Đại học Khoa học tự nhiên
Đại học Quốc gia
Thành phố Hồ Chí Minh

Ngành công nghệ thông tin

Đồ án cá nhân
Tìm hiểu về
Monte Carlo Tree Search

Môn học: Cơ sở Trí tuệ nhân tạo

Học viên:
Trương Ngọc Huy
20120109

Giáo viên:
Nguyễn Ngọc Đức

October 2022

Phụ lục

1 Giới thiệu về Monte Carlo Tree Search

Cờ vây, bắt nguồn từ Trung Quốc thế vào kỷ 4 trước Công nguyên, được thiết kế với bàn cờ gồm 17×17 ô, sau này trở thành 19×19 ô[1], và dần phát triển thành một trong những trò chơi đối kháng chiến thuật phức tạp nhất thế giới. Với số lượng ô trên mỗi bàn cờ là 19×19, không gian trạng thái (state space) của trò chơi là 10^{172} kích thước của cây trò chơi (game tree size) lên đến 10^{360}; điều này làm cho việc máy tính có thể tính toán để đưa ra các nước đi tối ưu trong thời gian ngắn có thể chấp nhận được là một điều rất khó.

Năm 2015, máy tính đánh cờ vây AlphaGo của Google DeepMind chiến thắng vang dội trước nhà đương kim vô địch cờ vây Châu Âu - Fan Hui với tỉ số 5-0. Và trong hai năm sau đó, AlphaGo lần lượt đánh bại những người đứng đầu trong giới cờ vây - nhà vô địch thế giới Lee Sedol (4-1) và người chơi cờ vây top 1 thế giới Ke Jie(3-0)[2]. Có nhiều thứ đã giúp AlphaGo vượt qua các vấn đề về chi phí tính toán cũng như chi phí về mặt không gian trạng thái để chiến thắng các cao thủ cờ vây, một trong số đó là Monte Carlo Tree Search (phần còn lại của bài viết sẽ gọi tắt là MCTS).

2 Ý tưởng của MCTS

MCTS là một thuật toán tìm kiếm heuristic được xây dựng bằng cách mở rộng bất đối xứng và mở các node với một hàm đánh giá gọi là tree policy. Hàm đánh giá sẽ hướng đến việc cân bằng giữa sự khai phá (exploration) và sự khai thác (exploitation). Sự khai phá sẽ tập trung vào những khu vực(nhánh) ít được khám phá hoặc chưa được khám phá, sự khai thác sẽ tập trung vào những khu vực(nhánh) có tiềm năng đem lại kết quả tốt. Ở mỗi vòng lặp của thuật toán, hàm đánh giá sẽ tìm ra(select) một node "phù hợp" nhất với tiêu chí trên của cây (được xây dựng đến thời điểm đang xét) và bắt đầu một mô phỏng (simulation) từ node đã chọn; khi có kết quả từ mô phỏng của node, cây sẽ được cập nhật dần về gốc (back-propagation)[3]. Sau khi cây đã được cập nhật, thuật toán sẽ bắt đầu vòng lặp mới và mở rộng cây đến một giới hạn cho phép nào đó.

Ở bước mô phỏng, việc đưa ra các bước đi đều dựa trên một chính sách (policy) nhất định, trường hợp dễ nhất là lấy **ngẫu nhiên** các bước đi mô phỏng để đến trạng thái kết thúc (terminal state). Giá trị cuối cùng của bước mô phỏng không phụ thuộc vào các trạng thái mô phỏng trung gian mà chỉ phụ thuộc vào trạng thái kết thúc[3].

