1. **Traveler’s dilemma:**
   1. **Bài toán:**

*Traveler’s Dilemma (Thế tiến thoái lưỡng nan của lữ khách )* trong lí thuyết trò chơi là một trò chơi có tổng khác không, trong đó hai người chơi cố gắng tối đa hóa phần thưởng của riêng họ mà không quan tâm đến người kia.

Nội dung cụ thể: Một hãng hàng không gây thiệt hại nghiêm trọng cho các cổ vật giống hệt nhau được mua bởi hai lữ khách khác nhau. Người quản lí hãng hàng không sẵn sàng đền bù cho họ vì mất đồ cổ, nhưng vì anh ta không biết gì về giá trị của chúng, anh ta nói với hai lữ khách hãy viết riêng ước tính của họ về giá trị hay bất kì con số nào trong khoảng từ 2 đến 100 đô la mà không trao cho người kia. Với một số lưu ý như sau:

* Nếu cả hai lữ khách đều viết cùng một số, anh ta sẽ hoàn trả cho mỗi người số tiền đó.
* Nếu họ viết các số khác nhau, người quản lí sẽ cho rằng giá thấp hơn là giá trị thực và người viết số cao hơn đang gian lận. Trong khi anh ta sẽ trả cho cả hai số tiền thấp hơn, người có số tiền thấp hơn sẽ nhận được tiền thưởng 2 đô la vì sự trung thực, trong khi người viết số cao hơn sẽ bị phạt 2 đô la.

Với các quy tắc trên, hãy đưa ra một chiến lược có lợi đối với từng người chơi.

* 1. **Thách thức:**

Phương pháp Hierachical Softmax với độ chính xác λ lớn đôi khi sẽ xảy ra hiện tượng tràn số trong tính toán. Độ phức tạp của các thuật giải lâu nếu không gian dữ liệu lớn.

* 1. **Thực nghiệm:**
     1. **Mô hình hóa bài toán:**

Dựa vào cấu trúc của một Simple game đã trình bày, cấu trúc của bài toán được mô hình hóa như sau:

* Trong bài toán này gồm có 2 người chơi, do đó gồm 2 agent đại diện:

.

* Mỗi người chơi chỉ được chọn mệnh giá từ 2 đến 100 đô la:

.

* Dựa vào quy tắc thưởng/phạt cho mỗi người chơi, hàm tính phần thưởng được định nghĩa như sau: với là tất cả lựa chọn bởi tất cả các người chơi khác ngoài người chơi i.
  + 1. **Phương pháp giải quyết:**

Trong bài báo cáo này, tôi sử dụng 2 phương pháp tìm để tìm policy tốt là *Iterated Best Response* và *Hierarchical Softmax*.

* + - 1. **Iterated Best Response:**

Lý do lựa chọn phương pháp:

*Nash Equilibrium* hay gọi là cân bằng Nash là trạng thái mà tất cả người chơi không có động cơ để đi chệch khỏi chiến lược được chọn của họ sau khi xem xét lại chiến lược của các đối thủ. Trong nhiều trò chơi không hợp tác (Non-cooperative game) với 2 hoặc nhiều người chơi, *Nash Equilibrium* thường là cách để tìm ra một lời giải hợp lí.

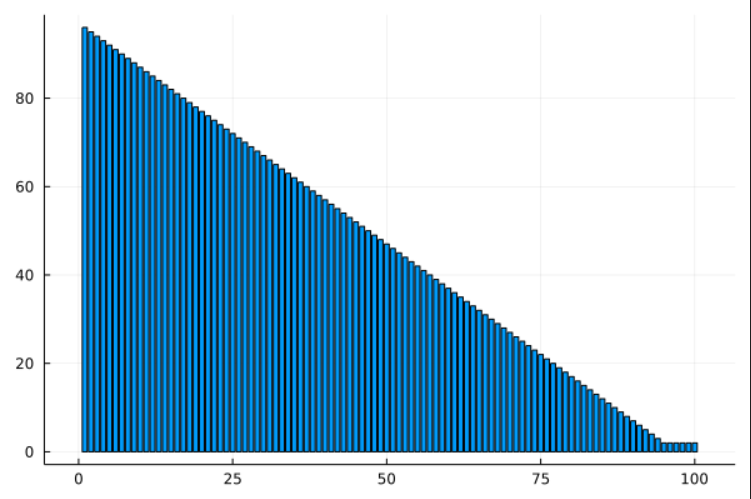
Bởi vì tính toán *Nash Equilibrium* có thể tốn kém về mặt tính toán, một cách tiếp cận thay thế là áp dụng lặp đi lặp lại các best response trong một loạt các trò chơi lặp lại giữa các agent. Trong *Iterated Best Response*, chúng tôi luân phiên ngẫu nhiên giữa những agent, lần lượt giải các *Best response policy* của từng agent.

