THÀNH PHẦN CHÍNH CỦA CÂY TÌM KIẾM MONTE CARLO VÀ HƯỚNG CẢI TIẾN

Nguyễn Quốc Huy[1], Nguyễn Khắc Chiến[2]
[1] Khoa Công nghệ thông tin, Trường Đại học Sài Gòn
[2] Bộ môn Toán – Tin học, Trường Đại học Cảnh sát nhân dân TP.Hồ Chí Minh.
nqhuy@sgu.edu.vn, nkchienster@gmail.com

TÓM TẮT *- Các chương trình đánh cờ là một phần nghiên cứu của ngành Trí tuệ nhân tạo. Các chương trình truyền thống được xây dựng trên cây tìm kiếm Minimax, Alpha-Beta với hàm lượng giá được xây dựng dựa trên tri thức của người chơi cờ. Việc thiết kế một hàm lượng giá tốt thường rất khó, hơn nữa các cây tìm kiếm truyền thống chỉ phù hợp với những trò chơi có hệ số phân nhánh thấp. Cây tìm kiếm Monte Carlo là một hướng tiếp cận hiện đại và hiệu quả trên nhiều trò chơi có hệ số phân nhánh cao như cờ Vây. Mô hình cây tìm kiếm Monte Carlo được kết hợp từ Cây tìm kiếm, Học tăng cường và giả lập Monte Carlo. Dưới góc nhìn này, ta có thể cải tiến hiệu suất của cây tìm kiếm Monte Carlo bằng cách tìm hiểu phương pháp cải tiến Học tăng cường, và cải tiến giả lập Monte Carlo. Bài báo này cô đọng cách thức cải tiến hiệu quả cây tìm kiếm Monte Carlo và chỉ ra một số kết quả thực nghiệm bước đầu rất hiệu quả.*

Từ khóa *- Monte Carlo Tree Search, Evaluation Function, Reinforcement Learning, Board Games, Feature Selection.*

I. GIỚI THIỆU

Tìm kiếm vừa là phương pháp giải quyết bài toán vừa là phương tiện để chương trình thể hiện tính thông minh của nó, nhất là trong các trò chơi đối kháng hai người chơi. Khi đối mặt với các bài toán phức tạp, máy tính phải xét quá nhiều trạng thái và cần quá nhiều thời gian tính toán. Thông thường tri thức bổ sung của các lĩnh vực liên quan được dùng trong việc thiết kế hàm lượng giá trạng thái để giảm không gian tìm kiếm. Dùng tri thức bổ sung tiết kiệm đáng kể thời gian giải quyết bài toán. Hiện nay tri thức bổ sung đóng vai trò quan trọng trong việc loại bỏ nhánh. Việc đánh giá trạng thái có tốt hay không tùy thuộc vào chất lượng của hàm lượng giá trạng thái.

Tuy nhiên, nếu độ sâu trung bình của cây tìm kiếm lớn thì còn có thể kiểm soát bằng hàm lượng giá trạng thái, nhưng hệ số phân nhánh lớn thì nên dùng cách khác. Trong bài toán giải quyết các trò chơi, nếu có nhiều nước đi "có khả năng, nhưng không hứa hẹn", thì nên tránh những nước đó càng nhiều càng tốt, để giảm tổng chi phí tìm kiếm trên cây. Tìm kiếm Minimax với phương pháp tỉa Alpha-Beta thường được dùng để tỉa các nước đi không cần thiết theo cơ chế nhánh cận. Để phương pháp tỉa Alpha-Beta hiệu quả, thì thứ tự các khả năng đi được là khá quan trọng, thường người ta dùng "hàm lượng giá hành động" để sắp thứ tự. Đôi khi các hàm lượng giá hành động còn được dùng để giới hạn số nước tìm kiếm. Lấy ví dụ, một số chương trình cờ Vây (Nomitan) chỉ tìm kiếm 20 nước đi trên khoảng 300 khả năng có thể đi được [16].

Hàm lượng giá trạng thái cần thiết cho tìm kiếm Alpha-Beta, nhưng trong một số trò chơi như cờ Vây, rất khó để thiết kế một hàm lượng giá trạng thái. Cây tìm kiếm Monte Carlo [4] là một phương pháp mới, có thể áp dụng cho hầu hết các trò chơi mà không cần đến tri thức hay dùng trong hàm lượng giá trạng thái. Giá trị đánh giá của một nút có thể được tính theo nhiều giả lập Monte Carlo mà ở đó các nước đi được sắp xếp ngẫu nhiên cho đến khi kết thúc ván cờ, và kết quả thắng thua mỗi ván được dùng để tích lũy thành giá trị mỗi nút. Gần đây cây tìm kiếm Monte Carlo đã được sử dụng rộng rãi đặc biệt là trong cờ Vây. Hầu hết tất cả những chương trình cờ Vây mạnh đều sử dụng cây tìm kiếm Monte Carlo hoặc các biến thể của nó [10, 11, 14, 20], các chương trình như MoGo/MoGoTW, Crazy Stone, Fuego, Many Faces of Go và Zen đã đạt được một mức độ chơi tốt nhất trong 10 năm trước đây. Các chương trình này hiện nay có thể đấu sòng phẳng với những người chơi chuyên nghiệp trên bàn cờ Vây 9x9 và với những người chơi không chuyên trên bàn cờ Vây 19x19. Cây tìm kiếm Monte Carlo nhận được sự quan tâm của nhiều nhà nghiên cứu vì hiệu quả và ưu điểm của nó là không cần đến hàm lượng giá trạng thái, và hiện nay vẫn còn nhiều câu hỏi đại loại như làm cách nào để cải tiến cây tìm kiếm Monte Carlo, ví dụ loại giả lập nào tốt hơn, quản lý việc khai thác và khám phá thế nào là hiệu quả.