3 Xây dựng Monte Carlo Tree Search

Các vòng lặp được thực hiện vô tận đến một giới hạn cho phép, có thể là giới hạn về độ sâu, giới hạn về thời gian, giới hạn về bộ nhớ,... Mỗi lặp có bốn bước:

- Select
- Expand
- Simulation
- Back-propagation

3.1 Select

Ở mỗi vòng lặp, việc đầu tiên và quan trọng nhất là chọn ra một node **lá** phù hợp nhất với tiêu chí cân bằng giữa sự khám phá và sự khai phá từ cây hiện tại. Hàm đánh giá phù hợp với tiêu chí trên là UCT (Upper Confidence Bound 1 applied to trees) được đưa ra bởi Levente Kocsis and Csaba Szepesvári[4].

$$\mathbf{UCT}(v_i, v) = \frac{Q(v_i)}{N(v_i)} + c\sqrt{\frac{\log N(v)}{N(v_i)}}$$

trong đó:

- v_i là các node con của node v.
- $Q(v_i)$ là tổng giá trị các mô phỏng bắt đầu từ node v_i và các node con của v_i.
- $N(v)$ là tổng số lần node v được thăm.
- c là tham số khám phá (exploration parameter), về mặt lý thuyết có thể bằng $\sqrt{2}$, trong thực tế có thể chọn bằng kinh nghiệm.

Bắt đầu từ node gốc, chọn node gốc làm node đang xét. Nếu node đang xét không phải là node lá, từ các node con của node đang xét, chọn ra một node có giá trị **UCT** cao nhất làm node đang xét và lặp lại quá trình đến khi node đang xét là node lá. Sau khi hoàn thành vòng lặp, node đang xét chính là node được chọn.

3.2 Expand

Xét node được chọn từ bước trước:

- Nếu là node kết thúc (terminal node) thì đến bước back-propagation.
- Nếu node chưa được thăm ($N(v) = 0$), bỏ qua bước này và bắt đầu bước simulation cho node được chọn.

- Nếu node đã được thăm ($N(v) \neq 0$), thêm các node con v_i là các trạng thái có thể xảy ra bắt đầu từ node v. Chọn một node con bất kỳ làm node được chọn bắt đầu bước simulation cho một node đó.

3.3 Simulation

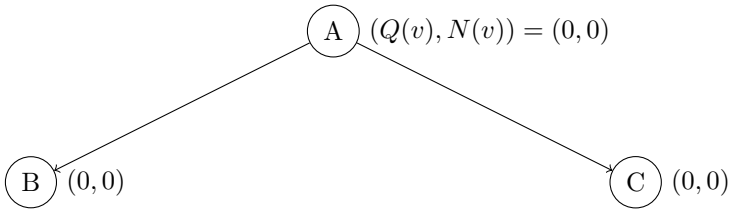
Simulation hay gọi là mô phỏng của một node, là một tập các trạng thái được bắt đầu từ node đó và kết thúc ở terminal node (node kết thúc). Ở bước này, giá trị của các trạng thái trung gian đều được bỏ qua mà chỉ quan tâm giá trị của node kết thúc. Sau khi kết thúc mô phỏng, giá trị $Q(v)$ của node sẽ chính là giá trị của node kết thúc.

3.4 Back-propagation

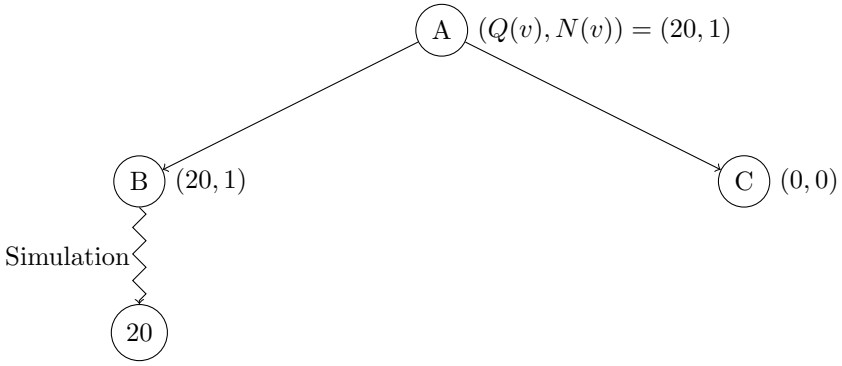
Tiến hành cập nhật giá trị các thuộc tính $Q(v)$ và $N(v)$ của node và các node tổ tiên. Back-propagation làm cho kết quả simulation của node được thể hiện ở các node tổ tiên và làm thay đổi giá trị **UCT** của các node đó, đồng thời ảnh hưởng đến bước select tiếp theo.