Thuật giải:

|  |
| --- |
| function solve(M::IteratedBestResponse, )  π = M.π # Policy hiện tại của agent  for k in 1:M.k\_max  π = [best\_response(, π, i) for i in .ℐ]  # Mảng chứa các best response của các agent  end  return π  end |

Quá trình này có thể hội tụ về trạng thái *Nash Equilibrium.* Biểu đồ sau thể hiện sự thay đổi Best Response đối với mỗi Agent qua mỗi vòng lặp sau vòng lặp.

Kết quả thực nghiệm:



Best response của agent qua 100 vòng lặp.

Như vậy lựa chọn Best Response hội tụ dần về giá trị thấp nhất có thể là 2 đô la. Đây cũng là *Nash Equilibrium.*

Giải thích:

Trong Traveler’s dilemma, sự lựa chọn hợp lí, về mặt cân bằng Nash, là 2 đô la. Các lập luận là như sau: Với sự lựa chọn đầu tiên, lữ khách A có thể là viết là 100 đô la, nếu lữ khách B cũng ghi là 100 đô la thì đó sẽ là số tiền cả hai nhận được từ người quản lí hãng hàng không. Nhưng với lựa chọn thứ hai, lữ khách A lí do rằng nếu anh ta viết 99 đô la và lữ khách B viết 100 đô la, thì A sẽ nhận được 101 đô la (99 đô la + 2 đô la tiền thưởng). Nhưng A tin rằng B cũng nghĩ như vậy và nếu B cũng giảm 99 đô la, cả hai sẽ nhận được 99 đô la. Vì vậy, A thực sự sẽ được lợi hơn khi viết 98 đô la và nhận 100 đô la (98 đô la + 2 đô la tiền thưởng) nếu B viết 99 đô la. Nhưng vì có cùng suy nghĩ viết 98 đô la có thể xảy ra với B, A đã cân nhắc việc giảm 97 đô la, v.v... Dòng qui nạp ngược này sẽ đưa lữ khách xuống con số nhỏ nhất cho phép, đó là 2 đô la. Tại trạng thái này, cả A và B đều không còn động cơ để thay đổi lựa chọn của mình, do đó đây là *Nash Equilibrium.*

* + - 1. **Hierarchical Softmax:**

Lý do lựa chọn phương pháp:

Khi xây dựng các hệ thống *decision-making* phải tương tác với con người, việc tính toán *Nash Equilibrium* không phải lúc nào cũng hữu ích. Con người thường không sử dụng chiến lược *Nash Equilibrium* . Trước hết, có thể không rõ nên áp dụng *Equilibrium* (điểm cân bằng) nào nếu có nhiều điểm cân bằng khác nhau trong trò chơi. Đối với các trò chơi chỉ có một *Equilibrium*, con người có thể khó tính toán *Nash Equilibrium* vì những hạn chế về nhận thức. Đối với bài toán Traveler’s dilemma, *Nash Equilibrium* là câu trả lời không thích hợp bởi quả thực tế, đa số các người chơi đều lựa chọn các giá trị khoảng từ 95 đô la đến 100 đô la. Lựa chọn 2 đô la không đem lại nhiều giá trị cho người chơi.

Phương pháp *Hierarchical Softmax* áp dụng lại ý tưởng của phương pháp Iterated Best Response nhưng áp dụng mô hình Softmax response thay vì Best Response. Phương pháp này mô phỏng chiều sâu lý luận của các agent với tham số level . Một agent có level 0 lựa chọn hành động trong trò chơi một cách ngẫu nhiên, một agent có level 1 cho rằng đối thủ sử dụng chiến lược level 0 chọn lựa hành động dựa theo đó với mô hình *softmax response* với độ chính xác là λ. Với λ càng tiến về 0 thì policy có được thường cho các agent với cách lựa chọn số tiền gần như là ngẫu nhiên, và với λ càng tiến về vô cùng thì policy càng giống với policy của model best response.

Thuật giải:

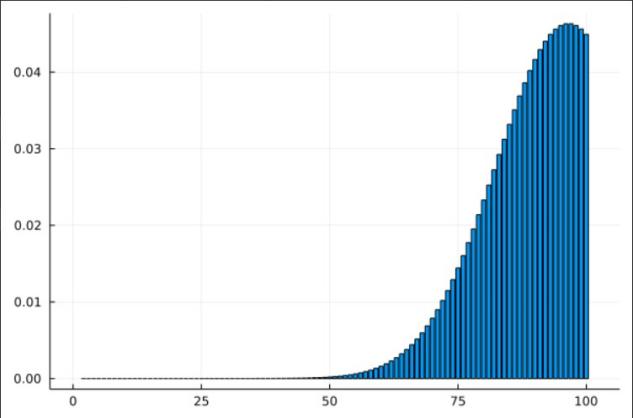
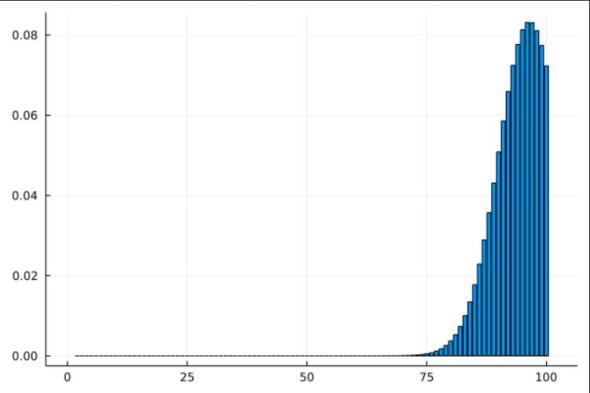
|  |
| --- |
| function solve(M::HierarchicalSoftmax, )  π = M.π # Policy hiện tại của agent  for k in 1:M.k\_max  π = [best\_response(, π, I, M.λ) for i in .ℐ]  # Mảng chứa các softmax response của các agent  end  return π  end |

Kết quả thực nghiệm:

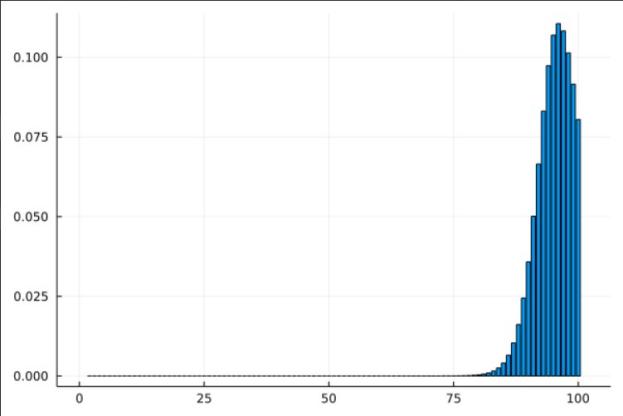
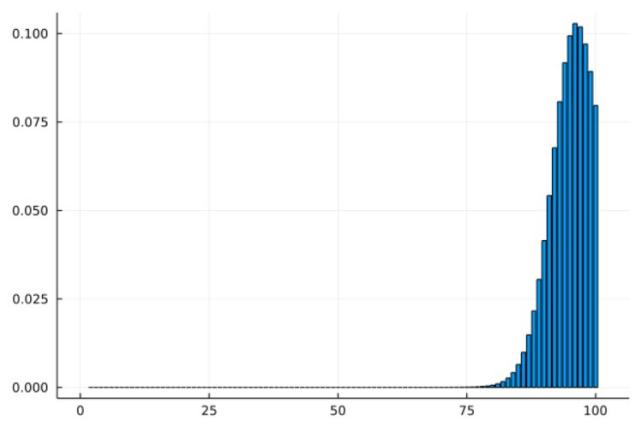
Nhằm đánh giá mô hình, chung tôi lựa chọn các giá trị k và λ ngẫu nhiên và sau đây là một số báo cáo kết quả thực nghiệm.

* k = 5, λ = 0.3.

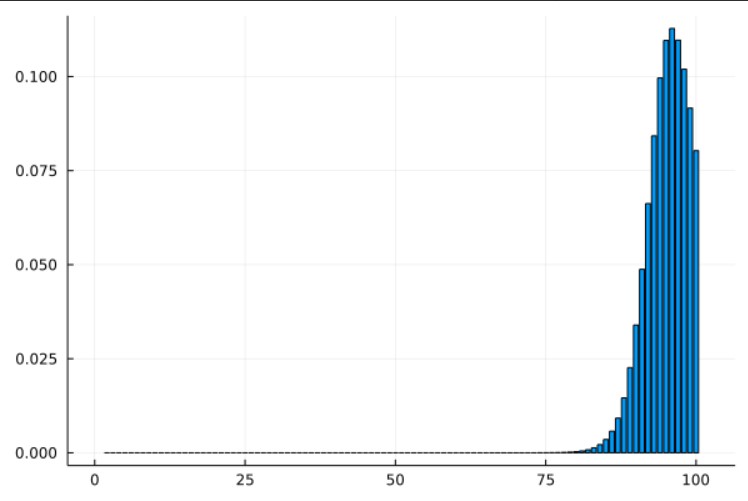
Vòng lặp 1, 2:

 ****

Vòng lặp 3, 4:

****

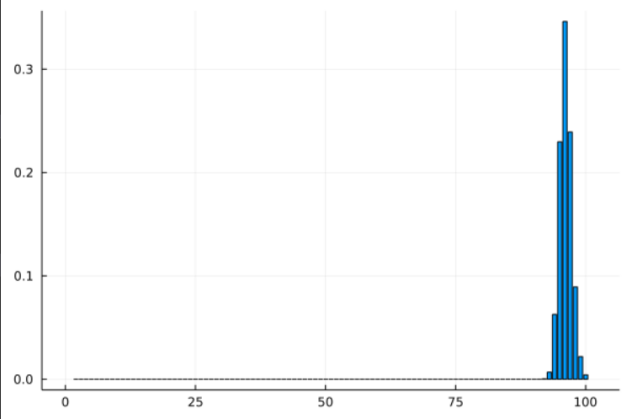
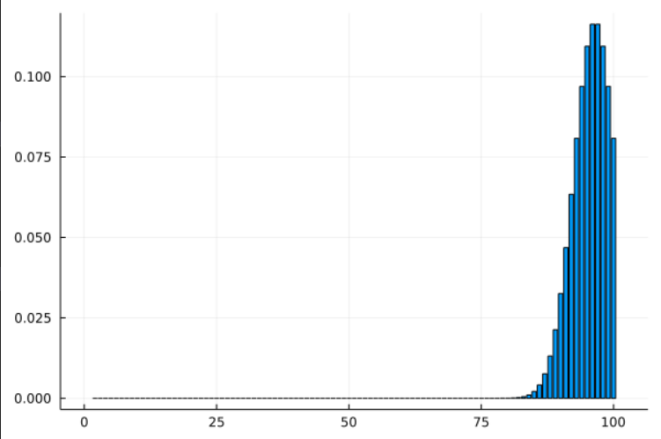
Vòng lặp 5:



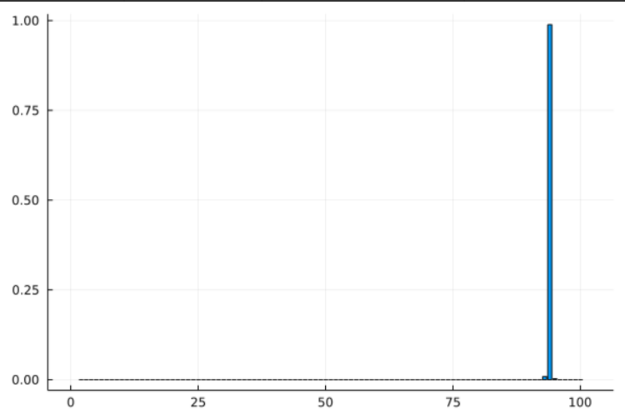
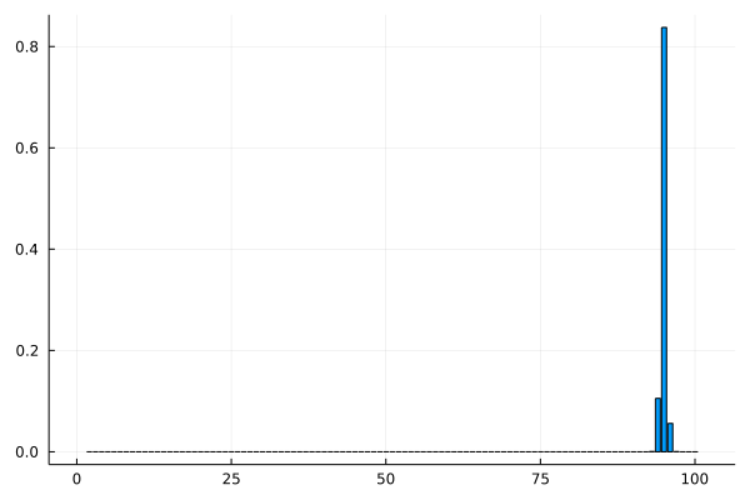
Với giá trị k và λ thấp cho thấy policy giải được có xác suất lớn nhất ở các lựa chọn nằm trong khoảng từ 90 đô đến 100 đô, và xác suất lớn dần hội tụ về một khoảng các lựa chọn nhỏ hơn sau mỗi vòng lặp. Tuy nhiên ta có để ý thấy sự dịch chuyển giá trị có xác suất lơn nhất thấp dần. Ở vòng lặp 1 các lựa chọn có xác suất cao nhất là 96 đến 97 đô với xác suất gần như là ngang nhau và bằng khoảng 0.046. Đến vòng lặp 5, xác suất cao nhất là ở lựa chọn 96 đô la khoảng 0.1125. Có sự chuyển dịch xác suất lớn nhất từ 97 đô sang 96 đô. Để kiếm tra giải thiết này chúng tôi tăng giá trị k và λ.

* k = 10, λ = 6.

