Tạp chí Khoa học Trường Đại học Cần Thơ
website: sj.ctu.edu.vn

PHƯƠNG PHÁP CẢI TIẾN HIỆU QUẢ CÂY TÌM KIẾM MONTE CARLO

Nguyễn Quốc Huy[1] và 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*

Thông tin chung:
Ngày nhận: 19/09/2015
Ngày chấp nhận: 10/10/2015

Title:
The efficient approach for improving the Monte Carlo Tree Search

Từ khóa:
Cây tìm kiếm Monte Carlo, Hàm lượng giá, Học tăng cường, Trò chơi bàn cờ, Chọn lựa đặc trưng

Keywords:
Monte Carlo Tree Search, Evaluation Function, Reinforcement Learning, Board Games, Feature Selection

ABSTRACT

Chess playing is an area of research in artificial intelligence. Traditional programs were built by using Minimax, Alpha-Beta with any heuristic evaluation function based on knowledge of chess players. It is difficult to design a good state evaluation function. Moreover, A traditional tree search is suitable for the games that their branch factor is low. Monte Carlo Tree Search is a novel framework, and very effective in some high branch factors 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 compacts the most efficient way of Monte Carlo Tree Search improvement and shows its efficiency based on the experimental results.

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 trạng thái 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. Với cách tiếp cận 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 nghiên cứu các thành phần chính của cây tìm kiếm Monte Carlo và xác định hướng cải tiến hiệu quả nhất cũng như thực nghiệm đã chứng minh tính hiệu quả.

1 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. Đối với các trò chơi như cờ Vua, cờ Tướng, cờ Vây, cờ Caro (Go-Moku), cờ Connect-6,… trong quá trình chơi sẽ xuất hiện các trạng thái bàn cờ khác nhau có thể biểu diễn thành một cây tìm kiếm (hay còn gọi là cây trò chơi). Một cây trò chơi bao gồm tất cả các nước đi có thể của hai người chơi và mỗi nút của cây thể hiện một trạng thái bàn cờ là kết quả của các nước đi. Từ một nút hiện tại có thể có nhiều lựa chọn cho nước

đi tiếp theo đó được gọi là hệ số phân nhánh. Độ sâu của cây trò chơi là số tầng của cây. Khi đối mặt với các cây trò chơi phức tạp, máy tính phải xét nhiều trạng thái và cần 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 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. Phương pháp tìm kiếm truyền thống Minimax với 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, chương trình cờ Vây (Nomitan) [10] của Ikeda và Viennot chỉ tìm kiếm 20 nước đi trên khoảng 300 khả năng có thể đi được.

Với hiệu quả của cây tìm kiếm Monte Carlo (MCTS) [4] trong nhiều lĩnh vực, trong đó có lĩnh vực trò chơi, bài báo này giới thiệu các thành phần cơ bản của MCTS, với hy vọng sẽ giúp người đọc có thể hiểu và áp dụng trong tương lai. Theo một khía cạnh khác, chúng ta có thể hiểu MCTS được tổng hợp hiệu quả từ ba mô hình kinh điển: Cây tìm kiếm, Học tăng cường và Giả lập Monte Carlo. Việc so sánh mô hình cây tìm kiếm truyền thống và MCTS được thể hiện trong Bảng 1.

Đóng góp chính của bài báo gồm có:

– Phân tích các thành phần cơ bản của MCTS để xác định hướng cải tiến phù hợp nhất (bảng 1).

– Xác định hướng cải tiến hiệu quả từ nhiều công trình đã công bố (phần 5).

– Cài đặt thực nghiệm để chứng minh hướng cải tiến được nêu trong bài báo là đúng đắn (phần 6).

– Bài báo là một trong số ít tài liệu tiếng Việt mô tả về MCTS.

Phần còn lại của bài báo được cấu trúc như sau: Phần 2 là bài toán "Tên cướp nhiều tay" trong học tăng cường. Cây tìm kiếm Monte Carlo (MCTS) được trình bày trong phần 3; Phần 4 bàn luận về vai trò của học tăng cường trong MCTS. Phân tích hướng cải tiến hiệu quả MCTS hiện nay được trình bày trong phần 5; phần 6 là một số kết quả thực nghiệm trên cờ Othello. Cuối cùng là kết luận và các vấn đề mở được trình bày trong phần 7.

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.
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.