3.5 Ví dụ

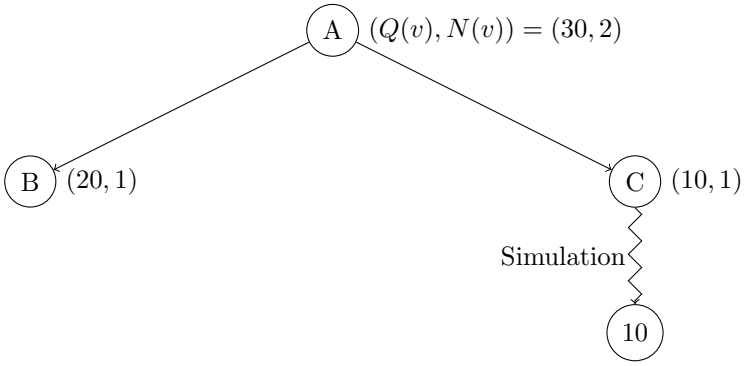
Bắt đầu xây dựng một MCTS từ node gốc là A có hai node con là B và C.



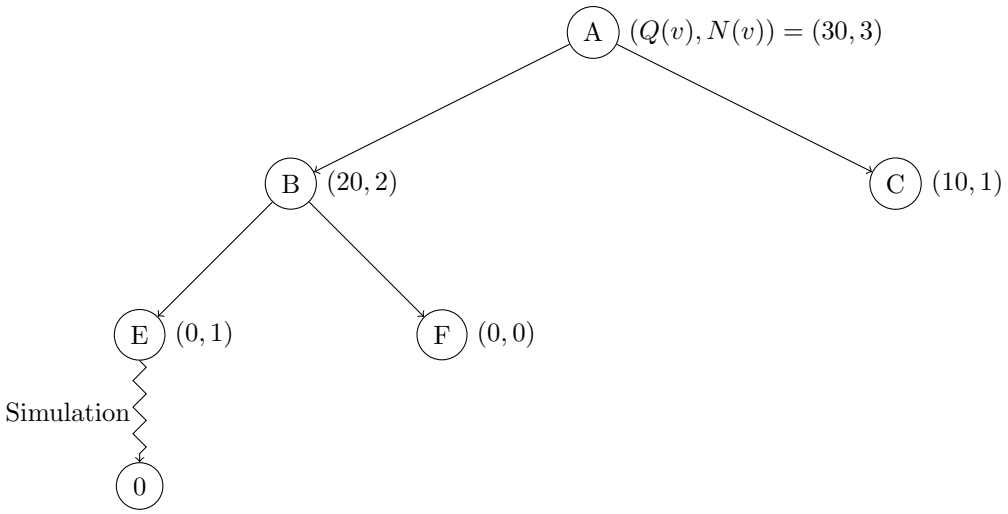
- Do $\mathbf{UCT}(B, A) = \mathbf{UCT}(C, A) = \infty$ nên chọn ngẫu nhiên một trong hai node con, ở ví dụ này sẽ ưu tiên chọn từ trái sang phải. Node **B** là node đang xét và cũng là node lá nên node được chọn sẽ là node **B**.
- Do node **B** không phải là node kết thúc và chưa được thăm nên bỏ qua bước expand.
- Từ B bắt đầu simulation. Giả sử giá trị của simulation là 20.
- Cập nhật lại cây.



