước đó. Nếu 2 nước đi trước đó là (X3,O4) thì nước đi hồi đáp tốt nhất cho người chơi X là X2.

Nước đi trước đó của X	Nước đi phản hồi tốt nhất của X
X3, O4	X2
X4, O5	X6

- **Bảng 2b:** Nếu 2 nước đi trước đó là (O3, X4) thì nước đi hồi đáp tốt nhất cho người chơi O là O2.

Nước đi trước đó của X	Nước đi phản hồi tốt nhất O
O3, X4	O1
O4, X5	O2

Điều chỉnh tỷ lệ LGRP trong MCTS

- **Xác định tỷ lệ khởi tạo:** Bắt đầu với tỷ lệ LGRP ban đầu (ví dụ: 0.7), nghĩa là 70% thuật toán sẽ chọn đề xuất của LGRP và 30% sẽ chọn ngẫu nhiên.
- **Theo dõi hiệu suất:** Ghi lại tỷ lệ chiến thắng của thuật toán khi chơi với các đối thủ khác nhau. Phân tích hiệu suất theo các giai đoạn khác nhau của trò chơi (ví dụ: giai đoạn đầu, giai đoạn giữa, giai đoạn cuối).
- **Điều chỉnh tỷ lệ LGRP:** Nếu tỷ lệ chiến thắng thấp, tăng tỷ lệ LGRP (ví dụ: tăng lên 0.8) để thuật toán tập trung hơn vào các đường đi có khả năng chiến thắng cao. Nếu tỷ lệ chiến thắng cao nhưng thuật toán dễ đoán trước, giảm tỷ lệ LGRP (ví dụ: giảm xuống 0.6) để tăng tính ngẫu nhiên và khả năng thích ứng. Điều chỉnh tỷ lệ LGRP một cách từ từ và theo dõi hiệu suất sau mỗi thay đổi.
- **Cân nhắc các yếu tố khác:** Ngoài tỷ lệ chiến thắng, cần xem xét các yếu tố khác như thời gian chơi, v.v. Điều chỉnh tỷ lệ LGRP để đạt được sự cân bằng giữa hiệu quả, tính ngẫu nhiên và các yếu tố quan trọng khác.

Lưu ý:

- Việc điều chỉnh tỷ lệ LGRP là một quá trình thử nghiệm và điều chỉnh.
- Không có giá trị tỷ lệ LGRP tối ưu cho tất cả các trò chơi và mục tiêu.
- Cần theo dõi và đánh giá hiệu suất liên tục để tìm ra tỷ lệ phù hợp nhất.

Ứng dụng của Thuật toán Monte Carlo Tree Search

I. Ứng dụng trong Robotics

MCTS có thể được sử dụng trong lĩnh vực robotics để tối ưu hóa các hành động của robot trong môi trường không chắc chắn. Robot có thể sử dụng MCTS để quyết định hành động tiếp theo dựa trên các mô phỏng của các trạng thái môi trường khác nhau và chọn hành động mang lại kết quả tốt nhất.

II. Ứng dụng trong Tối ưu hóa Đầu Tư Tài Chính

MCTS có thể được áp dụng để xây dựng các chiến lược đầu tư tài chính. Bằng cách mô phỏng các kịch bản thị trường khác nhau và phân tích kết quả của các quyết định đầu tư, MCTS giúp tối ưu hóa danh mục đầu tư nhằm đạt được lợi nhuận cao nhất.

III. Ứng dụng trong Lập Kế Hoạch Đô Thị

Trong lĩnh vực lập kế hoạch đô thị, MCTS có thể được sử dụng để tối ưu hóa việc phân bổ không gian, quản lý giao thông và cải thiện quy hoạch xây dựng. Các nhà quy hoạch có thể sử dụng MCTS để mô phỏng các phương án quy hoạch khác nhau và chọn ra phương án tối ưu nhất.

IV. Ứng dụng trong Dược Phẩm

Trong nghiên cứu dược phẩm, MCTS có thể được sử dụng để tối ưu hóa quy trình phát hiện và phát triển thuốc. Bằng cách mô phỏng các phản ứng hóa học và sinh học, MCTS giúp các nhà nghiên cứu xác định các hợp chất tiềm năng có thể phát triển thành thuốc hiệu quả.

