ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA KHOA HỌC MÁY TÍNH

LỚP CS106.M21

BÁO CÁO ĐỀ TÀI

MONTE CARLO TREE SEARCH

| Giáo viên hướng dẫn |
TS. Lương Ngọc Hoàng
Môn học: Trí Tuệ Nhân Tạo

Thành phố Hồ Chí Minh – 2022

DANH SÁCH THÀNH VIÊN

STT	Họ Tên	MSSV	Email
1	Trần Văn Lực	20521587	20521587@gm.uit.edu.vn
2	Lê Minh Quân	20520709	20520709@ms.uit.edu.vn
3	Lê Nguyễn Bảo Hân	20520174	20520174@gm.uit.edu.vn
4	Nguyễn Trần Minh Anh	20520394	20520394@gm.uit.edu.vn

LỜI CẢM ƠN

Lời đầu tiên, nhóm chúng em xin gởi lời cảm ơn chân thành đến quý thầy cô giảng viên Trường Đại học Công nghệ Thông tin – Đại học Quốc gia TP.HCM đã giúp cho nhóm chúng em có những kiến thức cơ bản làm nền tảng để thực hiện đề tài này, luôn tạo điều kiện tốt nhất để sinh viên có thể hoàn thành tốt quá trình học tại trường nói chung và trong môn học này nói riêng.

Tiếp theo, nhóm chúng em xin gởi lời cảm ơn và lòng biết ơn sâu sắc nhất tới Thầy Lương Ngọc Hoàng – người đã dẫn dắt nhóm em trong suốt thời gian làm đề tài. Thầy đã tận tình hướng dẫn, chỉ bảo với những phân tích định hướng rõ ràng cho nhóm trong suốt quá trình thực hiện đề tài, là tiền đề để nhóm có thể hoàn thành đề tài đúng hạn. Thầy cũng tạo điều kiện thuận lợi nhất có thể với các tài liệu cần thiết liên quan, giải đáp thắc mắc tại lớp khi các nhóm gặp khó khăn. Một lần nữa em chân thành cảm ơn Thầy và chúc Thầy dồi dào sức khoẻ.

Trong thời gian thực hiện đề tài, nhóm chúng em đã vận dụng những kiến thức nền tảng đã tích lũy đồng thời kết hợp với việc học hỏi và nghiên cứu những kiến thức mới từ thầy cô, bạn bè cũng như nhiều nguồn tài liệu tham khảo. Từ đó, nhóm chúng em vận dụng tối đa những gì đã thu thập được để hoàn thành một báo cáo đồ án tốt nhất. Tuy nhiên, vì kiến thức chuyên môn còn hạn chế và bản thân còn thiếu nhiều kinh nghiệm thực tiễn nên nội dung của báo cáo không tránh khỏi những thiếu sót, chúng em rất mong nhận được sự góp ý, chỉ bảo thêm của quý thầy cô nhằm hoàn thiện những kiến thức của mình để nhóm chúng em có thể dùng làm hành trang thực hiện tiếp các đề tài khác trong tương lai cũng như là trong việc học tập và làm việc sau này.

Một lần nữa xin gởi đến thầy cô, bạn bè lời cảm ơn chân thành và tốt đẹp nhất!

Thành phố Hồ Chí Minh, tháng 7 năm 2022

Nhóm sinh viên thực hiện

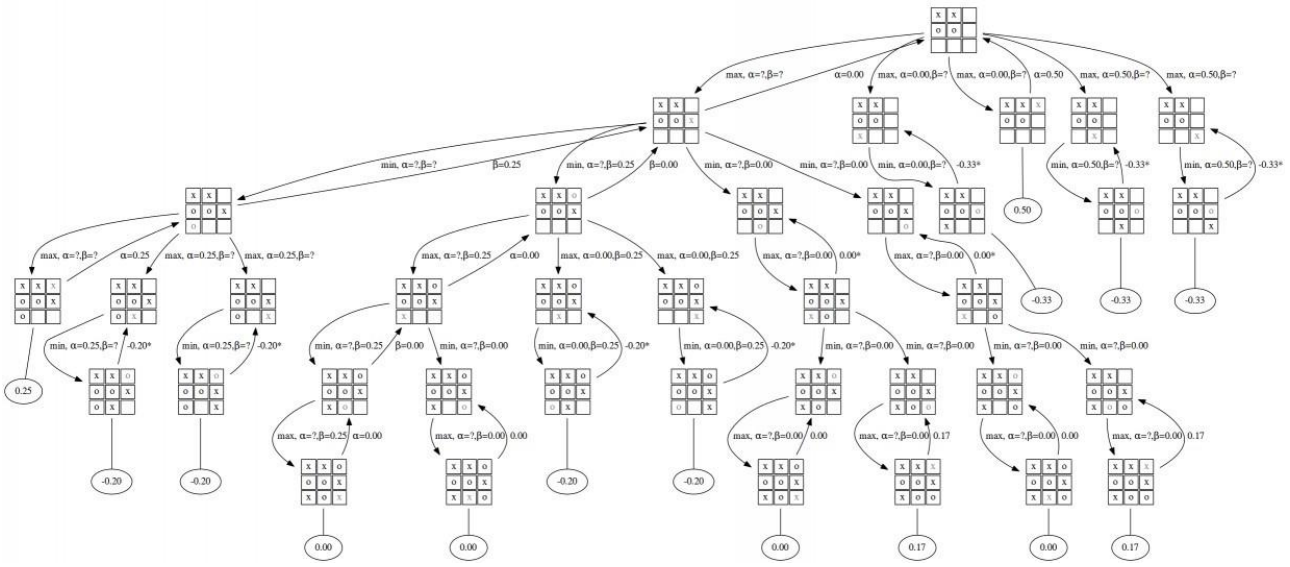
MỤC LỤC

DANH MỤC HÌNH ẢNH

I. GIỚI THIỆU

1. Cây tìm kiếm là gì?

Là một phương pháp dùng để tìm kiếm, dò tìm mọi bước đi có thể xảy ra sau mỗi lượt chơi trong một trò chơi. Ví dụ: người chơi có thể chọn rất nhiều phương án trong một ván chơi Tic-Tac-Toe, tất cả những lựa chọn đó có thể được sơ đồ hoá dưới dạng một cây tìm kiếm. Với mỗi nước chơi được sinh ra, các nhánh cây sẽ được nhân rộng, kéo dài trở thành một cây tìm kiếm hoàn chỉnh.



