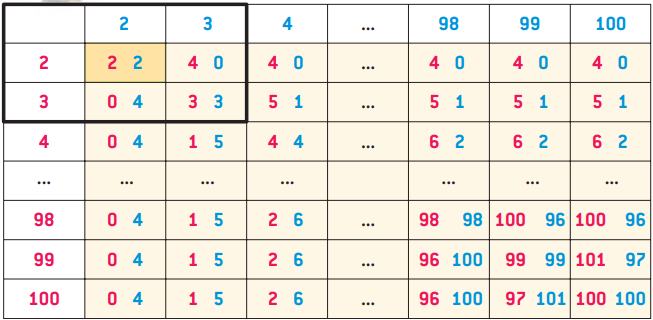
# I - Phát biểu bài toán: mô tả các bài toán cần giải quyết.

Bài toán “Travel’s dilemma” nói về việc 2 người bị mất hành lý ở 1 sân bay và 2 hành lý đó giống y hệt nhau. Người quản lý sân bay vì muốn tránh lừa đảo nên đã đưa ra phương pháp sau: Cho 2 người viết xuống 1 tờ giấy giá tiền của hành lý trong khoảng giữa 2$ và 100$ và họ không được trao đổi với nhau. Nếu 2 người viết giá tiền bằng nhau, thì quản lý sẽ cho là 2 người đều thành thật và 2 người đều nhận được số tiền đó. Nếu 2 người viết giá tiền khác nhau, thì người viết giá tiền cao hơn sẽ nhận được số tiền bằng giá tiền thấp hơn -2, còn người viết giá tiền thấp hơn sẽ được thưởng 2$ vào giá tiền thấp hơn đó.

Ví dụ với 2 người Tuấn và Lan, ban đầu Tuấn nghĩ rằng mình sẽ ghi số lớn nhất là 100$ và sẽ nhận được 100$ nếu Lan cũng tham lam như vậy. Nhưng rồi Tuấn nghĩ là nếu mình viết 99$, thì mình sẽ nhận được 101$ và Lan nhận được 98$, nhưng sau đó Tuấn chợt nhận ra rằng Lan cũng sẽ nghĩ giống mình là cũng sẽ viết 99$. Thế là Tuấn lại viết 98$, và lại nghĩ rằng Lan nghĩ giống mình, quy trình này lặp lại cho tới khi cả 2 người viết 2$ trên giấy và cùng đi về với 2$. Kết quả này là Nash Equilibrium của bài toán, và cách suy nghĩ này được gọi là quy nạp ngược (Backward induction).

Nhưng thực tế con người sẽ không có suy nghĩ như vậy và chỉ viết trong khoảng từ 95$ tới 97$. Bài toán này là một nghịch lý trong Game Theory, con người hành động bất hợp lý (irrational) lại mang về lợi ích nhiều hơn hành động hợp lý (rational). Như John Nash từng nói “Game theory predicts that the Nash equilibrium will occur when Traveler’s Dilemma is played rationally.”.

Việc thưởng phạt dựa trên lựa chọn của 2 người có thể được biểu diễn bằng 1 payoff matrix như sau:



Hình 1: The Traveler’s Dilemma, Kaushik Basu, Scientific American , Vol. 296, No. 6 (JUNE 2007), pp. 90-95

Payoff matrix này có thể tóm gọn hết ý tưởng của bài toán Travel’s Dilemma cho người xem, cột bên trái cùng là đại diện cho những lựa chọn của Lan, hàng trên cùng là đại diện cho những lựa chọn của Tuấn. Như ta thấy thì khi Lan chọn 100, Tuấn chọn 100 thì cả 2 người đều nhận được 100, và khi Tuấn chọn 100, Lan chọn 99 thì Tuấn nhận được 97 còn Lan lại nhận được 101, và ngược lại. Bảng cứ theo quy luật này đi lên cho tới khi gặp (2; 2) là Nash equilibrium, (2; 2) là Nash equilibrium vì nó là trạng thái cuối cùng mà người chơi sẽ nghĩ tới và sẽ không muốn chuyển qua trạng thái nào nữa.