V. Ứng dụng trong Quản lý Rủi Ro

MCTS cũng có thể được áp dụng trong lĩnh vực quản lý rủi ro, giúp các tổ chức phân tích và tối ưu hóa các chiến lược quản lý rủi ro. Bằng cách mô phỏng các kịch bản rủi ro khác nhau, các nhà quản lý có thể xác định các biện pháp phòng ngừa hiệu quả nhất.

VI. Ứng dụng MCTS trong Quản lý Chuỗi Cung Ứng

Trong bối cảnh quản lý chuỗi cung ứng, việc phân bổ hàng tồn kho là một vấn đề quan trọng, đặc biệt khi nhà cung cấp phải tuân thủ các điều khoản của Service Level Agreement (SLA). SLA là hợp đồng giữa nhà cung cấp và nhà bán lẻ, trong đó nhà cung cấp cam kết cung cấp một mức độ dịch vụ nhất định cho nhà bán lẻ .

Áp dụng MCTS vào thỏa thuận mức độ dịch vụ (SLA)

I. Service Level Agreement (SLA)

SLA là một thỏa thuận chính thức, có ràng buộc pháp lý giữa nhà cung cấp dịch vụ và khách hàng. Mục tiêu của SLA là thiết lập kỳ vọng rõ ràng, đảm bảo tính minh bạch và trách nhiệm giải trình, cung cấp cơ sở pháp lý để giải quyết tranh chấp giữa hai bên. SLA bao gồm nhiều yếu tố như số lượng hàng hóa mong muốn, chất lượng, trách nhiệm của nhà cung cấp, thời gian phản hồi…đã được 2 bên thỏa thuận.

Môt số thành phần chính cần quan tâm của một SLA bao gồm:

- Nhu cầu của khách hàng: số lượng hàng mà khách mong muốn
- Thời gian xem xét hiệu suất (Performance Review Period) (PRP): là khoảng thời gian mà hiệu suất (tỉ lệ lập đầy) của nhà cung cấp được đo lường và đánh giá với các mục tiêu dã thỏa thuận. Ví dụ PRP là 5 ngày thì hiệu suất sẽ được tính toán tại ngày thứ 5, 10, 15
- Tỉ lệ lấp đầy (fill rate): là thước đo hiệu suất cho biết có bao nhiêu phần trăm đơn hàng của khách hàng được đáp ứng ngay lập tức với lượng hàng hiện có trong kho
    - Target fill rate: là hiệu suất mà khách hàng yêu cầu phải đạt ở mỗi lần giao hàng
    - Real fill rate: là hiệu suất thực tế mà nhà cung cấp đáp ứng được
- Các biện pháp khắc phục: là các biện pháp mà nhà cung cấp sẽ thực hiện nếu như không đáp ứng đúng các mục tiêu đã thỏa thuận

Trong đó thời gian xem xét hiệu suất và tỉ lệ lấp đầy ảnh hưởng trực tiếp đến hiệu suất tổng thể của hợp đồng.

Việc đề ra một chính sách phân bố hàng hóa hợp lí là điểm mấu chốt để quản lí hiệu quả SLA. Chính sách phân bố sẽ chỉ ra các quy luật cần tuân theo và hướng dẫn phân bố hàng hóa dựa trên nhu cầu của khách hàng đảm bảo đạt được hiệu suất đã thỏa thuận. Việc có một chính sách phân bố hợp lí sẽ cân bằng được chi phí hàng tồn kho và sự hài lòng của khách, giảm thiểu các mức phạt phải nhận đồng thời giúp nhà cung cấp phân bố hàng hóa linh hoạt với sự thay đổi nhu cầu của khách hàng trong thực tế.

II. Hạn chế của các chính sách phân bố hàng hóa không có chiến lược cụ thể

Đối với các chính sách phân bố không có chiến lược cụ thể, hàng hóa phân bổ cho các khách hàng được định trước và không thay đổi theo thời gian. Chính sách phân bố này chỉ tập trung vào nhu cầu hiện tại, ưu tiên lợi nhuận trong ngắn hạn mà bỏ qua các yếu tố khách có thể ảnh hưởng đến lợi nhuận trong tương lai.