Hình 1. Minh họa sơ đồ cây tìm kiếm của một ván chơi Tic-Tac-Toe

Áp dụng thuật toán Brute Force, các nhánh cây của một cây tìm kiếm sẽ được mở rộng theo cấp số nhân. Nhờ vào đó, đường đi tới nút chiến thắng của trò chơi có thể được tìm ra. Tuy vậy, việc sơ đồ hoá lên cây tìm kiếm, đồng thời suy xét mọi trường hợp có thể xảy ra trong một trạng thái của trò chơi đặt ra yêu cầu rất lớn cho bộ xử lý của máy tính, dẫn tới việc tuỳ vào độ đa dạng trạng thái của trò chơi mà kết quả được tìm thấy có thể tốn rất nhiều thời gian.

Nên để tối ưu hoá quá trình tìm kiếm đường đi tốt nhất trên cây tìm kiếm, các nút trên cây sẽ được sắp xếp theo thứ tự ưu tiên, tuỳ vào mức độ ảnh hưởng của chúng tới bài toán. Và khi quá trình tìm kiếm lời giải được thực hiện, các nút có độ ưu tiên cao hơn sẽ được suy xét trước tiên.

Một trong những cây tìm kiếm được tối ưu hoá như vậy là cây tìm kiếm Monte Carlo.

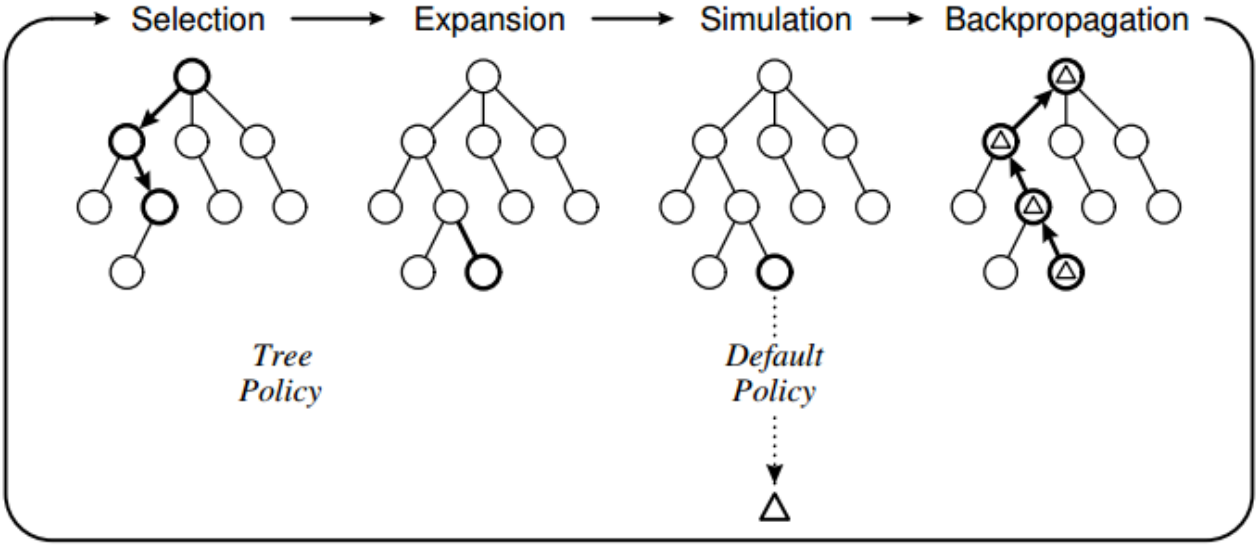
2. Cây tìm kiếm Monte Carlo là gì?

Cây tìm kiếm Monte Carlo là một kỹ thuật tìm kiếm trong lĩnh vực Trí tuệ nhân tạo (AI) sử dụng phương pháp tìm kiếm dựa theo lấy mẫu dùng giả lập ngẫu nhiên để ước lượng tỷ lệ thắng thua của một trạng thái nhằm tìm ra nước đi tốt nhất và để cân bằng giữa việc khám phá và khai thác của tất cả các nước đi. Điểm mấu chốt của cây tìm kiếm Monte Carlo so với các phương pháp tìm kiếm truyền thống như AlphaBeta và A* là nó không phụ thuộc vào tri thức đặc trưng của trò chơi, nói cách khác là không phụ thuộc vào hàm lượng giá trạng thái. Khi đó, MCTS có thể áp dụng vào nhiều trò chơi dạng không may rủi và thông tin trạng thái trò chơi rõ ràng sau mỗi lượt đi. Đối với các trò chơi mà khó xây đựng hàm lượng giá trạng thái tốt như cờ Vây thì việc áp dụng cây tìm kiếm Monte Carlo rất hiệu quả.