2 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 "tên cướp nhiều tay" (multi-armed bandit) trong lý thuyết xác suất.

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 nhiều tay" được Auer *et al*. [1] định nghĩa như sau:

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 mang 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.

3 CÂY TÌM KIẾM MONTE CARLO

Các phương pháp Monte-Carlo hiện nay đang rất hiệu quả cho các chương trình cờ Vây. Năm 2006, Kocsis và Szepesvári [1,4] đề xuất thuật toán UCT (Upper Confidence Bounds for Tree) áp dụng 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 MCTS. MCTS là phương pháp tìm kiếm dựa theo lấy mẫu dùng giả lập ngẫu nhiên để ước lượng tỷ lệ thắng thua của một trạng thái bàn cờ 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 MCTS so với các phương pháp tìm kiếm truyền thống 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, 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 MCTS rất hiệu quả.

3.1 Cấu trúc cây tìm kiếm Monte Carlo

MCTS là một quá trình lặp đi lặp lại bốn bước trong một khoảng thời gian hữu hạn (xem Hình 1). ***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.

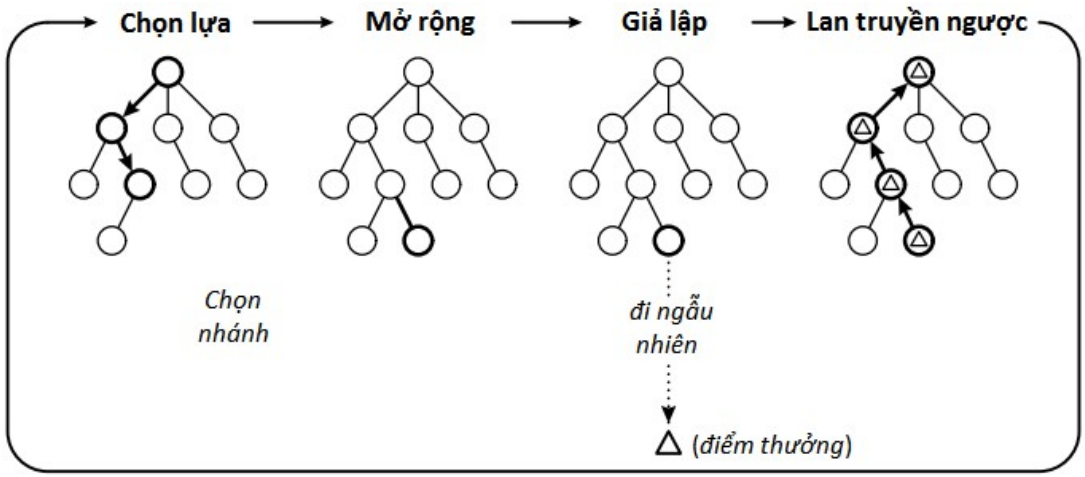
Cây trò chơi tăng trưởng sau mỗi lần lặp của MCTS, tăng trưởng rộng hơn và sâu hơn. Nước đi hứa hẹn là nút con nào có tỷ lệ thắng thua cao hơn được chọn trong giai đoạn chọn lựa, và rồi các cây con cũng tăng trưởng ngày càng rộng hơn và sâu hơn, và việc lấy mẫu ngày càng chính xác hơn. Sau khi kết thúc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

3.2 Thuật toán

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. Hàm *Selection()* thực hiện công việc tìm nút con của nút v dựa trên giá trị *UCB [2]* được tính theo công thức (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.

4 QUAN 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. 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: (1) kỹ thuật lập trình tối ưu để số lần giả lập càng nhiều càng tốt trong thời gian giới hạn, và (2) việc cân bằng giữa khai thác và khám phá được thực hiện một cách hợp lý.



Hình 1: Cây tìm kiếm Monte Carlo

Với mỗi trạng thái, trong khoảng thời gian giới hạn để chọn một 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 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, 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 (Upper Confidence Bounds) [1], 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 [1], ta có thể hình dung người đánh bạc chính là người chơi cờ trong lượt đi hiện hành, và các máy đánh bạc được xem như 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 (cận tin cậy trên) [1] được tích hợp vào MCTS trở thành thuật toán UCT.

$$UCB_i = \frac{w_i}{n_i} + C\sqrt{\frac{\ln N}{n_i}} \quad (1),$$

Tỷ số w_i/n_i là tỷ lệ thắng của nút con i, w_i là số lần thắng ván đấu giả lập 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 (1) có giá trị càng cao thì khả năng khai thác càng cao vì nút con có tỷ lệ thắng thua cao sẽ được chọn. Vế thứ hai của công thức (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.