# II - Thách thức: khó khăn, thách thức cụ thể đối với từng bài toán.

Thách thức lớn nhất về bài Traveler’s Dilemma đó chính là sự nghịch lý về hành vi hợp lí và hành vi thực tế của con người như đã có đề cập ở trên. Hành vi hợp lí theo Lý thuyết trò chơi đó chính là người chơi sẽ luôn cố gắng làm cho mình hơn đối thủ, nên sẽ cứ tiếp tục suy nghĩ tới số tiền nhỏ hơn vì nó sẽ làm cho mình nhận được tiền thưởng cao hơn đối thủ, hành vi này hội tụ về việc cả 2 người cùng chọn 2 – Nash equilibrim – vì sau đó không còn lựa chọn nào mà 2 người sẽ muốn chọn nữa. Còn hành vi thực tế của con người là sẽ không quan tâm nhiều tới người kia mà quan tâm tới số tiền mình nhận được, nên thường lựa chọn sẽ nằm dao động ở khoảng trên 95. Việc mâu thuẫn như vậy dẫn tới việc chọn policy để mô phỏng lại bài toán phải được cân nhắc kỹ cho cả 2 loại hành vi.

Còn khó khăn đối với em đó chính là việc hiểu và tìm được code tham khảo cho bài toán này, vì lý thuyết trò chơi và những khái niệm liên quan là rất mới đối với em nên em còn hơi lúng túng khi tra cứu trên mạng.

# III - Thực nghiệm:

1. **Mô hình hoá tính toán:**
2. **Simple game**

struct SimpleGame

      γ  # discount factor

      ℐ  # agents

      𝒜  # joint action space

      R  # joint reward function

end

Trong simple game, những agent chọn một hành động ai để tối ưu hoá phần thưởng ri của họ.

Một joint action space bao gồm mọi hoán vị của những hành động mà mỗi agent có thể có. Những hành động được chọn cùng một lúc qua khắp các agent có thể được gộp lại thành 1 joint action từ joint action space trên.

Joint reward function mô tả phần thưởng nhận được từ joint action . Joint reward là .

Còn với γ (discount factor), đó chỉ là biến đại diện cho xác suất “đi tiếp” của lần lặp hiện tại để cho nó không lặp vô tận, giá trị này nằm giữa 0 và 1 nhưng không thể bằng 0 và 1.

Với bài toán traveler’s dilemma, discount factor sẽ bằng 0.9, agents sẽ gồm 1 và 2, joint action space sẽ là một list chứa các hành động chọn 2…100 của mỗi agent, và joint reward function sẽ gồm 2 hàm reward giống hệt nhau của 2 agent có dạng như sau:

1. **Iterated best response**

struct IteratedBestResponse

      k\_max # number of iterations

      π # initial policy

end

1. **Hierarchical softmax**

struct HierarchicalSoftmax

      λ # precision parameter

      k # level

      π # initial policy

end

Cụ thể của 2 mô hình tính toán Iterated best response và Hierarchical softmax sẽ được nói rõ ở phần 2.

1. **Phương pháp giải quyết, phương pháp, thuật toán được sử dụng cho việc giải quyết bài toán**

Đối với bài này thì em sẽ chọn 2 cấu hình policy Iterated Best Response và Hierarchical Softmax, tương ứng cho 2 hành vi là hành vi hợp lí và hành vi thực tế.

1. **Iterated Best Response**
2. **Best response**

Với việc người chơi biết những gì mà những người chơi khác đang làm, một chiến thuật là một best response nếu và chỉ nếu người chơi không thể có lợi hơn khi chuyển sang chiến thuật khác. Có thể có nhiều best response.

Nếu chúng ta chỉ sử dụng những chính sách xác định (deterministic policies), thì một phản ứng mang tính xác định (deterministic best response) cho đối thủ đang 1 dùng 1 chính sách nào đó sẽ được tính một cách dễ dàng. Chúng ta chỉ cần duyệt qua hết hành động của agent thứ tự I và trả về cái mà cho mình lợi ích tốt nhất:

Định nghĩa Nash Equilibrium bằng Best response: một trò chơi đang ở trong trạng thái Nash equilibrium nếu và chỉ nếu tất cả người chơi đều đang có phản ứng tốt nhất (best response) cho việc mà những người chơi khác đang làm.

Cách tìm best response cho 1 trò chơi là cô lập 1 chiến thuật của 1 người chơi, và tìm xem phản ứng tốt nhất mà những người chơi còn lại nên có để đáp trả chiến thuật đó. Cụ thể với traveler’s dilemma:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 2 | 3 | 4 | 5 | … |
| 2 | 2; 2 | 4; 0 | 4; 0 | 4; 0 |  |
| 3 | 0; 4 | 3; 3 | 5; 1 | 5; 1 |  |
| 4 | 0; 4 | 1; 5 | 4; 4 | 6; 2\* |  |
| 5 | 0; 4 | 1; 5 | 2; 6 | 5; 5 |  |
| … |  |  |  |  |  |

Những lựa chọn ở cột trái cùng sẽ là của player1, còn những lựa chọn ở hàng trên cùng sẽ là của player2. Em chỉ xét những giá trị từ 2 tới 5 để dễ ví dụ.