Cây tìm kiếm **Monte Carlo** là một quá trình lặp lại nhiều lần bốn bước sau:

- Chọn lựa (Selection) : Từ node gốc, chọn đường đi tiềm năng nhất (dựa trên một vài statistics) cho đến khi gặp node lá
- Mở rộng (Expansion) : Một nút, hay một trạng thái hoàn toàn mới, chưa được truy cập sẽ được chọn ngẫu nhiên và được gắn thêm vào cây tìm kiếm
- Giả lập (Simulation) : Mở rộng hay mô phỏng với các nước đi ngẫu nhiên đến khi đạt trạng thái kết thúc
- Lan truyền ngược (Backpropagation): Là quá trình truyền ngược trở lại từ nút lá (nơi bắt đầu mô phỏng) đến nút gốc và Tất cả các nút trên đường dẫn đã chọn đều được cập nhật thông tin, tương ứng với kết quả của ván đấu thu được từ quá trình mô phỏng

Các bước này sẽ được lặp đi lặp lại cho đến khi tìm ra được lời giải và truy xuất được chiến lược tối ưu cho bài toán. Các bước trên được sơ đồ hoá như sau:



Hình 2. Sơ đồ các bước thực hiện quá trình tìm kiếm của cây tìm kiếm Monte Carlo

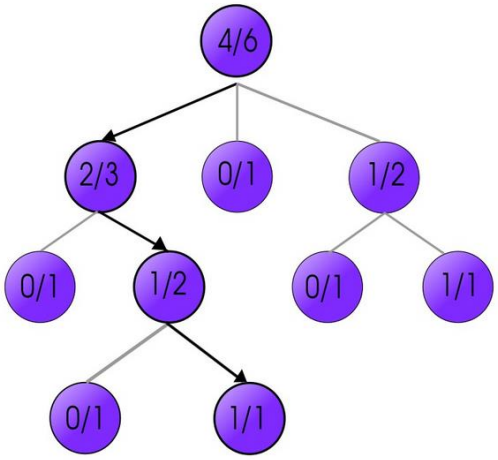
II. CÂY TÌM KIẾM MONTE CARLO

1. Cơ chế hoạt động

1.1. Chọn lựa (Selection)

Sau mỗi lần hoàn thành một lượt chơi, cây Tìm kiếm Monte Carlo sẽ lựa chọn nút có độ ưu tiên (có khả năng chiến thắng) cao nhất trong các nút con kế tiếp. Quá trình này được là là Chọn lựa - Selection.

Ví dụ:



Hình 3. Ví dụ quá trình Chọn Lựa

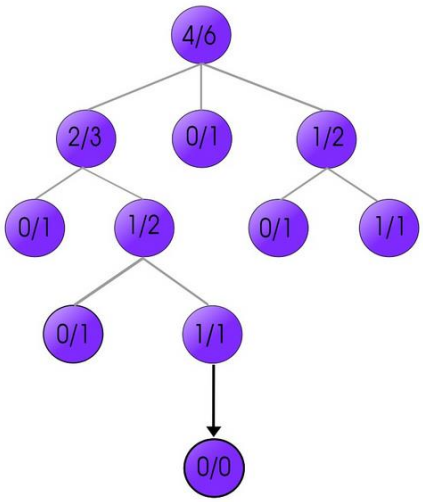
Xét cây tìm kiếm trên với mỗi nút là một phân số có ý nghĩa như sau:

- *Tử số*: số ván thắng trong tổng số ván đã chơi khi người chơi quyết định chọn nút này
- *Mẫu số*: tổng số ván đã chơi dựa trên nút này

Tại nút gốc 4/6, ta có các lựa chọn báo gồm các nút: 2/3, 0/1 và 1/2 - dễ dàng thấy được rằng nút 2/3 có tỉ lệ thắng cao nhất.

Cứ mỗi lần tiến tới một trạng thái sau một lựa chọn, tại nút cha, quá trình chọn các nút con sẽ được diễn ra cho đến khi chạm tới nút lá. Và mỗi lần lựa chọn như vậy đều là các nút có độ ưu tiên cao (khả năng chiến thắng cao) dẫn tới việc tập hợp các nút được chọn cũng sẽ là một đường đi có khả năng chiến thắng cao.

1.2. Mở rộng (Expansion)

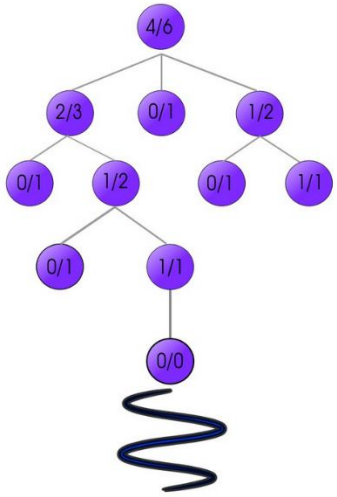


Hình 4. Ví dụ quá trình Mở Rộng

Sau khi đã hoàn thành việc chọn lựa giữa các nút con, các nhánh cây sẽ tiếp tục mở rộng ra nút con hơn bằng cách phát triển các khả năng có thể xảy ra trên nút cha. Các nút con này sẽ là tập hợp các nút được suy xét để lựa chọn trong tương lai.

Quá trình chỉ dừng lại khi cây tìm kiếm không thể mở rộng hơn được nữa. Các nút cuối cùng này gọi là nút lá.

1.3. Giả lập (Simulation)



Hình 5. Ví dụ quá trình Giả Lập

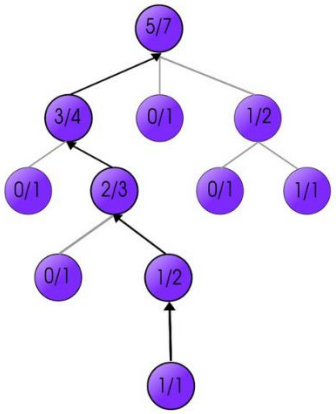
Giả lập, hay còn gọi là khám phá. Sau khi hoàn thành chọn lựa và mở rộng, ta đã có nút mà ta muốn chọn lựa. Tuy vậy, không thể đảm bảo nút đã được chọn đó là nút tốt nhất trên cây, và cũng không thể đảm bảo nút đó sẽ dẫn tới kết quả tốt nhất của cây. Vậy làm thế nào để đảm bảo được điều đó?

Có một cách đơn giản đó chính là áp dụng Học tăng cường (Reinforcement learning) trong quá trình giả lập.

