19120476 – Trần Phương Đình

# **I - Phát biểu bài toán: PREDATOR PREY HEX WORLD**

- Predator prey hex world là bài toán mà trong đó sử dụng hex world dynamics để chứa nhiều agents bao gồm “kẻ săn mồi”(predator) và “con mồi”(prey). Khi đó, predator cố gắng để bắt được các con mồi nhanh nhất có thể và con mồi phải chạy khỏi các predator càng lâu càng tốt.

- Một hoặc nhiều predator có thể bắt được nhiều con mồi cùng lúc nếu điều đó xảy ra trong một hex cell và phần thưởng sẽ được chia đều nhau cho mỗi predator.

# **II - Khó khăn, thách thức cụ thể đối với từng bài toán.**

1. ***Về các khó khăn chung khi thực hiện đồ án:***

* Ít nguồn tài liệu để tham khảo
* Các thuật toán và cấu trúc dữ liệu chứa nhiều khái niệm mới và nhiều công thức toán học phức tạp, dẫn đến việc gặp không ít khó khăn khi cài đặt.
* Một số thuật ngữ bằng tiếng Anh khi dịch sang tiếng Việt sẽ mất đi ý nghĩa, dẫn đến một số câu từ trong báo cáo thiếu tự nhiên và rành mạch.

1. ***Về bài toán Predator Prey Hex World:***

* Gặp nhiều khó khăn khi code vì thiếu nguồn tài liệu để tham khảo về cách thực thi và luồng chạy của bài toán này.
* Việc mô phỏng các predator và prey được thực hiện dựa trên mô hình học máy phức tạp, dẫn đến gặp nhiều khó khăn trong lúc cài đặt và thể hiện kết quả.

# **III - Thực nghiệm:**

1. ***Mô hình tính toán (cấu trúc dữ liệu / phương pháp thể hiện)***
   1. ***Markov Game***
   * Markov Game là một phiên bản mở rộng hơn của Simple Game với một trạng thái chia sẻ *s* ∈ S. Khả năng chuyển đổi từ trạng thái s đến trạng thái s’ dưới một joint action a được cung cấp bởi hàm phân phối chuyển tiếp *T*(*s*′ | *s*, **a**). Mỗi agent i nhận được một phần thưởng dựa trên một hàm phần thưởng riêng biệt với mỗi agent *Ri*(*s*, **a**), cũng phụ thuộc vào trạng thái s. Cấu trúc của một Markov Game được minh họa với đoạn mã nguồn sau:

struct MG

    γ  # discount factor

    ℐ  # agents

    𝒮  # state space

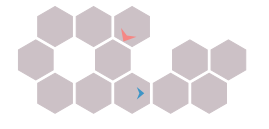
    𝒜  # joint action space

    T  # transition function

    R  # joint reward function

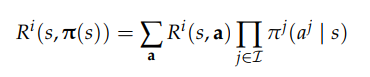
end

* 1. ***Predator Prey Hex World***
  + Predator Prey Hex World có trạng thái khởi tạo như hình bên dưới với mũi tên màu đỏ là predator và mũi tên màu xanh là prey. Hướng của các mũi tên chỉ ra các hành động tiềm năng được thực hiện bởi chủ thể của nó. Predator Prey Hex World không có trạng thái kết thúc.

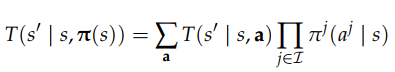


* + Ta có một tập các predator và một tập các prey , với = . Các trạng thái bao gồm các vị trí của mỗi agent: , với mỗi là một vị trí hex tương ứng. Không gian joint action là , với mỗi chứa tất cả các hướng chuyển động trong hex.
  + Nếu một predator và ở cùng một hex với thì prey sẽ bị predator ăn và sau đó prey sẽ được chuyển đến một hex ngẫu nhiên, được thể hiện như là những thế hệ sau của prey đã bị ăn thịt. Mặt khác, các trạng thái chuyển đổi là độc lập và được mô tả trong Original Hex World (tham khảo bài Hex World)
  + Một hoặc nhiều predator có thể bắt một hoặc nhiều prey nếu điều đó cùng xảy ra trong một hex. Giả sử ta có *n* predator và *m* prey ở cùng một ô thì, mỗi predator nhận được phần thưởng là *m / n*. Ví dụ: nếu 5 predator bắt được 3 prey, thì phần thưởng cho mỗi predator sẽ là 3/5 prey.
  + Khi một predator hoặc prey di chuyển, nó sẽ bị -1 điểm. Predator sẽ được 10 điểm nếu bắt được prey và prey sẽ bị trừ 100 điểm.

