UNIVERSIDAD NACIONAL DE INGENIERIA

FACULTAD DE CIENCIAS

CIENCIAS DE LA COMPUTACIÓN

Optimal play of the dice game pig

Alumnos

Luis Ramos Grados
Carlos Espinoza Mansilla

Profesor

Cesar Lara Avila

2018

Optimal play of the dice game pig

Luis Ramos Grados, Carlos Alberto Espinoza Mansilla

Introducción

El objetivo de nuestro proyecto es encontrar una forma óptima de jugar el "play of the dice game pig" para esto es necesario conocer las reglas del juego:

El juego inicia

- El primer jugador tira el dado
- Si saca 1 el puntaje acumulado durante su turno se vuelve cero y su turno termina
- Si saca de 2 a 6 el resultado se suma al puntaje de ese turno
- Ahora el jugador decide si vuelve a tirar el dado o terminar su turno
- Si el jugador termina su turno el puntaje acumulado durante su turno se suma al puntaje del jugador
- Si el jugador decide volver a lanzar se repetira el proceso hasta que decida terminar su turno o saque 1
  *El juego termina cuando un jugador logra acumular 100 o mas puntos.

A pesar de que este juego tiene muchas mas variaciones y puede jugarse de 2 a 10 jugadores aqui tomaremos la forma estandar del juego que es la ya descrita y jugada por 2 personas con un dado de 6 lados justo. Trataremos de maximizar nuestrar probabilidades de ganar usando nuestro conocimientos sobre probabilidades y luego computaremos los algoritmos que sean necesarios para resolver el problema con las herramientas que nos da R.

Estado del arte

1. Practical Play of the Dice Game Pig por Todd W. Neller, Clifton G.M. Presser: Siendo conciente de lo complicado del resultado "óptimo" para que un humano lo aplique en un entorno casual, busca una forma que sin ser la mas óptima resulta mas conveniente para jugar de forma optimo de modo casual.
2. Pig (Pig-out): Análisis de una de las variaciones del juego original esta vez con 2 dados.

Diseño del experimento

Primero vamos a definir de que cosas depende la probabilidad de ganar de nuestro jugador: Para un turno cualquiera de nuestro jugador la posibilidad de ganar va a depender de cuantos puntos tengemos (i), de cuantos puntos tiene nuestro oponente (j), de cuantos puntos tenemos acumulados en nuestro turno (k) y de decidir si volver a lanzar el dado o terminar el turno dependiendo de cual me da mas posibilidades de ganar, entonces definimos nuestras posibilidades de ganar como:

- $P_{i,j,k} = max(P_{i,j,k,lanzar}, P_{i,j,k,terminar})$

Donde $P_{i,j,k}$ es la posibilidad de ganar total, $P_{i,j,k,lanzar}$ es la posibilidad de ganar lanzando de nuevo el dado y $P_{i,j,k,terminar}$ es la posibilidad de ganar terminando ese turno.

Ahora si consideramos la situación en la que $i + k \geq 100$ entonces $P_{i,j,k} = 1$ porque le basta al jugador con terminar allí su turno para ganar, también podemos establecer limites para i,j,k:

- $0 \leq i < 100$, $0 \leq j < 100$ y $0 \leq k < 100 - i$

ahora la probabilidad de ganar terminando nuestro turno va a ser igual a la probabilidad de que nuestro oponente no gane:

- $P_{i,j,k,terminar} = 1 - P_{j,i,0}$

Mientras que la probabilida de ganar volviendo a lanzar el dado es la suma de las probabilidades de ganar obteniendo (1,2,3,4,5,6) multiplicado por la posibilidad de que se de dicho resultado:

- $P_{i,j,k,lanzar} = [\frac{1}{6}P_{i,j,k+1} + \frac{1}{6}P_{i,j,k+2} + \frac{1}{6}P_{i,j,k+3} + \frac{1}{6}P_{i,j,k+4} + \frac{1}{6}P_{i,j,k+5} + \frac{1}{6}P_{i,j,k+6}]$

pero $P_{i,j,k+1}$ significa que todo el k acumulado se reduce a 0 y termina tu turno por lo $P_{i,j,k+1} = 1 - P_{j,i,0}$ ahora:

- $P_{i,j,k,lanzar} = \frac{1}{6}[(1 - P_{j,i,0}) + P_{i,j,k+2} + P_{i,j,k+3} + P_{i,j,k+4} + P_{i,j,k+5} + P_{i,j,k+6}]$

Bueno, ya hemos construido un modelo matemático para calcular la maxima probabilidad de ganar para cada jugada posible solo queda resolver.