Đầu tiên, sử dụng Học tăng cường để chọn lựa ngẫu nhiên giữa các nút trong tất cả các nút con mà ta có được. Tiếp đến, tính phần thưởng mà các nút đó có được - bằng cách tính toán tổng các giá trị nhận được trong quá trình đi từ nút đó tới nút lá. Nút nào có giá trị phần thưởng cao nhất - tương đương với nút mà tại đó sẽ dẫn tới kế quả tốt nhất - sẽ là nút được chọn.

Việc giả lập sẽ kết thúc khi tất cả các nút con đều đã có giá trị phần thưởng của riêng nó.

1.4. Lan truyền ngược (Backpropagation)



Hình 6. Ví dụ quá trình Lan Truyền Ngược

Cho rằng nút là có tiềm năng trong tương lai có tỉ số chiến thắng là 1/1.

Dựa trên kết quả của nút đó (dương hoặc âm dựa vào môi trường), giá trị thưởng của nút cha sẽ là tổng giá trị của các nút con khi được truy ngược lên từng nút một. Tuỳ vào giá trị của nút cha mới được cập nhập đó, trạng thái của cây tìm kiếm và việc chọn lựa tiếp theo giữa các nút con sẽ thay đổi.

Sau khi đã cập nhập giá trị cho nút cha, vòng lặp các bước trên sẽ quay lại từ chỗ chọn ra nút con tốt nhất → mở rộng các nút con của nút đó → sử dụng học tăng cường cho việc giả lập khám phá → truy ngược lại và cập nhập giá trị của nút → thật sự chọn một nút con có thể dẫn tới kết quả chiến thắng.

2. Đi tìm lời giải

2.1. Các thông số cần thiết

Để tìm được các thông số cần thiết cho việc tìm kiếm lời giải, đầu tiên phải xét đến mục đích của lan truyền ngược trên cây tìm kiếm. Đó là để cập nhập ***tổng giá trị phần thưởng giả lập Q(v)*** và ***số lần di chuyển qua một nút N(v)*** cho tất cả các nút trong quá trình lan truyền (bao gồm cả nút đầu tiên được xét đến).

Trong đó:

- ***Q(v) - Tổng giá trị phần thưởng giả lập***: là một trong các thông số ta cần để giải quyết bài toán, được tính bằng cách lấy tổng giá trị phần thưởng giả lập của các nút con mà nó đi qua.
- ***N(v) - Số lần di chuyển qua một nút***: là thông số thứ hai mà ta cần đến ngoài *Q(v)*, giá trị này đại diện cho số lần di chuyển mà nút *v* thực hiện trong quá trình lan truyền ngược (hoặc có thể nói là số nút con được xét vào khi tính tổng giá trị phần thưởng giả lập của nút *v*).

Cả hai giá trị này đều sẽ được tính cho mọi nút mà ta di chuyển tới trên cây. Một khi quá trình giả lập đã được thực thi một số lần nhất định, ta có thể chỉ ra được nút mà ta đang xét được khai thác/khám phá tới mức độ nào.

Nói cách khác, khi nhìn vào hai thông số trên, ta có thể biết được một nút có tiềm năng như thế nào (tổng giá trị phần thưởng giả lập) và nó đã khai thác các nút con tới mức nào (số lần di chuyển qua một nút). Một nút có giá trị phần thưởng cao thì rất tiềm năng (khai thác tốt), nhưng nếu nút đó có số lần di chuyển thấp cũng sẽ rất đáng chú ý (vì nút này chưa được khám phá kỹ càng).

Ta đã có các thông số cần thiết để xét các nút con trên một cây. Vậy làm thế nào để có thể áp dụng các thông số đó để di chuyển từ nút này tới nút khác?

2.2. Di chuyển trên cây

Khi vừa bắt đầu tại nút gốc, quá trình giả lập vẫn chưa được thực hiện, khi đó các nút chưa được đi qua sẽ được lựa chọn trước tiên. Từ các nút đó, quả trình ***giả lập*** sẽ xảy ra trên từng nút, kết quả của quá trình giả lập sẽ được ***lan truyền ngược*** lại với nút gốc, khi đó nút gốc sẽ được ***mở rộng*** ra hoàn toàn.

Bước tiếp theo là làm thế nào ta định vị được nút kế tiếp cần di chuyển tới trong hàng đống nút đã được giả lập kết quả?

Để có thể chọn lọc ra nút tiếp theo trên đường đi liền kề với nút đang xét *v*, ta cần phải bung ra mọi nút con có thể: $v_1, v_2, v_3, \dots, v_k$ của *v* và thông tin của chính nút *v* đó. Vậy hãy xét tới các thông tin mà ta đang có:

Với nút *v* hiện có, ta đã mở rộng ra được các nút con mà *v* có thể di chuyển tới. Tại các nút con này, ta có hai thông số đã được tính thông qua quá trình giả lập là: tổng giá trị phần thưởng giả lập và số lần di chuyển qua một nút trên cây. Các thông số này dẫn tới mảnh ghép cuối cùng mà ta cần, chính là ***Cận trên khoảng tin cậy của cây*** (***Upper Confidence Bound Applied To Trees***) hay gọi tắt là ***UCT***.

2.3. Cận trên khoảng tin cậy (Upper Confidence Bound Applied To Trees)

UCT là một hàm để tính ra giá trị nhằm quyết định nút tiếp theo mà ta sẽ di chuyển tới trên cây - cốt lõi của cây tìm kiếm Monte Carlo. ***UCT*** được biểu diễn dưới dạng công thức như sau:

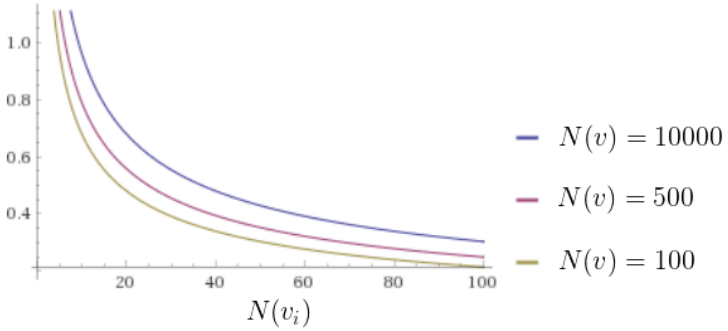
$$UCT(v_i, v) = \frac{Q(v_i)}{N(v_i)} + c\sqrt{\frac{log\big(N(v)\big)}{N(v_i)}}$$

Với công thức định nghĩa nút con v_i của nút v. Trong đó ta có $\frac{Q(v_i)}{N(v_i)}$ còn được gọi là ***thành phần khai thác***, hoặc chỉ số thắng/thua - lấy tổng giá trị phần thưởng giả lập chia cho số lần di chuyển qua một nút cho ra chỉ số thắng của nút v_i. Chỉ số này sẽ giúp ta chọn ra nút có chỉ số thắng cao nhất (tiềm năng lớn nhất) trên cây.