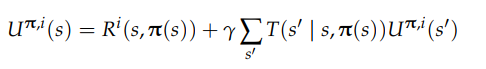
1. **Phương pháp giải quyết (thuật toán)**
   1. **Markov Game’s Policy**
   * Joint policy π trong Markov Game định nghĩa một khả năng phân phối xác suất dựa trên các joint action cho trạng thái hiện tại. Cùng với MDPs, ta sẽ tập trung trên các policy phụ thuộc vào các trạng thái hiện tại hơn các trạng thái quá khứ vì những trạng thái ở tương lai và các phần thưởng đều có điều kiện độc lập với lịch sử so với trạng thái hiện tại
   * Ta cũng sẽ chú ý đến các stationary policy (không phụ thuộc vào thời gian). Khả năng agent *i* chọn một action *a* ở trạng thái *s* được biểu diễn bởi . Hàm thường được dùng để biểu diễn sự phân bố trên những joint action.
   * Lợi ích của một joint policy từ quan điểm của agent *i* có thể được tính toán bằng cách sử dụng một biến thể của policy evaluation. Phần thưởng cho agent *i* từ trạng thái *s* khi tuân theo joint policy π là:



* + Xác suất chuyển trạng thái từ s sang s’ khi tuân theo π là:



* + Trong một infinite-horizon discounted game, lợi ích dành cho agent *i* từ trạng thái *s* là:



* + Đoạn code bên dưới mô tả một cách cài đặt của Markov Game’s Policy:

struct MGPolicy

  p # dictionary mapping states to simple game policies

  MGPolicy(p::*Base.Generator*) = new(Dict(p))

end

(πi::*MGPolicy*)(s, ai) = πi.p[s](ai)

(πi::*SimpleGamePolicy*)(s, ai) = πi(ai)

probability(𝒫::*MG*, s, π, a) = prod(πj(s, aj) for (πj, aj) in zip(π, a))

# Phần thưởng cho agent i từ trạng thái s khi tuân theo joint policy π

reward(𝒫::*MG*, s, π, i) =

  sum(𝒫.R(s,a)[i]\*probability(𝒫,s,π,a) for a in joint(𝒫.𝒜))

# Xác suất chuyển trạng thái từ s sang s’ khi tuân theo π

transition(𝒫::*MG*, s, π, s′) =

  sum(𝒫.T(s,a,s′)\*probability(𝒫,s,π,a) for a in joint(𝒫.𝒜))

# Đánh giá của policy π cho agent i

function policy\_evaluation(𝒫::*MG*, π, i)

  𝒮, 𝒜, R, T, γ = 𝒫.𝒮, 𝒫.𝒜, 𝒫.R, 𝒫.T, 𝒫.γ

  p(s,a) = prod(πj(s, aj) for (πj, aj) in zip(π, a))

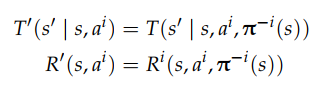
  R′ = [sum(R(s,a)[i]\*p(s,a) for a in joint(𝒜)) for s in 𝒮]

  T′ = [sum(T(s,a,s′)\*p(s,a) for a in joint(𝒜)) for s in 𝒮, s′ in 𝒮]

  return (I - γ\*T′)\R′

end

* 1. **Response Model**
  + Ta có thể tổng quát hóa các Response Model thành các Markov Game, với yêu cầu phải tính toán đến các mô hình chuyển đổi trạng thái
    1. *Best Response*
  + Một response policy cho agent *i* là một policy mà tối đa hóa được lợi ích mong muốn và đưa ra các fixed policy cho những agent khác. Nếu các policy của những agent khác đều cố định, thì vấn đề trên trở thành một MDP. MDP này có không gian trạng thái và không gian hành động . Ta có thể định nghĩa các hàm chuyển đổi và hàm phần thưởng như sau:



* + Bởi vì đây là best response cho agent *i* nên MDP chỉ dùng reward . Việc giải MDP này cho kết quả là một best response policy cho agent *i*. Đoạn code bên dưới mô tả cách cài đặt best response:

function best\_response(𝒫::*MG*, π, i)

  𝒮, 𝒜, R, T, γ = 𝒫.𝒮, 𝒫.𝒜, 𝒫.R, 𝒫.T, 𝒫.γ

  # Hàm chuyển đổi trạng thái

  T′(s,ai,s′) = transition(𝒫, s, joint(π, SimpleGamePolicy(ai), i), s′)

  # Hàm phần thưởng

  R′(s,ai) = reward(𝒫, s, joint(π, SimpleGamePolicy(ai), i), i)

  πi = solve(MDP(γ, 𝒮, 𝒜[i], T′, R′))

  return MGPolicy(s => SimpleGamePolicy(πi(s)) for s in 𝒮)

end

* + 1. *Softmax Response*
  + Ta có thể định nghĩa một softmax response policy bằng cách gán một stochastic response cho các policy của những agent khác tại mỗi trạng thái. Tương tự việc xây dựng nên một deterministic best response policy, ta giải một MDP mà trong đó các agent với các fixed policy được xếp vào môi trường. Tiếp theo, ta trích xuất hàm giá trị hành động *Q*(*s*, *a*) bằng cách sử dụng one-step lookahead. Ta có softmax response với tham số precision *λ* ≥ 0 như sau :



* + Đoạn code bên dưới mô tả cách cài đặt Softmax Response:

function softmax\_response(𝒫::*MG*, π, i, λ)

  𝒮, 𝒜, R, T, γ = 𝒫.𝒮, 𝒫.𝒜, 𝒫.R, 𝒫.T, 𝒫.γ

  T′(s,ai,s′) = transition(𝒫, s, joint(π, SimpleGamePolicy(ai), i), s′)

  R′(s,ai) = reward(𝒫, s, joint(π, SimpleGamePolicy(ai), i), i)

  mdp = MDP(γ, 𝒮, joint(𝒜), T′, R′)

  πi = solve(mdp)

  Q(s,a) = lookahead(mdp, πi.U, s, a)

  p(s) = SimpleGamePolicy(a => exp(λ\*Q(s,a)) for a in 𝒜[i])

  return MGPolicy(s => p(s) for s in 𝒮)

end

* 1. **Predator Prey Hex World (cài đặt thuật toán và kiểm thử)**
     1. *Cài đặt Predator Prey Hex World*
  + Khởi tạo các trạng thái ban đầu của trò chơi:

# Khởi tạo số agent là 2

n\_agents(mg::*PredatorPreyHexWorldMG*) = 2

* + Ta có một tập các predator và một tập các prey , với = .
  + Cài đặt các trạng thái các vị trí của mỗi agent:

# Đánh số thứ tự của các state

ordered\_states(mg::*PredatorPreyHexWorldMG*, i::*Int*) = vec(collect(1:length(mg.hexes)))

# Gán mỗi state với một agent tương ứng

ordered\_states(mg::*PredatorPreyHexWorldMG*) = vec(collect(Iterators.product([ordered\_states(mg, i) for i in 1:n\_agents(mg)]...)))

* + Cài đặt không gian joint action là

# Đánh số thứ tự cho các action

ordered\_actions(mg::*PredatorPreyHexWorldMG*, i::*Int*) vec(collect(1:n\_actions(mg.hexWorldDiscreteMDP)))

# Đánh số thứ tự cho các joint action

ordered\_joint\_actions(mg::*PredatorPreyHexWorldMG*) = vec(collect(Iterators.product([ordered\_actions(mg, i) for i in 1:n\_agents(mg)]...)))