Vòng lặp 1, 2:



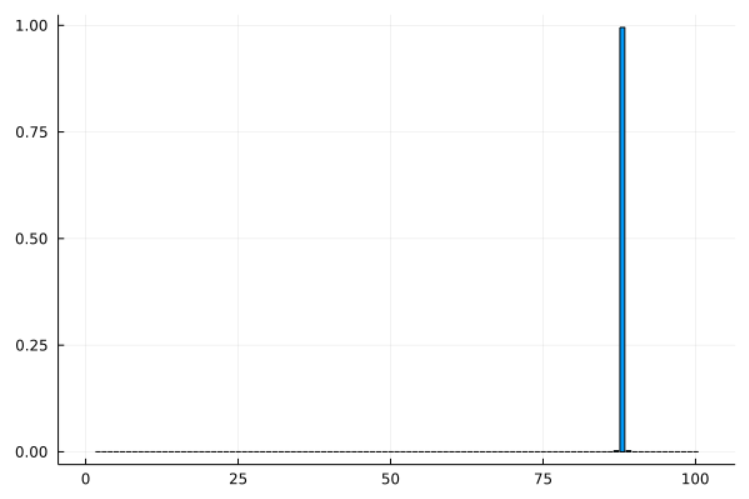
Vòng lặp 3, 4:



Qua 4 vòng lặp ta thấy với độ chính xác λ cao hơn thì sự hội tụ giá trị xác suất về một lựa chọn càng nhanh và ta có được xác suất lựa chọn 95 đô của policy tại vòng lặp này là gần đạt 1.0, do đó xác suất cho các hành động khác là gần như 0.0

…

Vòng lặp 10:

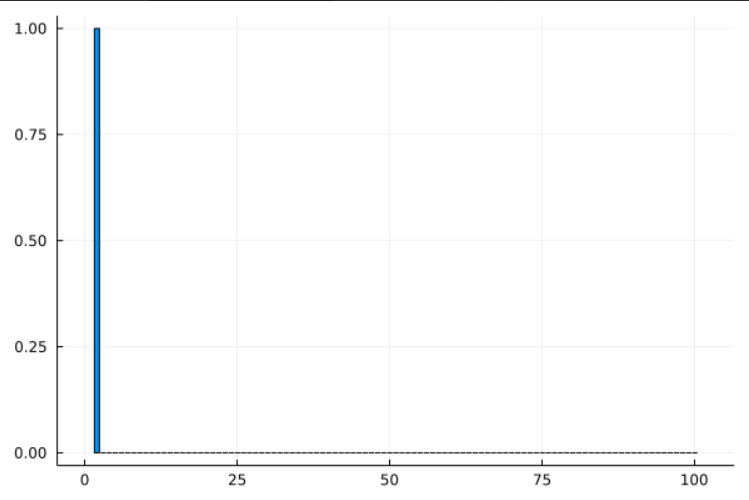


So với từ vòng lặp 4, giá trị xác suất cao nhất lúc này cũng tập trung tại một lựa chọn tuy nhiên lựa chọn lúc này đã giảm so với vong lặp 4. Tại đây lựa chọn 88 đô có xác suất gần bằng 1.0

Tại lúc này ta biết được lựa chọn có xác suất cao nhất trong policy sẽ giảm dần qua mỗi vòng lặp. Để kiểm chứng được kết quả giảm dần về đâu chúng tôi kiểm tra kết quả có được trong trường hợp vòng có số vòng lặp lớn và độ chính xác λ cao.

* k = 100, λ=7

Vòng lặp 100



Sau 100 vòng lặp, lựa chọn 2 đô la là lựa chọn có xác suất lớn nhất và gần như đạt 1.0 (cụ thể tính toán được là 0.99991692143713) trong policy. Và vì đây là lựa chọn bé nhất bởi không gian hành động là đoạn các số nguyên từ cho nên có thể biết rằng dù trải qua nhiều vòng lặp sau đó nữa, lựa chọn 2 đô la vẫn đạt xác suất lớn nhất.

Đây cũng chính là *Nash Equilibrium.* Điều này là hợp lí bởi vì xét trên thực tế, một người chơi thường không có mức độ lý luận cao như cách chúng ta chọn giá trị k, và điều đó dẫn đến đa số người chơi thường chọn giá trị khoảng từ 97 đô đến 100 đô.