Tại sao ta không chỉ xét đến ***thành phần khai thác*** mà còn cần các chỉ số khác? Đó là vì nếu chỉ đặt chú ý vào nút có tỉ lệ thắng cao nhất, kết quả ta nhận được cũng sẽ chỉ là những nút có tiềm năng lớn vào lúc bắt đầu.

Ví dụ: Giả sử ta bắt đầu quá trình tim kiếm trên cây Monte Carlo với công thức ***UCT*** chỉ bao gồm thành phần khai thác thôi. Từ nút gốc, ta giả lập các nút con có thể có của nút và tiếp sau đó chỉ di chuyển đến những nút mà tại đó đã có ít nhất một lần thắng. Những nút ***kém may mắn*** hơn (nhớ rằng quá trình giả lập được thực hiện ngẫu nhiên trên các nút con) sẽ bị ***loại ra*** ngay lập tức mà không có cơ hội cải thiện chỉ số.

Vì lẽ đó, một thành phần thứ hai được xét tới: $c\sqrt{\frac{log(N(v))}{N(v_i)}}$ – còn được gọi là ***thành phần khám phá***. Thành phần này sẽ ưu tiên các nút chưa được khám phá hết hoặc chưa từng được di chuyển tới (chỉ số $N(v_i)$ thấp). Nhìn vào ***thành phần khám phá***, tỉ số này tỉ lệ nghịch với số lần di chuyển của nút, từ đó tăng cơ hội đối với các nút chưa được khai phá hết, hướng cây tìm kiếm đi theo hướng ***khám phá***.



Hình 7. Thành phần khai thác

Cuối cùng là hằng số c, hằng số này sẽ cân bằng tỉ lệ của cây tìm kiếm Monte Carlo giữa hai hướng là ***khai thác*** (cái có sẵn) và ***khám phá*** (cái mới).

2.4. Dừng cây tìm kiếm Monte Carlo

Ta đã có đầy đủ các yếu tố cần thiết để sử dụng cây tìm kiếm Monte Carlo nhưng vẫn còn bỏ ngõ vài câu hỏi để trả lời. Đầu tiên là ***khi nào quá trình tìm kiếm bằng cây tìm kiếm Monte Carlo sẽ kết thúc?*** Câu trả lời là: tuỳ vào ngữ cảnh, vì khả năng vận hành của máy tính đều có giới hạn, nên cách hiệu quả nhất là để cây Monte Carlo thực thi càng lâu càng tốt trong điều kiện máy tính cho phép.

Sau khi cây Monte Carlo kết thúc thực thi, phương án tối ưu nhất trên cây thường là nút có ***tần suất di chuyển $N(v_i)$ cao nhất***, vì giá trị của nút đó được ước lượng tốt nhất (giá trị này cũng sẽ cao vì nút này được khai thác thường xuyên nhất).

III. CÂY TÌM KIẾM MONTE CARLO TRONG MỘT TRÒ CHƠI

1. Alpha Go/Zero

Hệ thống của Alpha Go/Zero là một công trình phức tạp bao gồm nhiều lớp lan kỹ thuật kết hợp lại thành một thể thống nhất. Trong số đó, cốt lõi của nó có thể kể đến bao gồm:

- ***Cây tìm kiếm Monte Carlo*** (với hàm duyệt cây PUCT).

- Residual Convolutional ***Neural Networks*** - với các mạng chiến lược và giá trị được thiết kế để định lượng giá trị trạng thái và dự đoán trước xác suất của mỗi nước đi.
- ***Học tăng cường*** (Reinforcement Leanring) để máy tự huấn luyện.

1.1. Quá trình giả lập

Trong *Alpha Go*, trọng số của một lá S_L bao gồm giá trị trạng thái được tính bằng một mạng nơ-ron tích chập (Convolutional Neural Networks) 13 lớp v0, hay còn gọi là mạng giá trị (Value Networks) được huấn luyện trong 30 phút bằng các trạng thái riêng biệt với dữ liệu lấy tự những lần tự chơi của Alpha Go.

$$V(S_L) = (1-\alpha)v_0(S_L) + \alpha z_l$$

Còn đối với Alpha Zero, các kỹ sư thiết kế đã cải tiến thêm một bước nữa. Đó chính là các playout hoàn toàn không được thể hiện ra, mà thay và đó trực tiếp tính toán giá trị trạng thái với một CNN Residual Neural Networks 19 lớp (ngoài ra, trong Alpha Zero, còn có một mạng f_0 khác dùng để lưu trữ giá trị trạng thái kế tiếp và xác suất di chuyển tới vị trí đó dưới dạng một vectơ)

$$V(S_L) = f_0(S_L)$$

1.2. UCT

Trong Alpha Go và Alpha Zero, các nốt trên cây sẽ được duyệt thông qua việc cực đại hoá giá trị được tính từ một công thức biến thể của UCT:

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

với $P(v, v_i)$ là xác suất được dự đoán trước của hành động di chuyển từ nút v sang nút v_i, giá trị này được tính toán từ một mạng nơron học sâu (Deep Neural Network) gọi là mạng chiến lược (***Policy Network***). Mạng chiến lược là một hàm tiếp thu các giá trị thông tin trạng thái và cho ra các xác suất dự báo của mỗi nước đi tiếp theo. Với mục đích khoanh vùng tìm kiếm đến những nước đi tốt - khiến ***thành phần khám phá*** hướng ***thành phần khai thác*** đến những nước đi hợp lý.