Như bảng thì em đã cô lập chiến thuật của player2 là chọn giá tiền 5, thì với chiến thuật của player2 như vậy, thì chiến thuật tốt nhất của player1 sẽ là chọn 4 vì nó cho ra giá tiền cao nhất cho player đó, là 6. Cho nên best response của player1 khi player2 chọn 5 là chọn 4.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 2 | 3 | 4 | 5 | … |
| 2 | 2; 2 | 4; 0 | 4; 0 | 4; 0 |  |
| 3 | 0; 4 | 3; 3 | 5; 1\* | 5; 1 |  |
| 4 | 0; 4 | 1; 5 | 4; 4 | 6; 2\* |  |
| 5 | 0; 4 | 1; 5 | 2; 6 | 5; 5 |  |
| … |  |  |  |  |  |

Tương tự, khi cô lập chiến thuật chọn 4 của player2, thì chiến thuật tốt nhất của player1 là 3 vì nó cho ra giá tiền lớn nhất cho player1 là 5, nên best response của player1 khi player2 chọn 4 là chọn 3.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 2 | 3 | 4 | 5 | … |
| 2 | 2\*; 2\* | 4; 0 | 4; 0 | 4; 0 |  |
| 3 | 0; 4 | 3; 3 | 5; 1\* | 5; 1 |  |
| 4 | 0; 4 | 1; 5 | 4; 4 | 6; 2\* |  |
| 5 | 0; 4 | 1; 5 | 2; 6 | 5; 5 |  |
| … |  |  |  |  |  |

Giờ em sẽ xét cùng lúc 2 chiến thuật của 2 người chơi là chọn 2. Em sẽ cô lập lựa chọn 2 của player2 trước (cột màu đỏ), thì khi player2 chọn 2, best response của player1 sẽ là 2 vì nó cho ra giá tiền cao nhất cho player1 là 2 thay vì 0 như những lựa chọn dưới. Tương tự, khi cô lập lựa chọn 2 của player1 (hàng màu xanh), thì player2 tốt nhất nên chọn 2 luôn vì nó cũng ra giá tiền cao nhất là 2 thay vì là 0 như những lựa chọn sau.

Vì ô chiến thuật đó, cả 2 player đều có best response cho lẫn nhau, nên (2; 2) chính là Nash equilibrium cho bài toán.

1. **Iterated best response**

Với phương pháp tìm Nash Equilibrium bằng Best response em vừa mô tả ở trên, thì chi phí sẽ rất là cao đối với 1 bài toán có nhiều hành vi như vậy, cho nên, ta phải có 1 phương pháp tối ưu hơn để tính được Nash Equilibrium cho bài toán.

Iterated best response tính được Nash Equilibrium bằng cách lặp lại trò chơi 1 số lần cố định, cụ thể là k\_max lần, với mỗi người chơi chọn 1 chính sách ban đầu, mỗi lần lặp như vậy 2 người chơi sẽ chọn chiến thuật dựa vào chiến thuật ở lần lặp trước, và ta sẽ tính best response cho 2 chiến thuật đó. Việc lặp lại như vậy sẽ có khả năng cao hội tụ về Nash Equilibrium. Nhưng nó chỉ hội tụ với 1 số loại trò chơi, nên việc quan sát quá trình lặp là một điều phổ biến.

1. **Code**

* **Flow chính:**

IteratedBestResponse(𝒫::SimpleGame, k\_max)

best\_response(𝒫::SimpleGame, π, i)

solve(M::IteratedBestResponse, 𝒫)

IteratedBestResponse: hàm tạo 1 cấu hình IteratedBestResponse (như phần mô hình hoá tính toán ở trên). Lấy vào 1 SimpleGame cũng đã có đề cập ở phần mô hình hoá tính toán, và số lần lặp k\_max. Chính sách ban đầu đó chính 1 danh sách các SimpleGamePolicy trong đó mỗi SimpleGamePolicy chứa 1 dictionary chứa các cặp (action; prob) đối với mọi action trong từng joint action trong joint action space của từng agent. Prob của các action sẽ bằng nhau và bằng 1.0/số action của joint action hiện tại.