Do tính hiệu quả của cây tìm kiếm Monte Carlo, đồng thời nhận được khá nhiều sự quan tâm của cộng đồng nghiên cứu trí tuệ nhân tạo, nên bài viết này tập trung phân tích các thành phần cơ bản cấu thành cây tìm kiếm Monte Carlo. Thật ra, dưới một góc nhìn khác thì mô hình cây tìm kiếm Monte Carlo được tổng hợp hiệu quả từ ba mô hình kinh điển: Cây tìm kiếm + Học tăng cường + Giả lập Monte Carlo. Chúng ta có thể so sánh mô hình cây tìm kiếm truyền thống và mô hình cây tìm kiếm Monte Carlo trong bảng 1.

Bảng 1. So sánh cây tìm kiếm truyền thống và Monte Carlo

	Cây tìm kiếm truyền thống	**Cây tìm kiếm Monte Carlo**
Mô hình	Minimax + Tỉa Alpha-Beta + tri thức Heuristic.	Cây tìm kiếm + Học tăng cường + giả lập Monte Carlo.
Tích cực	Phù hợp cây có hệ số phân nhánh nhỏ, và có sẵn hàm lượng giá trạng thái tốt.	Không phụ thuộc tri thức Heuristic, phù hợp trên cây có hệ số phân nhánh rất lớn.

	Cây tìm kiếm truyền thống	Cây tìm kiếm Monte Carlo
Hạn chế	Chi phí xây dựng hàm đánh giá trạng thái thường rất cao.	Phải cân bằng việc khai thác và khám phá, phải có chiến thuật giả lập hiệu quả.

Bài báo được tổ chức như sau. Phần I giới thiệu vấn đề cần quan tâm, phần II là bài toán "Tên cướp nhiều tay" trong học tăng cường, cây tìm kiếm Monte Carlo được trình bày trong phần III, phần IV bàn luận về vai trò của học tăng cường trong cây tìm kiếm Monte Carlo, phần V cô đọng cách thức cải tiến hiện quả, một số kết quả thực nghiệm ban đầu được chỉ ra trong phần VI, và phần cuối là kết luận và các vấn đề mở.

II. HỌC TĂNG CƯỜNG

Trong khi học có giám sát dựa trên cặp các thông tin đầu vào và đầu ra chính xác để xây dựng mối liên hệ giữa thông tin đầu vào và đầu ra, mối liên hệ này được xem như là bộ phân lớp dùng để dự đoán thông tin đầu ra khi biết được một thông tin đầu vào nào đó. Học tăng cường khác với học có giám sát ở chỗ không bao giờ có được các cặp thông tin đầu vào và đầu ra chính xác, các hành động gần tối ưu trong mỗi trạng thái cũng chưa chắc mang lại kết quả tối ưu về lâu về dài. Học tăng cường là một phương pháp học dành cho tác nhân đang ở một môi trường không cố định với mục tiêu cực đại hóa kết quả cuối cùng về lâu về dài, ở trên môi trường này cần có sự cân bằng giữa việc khai thác tri thức hiện hành và khám phá tri thức mới từ những vùng chưa khai thác. Việc cân bằng này trong học tăng cường hầu như giống bài toán nổi tiếng multi-armed bandit (tên cướp nhiều tay) trong lý thuyết xác suất (xem hình 1).

Bài toán tên cướp nhiều tay mô tả một người đánh bạc đứng trước nhiều máy đánh bạc, mỗi máy đánh bạc được xem như "một cánh tay" của tên cướp. Người đánh bạc không biết bất kỳ thông tin nào về các máy đánh bạc và các máy đánh bạc có thể khác nhau. Bài toán tên cướp được định nghĩa như sau [2, 17]:

Bài toán tên cướp với K cánh tay (các hành động) được xác định là dãy các phần thưởng (kết quả) ngẫu nhiên, $X_{i,t}$, i = 1, 2,…, K, t ≥ 1, trong đó mỗi i là chỉ số của máy đánh bạc ("cánh tay" của tên cướp). Việc chơi liên tục máy đánh bạc thứ i sẽ mạng lại các phần thưởng $X_{i,1}$, $X_{i,2}$,…

Câu hỏi đặt ra là người đánh bạc trong một thời điểm cần quyết định nên chơi máy nào, chơi bao nhiêu lần, với thứ tự như thế nào để cuối buổi chơi mang về số tiền thưởng nhiều nhất. Tất nhiên trong mỗi lần chọn một máy sẽ có một tỷ lệ thắng thua ngẫu nhiên nào đó mà người chơi không biết trước. Vì tính chất quan trọng của bài toán này nên John C. Gittins đã đưa ra một chính sách tối ưu theo kiểu Markov vào năm 1979 [13] để cực đại hóa phần thưởng mong muốn.

Các thành phần cơ bản trong mô hình học tăng cường bao gồm:

- Tập các trạng thái của môi trường *S*,
- Tập các khả năng có thể xảy ra *A*,
- Luật chuyển đổi giữa các trạng thái với xác suất $P_{s,s'}^{a} = p(s, a, s') \in [0,1]$,
- Luật xác định phần thưởng tức thì vô hướng cho một lần chuyển đổi *R(s)*,
- Luật mô tả những gì mà tác nhân quan sát.

Mục tiêu là tìm chính sách (chiến lược) $\pi: S \rightarrow A$ sao cho phần thưởng tích lũy được tối đa.