Experimentos y resultados

Para resolver $P_{i,j,k} = max(P_{i,j,k,lanzar}, P_{i,j,k,terminar})$ Notamos que nuestro modelo tiene resultados que dependen en parte de la probabilidad (resultado del dado), en parte de la decision que tome el jugado (lanzar el dado o terminar su turno) vemos que tal cual descrito, nuestro modelo se adecua a un MDP (Markov decision process), esto quiere decir que podemos intentar resolverlo mediante uno de los metodos conocidos de solución para MDP en este caso usaremos "VALUE ITERATION".

Value Iteration

Value iteration es un algoritmo que de forma ciclica estima valores para cada estado de un problema hasta llegar a un valor muy aproximado al ótimo. Ahora vamos a considerar la división del problema en estados(s),acciones(a) y recompensas(r) donde para nuestro problema los estados deben ser entendidos como cada situacion posible durante el juego para los i,j,k ,por ejemplo un estado seria la situación donde tengo i=96,j=99,k=2 , tengo 98 puntos , mi oponente 99 y tengo acumulado en ese turno 2 puntos, las acciones son lanzar el dado o terminar el turno y recompensa se define como un valor númerico que sirve para determinar que cambio de estado es mas optimo. Si se tiene 2 estados $s, s^{'} \in S$ y tomando la acción $a \in A$ se sabe que hay una $P^a_{ss'}$ de que haya un cambio de estado de s a s' a este resultado se le asocia la recompensa $R^a_{ss'}$. También es necesario definir V(s) como el valor de estar en el estado "s" basado en las recompensas necesarias para llegar a ese estado y los valores de los estados previos, decimos entonces que el valor estimado de una acción a en el estado s es dado por:

- $\sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V(s')]$

y entonces la opcion optima es:

- $max_{a \in A}(\sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V(s')])$

Donde $0 \leq \gamma < 1$ es llamado factor de descuento en indica que tanto nos interesa la recompensa futura de escoger la acción a.

Aplicando esto a nuestro problema es:

- $max(P_{i,j,k,lanzar}, P_{i,j,k,terminar})$

ya que vimos que solo hay 2 acciones y consideramos que la recompensa solo existe y es igual a 1 si se pasa de "no ganar" a "ganar",ademas de dar $\gamma = 1$

Ahora usamos el algoritmo 1 "value update" o "Bellman update/back-up".

Aplicar el algoritmo directamente al "play of the dice game pig" resulta muy complicado por lo que aquí se hará primero un ejemplo con un juego mucho mas simple.

Definamos pues las reglas basicas de modo similar al "play of the dice game pig"

- jugador lanza moneda, si sale cruz termina su turno
- si sale cara su puntaje de turno sube 1 punto y puede elegir si seguir lanzando o terminar su turno

Algorithm 1: Bellman update/back-up

Input: Vector de Probabilidades de$S_{i,j,k}$ arbitrariamente inicializado
Output: Vector de Probabilidades de$S_{i,j,k}$ óptimo

Repetir
  $\triangle < -0$
  Para cada $s \in S$
    $v < -V(s)$
    $V(s) < -max_{a \in A}(\sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V(s')])$
    $\triangle < -max(\triangle, |v - V(s)|)$
hasta que $\triangle < e$
donde $0 < e < 1$ **indica la aproximación a los valores óptimos.**

- gana quién llegue a 2 puntos primero

El ejemplo esta dado para que sea una versión mas simple y nos permita guiarnos a la solución buscada. Definiendo i como puntaje del jugador, j como puntaje del oponente y k como puntaje de turno, en el caso que i+k=2 $P_{i,j,k} = 1$ en el resto de casos donde 0< i,j < 2 y k<2-i , la probabilidad de ganar es:

- $max(P_{i,j,k,lanzar}, P_{i,j,k,terminar})$

Donde: * $P_{i,j,k,terminar} = 1$ -P_{j,i,0} $ * $P_{i,j,k,lanzar} = \frac{1}{2}[1 - P_{j,i,0} + P_{i,j,k+1}]$

luego todos los estados posibles son:

- $P_{0,0,0} = max(\frac{1}{2}[1 - P_{0,0,0} + P_{0,0,k+1}], 1 - P_{0,0,0})$
- $P_{0,0,1} = max(\frac{1}{2}[1 - P_{0,0,0} + 1], 1 - P_{0,1,0})$
- $P_{0,1,0} = max(\frac{1}{2}[1 - P_{1,0,0} + P_{0,0,k+1}], 1 - P_{1,0,0})$
- $P_{0,1,1} = max(\frac{1}{2}[1 - P_{1,0,0} + 1], 1 - P_{1,1,0})$
- $P_{1,0,0} = max(\frac{1}{2}[1 - P_{0,1,0} + 1], 1 - P_{0,1,0})$
- $P_{1,0,1} = 1$
- $P_{1,1,0} = max(\frac{1}{2}[1 - P_{1,1,0} + 1], 1 - P_{1,1,0})$
- $P_{1,1,1} = 1$