Giả sử như bởi sự hạn chế về thời gian, chất lượng, chi phí của việc sản xuất cũng như vẫn chuyển hàng hóa, kho hàng của một nhà cung cấp mất 3 tháng để được lấp đầy lại. Khi này họ sẽ phải lên kế hoạch phân bố hàng để sao cho trong 3 tháng chờ hàng về phải phân bố được hết hàng hóa tránh tính trạng tồn kho và đồng thời phải đáp ứng tốt nhu cầu của khách hàng ở mỗi lần giao hàng. Đối với các chính sách phân bố hàng hóa "tĩnh", số lượng hàng hóa giao cho khách hàng là như nhau ở mỗi lần giao hàng dẫn đến có thể sẽ bị thiếu hụt hàng hóa trong tương lai. Ngoài ra khi nhu cầu thay đổi đột ngột, các chính sách phân bố này cần thời gian để tính toán lại từ đó làm mất sự ổn định của tỉ lệ lấp đầy giữa các chu kì xem xét hiệu suất dẫn đến giảm lợi nhuận của hà cung cấp.

Khi áp dụng thuật toán MCTS, chính sách phân bố sẽ cân nhắc tinh chỉnh số lượng hàng giao cho các khách hàng khác nhau ở mỗi lần giao hàng tùy thuộc vào tình hình hiện tại của kho hàng và lợi nhuận của từng khách hàng. Thêm vào đó MCTS có thể phản ứng nhanh với các sự thay đổi đột ngột từ phía khách hàng. Các đặc điểm trên đảm bảo độ ổn định của tỉ lệ lấp đầy giữa các chu kì xem xét hiệu suất và tối ưu lợi nhuận cho nhà sản xuất.

III. Mô hình hóa vấn đề SLA

1. Thời điểm đưa ra quyết định

Gọi H là khoảng thời gian mà nhà cung cấp sẽ lên kế hoạch phân bố ta có T = {1, 2, 3, …, H} là tập chứa các thời điểm t phải đưa ra quyết định trong khoảng thời gian H. t ở đây có thể linh hoạt không nhất thiết là ngày, tháng, … miễn là các t cách nhau một đơn vị thời gian bằng nhau.

2. Các kí hiệu

R: tập hợp chứa các nhà bán lẻ

S: lượng hàng hiện có trong kho của nhà cung cấp

d_{rt}: nhu cầu của nhà bán lẻ r tại thời điểm t

a_{rt}: lượng hàng giao cho nhà bán lẻ r tại thời điểm t

p_s: lợi nhuận của nhà cung cấp trên một sản phẩm

π_t: lợi nhuận của nhà cung cấp tại thời điểm t

$\hat{\beta}_r$: tỉ lệ lấp đầy của nhà bán lẻ r mong muốn

$\beta_{r,\,[t_0,\,t_0+\tau-1]}$: tỉ lệ lấp đầy của nhà bán lẻ r được tính tại thời điểm kết thúc chu kì xem xét hiệu suất, bắt đầu từ thời điểm t_0 đến thời điểm $t_0 + \tau - 1$

τ: chu kì xem xét hiệu suất

λ_t: chi phí phạt cho mỗi đơn vị mà tỉ lệ lấp đầy thực tế lệch khỏi mong muốn của nhà bán lẻ

Ψ: tập hợp các trạng thái. ψ là các state (node) trong cây tìm kiếm

3. Các thành phần trong cây tìm kiếm

State: $\psi_t = (S, d_t)$

Mỗi state chứa tất cả thông tin cần biết của nh à cung cấp tại thời điểm t. Trong trường hợp này state chỉ cần biết sức chứa kho hàng của nhà cung cấp (S) và nhu cầu của các nhà bán lẻ tại thời điểm t được biểu diễn bằng vector d_t với các phần tử là nhu cầu của mỗi nhà bán lẻ d_{rt} ($d_t = (d_{rt})$ r ϵ R, d_{rt} ϵ Z)

Action: $a_t = (a_{rt})_{r \in R,\ a_{rt} \in Z,\ \sum_r a_{rt} \le S,\ a_{rt} < d_{rt}}$

