a cura di Francesco Romani

Il Gioco del Tris e l'apprendimento automatico

Vediamo un nuovo aspetto di "Enigmistica Computazionale". Cerchiamo di costruire un programma in grado di giocare e, contemporaneamente, di apprendere a giocare meglio. L'esempio scelto è il comunissimo Gioco del Tris che per la sua semplicità si presta bene ad applicazioni didattiche.

di Federico Curcio e Francesco Romani

Introduzione

Molti forse ricordano il film "Wargames", dove il computer WOPR (sì, proprio quello che, se lo chiami Joshua, ti consegna le chiavi dell'arsenale nucleare statunitense) viene convinto a desistere dallo sferrare un attacco di rappresaglia grazie a ciò che impara giocando a Tris. Al termine del film il computer impartisce la sua brava lezioncina di morale - se lo ha capito una macchina, come fanno gli umani a non capirlo? - osservando: "Strano gioco: l'unica mossa vincente è di non giocarlo". Il Tris ha infatti la caratteristica di portare alla patta se entrambi i giocatori effettuano sempre la mossa migliore in risposta a quella dell'avversario; ciò accade anche con la guerra termonucleare globale, dove nessuno risulterebbe vincitore, con risultati però decisamente diversi sulla sopravvivenza dei contendenti.

Da parte nostra, molto più modestamente, ci accingiamo a insegnare a *Mathematica* come giocare a Tris imparando dai suoi stessi errori. Vogliamo indurre il computer a considerare solo le mosse più convenienti, scartando quelle dubbie o decisamente cattive. Il metodo che useremo è quello dei premi e delle punizioni.

Il Problema

Il gioco si svolge su di una scacchiera 3x3.

In[1]:=
```
Pmat=
 ListDensityPlot[Array[0&,{3,3}],
  FrameTicks -> None,
  ColorFunction->(White&)];
```

Vedi figura 1

Uno dei giocatori (il bianco) mette un O sulla scacchiera l'altro risponde con una X; vince chi mette tre segni uguali su una riga, una colonna oppure una diagonale. Se la scacchiera è piena e nessuno ha vinto il gioco è pari.

La prima mossa ha 9 possibilità, la seconda 8 e così via. L'albero del gioco ha quindi al più 9! = 362880 rami.

In[2]:=
```
Sum[9!/i!, {i,9}]
```

Out[2]=
```
623530
```

In realtà i nodi sono molti meno, perché quando qualcuno vince non si va più avanti.

Se consideriamo invece le configurazioni possibili della scacchiera in modo indipendente da come ci si è arrivati, si vede che la dimensione del gioco è molto inferiore. Vi sono 3^9= 19.683 possibili scritture dei 3 simboli (blank, O, X), inoltre le configurazioni legali contengono o un numero pari di O e di X oppure un O in più. Giocando un poco con i coefficienti multinomiali si vede che le configurazioni con un numero pari di O e X sono:

In[2]:=
```
eq=Sum[9!/(x! x! (9-2x)!),{x,0,4}]
```

Out[2]=
```
3139
```

e quelle con un O in più:

In[2]:=
```
go=Sum[9!/(x! (x+1)! (8-2x)!),{x,0,4}]
```

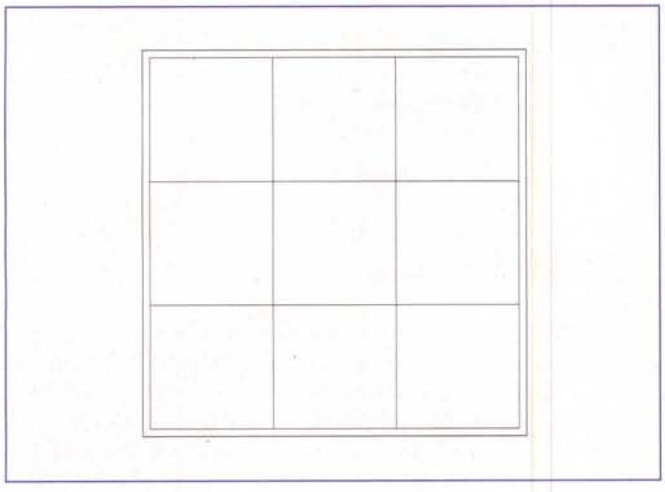

Figura 1

Out[2]=
2907

In totale abbiamo circa 6.000 configurazioni che costituiscono un grafo connesso. Una partita è identificabile con un cammino all'interno del grafo stesso.