5 HƯỚNG CẢI TIẾN

Dù MCTS 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 MCTS. Vì vậy, có rất nhiều kỹ thuật cải tiến MCTS dựa trên tri thức. Một trong các kỹ thuật đó được thể hiện như sau:

```
function UCT_SEARCH(s0)
    create root node v0 with state s0
    while within computational budget do   // Thực hiện vòng lặp để xây dựng cây
        v := Selection(v0, C)              // Bước 1: Chọn lựa nút con có giá trị UCB lớn nhất
        vl := Expand(v)                    // Bước 2: Mở rộng: chọn một nút bất kỳ vl là con của nút v
        reward := Simulation(s(vl))        // Bước 3: Thực hiện giả lập cho đến khi kết thúc ván cờ, trả về phần thưởng
    reward
        BackPropagation (vl, reward)       //Bước 4: Thực hiện lan truyền ngược kết quả của ván cờ giả lập.
    return the most visited child node of v0   //Trả về là nút v0 có giá trị UCB lớn nhất (nút được thăm nhiều nhất)
function Selection(v, C)                   // Thực hiện tính toán và chọn lựa nút con nào có giá trị UCB lớn nhất
        while v is not terminal do
            v := arg max UCB_{v' ∈ v_child}'
        return v
  function Expand(v)                       // Chọn 1 nút con bất kỳ của nút vừa được chọn ở Selection()
        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)                    // Thực hiện quá trình giả lập từ nút mở rộng Expand()
    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)       // Thực hiện lan truyền ngược từ nút bắt đầu được mở rộng //cho đến nút gốc
    while v is not null do
        N(v) := N(v) + 1
        Q(v) := Q(v) + reward
        reward := - reward
        v := parent of v
```

5.1 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 Viennot [10] đã đưa ra công thức (2) 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.

$$UCB_{Bias}(i) = \frac{w_i}{n_i} + C\sqrt{\frac{\ln N}{n_i}} + C_{BT}P(m_i)\sqrt{\frac{K}{N+K}} \quad (2),$$

với 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.

5.2 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 (2) 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.

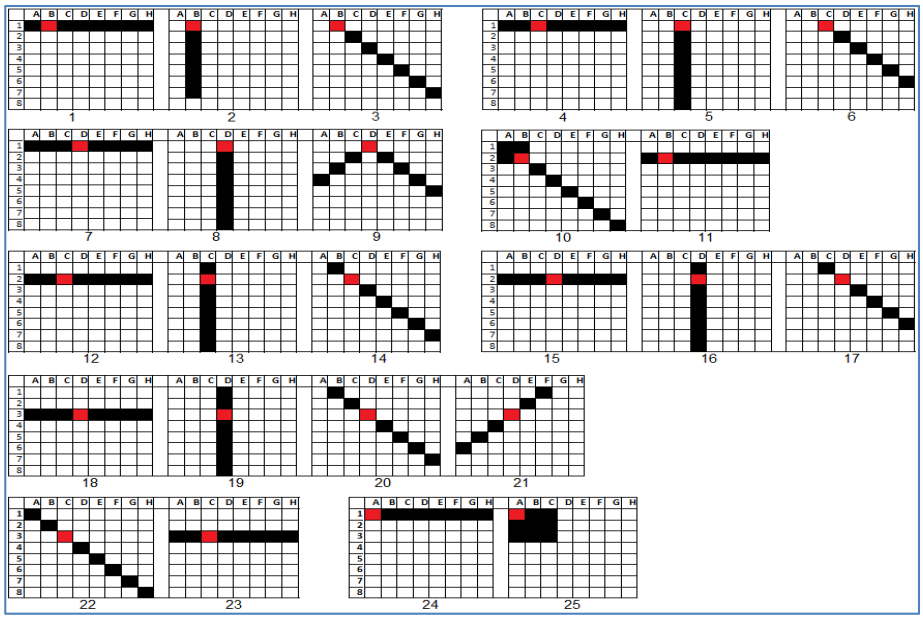
6 KẾT QUẢ THỰC NGHIỆM

Để so sánh chiến lược Alpha-Beta và chiến lược MCTS, 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 (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 (2) 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 MCTS.