1.3. Huấn luyện mạng chiến lược

Có 2 mạng chiến lược trong Alpha Go:

- ***Mạng chiến lược học có giám sát*** (SL Policy Network) được huấn luyện một cách có giám sát các dữ liệu có được từ người thật.
- ***Mạng chiến lược học tăng cường*** (RL Policy Network) - được nâng cấp và có cấu trúc tương tự mạng chiến lược có giám sát nhưng có thể tự huấn luyện thông qua học tăng cường (Reinforcement Learning).

Trong khi đó thì Alpha Zero chỉ có một mạng f_0 duy nhất bao gồm cả ***mạng giá trị*** (Value Network) và ***mạng chiến lược*** (Policy Network). Dữ liệu nó có được hoàn toàn thông qua tự huấn luyện, bắt đầu bằng các ván chơi với nước đi ngẫu nhiên. Có nhiều mạng được huấn luyện song song cùng lúc, và sau một số lần huấn luyện nhất định, mạng có kết quả huấn luyện tốt nhất trong số các mạng được huấn luyện đó sẽ được đưa ra đối chiếu với mạng nơron tốt nhất hiện tại để so sánh và cập nhật.

2. Thiết lập

Dưới góc nhìn của một trò chơi, lời giải của cây tìm kiếm Monte Carlo có thể được mô tả như sau: dựa vào ***trạng thái*** hiện tại của người chơi, chọn một ***nước đi tiếp theo tiềm năng nhất*** mà tại đó có thể dẫn người chơi đến gần hơn với chiến thắng.

Một trò chơi có môi trường phù hợp với cây tìm kiếm Monte Carlo có thể được thể hiện bằng một loạt các tính từ sau:

2.1. Trò chơi đối kháng tuần tự hai người giới hạn (Finite two person zero-sum sequential game)

Nghe có vẻ phức tạp và trừu tượng, nhưng để diễn giải một cách dễ hiểu hơn ta sẽ phân tích từng yếu tố một trong môi trường:

- ***Trò chơi (game)*** tức là một môi trường mà tại đó các người chơi có thể tương tác với nhau. Sự tương tác này có thể diễn ra trên một số hoặc nhiều phương diện.
- ***Đối kháng (zero-sum)*** có nghĩa là các lợi thế của người chơi sẽ đối chọi và tương khắc nhau, dẫn tới sự cạnh tranh khốc liệt.
- ***Hai người (two person)*** là số người chơi của trò chơi.
- ***Tuần tự (sequential)*** mang ý nghĩa lượt chơi của người chơi này sẽ kèm theo lượt chơi của người chơi khác.
- Và cuối cùng là ***giới hạn (finite)*** ám chỉ các cách tương tác qua lại sẽ có giới hạn nhất định.

Có thể nói các trò chơi như cờ vây, cờ ca rô nói riêng và các loại cờ nói chung là một trò chơi đối kháng tuần tự hai người giới hạn. Và đúng là vậy khi chỉ có hai người chơi có mặt trong một ván chơi, với số nước đi của mỗi người chơi bị giới hạn là tính cạnh tranh khốc liệt do đích đến của trò chơi mà mỗi người chơi hướng đến là xung khắc với nhau.

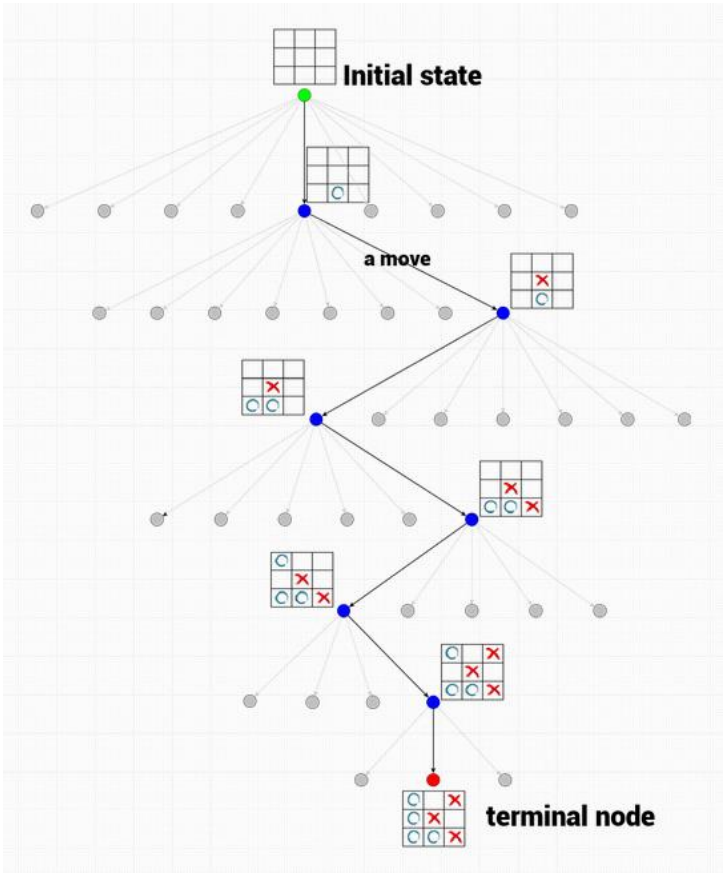
2.2. Làm thế nào để thể hiện được một trò chơi?

Để biểu diễn cấu trúc của một trò chơi dưới góc độ lập trình, ta có thể sử dụng một ***cây trò chơi*** (game tree).