Hình 1. Người đánh bạc đang giải bài toán "Tên cướp nhiều tay"

Các luật thường là ngẫu nhiên. Một tác nhân học tăng cường tương tác với môi trường trong nhiều bước thời gian rời rạc. Tại mỗi bước thời gian *t*, tác nhân có thông tin quan sát o_t, tương ứng là phần thưởng r_t. Rồi tác nhân chọn hành động a_t từ các khả năng có thể chọn, quá trình cứ tiếp tục như vậy trên môi trường của tác nhân. Môi trường chuyển sang một trạng thái mới s_{t+1} và phần thưởng r_{t+1} ứng với chuyển đổi (s_t, a_t, s_{t+1}). Xác suất đạt trạng thái s_{t+1} khi hành động a_t được chọn từ trạng thái s_t gọi là $p(s_t, a_t, s_{t+1})$ thông qua chính sách π, $r_{t+1} = R(s_{t+1})$ gọi là phần thưởng mà tác nhân nhận được.

III. CÂY TÌM KIẾM MONTE CARLO

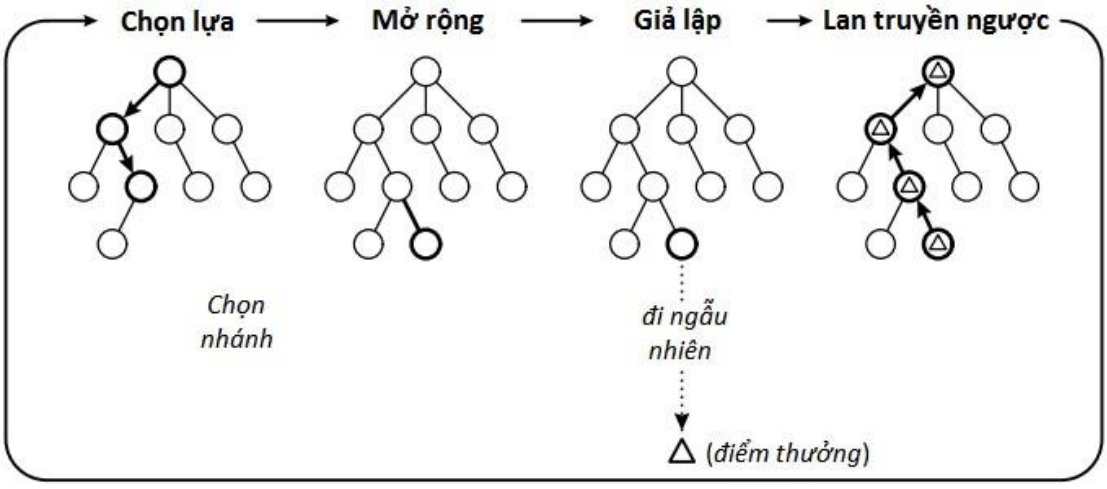
Các phương pháp Monte-Carlo hiện nay đang rất hiệu quả cho các chương trình cờ Vây. Từ năm 1993 Brugmann [5] đã áp dụng phương pháp này cho cờ Vây, và đến năm 2006, Kocsis và Szepesvari [17] mới xây dựng thuật toán UCT cho cờ Vây. Hiện nay, thuật toán UCT và các biến thể của nó là các thuật toán chủ đạo cho các chương trình có sử dụng cây tìm kiếm Monte Carlo. Cây tìm kiếm Monte Carlo là thuật toán tìm kiếm dựa theo phương pháp lấy mẫu dùng giả lập ngẫu nhiên để ước lượng tỷ lệ thắng thua của một trạng thái bàn cờ [4] nhằm tìm ra nước đi tốt nhất và để cân bằng giữa việc khám phá và khai thác của tất cả các nước đi [12]. Điểm mấu chốt của cây tìm kiếm Monte Carlo so với các phương pháp tìm kiếm cổ điển như Alpha-Beta 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 cây tìm kiếm Monte Carlo 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. Tuy nhiên 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 lần đầu tiên được triển khai trong trò chơi CRAZY STONE, đã chiến thắng trong giải đấu cờ Vây kích thước 9 × 9 tại Computer Olympiad năm 2006. Cùng với sự xuất hiện của thuật toán UCT [17], thành công lớn của thuật toán cây tìm kiếm Monte Carlo đã kích thích được sự quan tâm sâu sắc các nhà lập trình cờ Vây. Gần đây, cây tìm kiếm Monte Carlo đã được cải thiện hiệu suất trong các lĩnh vực khác nhau, như các trò chơi hai người chơi cờ Vây [20], trò chơi LOA (2010) [21] và Hex (2010) [1]. Hơn nữa, cây tìm kiếm Monte Carlo đã cho thấy thành công trong các lĩnh vực khác như trò chơi chiến thuật thời gian thực [4], trò chơi Ms Pac-Man (2012) [18] và Physical Traveling Salesman (2012) [19]. Ngoài ra, cây tìm kiếm Monte Carlo còn ứng dụng hiệu quả trong các lĩnh vực cuộc sống như tối ưu hóa, lập lịch và bảo mật [4].

1. Cấu trúc cây tìm kiếm Monte Carlo

Cây tìm kiếm Monte Carlo là một quá trình lặp đi lặp lại bốn bước trong một khoảng thời gian hữu hạn (xem hình 2). ***Chọn lựa***, từ một nút gốc (trạng thái bàn cờ hiện hành) cho đến nút lá, vì vậy sẽ có nhiều hướng đi được mang ra đánh giá. ***Mở rộng***, thêm một nút con vào nút lá của hướng được chọn trong bước chọn lựa, việc mở rộng không thực hiện trừ khi kết thúc ván cờ tại nút lá. ***Giả lập***, một ván cờ giả lập được chơi từ nút mở rộng, sau đó kết quả thắng thua của ván cờ giả lập sẽ được xác định. ***Lan truyền ngược***, kết quả thắng thua sẽ được cập nhật cho tất cả các nút của hướng được chọn theo cách lan truyền ngược.