* + 1. **Code:**
       1. **SimpleGame:**

|  |
| --- |
| struct SimpleGame      γ # discount factor      ℐ# agents      # joint action space      R # joint reward function  end |

* γ : Hệ số chiết khấu.
* ℐ : Tập hợp các agent.
* : Không gian hành động chung.
* R: Hàm thưởng chung.
  + - 1. **SimpleGamePolicy:**

|  |
| --- |
| struct SimpleGamePolicy      p # dictionary mapping actions to probabilities      function SimpleGamePolicy(p::Base.Generator)          return SimpleGamePolicy(Dict(p))      end      function SimpleGamePolicy(p::Dict)          vs = collect(values(p))          vs ./= sum(vs)          return new(Dict(k => v for (k,v) in zip(keys(p), vs)))      end        SimpleGamePolicy(ai) = new(Dict(ai => 1.0))  end |

* p là một thuộc tính kiểu dictionary nối tất cả các lựa chọn hành động (trong bài toán này là 2 đến 100 đô la) với xác suất cho từng hành động.
* Hàm khởi tạo SimpleGamePolicy(Base.Generator) truyền vào một đối tượng khởi tạo từ điển và gán cho thuộc tính p nội dung của từ điển đó.
* SimpleGamePolicy(p::Dict) truyền vào một đối tượng từ điển có sẵn giá trị, cần phải chuẩn hóa các giá trị trong từ điển để phù hợp với một giá trị xác suất bằng cách lấy từng giá trị trong từ điển chia cho tổng tất cả các giá trị (Nếu ban đầu các giá trị đều đã được chuẩn hóa thì tổng các giá trị trong từ điển sẽ là 1, nên việc chia cho tổng đó sẽ không gây ra việc thay đổi các giá trị xác suất).
* SimpleGamePolicy(ai) truyền vào một hành động ai, trong trường hợp này thuộc tính p của policy chỉ có một cặp key-value là với .

|  |
| --- |
| (πi::SimpleGamePolicy)(ai) = get(πi.p, ai, 0.0)  function (πi::SimpleGamePolicy)()      D = SetCategorical(collect(keys(πi.p)),collect(values(πi.p)))      return rand(D)  end |

* Nếu ta có một đối tượng SimpleGamePolicy πi thì πi(ai) trả về xác suất của hành động ai trong policy πi, πi() trả về một hành động ngẫu nhiên trong policy πi.

|  |
| --- |
| joint(X) = vec(collect(Iterators.product(X...)))  joint(π, πi, i) = [i == j ? πi : πj for (j, πj) in enumerate(π)] |

* Ta dùng joint() để khởi tạo một không gian hành động chung từ

|  |
| --- |
| function utility(::SimpleGame, π, i)      ,R = ., .R      p(a) = prod(πj(aj) for (πj, aj) in zip(π, a))      return sum(R(a)[i]\*p(a) for a in joint())  end |

* Hàm utility(::SimpleGame, π, i) tính toán utility ở trò chơi với policy chung các agent được thực thi là π trong sự tính toán của agent i. Ultility được tính với công thức tường minh:

: không gian hành động chung.

: hàm thưởng cho agent i.

: tập hợp các agent.

: policy của agent i.

* + - 1. **Response models:**

|  |
| --- |
| function best\_response(::SimpleGame, π, i)      U(ai) = utility(joint(π, SimpleGamePolicy(ai), i), i)      ai = argmax(U, [i])      return SimpleGamePolicy(ai)  end |
| function softmax\_response(::SimpleGame, π, i, λ)      i = [i]      U(ai) = utility(joint(π, SimpleGamePolicy(ai), i), i)      return SimpleGamePolicy(ai => exp(λ\*U(ai)) for ai in i  end |

* best\_response, softmax\_response trả về một policy theo lần lượt model best response và softmax response của agent i với các agent hiện tại đều đang sử dụng các policy có trong π.
* Công thức:
* best response:

: utility của agent i.

: không gian hành động cho agent i.

: policy chung của các agent khác I => là policy của các agent với agent i sử dụng policy riêng có hành động có xác suất 1.0.

* softmax response:

λ: độ chính xác.

: utility của agent i.

: policy chung của các agent khác I => là policy của các agent với agent i sử dụng policy riêng có hành động có xác suất 1.0.

: tỉ lệ với => xác suất của hành động trong policy tìm được có xác suất .

* + - 1. **Travelers:**

Code được tham khảo từ Mykel Kochenderfer, Tim Wheeler and Kyle Wray.

|  |
| --- |
| struct Travelers end  n\_agents(simpleGame::Travelers) = 2  ordered\_actions(simpleGame::Travelers, i::Int) = 2:100  ordered\_joint\_actions(simpleGame::Travelers) = vec(collect(Iterators.product([ordered\_actions(simpleGame,i)  for i in 1:n\_agents(simpleGame)]...)))  n\_joint\_actions(simpleGame::Travelers)=length(ordered\_joint\_actions(  simpleGame))  n\_actions(simpleGame::Travelers,i::Int)=length(ordered\_actions(  simpleGame, i)) |

* Struct Travelers rỗng nhằm tạo một đối tượng đại diện cho bài toán Traveler’s dilemma.