Solve: Hàm gọi hàm best\_response cho 2 agent, để 2 kết quả vào 1 list, rồi sau đó dùng lại list đó để tính best\_response cho lần lặp sau, cứ làm vậy k\_max lần, và sau cùng sẽ trả về list kết quả sau k\_max lần lặp đó.

Best\_response: tính best response cho joint policy π hiện tại khi đang là lượt của agent có thứ tự i.

* **Hàm hỗ trợ:**

utility(𝒫::SimpleGame, π, i)

Để tính toán lợi ích của việc thực hiện joint policy π trong simple game 𝒫 khi đang là lượt của agent có thứ tự i.

SimpleGamePolicy(p::Dict)

Tham số truyền vào chính là 1 từ điển chứa các cặp (action; prob), hàm này nó chỉ có tác dụng là với mỗi cặp đó, nó tính phần trăm prob trong tổng số tất cả prob trong tất cả cặp, rồi gán lại số phần trăm vào trong prob của mỗi cặp.

joint(X)

Hàm này chỉ dùng hàm Iterators.product để phân phối mọi mảng trong X với nhau.

joint(π, πi, i)

Hàm này thay thế policy vị trí i trong joint policy π bằng policy mới πi.

1. **Kết quả thử nghiệm**

Kết quả thử nghiệm của bài toán sẽ là giá trị trả về của hàm solve, là một joint policy, như trên đã nói, là 1 list các SimpleGamePolicy, mỗi SimpleGamePolicy chứa biến p chính là từ điển chứa các cặp (action; prob). Sau khi chạy xong thuật toán với k\_max = 100, giá trị trả về là 1 list chứa 2 SimpleGamePolicy, giá trị của biến p của mỗi SimpleGamePolicy đều là một từ điển có 1 phần tử có key là 2 và value là 1.0, tức là action = 2, prob = 1.0 (khi print ra có dạng Dict(2 => 1.0) Dict(2 => 1.0)), và đó chính là Nash equilibrium mà ta đã dự đoán.

Ý nghĩa: Iterated Best Response là một phương pháp hiệu quả, tối ưu và đáng tin cậy để tính được Nash equilibrium và đồng thời dễ cài đặt.

1. **Hierarchical Softmax**
2. **Softmax response**

Ngoài Best response thì ta cũng có thể sử dụng Softmax response để mô hình hoá cách mà một agent sẽ chọn hành động của họ. Cách suy nghĩ của con người hoàn toàn không giống những cỗ máy tối ưu - luôn mang về lợi ích cao nhất. Cơ chế cơ bản của Softmax response nói rằng con người sẽ càng dễ mắc lỗi hơn khi những lỗi đó càng không nghiêm trọng. Chon một tham số chính xác (precision parameter) λ ≥ 0, mô hình này lựa chọn hành động ai dựa theo:

Khi λ tiến về 0, agent sẽ ít nhạy cảm với sự khác biệt về lợi ích, cho nên mọi lựa chọn họ đều chọn ngẫu nhiên. Còn khi λ tiến về vô cực, chính sách này sẽ trở về deterministic best response. Chúng ta có thể xem λ như một biến mà ta có thể học từ dữ liệu, cách tiếp cận theo cách học như vậy sẽ dẫn tới việc hành vi là những dự đoán thay vì là những hành vi có căn cứ. Nhưng việc có một mô hình dự đoán hành vi như vậy sẽ rất có ích trong việc xây dựng 1 mô hình hành vi có căn cứ.

1. **Hierarchical Softmax**

Khi ta muốn xây dựng một hệ thống thực hiện lựa chọn mà phải tương tác với con người, thì việc tính toán Nash equilibrium luôn luôn không phải là tốt nhất, vì con người thường sẽ không dùng những chiến thuật tiến tới Nash equilibrium. Vì lí do đầu tiên là trong 1 trò chơi có thể có nhiều trạng thái equilibria khác nhau nên người chơi sẽ không biết nên chọn equilibrium nào để thích ứng. Với những game chỉ có 1 equilibrium thì con người cũng sẽ không thể nào tính ra, đơn giản vì độ lớn của game và sự giới hạn của bộ não con người. Cho dù họ nghĩ họ có khả năng tính ra được Nash equilibrium thì họ lại nghĩ đối phương của họ sẽ không tính ra, cho nên việc chơi tối ưu như vậy là một điều tốn sức.