Cây trò chơi tăng trưởng sau mỗi lần lặp của cây tìm kiếm Monte Carlo, tăng trưởng rộng hơn và sâu hơn. Nước đi hứa hẹn là nút con nào có tỷ lệ thắng cao hơn được chọn trong giai đoạn chọn lựa, và rồi các cây con cũng tăng trưởng ngày càng rộng hơn và sâu hơn, và việc ước lượng sẽ ngày càng chính xác hơn. Sau khi kết thúc thời gian tìm kiếm, nút con nào được thăm nhiều nhất tại nút gốc sẽ được chọn để đi.



Hình 2. Cây tìm kiếm Monte Carlo

2. Thuật toán cây tìm kiếm Monte Carlo

Thuật toán này có thể sử dụng được cho bất cứ bài toán đối kháng nào, trạng thái và hành động rời rạc. Mỗi nút v có bốn thành phần dữ liệu: trạng thái gọi là $s(v)$, hành động gọi là $a(v)$, tổng điểm thưởng giả lập $Q(v)$, và số lần viếng thăm $N(v)$. Kết quả của hàm UCT_SEARCH(s_0) là một khả năng có thể xảy ra tại trạng thái s_0 sao cho nút con có số lần viếng thăm nhiều nhất được chọn.

function *UCT_SEARCH(s_0)*

create root node v_0 with state s_0

while *within computational budget* ***do***

```
        v := Selection(v0, C)
        vl := Expand(v)
        reward := Simulation(s(vl))
        BackPropagation (vl, reward)
    return the most visited child node of v0
function Selection(v, C)
    while v is not terminal do
        v := arg max UCB_{v'∈v_child}'
return v
function Expand(v)
    choose a ∈ random actions from A(s(v))
    add a new child v' to v
    with s(v') = f(s(v),a)
    and a(v') = a
return v'

function Simulation(v)
    while s is non-terminal do
        choose a ∈ A(s) uniformly at random
        s := f(s,a)
return reward for state s

function BackPropagation(v, reward)
    while v is not null do
        N(v) := N(v) + 1
        Q(v) := Q(v) + reward
        reward := - reward
        v := parent of v
```

Hàm *Selection()* thực hiện công việc tìm nút con của nút *v* dựa trên giá trị *UCB [2]* được tính theo công thức (4.1), hằng số *C* dùng để điều chỉnh sự khám phá hay khai thác.Từ trạng thái nút *v* được chọn theo hàm *Selection()*, hàm *Expand()* chọn một khả năng bất kỳ trong số các khả năng để tạo thành một nút mở rộng. Hàm *Simulation()* thực hiện công việc giả lập một ván cờ bắt đầu từ nút mới được mở rộng của cây và trả về giá trị thắng thua của ván cờ giả lập đó. Giá trị thắng thua và giá trị viếng thăm sẽ được cập nhật từ nút mở rộng đến các nút cha bởi hàm *BackPropagation()*, lưu ý trong hàm này cứ mỗi bước cập nhật thì giá trị thắng thua được nhân cho giá trị -1 vì cứ mỗi bước, lượt đi bị thay đổi. Hàm *UCT_SEARCH()* thực hiện công việc của cây tìm kiếm Monte Carlo rõ ràng có bốn giai đoạn thực hiện thông qua bốn hàm *Selection(), Expand(), Simulation(), và BackPropagation().*

IV. MỐI LIÊN HỆ GIỮA HỌC TĂNG CƯỜNG VÀ CÂY TÌM KIẾM MONTE CARLO

Nói đến phương pháp Monte Carlo là nói đến giả lập, đây là một phương pháp lấy tập mẫu nhỏ để kết luận cho tập rất lớn. Phương pháp này dùng để tránh vét cạn trong những không gian tìm kiếm khổng lồ cần xem xét nhằm giảm chi phí tính toán. Vì vậy, vấn đề lấy mẫu nhỏ làm sao để biểu diễn chính xác tập khổng lồ là công việc không đơn giản. Có hai vấn đề cần quan tâm để việc lấy mẫu chính xác là: (1) kỹ thuật lập trình cần tối ưu để số lần giả lập được thực hiện càng nhiều càng tốt trong một khoảng thời gian giới hạn, và (2) việc cân bằng giữa khai thác và khám phá được thực hiện một cách hợp lý.

Với mỗi trạng thái, trong khoảng thời gian giới hạn chừng 10 giây để chọn một khả năng hợp lý trong số các khả năng có thể chọn. Thông thường tìm kiếm tất cả các hướng trong một khoảng thời gian giới hạn thì không hiệu quả. Trong cây tìm kiếm Monte Carlo ở bước chọn lựa, cần xác định một đường đi hợp lý tính từ nút gốc đến nút lá của

cây tìm kiếm hiện tại và thông tin về số lần viếng thăm cũng như tỷ lệ thắng thua được lưu trữ trong mỗi nút nằm trên đường đi đó. Trong bước này, việc cân bằng giữa khai thác và khám phá được kiểm soát theo một chính sách nào đó. Nói cách khác những vùng chứa các nước đi hứa hẹn sẽ được chọn thường xuyên gọi là khai thác, nhưng những vùng chứa các nước đi ít hứa hẹn vẫn có cơ hội được thử để tránh việc ước lượng thiếu khách quan thì được gọi là khám phá. Như trong thật toán UCT_SEARCH, vai trò hai hàm *Selection()* và *Expand()* là tìm vùng để giả lập, vùng để khai thác hay vùng để khám phá là tùy thuộc vào công thức UCB [2], một công thức được dựa trên ý tưởng của bài toán "Tên cướp nhiều tay". Việc chọn vùng rất quan trọng trong việc lấy mẫu sao cho tập mẫu không lớn nhưng biểu diễn chính xác tập không gian khổng lồ mà ta không thể vét cạn được.