Mỗi action là một cách phân bố hàng hóa đến cho tất cả các nhà bán lẻ tại thời điểm t được biểu diễn bằng vector a_t với các phần tử của vector là lượng hàng phân bố cho mỗi nhà bán lẻ a_{rt}. Mỗi action phải tuân thủ 2 điều kiện

- Tổng số hàng phân bổ phải nhỏ hơn hoặc bằng với số hàng hiện có trong kho
- Số lượng hàng cung cấp cho một nhà bán lẻ phải nhỏ hơn hoặc bằng nhu cầu của nhà bán lẻ đó⌊

Reward:

$$\pi_t(\psi_t, a_t) = \{if\ \tfrac{t}{\tau}{}^1\lfloor\tfrac{t}{\tau}\rfloor\colon\ p_s \sum_{r\epsilon R} a_{rt}\ if\ \tfrac{t}{\tau} = \lfloor\tfrac{t}{\tau}\rfloor\colon\ p_s \sum_{r\epsilon R} a_{rt} - \sum_{r\epsilon R} \lambda_r \left[\hat{\beta}_r - \beta_{r,\ [t_0,\ t_0+\tau-1]}\right]^+$$

Reward ở đây là lợi nhuận của nhà cung cấp tại thời điểm một thời điểm t. Cách tính lợi nhuận sẽ chia làm 2 trường hợp:

- Thời điểm t chưa phải là ngày hết hạn của chu kì xem xét hiệu suất, khi này lợi nhuận tính bằng lợi nhuận trên 1 sản phẩm (p_s) nhân với tổng số lượng hàng phân bố tại thời điểm t
- Thời điểm t là ngày hết hạn chu kì xem xét hiệu suất, khi này lợi nhuận sẽ tính bằng giá trị của trường hợp trên trừ cho tổng số tiền phạt. Tổng tiền phạt được tính bằng số tiền phạt nhân với lượng chênh lệch giữa tỉ lệ lấp đầy mà nhà bán lẻ r mong muốn với tỉ lệ lập đầy thực tế tại thời điểm t của nhà bán lẻ r $\left(\hat{\beta}_r - \beta_{r,\ [t_0,\ t_0+\tau-1]}\right)$. Lưu ý $[x]^+ =$ max(x, 0) tức chênh lệch tỉ lệ lấp đầy chỉ được tính khi tỉ lệ lấp đầy tại cuối PRP nhỏ hơn tỉ lệ lấp đầy mong muốn của nhà bán lẻ ($\hat{\beta}_r > \beta_{r,\ [t_0,\ t_0+\tau-1]}$). Trong trường hợp tỉ lệ lấp đầy ở cuối PRP lớn hơn nhu cầu của nhà bán lẻ mức chênh lệch sẽ bằng 0 ($\hat{\beta}_r < \beta_{r,\ [t_0,\ t_0+\tau-1]}$ thì chênh lệch là số âm dẫn đến $[\hat{\beta}_r - \beta_{r,\ [t_0,\ t_0+\tau-1]}]^+ = \max($ $\hat{\beta}_r - \beta_{r,\ [t_0,\ t_0+\tau-1]}, 0) = 0$). Tỉ lệ lấp đầy tại thời điểm kết thúc PRP tính bằng cách

chia tổng lượng hàng hóa trong chu kì PRP cho tổng lượng hàng mong muốn của các nhà bán lẻ trong chu kì PRP $\beta_{r,\,[t_0,\,t_0+\tau-1]} = \frac{\sum_{t=t_0}^{t_0+\tau-1} a_{rt}}{\sum_{t=t_0}^{t_0+\tau-1} d_{rt}}$

Từ một state ở thời điểm ta thực hiện một action để tạo ra state con của state đó tại thời điểm t + 1. Mỗi cấp t của cây đại diện cho trạng thái của nhà cung cấp tại thời điểm t trong khoảng thời gian lên kế hoạch H.