Hierarchial Softmax là một hệ thống rất tốt cho loại hệ thống nói trên, nó kết hợp phương thức tuần tự như IBR với softmax response. Phương thức Hierarchical softmax này mô hình hoá chiều sâu của sự hợp lí (depth of rationality) của một agent bằng 1 biến level k >= 0.

Một agent level 0 chọn hành động của nó 1 cách ngẫu nhiên, một agent level 1 thì sẽ giả sử đối phương dùng chiến thuật level 0 và sẽ chọn hành động theo một softmax response với độ chính xác λ (không đổi trong suốt quá trình chạy tuần tự) để đáp lại chiến thuật level 0 đó. Một cách tổng quát thì một agent level k sẽ chọn hành động theo softmax response khi đối phương của họ là level k-1.

1. **Code**

* **Flow chính:**

HierarchicalSoftmax(𝒫::SimpleGame, λ, k)

solve(M::HierarchicalSoftmax, 𝒫)

softmax\_response(𝒫::SimpleGame, π, i, λ)

HierarchicalSoftmax: hàm tạo 1 cấu hình HierarchicalSoftmax (cấu trúc như phần mô hình hoá tính toán ở trên), lấy vào 1 SimpleGame, tham số chính xác, và số level k mà mình sẽ lặp (số lần lặp).

solve: Hàm dường như có các bước thực hiện như IBR, vì cả 2 phương pháp này đều là phương pháp tuần tự, chỉ là thay vì gọi best\_response thì sẽ gọi softmax\_response cho mỗi joint policy mới mỗi lần lặp.

softmax\_response: tính softmax response cho joint policy π hiện tại khi đang là lượt của agent có thứ tự I, với độ chính xác λ.

* **Hàm hỗ trợ: sử dụng lại những hàm hỗ trợ được sử dụng ở IBR.**

1. **Kết quả thử nghiệm**

Tham số chính xác đó được em chọn là 0.5 vì kết quả nó đưa ra giống với mong đợi của em nhất (là lựa chọn tập trung ở 95 trở lên).

Kết quả thử nghiệm của bài toán cũng sẽ là giá trị trả về của hàm solve, cũng là một joint policy, là 1 list các SimpleGamePolicy, mỗi SimpleGamePolicy chứa biến p chính là từ điển chứa các cặp (action; prob). Sau khi chạy xong thuật toán với λ = 0.5, k = 10, giá trị trả về là 1 list chứa 2 SimpleGamePolicy tương đồng nhau, giá trị của biến p của mỗi SimpleGamePolicy đều là một từ điển có 99 phần tử, khi print ra các phần tử của từ điển đó của SimpleGamePolicy thứ nhất, ta có:

2 : 1.8290008908057377e-21

3 : 3.015512672800902e-21

4 : 4.971739885712644e-21

5 : 8.197013301962656e-21

6 : 1.3514590187157728e-20

7 : 2.2281792306362172e-20

8 : 3.673646492482178e-20

9 : 6.056819113188285e-20

10 : 9.986006504696612e-20

11 : 1.6464141333643143e-19

12 : 2.7144780020590605e-19

13 : 4.475417620842363e-19

14 : 7.378716226748965e-19

15 : 1.216544639350121e-18

16 : 2.0057430236527603e-18

17 : 3.3069111866546966e-18

18 : 5.4521748137538e-18

19 : 8.9891165870114e-18

20 : 1.4820547721809034e-17

21 : 2.4434952272372878e-17

22 : 4.028642556000359e-17

23 : 6.642108674125526e-17

24 : 1.0950985853332579e-16

25 : 1.8055123311525616e-16

26 : 2.976786584882597e-16

27 : 4.907891360830721e-16

28 : 8.091744880886994e-16

29 : 1.3341031902197235e-15

30 : 2.199564307024146e-15

31 : 3.626468459263477e-15

32 : 5.9790356863107305e-15

33 : 9.857763314295432e-15

34 : 1.625270405780575e-14

35 : 2.6796178886497197e-14

36 : 4.4179430103649365e-14

37 : 7.283956613928514e-14

38 : 1.2009214204237882e-13

39 : 1.9799846902913018e-13

40 : 3.264442874541648e-13

41 : 5.382156404236299e-13

42 : 8.873675745882943e-13

43 : 1.4630217951488644e-12

44 : 2.412115153147744e-12

45 : 3.976905560339961e-12

46 : 6.5568087888089345e-12

47 : 1.0810350117780378e-11

48 : 1.7823254182241037e-11

49 : 2.938557828156433e-11

50 : 4.8448627959768935e-11

51 : 7.987828344027057e-11

52 : 1.3169702493900404e-10

53 : 2.1713168620704838e-10

54 : 3.5798962932666043e-10

55 : 5.90225115838168e-10

56 : 9.731167010191914e-10

57 : 1.6043981985034359e-9

58 : 2.645205422028757e-9

59 : 4.361206405209267e-9

60 : 7.1904136589165336e-9

61 : 1.1854987653010073e-8

62 : 1.9545569514813027e-8

63 : 3.222519405253193e-8

64 : 5.313035703034358e-8

65 : 8.759713383619207e-8

66 : 1.4442321453408088e-7

67 : 2.3811350815434896e-7

68 : 3.925824859816133e-7

69 : 6.472582259639021e-7

70 : 1.0671460421185993e-6

71 : 1.7594199562340443e-6

72 : 2.9007756483383676e-6

73 : 4.782523058983758e-6

74 : 7.884918504918905e-6

75 : 1.2999682239840421e-5

76 : 2.1431899609680266e-5

77 : 3.533263849193083e-5

78 : 5.824663284775086e-5

79 : 9.601333218232245e-5

80 : 0.00015824724725372054

81 : 0.0002607644335682796

82 : 0.00042954464934018646

83 : 0.0007071600191436851

84 : 0.0011630947678960047

85 : 0.0019100130901855202

86 : 0.003128607447763954

87 : 0.005103447339925789

88 : 0.00826920041120103

89 : 0.013256029860947326

90 : 0.02089691869915554

91 : 0.03211305066565022

92 : 0.04755170974250347

93 : 0.06692074604726286

94 : 0.0882950984900803

95 : 0.10811653132611773

96 : 0.12242106735293848

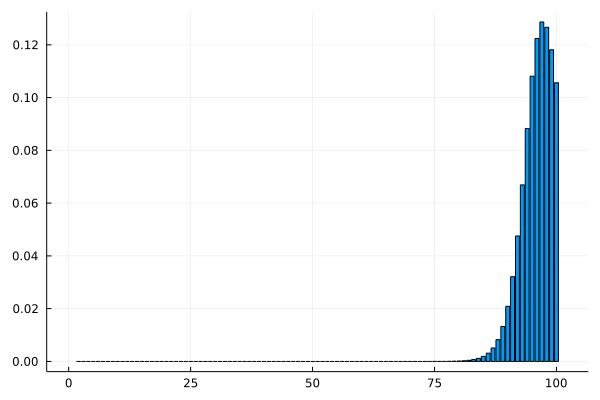
97 : 0.12865448100069946

98 : 0.1266419725577256

99 : 0.11813095698074508

100 : 0.10562729389624304

Khi vẽ ra biểu đồ:



Ý nghĩa: Ta thấy rằng Hierarchical Softmax đã dự đoán đúng hành động thực tế của con người, là tập trung chọn những giá tiền cao và hầu như không chọn những giá tiền thấp – những lựa chọn gần Nash equilibrium. Suy ra Hierarchical là một trong những phương pháp hiệu quả và rất tốt cho mô phỏng sự “bất hợp lí” của hành vi con người, và đồng thời dễ cài đặt.

# IV - Tóm tắt kết quả:

1. **Dựa trên các tiêu chí đánh giá kết quả của bạn.**

* Hiểu cũng như mô tả rõ ràng các bài toán cần giải quyết: 18/20%.
* Sử dụng các giải pháp phù hợp cho từng bài toán: 20/20%
* Code: 15/20%
* Báo cáo trình bày tốt, giải thích rõ ràng: 17/20%

1. **Điểm mạnh và điểm yếu trong đồ án này của bạn.**

* **Điểm mạnh:**
* Khả năng tự học cao, sớm lĩnh hội được cú pháp của Julia.
* Hiểu được ý tưởng của bài toán Traveler’s Dilemma khá nhanh, tìm được nhiều bài báo khoa học liên quan.
* Tìm ra và phân loại được policy nào cho hành vi nào.
* Phân code ra thành nhiều file để tiện quản lý.
* **Điểm yếu:**
* Bước đầu gặp nhiều khó khăn trong việc tìm được mã nguồn để tham khảo.
* Mã nguồn tham khảo, dù biết cú pháp Julia nhưng có những hàm thuộc những thư viện của Julia thì phải tìm hiểu thêm, cộng với việc tìm hiểu cách hệ thống Pkg hoạt động.