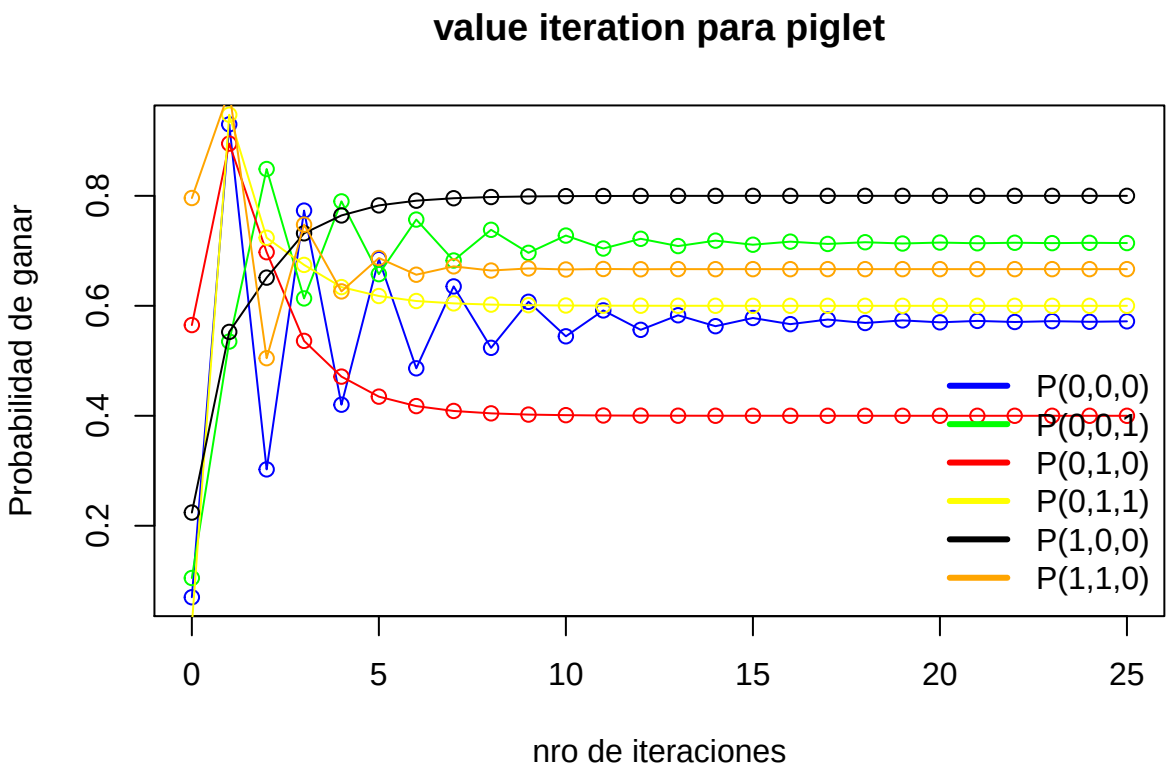
Implementando el algoritmo: archivo 'value iteration'.R

```
## Arreglo generado
```

```
## , , 1
##
##       [,1]  [,2]
## [1,] 0.070 0.565
## [2,] 0.105 0.019
##
## , , 2
##
##       [,1]  [,2]
## [1,] 0.224 0.796
## [2,] 1.000 1.000
```

```
## Valores iniciales
```

```
## [1] 0.070 0.105 0.565 0.019 0.224 1.000 0.796 1.000
```




```
## P(0,0,0)=0.571788
## P(0,0,1)=0.714106
## P(0,1,0)=0.400000
## P(0,1,1)=0.600000
## P(1,0,0)=0.800000
## P(1,0,1)=1.000000
## P(1,1,0)=0.666667
## P(1,1,1)=1.000000
```

Aplicando Value iteration al "play of the dice game pig"

Ahora que ya tenemos el algoritmo implementando y una mejor comprension de como funciona vamos a aplicarlo a nuestro objetivo principal el "play of the dice game pig"

Discusión

Conclusiones

Bibliografía o Referencias

[1] http://cs.gettysburg.edu/~tneller/papers/umap10.pdf Practical Play of the Dice Game Pig por Todd W. Neller

[2] http://www.durangobill.com/Pig.html Pig (Pig-out)

[3] https://people.eecs.berkeley.edu/~pabbeel/cs287-fa12/slides/mdps-exact-methods.pdf

[4] https://people.eecs.berkeley.edu/~pabbeel/cs287-fa12/slides/mdps-exact-methods.pdf

[5]https://en.wikipedia.org/wiki/Markov_decision_process