Một số hàm nhằm khởi tạo các thuộc tính cho một simple game cơ bản:

* Hàm n\_agents(simpleGame::Travelers) nhận tham số là đối tượng đại diện cho bài toán, kết quả trả về cho biết số lượng agent trong bài toán, ở đây là 2 agents.
* Hàm ordered\_actions(simpleGame::Travelers, i::Int) nhận tham số là đối tượng đại diện cho bài toán và số thứ nguyên i đại diện cho agent thứ i, kết quả trả về là tất cả các hành động (lựa chọn) hợp lệ của agent i có sắp thứ tự, trong bài toán này thì các hành động của các agent là như nhau và đó là lựa chọn từ 2 đến 100.
* Hàm ordered\_joint\_actions(simpleGame::Travelers) nhận tham số là đối tượng đại diện cho bài toán, kết quả trả về là một vector chưa tất cả các hành động chung dạng được sắp xếp có thứ tự.
* Hàm n\_joints\_actions(simpleGame::Travelers) nhận tham số là đối tượng đại diện cho bài toán, kết quả trả về là số lượng các hành động chung của các agent trong bài toán.
* Hàm n\_actions(simpleGame::Travelers, i::Int) nhận tham số là đối tượng đại diện cho bài toán, số nguyên i đại diện cho số thứ tự của agent i, kết quả trả về là số lượng hành động (lựa chọn) của agent i.

|  |
| --- |
| function reward(simpleGame::Travelers, i::Int, a)     if i == 1          noti = 2      else          noti = 1      end      if a[i] == a[noti]          r = a[i]      elseif a[i] < a[noti]          r = a[i] + 2      else          r = a[noti] - 2      end      return r  end |

* Hàm reward(simpleGame::Travelers, i::Int, a) nhận tham số đầu vào là đối tượng đại diện cho bài toán, số nguyên i đại diện cho agent i, a là một hành động chung được lựa chọn, kết quả trả ra là phần thưởng nhận được của agent i sau khi hành động a được lựa chọn.

|  |
| --- |
| function joint\_reward(simpleGame::Travelers, a)      return [reward(simpleGame, i, a) for i in 1:n\_agents(simpleGame)]  end |

* Hàm joint\_reward(simpleGame::Traverlers, a) nhận tham số đầu vào là đối tượng đại diện cho bài toán, và một hành động chung a tạo bởi hành động được lựa chọn của các agent, kết quả trả về là một phần thưởng chung có dạng với là phần thưởng cho agent i.

|  |
| --- |
| function SimpleGame(simpleGame::Travelers)  return SimpleGame(  0.9,  vec(collect(1:n\_agents(simpleGame))),  [ordered\_actions(simpleGame, i) for i in 1:n\_agents(simpleGame)],  (a) -> joint\_reward(simpleGame, a)  )  end |

* Hàm SimpleGame(simpleGame::Travelers) là một hàm khởi tạo cho đối tượng SimpleGame với tham số đầu vào là một đối tượng đại diện cho bài toán Traveler’s dilemma. Hàm này trả về một đối tượng SimpleGame với các thuộc tính phù hợp với bài toán đang xét, cụ thể sẽ là:

, , ,

* + - 1. **Iterated Best Response:**

|  |
| --- |
| struct IteratedBestResponse      k\_max # number of iterations      π # initial policy  end |

* Struct IteratedBestResponse gồm 2 thuộc tính: số nguyên k\_max và policy hiện tại của Agent là π. Thuật toán Iterated Best Response luân phiên từng agent và áp dụng mô hình Best Response cho từng agent, policy được giải ra sẽ gán vào policy π, thuật toán kết thúc sau k\_max lần duyệt.

|  |
| --- |
| function IteratedBestResponse(::SimpleGame, k\_max)      π = [SimpleGamePolicy(ai => 1.0 for ai in i) for i in ]      return IteratedBestResponse(k\_max, π)  end |

* Hàm khởi tạo IteratedBestResponse(::SimpleGame, k\_max) nhận tham số đầu vào là đối tượng SimpleGame đại diện cho bài toán, và giá trị nguyên cho thuộc tính k\_max, policy được khởi tạo là policy mà trong đó tất cả các lựa chọn hành động đều có xác suất là 1.0, nghĩa là mỗi agent có xu hướng chọn ngẫu nhiên các lựa chọn trong không gian hành động.

|  |
| --- |
| function solve(M::IteratedBestResponse, , plot = false)      agent\_best\_responses =[[],[]]      π = M.π      for k in 1:M.k\_max          π = [best\_response(π, i) for i in ]          append!(agent\_best\_responses[1], collect(keys(π[1].p))[1])          append!(agent\_best\_responses[2], collect(keys(π[2].p))[1])      end      if plot          display(bar(collect(1:k\_max),agent\_best\_responses[1], orientation = :vertical, legend = false)) # agent 1          display(bar(collect(1:k\_max),agent\_best\_responses[2], orientation = :vertical, legend = false)) # agent 2      end      return π  end |