IV. Các cải tiến của thuật toán MCTS khi áp dụng vào SLA

Loại bỏ các phân bố không hợp lệ

Giả sử như chỉ có 2 nhà bán lẻ. Khi này, tập các phân bố hợp lệ F_ψ sẽ chứa các phân bố cho các nhà bán lẻ sao cho lượng hàng phân bố cho mỗi nhà bán lẻ sẽ không lớn hơn nhu cầu của họ.

$$F_\psi = \{(a_1, a_2) |\ a_1, a_2 \in Z \text{ Ù } 0 \le a_1 \le (S,\ d_1) \text{ Ù } 0 \le a_2 \le (S,\ d_2)\}$$

Tại mỗi State sẽ có thể xảy 2 trường hợp

Trường hợp 1 (c1): $\sum_{i=1}^{R} d_i \ge S$

Khi tổng nhu cầu của các nhà bán lẻ cao hơn hoặc bằng số hàng có trong kho thì tổng lượng hàng hóa đang có phải được phân phối hết, tất cả các phân bố có tổng lượng hàng phân bố nhỏ hơn lượng hàng hiện có đều dẫn đến việc giảm lợi nhuận cho nhà cung cấp

$$F_{\psi,\,c1} = \{(a_1, a_2) |\ a_1, a_2 \in F_\psi \text{ Ù } a_1 + a_2 = S\}$$

Trường hợp 2 (c2): $\sum_{i=1}^{R} d_i < S$

Khi tổng số hàng trong kho lớn hơn tổng nhu cầu của các nhà bán lẻ thì các lượng hàng phân bố cho mỗi nhà bán lẻ phải đúng bằng nhu cầu của họ

$$F_{\psi,\,c2} = \{(a_1, a_2) |\ a_1 = d_1 \text{ Ù } a_2 = d_2\}$$

Cải tiến quá trình lựa chọn trong thuật toán MCTS

Ở cải tiến này ta sẽ áp dụng các số liệu trong SLA vào công thức UCT để điều chỉnh quá trình lựa chọn trong thuật toán. Ta có $\left|\left|\vec{a}_j - \vec{d}_j\right|\right|_2$ là khoảng cách giữa vector phân bố và vector nhu cầu của nhà bán lẻ hay dễ hiệu là sự chênh lệch của tổng hàng phân bố với tổng nhu cầu của nhà bán lẻ tại mỗi node. Nếu chênh lệch tại node lớn thì tức node đó không tối ưu lợi nhuận nên ta sẽ trừ UCT cho phần chênh lệch để điều chỉnh quá trình lựa chọn của thuật toán. Công thức mới gọi là AUCT (Allocation distance UCT)

$$AUCT = R_j + 2C_p\sqrt{\frac{lnN_j}{n_j}} - C_d\left|\left|\vec{a}_j - \vec{d}_j\right|\right|_2$$

Ngoài ta kết hợp thêm tham số C_d có thể điều chỉnh mức độ ảnh hưởng của khoảng cách phân bố lên quá trình lựa chọn

Kết luận

Monte Carlo Tree Search (MCTS) là một phương pháp mạnh mẽ và linh hoạt, được sử dụng rộng rãi để giải quyết các vấn đề lựa chọn trong môi trường phức tạp, đặc biệt là trong các trò chơi trí tuệ nhân tạo. Phương pháp này nổi bật với khả năng cân bằng giữa thăm dò và khai thác, giúp tìm ra các hành động tối ưu mà không cần đến kiến thức chuyên môn hay các heuristic phức tạp. Một trong những lợi ích chính của MCTS là không yêu cầu kinh nghiệm trước, chỉ cần các luật chơi cơ bản và dễ dàng áp dụng cho nhiều loại trò chơi và vấn đề khác nhau. Ngoài ra, MCTS có khả năng mở rộng linh hoạt, cho phép áp dụng trong các môi trường có không gian trạng thái lớn và phức tạp, có thể xử lý tốt các tình huống không chắc chắn nhờ vào quá trình thăm dò ngẫu nhiên và tính toán xác suất.

Tuy nhiên, MCTS cũng có một số hạn chế. Đầu tiên, nó yêu cầu một lượng lớn tính toán để mô phỏng các kịch bản và cập nhật cây tìm kiếm, điều này có thể trở nên tốn kém về mặt tài nguyên tính toán, đặc biệt trong các môi trường rất phức tạp. Thứ hai, kết quả của MCTS phụ thuộc nhiều vào chất lượng và độ chính xác của các mô phỏng ngẫu nhiên. Nếu các mô phỏng không phản ánh đúng bản chất của môi trường, các quyết định đưa ra có thể không tối ưu.