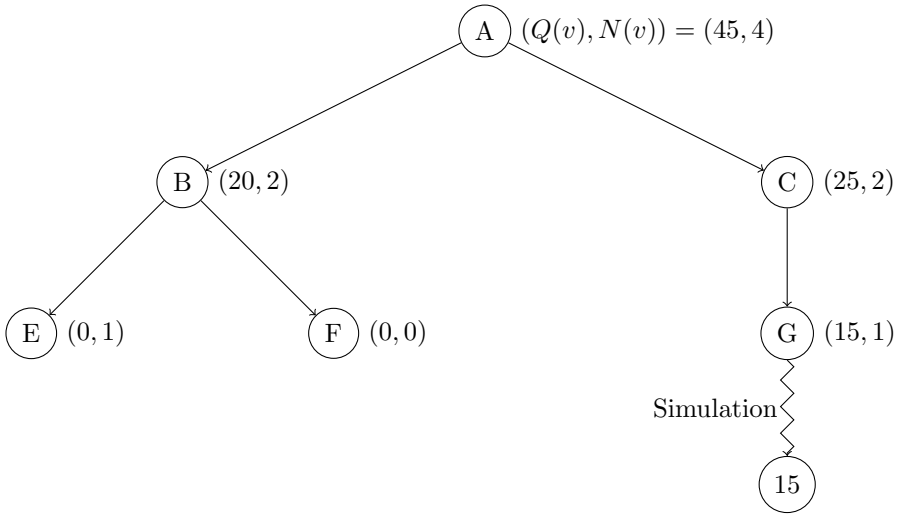
- Do $\mathbf{UCT}(B, A) = 20 < \mathbf{UCT}(C, A) = \infty$ nên node đang xét là node **C**. Node **C** là node lá nên node được chọn sẽ là node **C**.
- Do node **C** không phải là node kết thúc và chưa được thăm nên bỏ qua bước expand.
- Từ **C** bắt đầu simulation. Giả sử giá trị của simulation là 10.
- Cập nhật lại cây.



- Do $\mathbf{UCT}(B, A) = 20 > \mathbf{UCT}(C, A) = 10$ nên node đang xét là node **B**. Node **B** là node lá nên node được chọn sẽ là node **B**.
- Do node **B** không phải là node kết thúc và đã được thăm nên thêm các node con **E** và **F** của **B** vào cây. Chọn **E** làm node được xét.
- Từ **E** bắt đầu simulation. Giả sử giá trị của simulation là 0.
- Cập nhật lại cây.



- Do $\mathbf{UCT}(B, A) = 10.6907 < \mathbf{UCT}(C, A) = 10.9769$ nên node đang xét là node **C**. Node **C** là node lá nên node được chọn sẽ là node **C**.
- Do node **C** không phải là node kết thúc và đã được thăm nên thêm các node con **G** của **C** vào cây. Chọn **G** làm node được xét.
- Từ **G** bắt đầu simulation. Giả sử giá trị của simulation là 15.
- Cập nhật lại cây.



Vòng lặp được thực hiện vô tận và MCTS được mở rộng đến khi máy tính sử dụng hết tài nguyên cho phép. Khi đó, từ các node con của node gốc sẽ chọn ra node có số lần được thăm ($N(v)$) nhiều nhất và có giá trị ($Q(v)$) lớn nhất để làm nước đi tiếp theo. Trong ví dụ này, giả sử đến bước này node **A** bắt buộc phải đưa ra lựa chọn thì node **C** sẽ là nước đi tiếp theo (do $N(C) = N(B)$ và $Q(C) > Q(B)$).