Il gioco casuale

Mettiamo su una coppia di giocatori di Tris. Dapprima decidiamo la rappresentazione di una configurazione (un vettore **A** di nove valori scelti tra **0** (zero), **X** e **O**).

La funzione **verify** controlla se la partita è terminata e restituisce uno dei valori **-1** (ha vinto **X**), **0** (pari), **1** (ha vinto **O**), **None** (si può andare avanti).

In[1]:=
```
verify:= Module[{B,v,full},
   B=A/.{X->-1,O->1};
   full= Plus@@Abs[B]==9;
   B=Partition[B,3];
   v=Flatten[{
      Plus@@B,
      Plus@@Transpose[B],
      B[[1,1]]+B[[2,2]]+B[[3,3]],
      B[[1,3]]+B[[2,2]]+B[[3,1]]}];
   Which[Max[v]==3,                1,
         Max[-v]==3,        -1,
         full,               0,
         True,            None]]
```

La funzione **lc** rende la primitiva grafica adatta a scrivere **X** oppure **O**; **mess** è il messaggio associato ad uno dei possibili risultati di **verify**.

In[2]:=
```
lc[___]:= 0;
lc[i_,j_]:=
  Text[FontForm[
    ToString[A[[3(i-1)+j]]],
    {"Courier",24}],
   {j-0.4,3-i+0.4},{0,0}]/;
  LetterQ[ToString[A[[3(i-1)+j]]]];
```

In[3]:=
```
mess[None] = " ";
mess[1]   = "Vince O ";
mess[0]   = "Pari ";
mess[-1] = "Vince X ";
```

Infine **showtab** stampa la scacchiera.

In[4]:=
```
showtab:=(
 Block[{$DisplayFunction=Identity},
  aaa=Select[Flatten[Array[
    lc,{3,3}]],#=!=0&]];
 Show[Pmat,Graphics[{aaa}],
    AspectRatio->1,
    PlotLabel->mess[RES]];)
```

Vediamo un esempio di stampa di configurazione:

In[5]:=
```
A=Array[0&,{9}];
A[[1]]=X;
A[[2]]=O;
A[[5]]=O;
A[[7]]=X;
A[[8]]=O;
RES=verify
showtab
```

Vedi figura 2

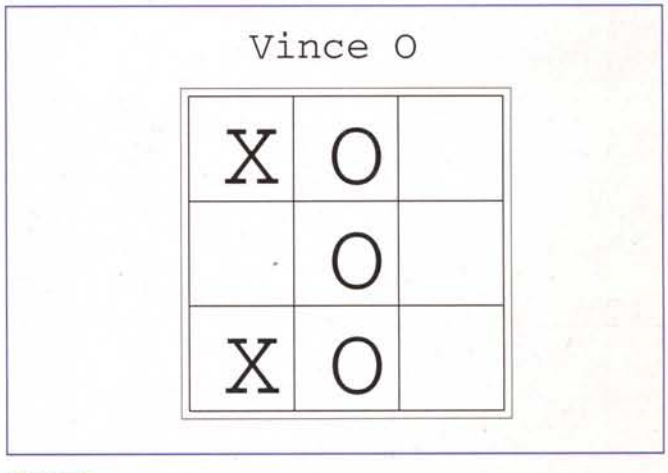

Figura 2

La funzione **init** inizializza il gioco con una scacchiera pulita e dando il valore opportuno alle variabili **nply** (il numero delle mezze mosse), **RES**, **LLX** e **LLO**.