Trong tương lai, các nghiên cứu sẽ tiếp tục tập trung vào việc tối ưu hóa MCTS để giảm thiểu yêu cầu tính toán và cải thiện hiệu quả tìm kiếm. Một xu hướng mới là kết hợp MCTS với các phương pháp học máy để nâng cao chất lượng của các mô phỏng và quyết định. Ví dụ, AlphaGo của Google DeepMind đã sử dụng sự kết hợp này để đạt hiệu suất vượt trội trong trò chơi cờ vây. Ngoài ra, MCTS đang được mở rộng ứng dụng sang các lĩnh vực khác ngoài trò chơi, bao gồm lập kế hoạch chiến lược, robot, và các hệ thống gợi ý. Tổng kết lại, Monte Carlo Tree Search là một công cụ quan trọng và hiệu quả trong trí tuệ nhân tạo và khoa học máy tính, với tiềm năng lớn để tiếp tục phát triển và mở rộng trong tương lai.

Tài liệu tham khảo

1. Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., ... & Colton, S. (2012). "A Survey of Monte Carlo Tree Search Methods." *IEEE Transactions on Computational Intelligence and AI in Games, 4*(1), 1-43. doi:10.1109/TCIAIG.2012.2186810
2. Chaslot, G. M. J.-B., Winands, M. H. M., & van den Herik, H. J. (2008). "Parallel Monte-Carlo Tree Search." *Computers and Games, 5131*, 60-71. doi:10.1007/978-3-540-87608-3_6
3. Gelly, S., & Silver, D. (2011). "Monte-Carlo Tree Search and Rapid Action Value Estimation in Computer Go." *Artificial Intelligence, 175*(11), 1856-1875. doi:10.1016/j.artint.2011.03.007
4. Kocsis, L., & Szepesvári, C. (2006). "Bandit based Monte-Carlo Planning." *Proceedings of the 17th European Conference on Machine Learning (ECML-06)*, 282-293. doi:10.1007/11871842_29
5. Coulom, R. (2006). "Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search." *Proceedings of the 5th International Conference on Computers and Games (CG-06)*, 72-83. doi:10.1007/978-3-540-75538-8_7
6. Silver, D., & Veness, J. (2010). "Monte-Carlo Planning in Large POMDPs." *Proceedings of the 24th International Conference on Neural Information Processing Systems (NIPS-10)*, 2164-2172. doi:10.5555/2984093.2984318
7. Browne, C. B. (2014). "Introduction to Monte Carlo Tree Search." *Lecture Notes in Computer Science, 8768*, 1-14. doi:10.1007/978-3-319-09165-5_1
8. Lorentz, R. J. (2016). "An MCTS Program to Play Kriegspiel." *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence (AAAI-16)*, 39-44. doi:10.1609/aaai.v22i2.16004
9. Lanctot, M., Winands, M. H. M., & Sturtevant, N. R. (2014). "Monte Carlo Tree Search with Heuristic Evaluations Using Implicit Minimax Backups." *Proceedings of the 2014 IEEE Conference on Computational Intelligence and Games (CIG-14)*, 1-8. doi:10.1109/CIG.2014.6932871
10. Björnsson, Y., & Finnsson, H. (2009). "CadiaPlayer: A Simulation-Based General Game Player." *IEEE Transactions on Computational Intelligence and AI in Games, 1*(1), 4-15. doi:10.1109/TCIAIG.2009.2018702
11. Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., ... & Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587), 484-489.

12. Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., ... & Colton, S. (2012). A survey of Monte Carlo Tree Search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1), 1-43.
13. Coulom, R. (2006). Efficient selectivity and backup operators in Monte-Carlo tree search. In *International conference on computers and games* (pp. 72-83). Springer, Berlin, Heidelberg.
14. Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. MIT press.
15. Kocsis, L., & Szepesvári, C. (2006). Bandit based monte-carlo planning. In *European conference on machine learning* (pp. 282-293). Springer, Berlin, Heidelberg.
16. DeepMind. (2016). AlphaGo: The story so far.
17. Tesauro, G. (1995). Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38(3), 58-68.