Một cây trò chơi là một cây bao gồm với các nút của nó đại diện cho một ***trạng thái*** có thể xảy ra trong một trò chơi. Sự di chuyển từ một nút này tới nút con của nó (nếu tồn tại) gọi là một ***nước đi***. Số nút con của một nút là ***số nước đi khả thi***. Còn nút gốc đại diện cho ***trạng thái bắt đầu*** của trò chơi đó.

Chúng ta cũng cần phân biệt giữa ***nút kết thúc*** - nút không có nút con nào/nút lá, với việc trò chơi không thể tiếp tục được nữa. Về cơ bản, ***nút kết thúc*** là nút mà tại đó ta có được kết quả chung cuộc của ván chơi và nó có thể lập trình được.

Ví dụ với một cây trò chơi của cơ ca-rô:



Hình 8. Ví dụ minh họa cây trò chơi

Cây trò chơi trên thể hiện những đặc tính sau của cờ ca-rô:

- Ở phần đầu, ta dễ dàng thấy được ***nút gốc*** của cây, đó là ***trạng thái bắt đầu*** của một ván cờ ca-rô, khi chưa có ô nào được điền (đánh dấu xanh lá).
- Mỗi bước di chuyển từ nút này sang nút khác là một ***nước đi.***
- ***Số bước di chuyển khả thi*** là độ sâu của cây.
- Trò chơi kết thúc ở ***nút lá*** (đánh dấu đỏ).
- Cây duyệt từ nút gốc tới nút kết thúc thể hiện một ván chơi.

Cây trò chơi thật ra là một cấu trúc dữ liệu hồi quy. Nên mỗi khi khi di chuyển từ một nút này sang nút khác thì nút đó là nút gốc trong cây con của chính nó. Do đó một trò chơi có thể được coi như là tập hợp các kết quả của bài toán "nước đi tiềm năng nhất" trong mỗi lượt chơi (với mỗi lượt chơi dẫn đến một nút gốc khác). Và ta cũng không cần phải ghi nhớ đường đi tới trạng thái hiện tại, vì thông tin về vị trí hiện tại là không cần thiết với bài toán mà ta đang giải quyết.

2.3. Bài toán "nước đi tiềm năng nhất"

Mục đích cuối cùng của bài toán này chính là ***tìm ra lời giải tốt nhất*** mà có thể dựa trên trạng thái hiện tại và cây trò chơi mà ta có. Nhưng điều đó có nghĩa là gì?

Không có đáp án đúng cho câu hỏi đó. Do trong một ván chơi, ta không có cách nào xác định được chắc chắn đối thủ sẽ lựa chọn nước đi như thế nào - người đó có thể là một kiện tướng với nước đi thần sầu hoặc một người mới tập chơi với nước đi chập chững. Lấy ví dụ như trong cờ vua, khi biết được đối thủ là một người chơi nghiệp dư, từ đó có thể đoán được nước đi của của người đó rất ***lộn xộn*** và ***thiếu chặt chẽ***. Dựa vào thông tin đó, ta có thể chọn một ***chiến lược*** đơn giản để bẫy đối thủ và đi đến chiến thắng. Nhưng một ***chiến thuật*** đơn giản chắc chắn sẽ không có tác dụng với một đối thủ thông mình và đầy kinh nghiệm.

2.4. Minimax

Đó là khi ta biết về đối thủ, còn trong trường hợp ta không có bất cứ thông tin gì về năng lực và chiến thuật của đối thủ, ta có thể áp dụng một chiến lược căn bản có tên gọi là ***Minimax***. Chiến lược này sẽ xem đối thủ của ta như một đại kiện tướng và luôn tung ra nước đi tốt nhất có thể. Trong một trò chơi đối khoáng tuần tự hai người giới hạn giữa A và B (A sẽ cố gắng cực đại hoá lợi ích của mình còn B sẽ cố gắng cực tiểu hoá nó), ***thuật toán Minimax*** có thể được mô tả bằng hai công thức sau:

$$\mathrm{v_A}(\mathrm{s_i}) = \max_{\mathrm{a_i}} \mathrm{v_B}\big(move(\mathrm{s_i}, \mathrm{a_i})\big) \qquad \mathrm{v_a}(\hat{s}) = eval(\hat{s})$$

$$\mathrm{v_B}(\mathrm{s_i}) = \min_{\mathrm{a_i}} \mathrm{v_A}(\mathrm{move}(\mathrm{s_i}, \mathrm{a_i})) \qquad \mathrm{v_B}(\hat{s}) = \text{-}\mathrm{eval}(\hat{s})$$

Trong đó:

- v_A, v_B là hàm lợi ích (utility function) của người chơi A và B (utility: lợi ích, phần thưởng)
- $move$ là hàm trả trạng thái tiếp theo (một trong số nút con của nút hiện tại) của trạng thái s_i và hành động đã thực hiện tại trạng thái s_i là a_i
- $eval$ là hàm đánh giá kết quả cuối cùng của ván chơi (tại nút lá)
- $\hat{s}$ là một trạng thái kết thúc bất kỳ của trò chơi (nút lá)
- dấu (-) ở trạng thái kết thúc của v_B là để tổng phần thưởng của trò chơi là 0 (zero-sum), thể hiện tính đối kháng của trò chơi

Nói một cách đơn giản, với trạng thái *s* bất kỳ, ta muốn có được kết quả tốt nhất trong khi đối thủ của ta sẽ cố gắng cực tiểu hoá nó. Nên thuật toán này có tên gọi là ***Minimax***. Những gì cần làm chỉ là ***bung ra tất cả các nhánh có thể trên cây*** và lan truyền ngược giá trị dựa trên cấu trúc hồi quy của cây.

Nhưng việc yêu cầu cây phải bung ra tất cả các nhánh trên cây là điểm yếu của Minimax và điều đó sẽ tiêu tốn bộ xử lý và bộ nhớ khổng lồ để lưu trữ toàn bộ cây. Đặc biệt là với những trò chơi có số nước đi lớn như cờ vây và cờ vua.