Việc cân bằng như vậy là cốt lõi của bài toán "Tên cướp nhiều tay" - MAB [2], ta có thể hình dung người đánh bạc chính là người chơi cờ trong lượt đi hiện hành, và các máy đánh bạc chính là các nước đi hợp lệ trên trạng thái hiện hành, và nhiệm vụ của người chơi sẽ chọn nước đi hợp lệ nào để góp phần cho chiến thắng cuối cùng của ván cờ. Để cân bằng giữa việc khai thác và khám phá trong giai đoạn này thì công thức UCB (Upper Confidence Bounds) – Cận tin cậy trên – được đưa ra (công thức 4.1) [2], và được tích hợp vào cây tìm kiếm Monte Carlo trở thành thuật toán UCT (Upper Confidence Bounds for Tree) [17].

$$UCB_i = \frac{w_i}{n_i} + C\sqrt{\frac{\ln N}{n_i}} \qquad (4.1)$$

Trong công thức này tỷ số w_i/n_i là tỷ lệ thắng của nút con i, w_i là số lần thắng ván đấu giả lập mà có đi qua nút này, n_i là số lần nút này được đi qua, N là số lần nút cha được đi qua, và C là tham số có thể điều chỉnh. Tại nút gốc, nút con được chọn là nút có giá trị UCB cao nhất, rồi các nút con của nút hiện hành được so sánh với nhau bởi giá trị UCB và tiếp tục chọn nút có giá trị cao nhất. Vế thứ nhất trong công thức (4.1) có giá trị càng cao thì khả năng khai thác càng cao vì nút con có tỷ lệ thắng cao sẽ được chọn. Vế thứ hai của (4.1) thể hiện sự tăng cường khám phá vì nút có số lần đi qua càng thấp thì càng có khả năng được chọn. Nếu hằng số C thấp, công thức sẽ nghiêng về khai thác, ngược lại nếu C cao, công thức sẽ nghiêng về khám phá.

Cho đến nay, nhiều cải tiến của cây tìm kiếm Monte Carlo đã được đề xuất và phát triển, chẳng hạn như Rapid Action Value Estimation (RAVE) do Gelly và Silver đề xuất (2007 [10] và 2011 [11]), và phương pháp Progressive Bias do Chaslot và đồng sự đề xuất năm 2007 [8] nhằm để tăng cường hiệu suất của cây tìm kiếm Monte Carlo. Rất nhiều nghiên cứu toàn diện cũng đã được tập trung vào các chính sách và chất lượng nhằm cải thiện giai đoạn giả lập tốt hơn (Coulom 2007 [9], Chaslot và đồng sự 2009 [7], Hendrik 2010 [14]).

V. HƯỚNG CẢI TIẾN

Dù cây tìm kiếm Monte Carlo có thể làm việc không cần đến tri thức đặc trưng của trò chơi. Tuy nhiên, tri thức rất cần thiết để cải tiến các phương pháp học tăng cường, và học tăng cường là một phần của cây tìm kiếm Monte Carlo. Vì vậy, có rất nhiều kỹ thuật cải tiến cây tìm kiếm Monte Carlo dựa trên tri thức. Các kỹ thuật được thể hiện như sau:

1. Tìm kiếm lệch

Thông thường công thức UCB chỉ tìm một số nước đi hợp lệ một cách công bằng, và các nước đi hợp lệ còn lại thường được bỏ qua. Tuy nhiên, ta có thể thêm giá trị cộng thêm vào giá trị UCB của một nước đi để điều chỉnh các nước đi thực sự tốt hay xấu dựa vào tri thức đặc trưng từ các ván cờ. Lúc đó giá trị UCB sẽ bị lệch đi so với công thức thông thường, và vì thế việc tìm kiếm cũng chính xác hơn. Theo cách đó, có rất nhiều công thức được đưa ra, như Ikeda và Simon [16] đã đưa ra công thức (5.1) rất chuẩn mực và tinh gọn từ những công thức trước đó vào năm 2013 để cải tiến việc tìm kiếm thông qua tri thức đặc trưng từ các ván cờ.

$$UCB_{Bias}(i) = \frac{w_i}{n_i} + C\sqrt{\frac{\ln N}{n_i}} + C_{BT}P(m_i)\sqrt{\frac{K}{N+K}} \qquad (5.1)$$

Trong công thức này, giá trị thêm vào gồm những thành phần như C_{BT} là hệ số điều chỉnh ảnh hưởng độ lệch, K 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. Hàm lượng giá hành động được xây dựng ít tốn chi phí và hoàn toàn khác với hàm lượng giá trạng thái vốn rất tốn chi phí xây dựng.

2. Giả lập lệch

Trong giai đoạn giả lập, một nước đi được chọn ngẫu nhiên từ tập các nước đi hợp lệ trên một trạng thái của ván cờ. Tuy nhiên, để việc giả lập ngẫu nhiên hội tụ nhanh cần có định hướng của tri thức đặc trưng từ các ván cờ. Lúc đó cần vai trò của hàm lượng giá hành động, hàm này cũng được xây dựng từ tri thức nhưng ít tốn chi phí hơn việc xây dựng một hàm lượng giá trạng thái. Có nhiều phương pháp xác suất chọn lựa nước đi ngẫu nhiên như Roulette Wheel, chọn theo tua, chọn theo mẫu, và chọn theo xếp hạng. Giả sử hàm lượng giá hành động là $f(a)$, thì xác suất chọn nước

đi a trong tập các nước đi hợp lệ A được tính như $p(a) = f(a) / \sum_{a' \in A} f(a')$ theo Roulette Wheel, gần như toàn bộ các giả lập Monte Carlo đều theo cách này.

Thông thường giá trị $P(m_i)$ trong công thức (5.1) và hàm $f(a)$ là một, và đây chính là hàm lượng giá hành động được xây dựng dựa trên tri thức đặc trưng của trò chơi.