* Hàm solve mô tả lại cách thuật toán Iterated Best Response hoạt động như đã giải thích, ngoài ra thì ta còn có tham số plot được mặc định giá trị là false. Các best response policy của các agent sau mỗi vòng lặp sẽ có hành động tốt nhất trong đó được lưu lại trong các vector agent\_best\_responses[]. Nếu người sử dụng hàm đặt giá trị plot = true thì biểu đồ mô tả sự thay đổi best response của các agent qua mỗi vòng lặp sẽ được vẽ ra.
  + - 1. **Hierarchical Softmax:**

|  |
| --- |
| struct HierarchicalSoftmax      λ # precision parameter      k # level      π # initial policy  end |

* Struct HierarchicalSoftmax gồm có 3 thuộc tính, số chấm động λ thể hiện giá trị độ chính xác trong model Softmax response, số nguyên thể hiện cho thuộc tính level đã nêu trong phần mô tả thuật toán và một policy π là policy hiện tại của agent.

|  |
| --- |
| function HierarchicalSoftmax(::SimpleGame, λ, k)  π = [SimpleGamePolicy(ai => 1.0 for ai in i) for i in ]  return HierarchicalSoftmax(λ, k, π)  end |

* Hàm khởi tạo HierarchicalSoftmax(::SimpleGame, λ, k) nhận tham số đầu vào là đối tượng SimpleGame đại diện cho bài toán, và giá trị nguyên cho thuộc tính k, giá trị số chấm động truyền cho thuộc tính λ, policy được khởi tạo là policy mà trong đó tất cả các lựa chọn hành động đều có xác suất là 1.0, nghĩa là mỗi agent có xu hướng chọn ngẫu nhiên các lựa chọn trong không gian hành động.

|  |
| --- |
| function solve(M::HierarchicalSoftmax, , plot = false)      π = M.π      for k in 1:M.k          π = [softmax\_response(π, i, M.λ) for i in ]          if plot              display(bar(collect(keys(π[1].p)),collect(values(π[1].p)),  orientation = :vertical, legend = false )) # agent 1  display(bar(collect(keys(π[2].p)),collect(values(π[2].p)),  orientation = :vertical, legend = false )) # agent 2          end      end      return π  end |

* Hàm solve mô tả lại cách thuật toán Hierarchical Softmax hoạt động như đã giải thích, ngoài ra thì ta còn có tham số plot được mặc định giá trị là false. Nếu người dùng đặt giá trị tham số plot = true thì qua mỗi vòng lặp, policy hiện tại của mỗi agent sẽ được mô tả trên một biểu đồ.

* + 1. **Phân tích:**

Với phương pháp Iterated Best Response áp dụng mô hình best response cho agent, trong những lần lặp đầu tiên thường nhận được kết quả từ 97 đô và sau mỗi lần lặp, lựa chọn được tính toán ngày càng giảm xuống và khi chạm đến 2 đô la, best response không đổi. Kết quả này tương đương *Nash Equilibrium.*

Với phương pháp Hierarchical Softmax áp dụng mô hình softmax response cho agent, trong những lần lặp đầu tiên, policy có được thường có các lựa chọn có xác suất cao nhất nằm trong khoảng 96 đến 100 đô la. Và các lựa chọn đó cũng giảm dần về 2 đô la sau nhiều vòng lặp. Kết quả này cũng tương đương *Nash Equilibrium.* Với thuộc tính độ chính xác λ của thuật toán được gán càng cao thì tốc độ hội tụ về *Nash Equilibrium* càng nhanh sau các vòng lặp. Hierarchical Softmax nhằm mô phỏng lại lựa chọn của người chơi ngoài thực tế, và tại đó hầu hết người chơi đều chọn 97 đến 100 đô. Điều này cũng là hợp lí vì trong thực tế con người thường không tư duy với độ chính xác cao và xem xét chiến lược của đối thủ quá nhiều lần. Nếu ta lựa chọn thuộc tính độ chính xác λ thấp thì kết quả nhận được policy sẽ sát với thực tế hơn.

* 1. **Tóm tắt kết quả:**

Kết quả bài toán giải được phù hợp với dự đoán sau nhiều thử nghiệm khác nhau với các giá trị tham số đầu vào khác nhau.

Điểm mạnh:

Đánh giá được 2 phương pháp giải quyết bài toán áp dụng 2 model khác nhau là best response và softmax response.

Điểm yếu:

Trong phương pháp Hierarchical Softmax, với λ cao có thể xảy ra lỗi trong tính toán.

* 1. **Trích dẫn:**

1. Zerilli, John & Knott, Alistair & Maclaurin, James & Gavaghan, Colin. (2019). Algorithmic Decision-Making and the Control Problem. Minds and Machines. 29. 10.1007/s11023-019-09513-7.
2. Mykel Kochenderfer & Tim Wheeler & Kyle Wray,

<https://github.com/algorithmsbooks/DecisionMakingProblems.jl.git>