Trong OthelloMCTS2, hàm lượng giá hành động xây dựng dựa trên tri thức của cờ Othello theo Michael Buro [2] trong Hình 2. Đặc trưng 1, 2, 3 của ô B1. Đặc trưng 4, 5, 6 của ô C1. Đặc trưng 7, 8, 9 của ô D1. Đặc trưng 10, 11 của ô B2, tương tự cho các ô còn lại. Hàm lượng giá hành động được xây dựng theo mô hình Bradley-Terry (công thức 3) như đề xuất của Remi Coulom [5]. Ta có thể hiểu OthelloMCTS2 là cải tiến của OthelloMCTS1. Cách thức cải tiến mô tả trong phần 5 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 2. Hình 2 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.

$$P(2\_4 \text{ against } 1\_2\_5 \text{ and } 1\_3\_6\_7) = \frac{\gamma_2\gamma_4}{\gamma_2\gamma_4 + \gamma_1\gamma_2\gamma_5 + \gamma_1\gamma_3\gamma_6\gamma_7} \quad (3)$$

Công thức (3) 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 (3) tính xác suất của nước đi A so với nước đi B và C. γ_i là trọng số của đặc trưng thứ *i*.



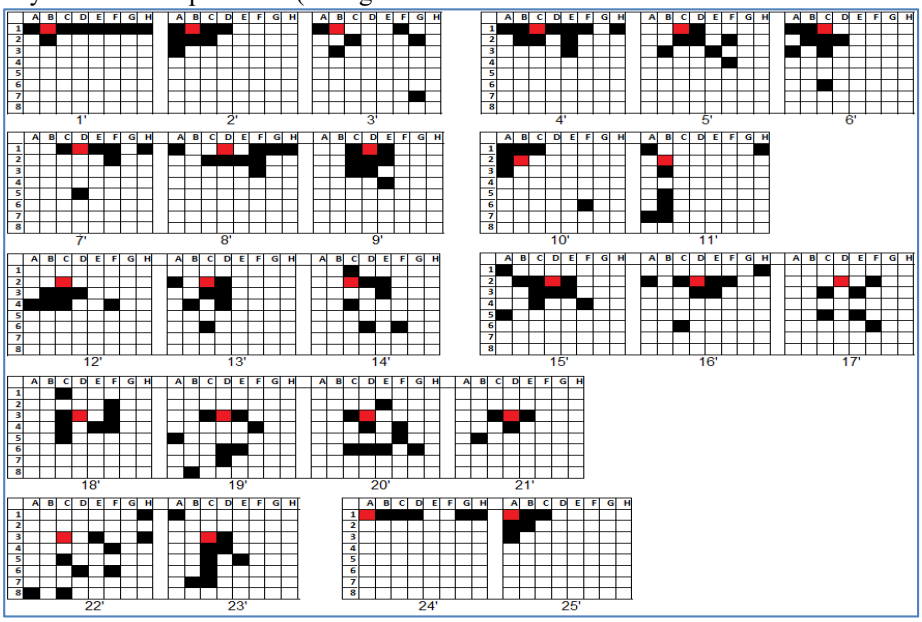
Hình 2: Các đặc trưng của 9 vị trí trên bàn cờ Othello [6] của M. Buro

OthelloMCTS3 giống OthelloMCTS2, nhưng hàm lượng giá hành động được xây dựng trên tri thức của cờ Othello do Huy *et al.* [9] tìm ra như trong Hình 3 bằng phương pháp tối ưu ngẫu nhiên. Như vậy, ta có thể hiểu chương trình OthelloMCTS3 là một cải tiến của chương trình OthelloMCTS2, cải tiến là do hàm lượng giá hành động sử dụng các đặc trưng tối ưu (Hình 3) thay cho các đặc trưng chưa được tối ưu (Hình 2).

Kết quả thực nghiệm cho thấy chương trình OthelloMCTS1 chỉ thắng 314 ván trên 1000 ván đấu với chương trình Riversi, kết quả như vậy chúng ta có thể nhận thấy tại sao chương trình Riversi được cài đặt theo cây tìm kiếm Alpha-Beta lại có kết quả tốt hơn OthelloMCTS1 được cài đặt theo MCTS với hai lý do sau đây:

– 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 MCTS. Tuy nhiên, trò chơi Riversi phổ biến trong [8], và dễ cài đặt nên bài báo chọn làm chương trình minh họa).