VI. KẾT QUẢ THỰC NGHIỆM

Để minh họa cho tính hiệu quả của Cây tìm kiếm Monte Carlo trong các trò chơi có hệ số phân nhánh cao so với phương pháp truyền thống Minimax kết hợp cắt tỉa Alpha-Beta, nhóm tác giả đã tiến hành cài đặt thử nghiệm trên cờ Othello. Cờ Othello hay còn gọi là Reversi là một trò chơi trên bàn cờ và là môn thể thao trí tuệ dành cho hai người chơi. Bàn cờ được chia lưới ô vuông 8 x 8 còn những quân cờ có hình dạng giống đồng xu có hai mặt màu nhạt và sẫm (có thể là màu trắng hoặc đen). Các nhà phân tích đã ước tính số lượng vị trí hợp lệ ở cờ Othello nhiều nhất là 10^{28} và nó có độ phức tạp xấp xỉ 10^{58}.

Để so sánh chiến lược Alpha-Beta và chiến lược Cây tìm kiếm Monte Carlo, nhóm tác giả đã tiến hành thử nghiệm như sau: Cài đặt chương trình thứ nhất bằng ngôn ngữ C# gọi là OthelloMCTS1 với công thức UCB (4.1) thiết lập hằng số C=0.85 và giả lập thông thường, chương trình thứ 2 bằng ngôn ngữ C# gọi là OthelloMCTS2 với công thức UCB (5.1) thiết lập các hệ số C_{BT}=0.5, K=5000, C=0.85 và giả lập lệch. Sau đó tiến hành chơi với cùng một chương trình Othello [15] trên Internet được cài đặt theo chiến lược Alpha-Beta mức tốt gọi là Riversi, kích thước bàn cờ là 8x8. Chúng ta có thể tham khảo và chơi thử để xác định độ mạnh của Riversi. Chương trình OthelloMCTS1 và OthelloMCTS2 sử dụng cây tìm kiếm Monte Carlo.

Trong chương trình OthelloMCTS2, hàm lượng giá hành động được xây dựng dựa trên tri thức của cờ Othello mà Michael Buro [6] tìm ra như trong hình 3. Hàm lượng giá hành động được xây dựng theo mô hình Bradley-Terry (công thức 6.1) như đề xuất của Remi Coulom [9]. Như vậy chúng ta có thể hiểu chương trình OthelloMCTS2 là một cải tiến của chương trình OthelloMCTS1. Cách thức cải tiến được mô tả trong phần V của bài báo, chủ yếu vào giai đoạn chọn lựa và giả lập dựa trên tri thức đặc trưng của bàn cờ. Tri thức đặc trưng cụ thể trong thực nghiệm này chính là các mẫu của từng vị trí trên bàn cờ Othello như trong hình 3. Hình 3 mô tả các mẫu đặc trưng của 9 vị trí cần thiết trên bàn cờ 8x8, do bàn cờ có tính đối xứng nên chỉ cần tìm 9 vị trí là đủ.

$$P(2\_4 \text{ against } 1\_2\_5 \text{ and } 1\_3\_6\_7) = \frac{\gamma_2\gamma_4}{\gamma_2\gamma_4 + \gamma_1\gamma_2\gamma_5 + \gamma_1\gamma_3\gamma_6\gamma_7} \qquad (6.1)$$

Công thức (6.1) có ý nghĩa như sau: Giả sử trạng thái hiện hành có 3 nước đi A, B, C. Nước đi A được xây dựng dựa trên đặc trưng 2 và 4, nước đi B được xây dựng dựa trên đặc trưng 1, 2, và 5, nước đi C được xây dựng dựa trên đặc trưng 1, 3, 6, và 7. Công thức (6.1) tính xác suất của nước đi A so với nước đi B và C. γ_i là trọng số của đặc trưng thứ i.

Bảng 2: Kết quả thực nghiệm

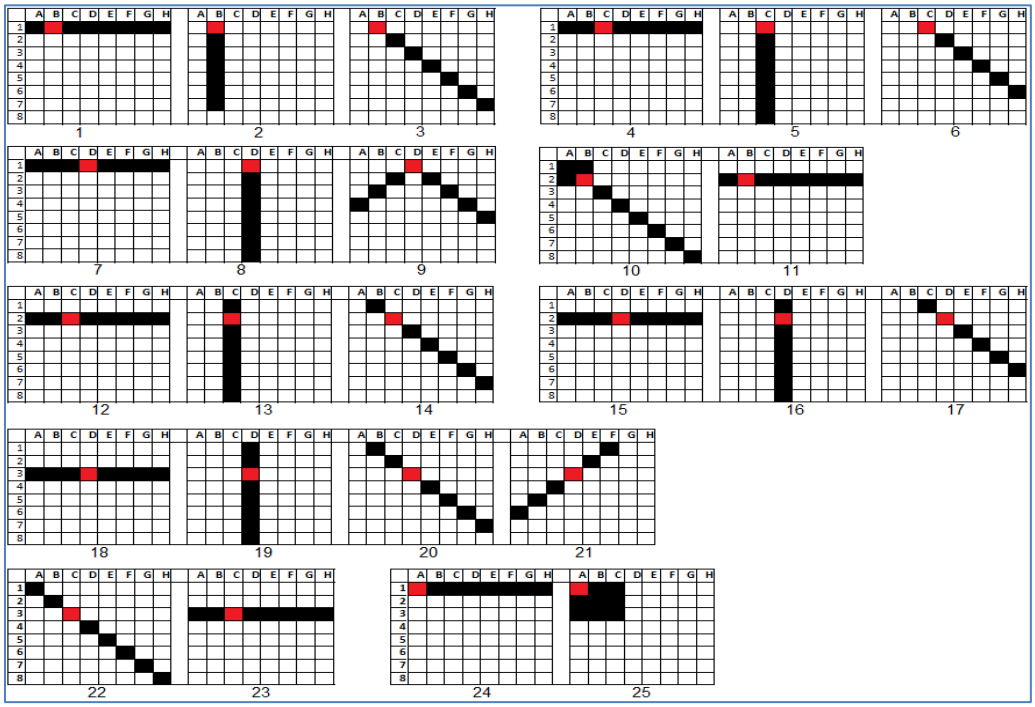
Thời gian suy nghĩ của MCTS: 4s	OthelloMCTS1	OthelloMCTS2
Riversi	314/1000	502/1000