In[5]:=
```
init:=(
  A=Array[0&,{9}];
  nply=0;
  RES=None;
  LLX={};
  LLO={};)
```

La funzione **move** esegue la i-esima delle possibili mosse lecite rendendo la nuova scacchiera. **MoveX** e **MoveO** sono due giocatori random (eseguono una a caso delle mosse lecite) e GiocoShow mostra passo passo una partita

In[6]:=
```
move[i_,A_,S_]:=(
   B=A;
   j=i;
   k=0;
   While[j>0,If[A[[++k]]==0,--j]];
   B[[k]]=S;
   B)
```

In[7]:=
```
MoveX:=(
  rp=Random[Integer,{1,9-nply}];
  A=move[rp,A,X];
  nply++;
  RES=verify;)
```

In[8]:=
```
MoveO:=(
  rp=Random[Integer,{1,9-nply}];
  A=move[rp,A,O];
  nply++;
  RES=verify;)
```

In[6]:=
```
GiocoShow:= (
  init;
  While[RES==None,
       MoveO;
       If[RES==None,MoveX];
       showtab]);
```

Vedi figura 3

Apprendimento

Per far sì che vengano scelte sempre le mosse migliori, associamo un peso (inizialmente nullo) a tutte le configurazioni di gioco e, per le sole configurazioni toccate durante una partita, il peso viene aumentato (premio) se essa si è conclusa con la vittoria, diminuito in caso di sconfitta (punizione), lasciato invariato se si è verificata una patta. E' chiaro che con l'aumentare delle partite giocate, il programma aumenta la sua "esperienza" e diventa sempre più improbabile una sua sconfitta.

Il programma "furbo" memorizza le mosse giocate durante una partita e poi modifica il valore di quelle configurazioni una volta visto il risultato finale. Quando si tratta di giocare, la mossa non viene scelta a caso ma solo la prima di quelle a più alto punteggio.

Vediamo la strategia per il giocatore O.

In[1]:=
```
initlearn:=(
  Clear[valueO];
  valueO[_]=0;);
```

In[2]:=
```
learnO := Scan[valueO[#]+=RES&,LLO];
```

In[3]:=
```
MoveOL:=(
  LM=move[#,A,O]&/@Range[9-nply];
  v=valueO/@LM;
  p=Position[v,Max[v]];
  A=move[p[[1,1]],A,O];
```

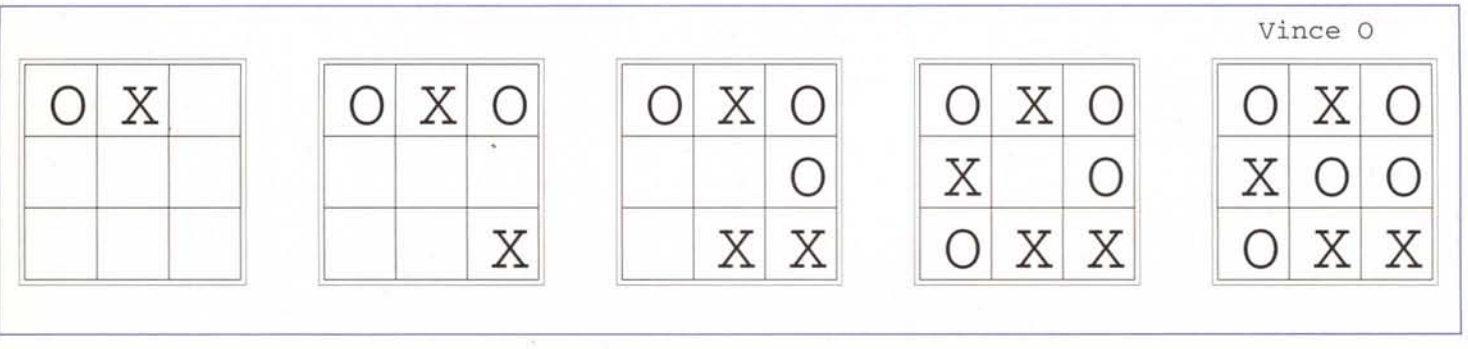

Figura 3

```
  AppendTo[LLO,A];
  nply++;
  RES=verify;)
```

In[4]:=

```
GiocoLO:= (
   init;
   While[RES==None,
        MoveOL;
        If[RES==None,MoveX]];
   learnO;);
```

Si noti la concisione (forse eccessiva ai fini della chiarezza) del codice che aggiorna la tabella dei valori delle posizioni.