Hình 3: Các đặc trưng của 9 vị trí trên bàn cờ Othello của Huy *et al.*

Bảng 2: Thực nghiệm MCTS so với Riversi

Thời gian suy nghĩ của MCTS: 4s	Riversi
OthelloMCTS1	314/1000
OthelloMCTS2	502/1000
OthelloMCTS3	981/1000

OthelloMCTS3 thắng 981 ván trên 1000 ván đấu với Riversi. Như vậy, OthelloMCTS3 mạnh hơn OthelloMCTS1 với tỷ lệ 98.1% so với tỷ lệ 31.4%, và mạnh hơn so với OthelloMCTS2 với tỷ lệ 98.1% so với 50.2%. Các đặc trưng trên Hình 3 là những đặc trưng tối ưu so với các đặc trưng chưa tối ưu trên Hình 2. Tỷ lệ thắng cao hơn khi các đặc trưng được tối ưu.

Bảng 3 so sánh giữa các chương trình MCTS với nhau. OthelloMCTS1 là một chương trình Othello được xây dựng trên MCTS hoàn toàn không có sự hỗ trợ của hàm lượng giá hành động. OthelloMCTS2 được xây dựng trên MCTS có hàm lượng giá hành động nhưng sử dụng tri thức chưa tối ưu của M. Buro (Hình 2). OthelloMCTS3 được xây dựng trên MCTS có hàm lượng giá hành động nhưng sử dụng tri thức tối ưu của Huy *et al.* [9] (Hình 3). Theo Bảng 3, OthelloMCTS3 mạnh nhất so với hai chương trình còn lại khi thắng 999 ván trên tổng số 1000 ván so với OthelloMCTS1, và thắng 835 trên tổng số 1000 ván so với OthelloMCTS2. Do đó, cải tiến hiệu suất MCTS có sử dụng tri thức bổ sung có thể mang lại hiệu quả cao, và đặc biệt là tri thức bổ sung được tối ưu. Như vậy, tri thức bổ sung rất cần thiết cho giai đoạn học tăng cường cũng như trong giai đoạn giả lập.

Bảng 3: Thực nghiệm MCTS so với Riversi

Thời gian suy nghĩ của MCTS: 4s	OthelloMCTS3
OthelloMCTS1	999/1000
OthelloMCTS2	835/1000

7 KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

Có nhiều cải tiến MCTS 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 khảo sát của nhóm tác giả, có ít công trình đề cập đến mô hình MCTS là sự kết hợp của Cây tìm kiếm, Học tăng cường và giả lập Monte Carlo. Có nhiều công trình 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 MCTS. Hầu hết những cải tiến phương pháp học tăng cường và cải tiến giai đoạn giả lập hiệu quả đều dựa vào tri thức là các đặc trưng của trò chơi.

Cải tiến hiệu suất MCTS 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.

Nhóm tác giả 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 có cơ sở và có tiềm năng trong công việc nghiên cứu.

Hướng nghiên cứu tiếp theo, nhóm tác giả sẽ thực hiện tối ưu tri thức đặc trưng từ những ván cờ Connect-6 có sẵn và bổ sung tri thức vào hai giai đoạn ***Chọn lựa*** và giai đoạn ***Giả lập*** để nâng cao hiệu quả của cây tìm kiếm.

TÀI LIỆU THAM KHẢO

1. 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.
2. Buro, M. (2003), "The evolution of strong othello programs", *In: The International Federation for Information Processing*, Volume 112. pp. 81 – 88.
3. 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.
4. 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.
5. Coulom, R. (2007), "Computing elo ratings of move patterns in the game of go", *ICGA Journal 30*, pp. 198 – 208.
6. 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.
7. http://www.codeproject.com/Articles/4672/Reversi-in-C
8. https://en.wikipedia.org/wiki/Game_complexity
9. Huy Nguyen, Kokolo Ikeda, Simon Viennot (2014), "Fast Optimization of the Pattern Shapes in Board Games with Simulated Annealing", *Proceedings of the Sixth International Conference KSE 2014,* pp. 325 – 337.
10. Ikeda, K., Viennot, S. (2013), "Efficiency of static knowledge bias in monte-carlo tree search", *In: Computers and Games 2013*.