Kết quả thực nghiệm cho thấy chương trình OthelloMCTS1 chỉ thắng 314 ván trên 1000 ván đấu với chương trình Riversi, nhìn kết quả như vậy chúng ta có thể ngạc nhiên khi nhận định tại sao chương trình Riversi được cài đặt theo cây tìm kiếm Alpha-Beta lại có kết quả tốt hơn chương trình OthelloMCTS1 được cài đặt theo cây tìm kiếm Monte Carlo. Tuy nhiên chúng tôi xin nhắc lại hai lý do sau đây để tiện so sánh:

- Chi phí xây dựng hàm lượng giá trạng thái cho Riversi rất cao (tốn nhiều thời gian tích lũy tri thức và rút trích tri thức heuristic).

- Độ phức tạp của trò chơi Riversi vẫn còn phù hợp với cây tìm kiếm Alpha-Beta (những trò chơi cờ Vây, cờ Connect-6, cờ Shogi là những trò chơi có độ phức tạp cao phù hợp hơn với cây tìm kiếm Monte Carlo. Tuy nhiên, trò chơi Riversi quen thuộc với chúng ta, và dễ cài đặt nên chúng tôi chọn làm chương trình minh họa).

Chương trình OthelloMCTS2 thắng 502 ván trên 1000 ván đấu với chương trình Riversi. Như vậy chương trình OthelloMCTS2 hoàn toàn mạnh hơn OthelloMCTS1 với tỷ lệ 50.2% so với tỷ lệ 31.4%. Tuy nhiên, các đặc trưng trên hình 3 vẫn chưa là những đặc trưng tối ưu. Tỷ lệ thắng sẽ cao hơn khi các đặc trưng được tối ưu, và chi phí tối ưu tự động hoàn toàn nhỏ hơn rất nhiều so với chi phí xây dựng hàm lượng giá trạng thái như trong chương trình Riversi.



Hình 3. Các đặc trưng của 9 vị trí trên bàn cờ Othello [6]

VII. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

Có nhiều cải tiến cây tìm kiếm Monte Carlo ở hai giai đoạn ***Chọn lựa*** và ***Giả lập***, nhưng không có nhiều cải tiến cây Monte Carlo trên giai đoạn ***Mở rộng*** và ***Lan truyền ngược***. Theo như góc nhìn mới, ít có tài liệu đề cập thì mô hình cây tìm kiếm Monte Carlo là sự kết hợp của Cây tìm kiếm, Học tăng cường và giả lập Monte Carlo. Có rất nhiều tài liệu đề cập đến vấn đề cải tiến việc Học tăng cường, cũng như cải tiến việc giả lập Monte Carlo. Cải tiến hai thành phần này có khả năng cải tiến được hiệu suất của cây tìm kiếm Monte Carlo. Dựa vào một số tài liệu về cải tiến phương pháp học tăng cường và cải tiến giai đoạn giả lập, hầu như toàn bộ các công trình cải tiến hiệu quả đều dựa vào tri thức là các đặc trưng của trò chơi. Dựa vào công trình [4] và một số công trình gần đây về cải tiến hiệu suất cây tìm kiếm Monte Carlo, vấn đề cải tiến dựa vào tri thức là đặc trưng của các bàn cờ trò chơi vẫn là chủ đề cần được quan tâm và nghiên cứu trong thời gian tới.

Bảng 3: So sánh độ phức tạp giữa các trò chơi [22]

Trò chơi	**Kích thước bàn cờ**	**Không gian nước đi hợp lệ (log_{10})**	**Kích thước cây tối đa (log_{10})**
Tic-tac-toe	9	3	5
Reversi (Othello)	64	28	58
Cờ vua	64	47	123
Connect6	361	172	140
Shogi	81	71	226
Cờ vây (19x19)	361	171	360

Chúng tôi cũng có kế hoạch tối ưu tri thức đặc trưng từ những ván cờ Othello có sẵn và bổ sung tri thức vào hai trong bốn giai đoạn của cây tìm kiếm Monte Carlo đó là giai đoạn ***Chọn lựa*** và giai đoạn ***Giả lập*** để nâng cao hiệu quả của cây tìm kiếm. Với hy vọng chương trình mới sẽ mạnh hơn chương trình OthelloMCTS2 của thực nghiệm hiện hành khi hàm lượng giá hành động được xây dựng trên các đặc trưng tối ưu hơn các đặc trưng trong hình 3.

Trò chơi Riversi có độ phức tạp thấp hơn rất nhiều so với cờ Vây, cờ Connect-6, cờ Shogi (xem bảng 3). Chúng tôi sẽ áp dụng phương pháp đã thực nghiệm tốt trên cờ Riversi vào trong các loại cờ có độ phức tạp cao nhằm nâng cao vai trò của cây tìm kiếm trong tương lai gần. Từ đó cho thấy hướng nghiên cứu của bài báo hoàn toàn thú vị, mới lạ và rất tiềm năng trong công việc nghiên cứu.

VIII. TÀI LIỆU THAM KHẢO

[1] Arneson, B., Hayward, R. B., and Henderson, P. (2010), "Monte-Carlo tree search in Hex", *IEEE Trans. Comput. Intell. AI in Games*, 2(4), pp. 251– 258.

[2] Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002), "Finite-Time Analysis of the Multiarmed Bandit Problem," *Machine Learning*, vol. 47, no. 2-3, pp. 235–256.

[3] Bouzy, B., Cazenave, T. (2001), " Computer go: An AI oriented survey", *Artificial Intelligence*, 2001.