4 Ưu điểm và khuyết điểm

4.1 Ưu điểm

- MCTS là một thuật toán heuristic và dễ cài đặt. Với một heuristic hợp lý và nhất quán thì MCTS sẽ mở rộng cây bất đối xứng một cách hiệu quả.
- MCTS hoạt động nhờ vào luật chơi và trạng thái kết thúc nên cây có thể tự tìm các nước đi thông các mô phỏng ngẫu nhiên trong quá trình xây dựng cây mà không phụ thuộc vào bất kỳ kiến thức đặc thù nào[5].
- Có thể xây dựng một MCTS mới từ MCTS cũ.

4.2 Nhược điểm

- Độ phức tạp về không gian của MCTS là cấp số mũ nên nếu độ sâu lớn sẽ dẫn đến vấn đề về bộ nhớ.
- Để đưa ra một nước đi "tối ưu" nhất thì cần tối đa số simulation có thể tạo ra và dẫn đến tốc độ để đưa ra một nước đi chậm hơn.
- Do các nhánh được mở rộng bất đối xứng nên có thể xảy ra trường hợp chọn một nhánh chưa được mở rộng đủ nhiều hoặc chưa đủ sâu dẫn đến kết quả không mong muốn. Ván thua của AlphaGo trước Lee Sedol là một ví dụ cho trường hợp này[5].

5 Monte Carlo Tree Search trong AlphaGo

> "MCTS use Monte Carlo rollouts to estimate the value of each state in a search tree. As more simulations are executed, the search tree grows larger and the relevant values become more accurate. The policy used to select actions during search is also improved over time, by selecting children with higher values. Asymptotically, this policy converges to optimal play, and the evaluations converge to the optimal value function. The strongest current Go programs are based on MCTS..."[6]

> "We pass in the board position as a 19×19 image and use convolutional layers to construct a representation of the position. We use these neural networks to reduce the effective depth and breadth of the search tree: evaluating positions using a value network, and sampling actions using a policy network."[6]

> "Our program AlphaGo efficiently combines the policy and value network, and sampling actions using a policy network."[6]

AlphaGo sử dụng hai deep neural network là value network và policy network. Value network dùng để tính xấp xỉ giá trị của simulation bắt đầu từ trạng thái(node) bất kỳ dựa trên những tập dữ liệu được cung cấp từ trước; giá trị trả về sẽ là xác suất chiến thắng. Policy network cũng tương tự như value network nhưng không dựa trên những tập dữ liệu có sẵn mà thông qua quá trình tự học(Reinforcement Learning). Cả hai deep neural network đều "huấn luyện" bằng việc áp dụng MCTS.

Hàm đánh giá các nước đi được sử dụng trong AlphaGo là:

$$\mathbf{UCT}(v_i, v) = \frac{Q(v_i)}{N(v_i)} + cP(v, v_i)\frac{\sqrt{N(v)}}{1 + N(v_i)}$$

trong đó $P(v, v_i)$ là giá trị xác suất ưu tiên(prior probability) của nước đi từ v đến v_i được tính thông qua policy network[7].

6 Tài liệu tham khảo

[1]: Go Wikipedia

[2]: MONTE CARLO TREE SEARCH: A TUTORIAL

[3]: Browne, C., E. Powley, D. Whitehouse, S. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. 2012. “A Survey ofMonte Carlo Tree SearchMethods”.IEEE Transactions on Computational Intelligence and AI in Games 4(1):page 1-2

[4]: Kocsis, Levente; Szepesvári, Csaba (2006). "Bandit based Monte-Carlo Planning". In Fürnkranz, Johannes; Scheffer, Tobias; Spiliopoulou, Myra (eds.). Machine Learning: ECML 2006, 17th European Conference on Machine Learning, Berlin, Germany, September 18–22, 2006, Proceedings. Lecture Notes in Computer Science. Vol. 4212. Springer. pp. 282–293.

[5]: MCTS Geeksforgeeks

[6]: Silver, D., A. Huang, Aja, C. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. 2016. ”Mastering the Game of Go with Deep Neural Networks and Tree Search”. Nature 529 (28 Jan):484–503.

[7]: MCTS int8