* + Cài đặt hàm xử lý khi một predator và ở cùng một hex với :

function transition(mg::*PredatorPreyHexWorldMG*, s, a, s′)

    # Khi một prey bị bắt, một prey mới sẽ được tạo ra ở một vị trí ngẫu nhiên và predator sẽ vẫn đứng yên.

    if s[1] == s[2]

        prob = Float64(s′[1] == s[1]) / length(mg.hexes)

    else

        # Ngược lại, predator và prey sẽ tuân theo theo HexWorldMDP.

        prob = mg.hexWorldDiscreteMDP.T[s[1], a[1], s′[1]] mg.hexWorldDiscreteMDP.T[s[2], a[2], s′[2]]

    end

    return prob

end

* + Cài đặt hàm reward: khi một predator hoặc prey di chuyển, nó sẽ bị -1 điểm. Predator sẽ được 10 điểm nếu bắt được prey và prey sẽ bị trừ 100 điểm.

function reward(mg::*PredatorPreyHexWorldMG*, i::*Int*, s, a)

    r = 0.0

    if i == 1

        # Predator bị -1 điểm khi di chuyển và được 10 điểm khi bắt được con mồi

        if s[1] == s[2]

            return 10.0

        else

            return -1.0

        end

    elseif i == 2

        # Prey bị -1 điểm khi di chuyển và -100 điểm khi bị bắt

        if s[1] == s[2]

            r = -100.0

        else

            r = -1.0

        end

    end

    return r

end

* + Cài đặt các hàm khởi tạo cho MG và Predator Prey Hex World:

# Khởi tạo một MG theo các tham số của một PredatorPreyHexWorld

function MG(mg::*PredatorPreyHexWorldMG*)

    return MG(

        mg.hexWorldDiscreteMDP.γ,

        vec(collect(1:n\_agents(mg))),

        ordered\_states(mg),

        [ordered\_actions(mg, i) for i in 1:n\_agents(mg)],

        (s, a, s′) -> transition(mg, s, a, s′),

        (s, a) -> joint\_reward(mg, s, a)

    )

end

# Khởi tạo một PredatorPreyHexWorld theo MG

function PredatorPreyHexWorldMG(hexes::*Vector{Tuple{Int,Int}}*,

                                r\_bump\_border::*Float64*,

                                p\_intended::*Float64*,

                                γ::*Float64*)

    hexWorld = HexWorldMDP(hexes,

                           r\_bump\_border,

                           p\_intended,

                           Dict*{Tuple{Int64,Int64},Float64}*(),

                           γ)

    mdp = hexWorld.mdp

    return PredatorPreyHexWorldMG(hexes, mdp)

end

* + 1. *Kiểm thử*

Ta có chương trình kiểm thử thuật toán đã cài đặt như sau:

@testset "predator\_prey.jl" begin

    m = PredatorPreyHexWorld()

    hexes = m.hexes

    @test p.n\_agents(m) == 2

    @test length(p.ordered\_states(m, rand(1:2))) == length(hexes) && length(p.ordered\_states(m)) == length(hexes)^2

    @test length(p.ordered\_actions(m, rand(1:2))) == 6 && length(p.ordered\_joint\_actions(m)) == 36

    @test p.n\_actions(m, rand(1:2)) == 6 && p.n\_joint\_actions(m) == 36

    @test 0.0 <= p.transition(m, rand(p.ordered\_states(m)), rand(p.ordered\_joint\_actions(m)), rand(p.ordered\_states(m))) <= 1.0

    @test -1.0 <= p.reward(m, rand(1:2), rand(p.ordered\_states(m)), rand(p.ordered\_joint\_actions(m))) <= 10.0

    @test [-1.0, -1.0] <= p.joint\_reward(m, rand(p.ordered\_states(m)), rand(p.ordered\_joint\_actions(m))) <= [10.0, 10.0]

    mg = p.MG(m)

end

# **IV - Tóm tắt kết quả:**

1. **Đánh giá kết quả kiểm thử**
2. **Điểm mạnh**
3. **Điểm yếu**

# **V – Tài liệu tham khảo**

* Code: <https://github.com/algorithmsbooks/DecisionMakingProblems.jl>
* Document: Sách Algorithms for decision making