[4] Browne, C., Powley, Whitehouse, Lucas, Cowling, Tavener, Perez, Samothrakis, Colton (2012), "A survey of Monte Carlo tree search methods", *IEEE transactions on computational intelligence and AI in games 4*, pp. 1 – 43.

[5] Brugmann, B. (1993), "Monte Carlo Go", *In: AAAI Symposium on Games*.

[6] Buro, M. (2003), "The evolution of strong othello programs", *In: The International Federation for Information Processing*, Volume 112. pp. 81 – 88.

[7] Chaslot, G., Fiter, C., Hoock, J.-B., Rimmel, A., and Teytaud, O. (2009), "Adding Expert Knowledge and Exploration in Monte-Carlo Tree Search", *Proceedings of the Twelfth International Advances in Computer Games Conference*, pp. 1-13, Pamplona, Spain.

[8] Chaslot, G., Winands, M., Bouzy, B., Uiterwijk, J. W. H. M., and Herik, H. J. van den (2007), "Progressive Strategies for Monte-Carlo Tree Search*", Proceedings of the 10th Joint Conference on Information Sciences (ed.P. Wang)*, pp. 655–661, Salt Lake City, USA.

[9] Coulom, R. (2007), "Computing elo ratings of move patterns in the game of go", *ICGA Journal 30*, pp. 198 – 208.

[10] Gelly, S. and Silver, D. (2007), "Combining Online and Offline Knowledge in UCT", *Proceedings of the 24th International Conference on Machine Learning*, pp. 273-280, Corvallis Oregon USA.

[11] Gelly, S. and Silver, D. (2011), "Monte-Carlo Tree Search and Rapid Action Value Estimation in Computer Go", *Artificial Intelligence*, Vol. 175, No. 11, pp. 1856-1875.

[12] Gelly, S. and Wang, Y. (2006), "Exploration exploitation in Go: UCT for Monte-Carlo Go," *in Proc. Adv. Neur. Inform. Process. Syst., Vancouver, Canada*.

[13] Gittins, J.C. (1979), "Bandit Processes and Dynamic Allocation Indices", *Journal of the Royal Statistical Society*. Series B (Methodological) 41 (2), pp. 148–177.

[14] Hendrik, B. (2010), *Adaptive Playout Policies for Monte-Carlo Go*, Master thesis, Institut für Kognitionswissenschaft, Universität Osnabrück.

[15] http://www.codeproject.com/Articles/4672/Reversi-in-C

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

[17] Kocsis, L. and Szepesvári, C. (2006), "Bandit Based Monte-Carlo Planning," *in 17th European Conference on Machine Learning, ECML 2006, ser.Lecture Notes in Computer Science, J. F¨urnkranz, T. Scheffer, and M. Spiliopoulou, Eds.*, vol. 4212. Springer, pp. 282–293.

[18] Pepels, T. and Winands, M. H. M. (2012), "Enhancements for Monte-Carlo tree search in Ms Pac-Man", *in IEEE Conf. Comput. Intell. Games*, pp. 265– 272.

[19] Powley, E. J., Whitehouse, D., and Cowling, P. I. (2012), "Monte Carlo tree search with macro-actions and heuristic route planning for the physical travelling salesman problem*", in IEEE Conf. Comput. Intell. Games*, pp. 234–241. IEEE.

[20] Rimmel, A., Teytaud, O., Lee, C., Yen, S., Wang, M., and Tsai, S. (2010), "Current frontiers in computer Go", *IEEE Trans. Comput. Intell. AI in Games*, 2(4), pp. 229–238.

[21] Winands, M. H. M., Bjornsson, Y., and Saito, J-T.(2010), "Monte Carlo Tree Search in Lines of Action", *IEEE Trans. Comp. Intell. AI Games*, 2(4), pp. 239–250.

[22] https://en.wikipedia.org/wiki/Game_complexity

THE MAIN ELEMENTS OF MONTE CARLO TREE SEARCH AND IMPROVEMENT APPROACHES

Nguyen Quoc Huy, Nguyen Khac Chien

Abstract - *Applications of board games are usually the testbed of AI field. The traditional programs were built up on Minimax, Alpha-Beta with the heuristic evaluation function. It is difficult to design a good state evaluation function. Moreover, the traditional tree search is suitable for the games in which the branch factor is low. Monte Carlo Tree Search is a novel framework, and it is very effective in some high branch factor such as Go. The Monte Carlo Tree Search model is combined from Tree search, Reinforcement learning, and Monte Carlo simulation. In our view, we can improve the performance of Monte Carlo Tree Search by studying how to improve the performance of reinforcement learning, or to improve the Monte Carlo simulation. This paper compact the efficient way of Monte Carlo Tree Search improvement and showed some initial experimental results very effective.*

Nguyễn Quốc Huy, tốt nghiệp tiến sỹ Khoa học máy tính tại Viện khoa học và công nghệ tiên tiến Nhật Bản (JAIST) vào năm 2014. Hiện đang là giảng viên của trường Đại học Sài Gòn, khoa CNTT (273 An Dương Vương P3 Q5 Tp.HCM). Chuyên ngành nghiên cứu Tối ưu ngẫu nhiên, Khai thác dữ liệu và Lý thuyết trò chơi. Email: nqhuy@sgu.edu.vn

Nguyễn Khắc Chiến, tốt nghiệp thạc sỹ Khoa học máy tính tại trường Đại học KHTN – Đại học Quốc gia TP.HCM vào năm 2008. Hiện đang là giảng viên trường Đại học Cảnh sát nhân dân TP.HCM, và là NCS Chuyên ngành Kỹ thuật máy tính tại Học viện Công nghệ Bưu chính Viễn thông. Email: nkchienster@gmail.com hoặc nkchien.dhcs@pup.edu.vn.