Il programma che implementa la stategia "furba" per X e quella "scema" per O è simmetrico; è facile anche implementare un gioco in cui entrambi i giocatori sono dotati di capacità di apprendimento.

Risultati

Sono state giocate fino a 200.000 partite con le seguenti percentuali dei risultati.

Entrambi i giocatori giocano a caso

```
Vince O 59. %, pari 13. %, su 100
Vince O 57.3%, pari 12.9%, su 1000
Vince O 58.3%, pari 13.3%, su 2000
Vince O 58.5%, pari 12.3%, su 5000
Vince O 58.9%, pari 12.7%, su 10000
Vince O 58.5%, pari 12.6%, su 20000
Vince O 58.4%, pari 12.6%, su 50000
Vince O 58.4%, pari 12.6%, su 100000
Vince O 58.6%, pari 12.6%, su 200000
```

Negli esempi successivi la strategia di apprendimento funziona in modo graduale ottenendo i migliori risultati col crescere dell'esperienza.

O è furbo, X gioca a caso

```
Vince O 80. %, pari 7.  %, su 100
Vince O 88.5%, pari 4.  %, su 1000
Vince O 89.3%, pari 4.15%, su 2000
Vince O 91.1%, pari 3.88%, su 5000
Vince O 92.2%, pari 3.66%, su 10000
Vince O 92.5%, pari 3.66%, su 20000
Vince O 92.8%, pari 3.7 %, su 50000
Vince O 93. %, pari 3.65%, su 100000
Vince O 93. %, pari 3.69%, su 200000
```

X è furbo, O gioca a caso

```
Vince O 55. %, pari 5.  %, su 100
Vince O 27.8%, pari 12.8%, su 1000
Vince O 24.1%, pari 13.3%, su 2000
Vince O 19.4%, pari 14.2%, su 5000
Vince O 16.8%, pari 13.7%, su 10000
Vince O 15.7%, pari 13.7%, su 20000
Vince O 14.8%, pari 13.8%, su 50000
Vince O 14.2%, pari 13.7%, su 100000
Vince O 13.9%, pari 13.7%, su 200000
```

Sia O che X sono furbi

```
Vince O 10.   %, pari  84. %, su 100
Vince O  1.   %, pari  98.4%, su 1000
Vince O  0.5  %, pari  99.2%, su 2000
Vince O  0.2  %, pari  99.7%, su 5000
Vince O  0.1  %, pari  99.8%, su 10000
Vince O  0.05 %, pari  99.9%, su 20000
Vince O  0.02 %, pari 100. %, su 50000
Vince O  0.01 %, pari 100. %, su 100000
Vince O  0.005%, pari 100. %, su 200000
```

Conclusioni

È impressionante come in questo tipo di gioco la strategia di apprendimento sia efficace. Si noti che per gli umani questo si traduce nel fatto che il gioco (in questa forma) è "troppo semplice" e non c'è gusto a giocarlo se non per i bambini piccoli.

La semplicità del gioco ci ha permesso di rappresentare tutte le configurazioni. Per situazioni in cui la rappresentazione dell'intero grafo non è proponibile date le sue dimensioni (esempio classico: gli scacchi), di solito si effettua un tipo differente di esplorazione di una parte del grafo valutando - sul breve o medio periodo, a seconda del numero di mosse che si cerca di prevedere - la "bontà" di ogni singola mossa possibile.

Ci stiamo organizzando per presentare (sia con uno che due giocatori) le strategie di ricerca euristica su alberi che formano la base dei programmi "intelligenti" per i giochi più complessi.

MC

Bibliografia

G. Dossena, R. Rinaldi **A che gioco si gioca?**, Le guide de "L'Espresso", 1978 (pagg. 9-45).

M. Gardner **Enigmi e giochi matematici, Vol. 4**, Enciclopedie pratiche Sansoni, 1977 (pagg. 77-89).

P. Berloquin **Il centogiochi**, Le guide hobby Vallardi, 1984.