Làm sao để giải quyết vấn đề đó? Một cách để làm đơn giản hoá kích thước của cây là giải thuật Alpha-Beta Prunning.

2.5. Alpha-Beta Pruning

Giải thuật ***Alpha-Beta Pruning*** là một giải thuật được dùng để cải thiện điểm yếu của ***Minimax***, nó duyệt qua cây như cách của giải thuật ***Minimax*** đồng thời sẽ loại bỏ bớt một số nhánh không cần thiết trên cây. Và điều quan trọng là ***Alpha-Beta Pruning*** sẽ không làm thay đổi kết quả chung cuộc dù có cắt tỉa đi một số lượng nhánh cây nhất định.

Với hai giải thuật ***Minimax*** và ***Alpha-Beta Pruning***, xem chừng là đã đủ để xây dựng bất kỳ một trí tuệ nhân tạo phức tạp cho mọi trò chơi. Nhưng có một điểm cốt lõi trong lập trình sẽ quyết định sự hiệu quả của thuật toán, đó là ***hàm lượng giá*** (Evaluation Function: hàm dùng để xác định giá trị trạng thái hiện tại của người chơi). Để thuật toán hoạt động tốt, buộc người phát triển phải xây dựng được nên một ***hàm lượng giá*** bền vững và hợp lý.

Ví dụ như trong cờ vua, một ***hàm lượng giá*** đơn giản có thể là tổng các quân cờ trắng trừ đi tổng các quân cờ đen trên bàn cờ. Khi bàn cờ nghiêng về phía quân trắng, giá trị trả lại của ***hàm lượng giá*** sẽ cao, trong khi giá trị đó sẽ thấp nếu lợi thế đang thuộc về quân đen. Nghe có vẻ nền tảng này là hợp lý với một ***hàm lượng giá*** và đáng để kiểm chứng, nhưng trong thực tế thì lại không đơn giản như vậy, để xác định được đúng lợi thế đang nghiêng về phía quân nào thì cần nhiều hơn là chỉ số lượng quân của hai phe. Và xây dựng được nên ***một hàm lượng giá đánh giá đúng cục diện của bàn cờ*** có thể là một vấn đề rất phức tạp và tối quan trọng, vì nó ảnh hưởng đến cách chọn lựa nước đi phù hợp nhất tại một trạng thái.

Và từ đó dẫn tới cây tìm kiếm Monte Carlo, nơi dữ liệu máy có được đến từ những nước đi ***ngẫu nhiên*** (random) mà không cần phải thông qua một ***hàm lượng giá*** nào.

IV. HƯỚNG CẢI TIẾN

1. Tìm kiếm lệch

Thông thường sau khi công thức UCT tìm được một số nước đi hợp lệ thông qua quá trình giả lập, các nước đi hợp lệ còn lại theo đó thường bị bỏ qua. Để hướng sự tìm kiếm này vào các nước đi phù hợp đã bị bỏ qua, ta có thể biến đổi công thức UCT ban đầu bằng cách thêm vào các ***giá trị mới***, hoặc các thuộc tính khác nhằm tối ưu hoá quá trình tìm kiếm và xác định xem các nước đi có thực sự tốt hay không dựa vào tri thức đặc trưng từ các ván cờ mà ta vừa thêm vào. Khi đó giá trị UCT sẽ bị lệch đi so với công thức thông thường, và từ đó việc tìm kiếm có thể trở nên chính xác hơn.

Một trong số công thức đã được cải tiến thông qua hàm lượng giá hành động có thể kể đến là:

$$UCT_{bias}(v_i,\ v) = \frac{Q(v_i)}{N(v_i)} + c\sqrt{\frac{log(N(v))}{N(v_i)}} + c_{BT}P(m_i)\sqrt{\frac{K(v)}{N(v)+K(v)}}$$

với c_{BT} là hệ số điều chỉnh ảnh hưởng độ lệch, $K(v)$ là tham số điều chỉnh tỷ lệ khi nước đi có xu hướng giảm số lần viếng thăm, và $P(m_i)$ là ***hàm lượng giá hành động*** được xây dựng từ các đặc trưng và ít tốn chi phí xây dựng hơn rất nhiều nếu so với ***hàm lượng giá trạng thái.***

2. Giả lập lệch

Trong giai đoạn giả lập, một nước đi được chọn ngẫu nhiên từ tập các nước đi khả thi trên một trạng thái bất kỳ của ván cờ. Để việc giả lập ngẫu nhiên hội tụ có thể diễn ra nhanh chóng cần có định hướng của tri thức đặc trưng rút trích được từ các ván cờ, dưới hình thức một ***hàm lượng giá hành động***. Giả sử ***hàm lượng giá hành động*** đó được biểu diễn dưới dạng hàm số $f(a)$, thì xác suất chọn nước đi a trong tập các nước đi hợp lệ A được tính bằng công thức $p(a) = \frac{f(a)}{\sum_{a\prime} f(a\prime)}$ dựa theo nguyên lý Roulette Wheel, gần như toàn bộ các giả lập Monte Carlo đều áp dụng nguyên lý này.

TÀI LIỆU THAM KHẢO

[1] C. B. Browne et al., "A Survey of Monte Carlo Tree Search Methods," in IEEE *Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1-43, March 2012, doi: 10.1109/TCIAIG.2012.2186810. https://ieeexplore.ieee.org/document/6145622

[2] Kamil. “Monte Carlo Tree Search - Beginners Guide.” *Int8.Io*, 26 Apr. 2021, *MONTE CARLO TREE SEARCH: A TUTORIAL*. (2018, December 1). IEEE Conference Publication | IEEE Xplore. https://ieeexplore.ieee.org/abstract/document/8632344

[3] Ikeda, K., Viennot, S. (2013), “Efficiency of static knowledge bias in monte-carlo tree search”, In: Computers and Games 2013.

[4] GeeksforGeeks. (2022, July 5). ML | Monte Carlo Tree Search (MCTS). https://www.geeksforgeeks.org/ml-monte-carlo-tree-search-mcts/