a cura di Francesco Romani

Cavalli e Topi

Riprendiamo questo mese l'interessantissimo articolo di Dani Ferrari apparso negli "Intelligiochi" di settembre 1998. Se dico la mia sull'argomento non è per fare il saccente ma solo per far vedere come con *Mathematica* si possano analizzare anche i problemi più difficili, magari in forma ridotta, tanto per chiarirsi le idee su come vanno affrontati in generale.

Il problema dei Cavalli

Riformuliamo il problema dei cavalli (MCmicrocomputer n. 187, pagina 150) usando una notazione leggermente diversa.

Dato un esperimento **X** con **k** possibili uscite $(x_1, x_2,\ldots,x_k)$ ognuna con probabilità $(p_1, p_2,\ldots,p_k)$ abbiamo **k** "cavalli di plastica" messi su una pista. Il cavallo **i** avanza di un passo se esce x_i. Vince il primo cavallo che fa **N** passi. Ogni cavallo ha allora una probabilità di vincere v_i che dipende dalla distribuzione $(p_1, p_2,\ldots,p_k)$.

Il problema diretto è: come dipendono le **v** dalle **p**? Il problema posto nell'articolo di Dani è quello inverso: come dipendono le **p** dalle **v** (nel caso speciale **k=6**, **n=49**)?

La prima considerazione da fare è che l'esperimento preso in esame termina dopo un numero finito di passi (al più **(N-1)(k-1)+N**) e le catene di Markov non c'entrano quindi nulla. Il modello adatto per studiare il problema in modo esaustivo è quello di un albero **k-ario** (ovvero dove ogni nodo non foglia ha esattamente **k** figli) di dimensione finita.

Vediamo un esempio, disegnato con *Mathematica*, per il caso banale **N=2**, **k=2**, ($x_1 = 0$ con probabilità $p_1 = p$ e $x_2 = 1$ con probabilità $p_2 = q = 1-p$).

Vedi Figura 1

Queste sono tutte le possibilità: se si arriva ad un nodo verde ha vinto il primo cavallo, se si arriva ad un nodo giallo ha vinto il secondo. La probabilità di raggiungere un nodo dell'albero è $p^z q^u$ dove **z** è il numero di zeri e **u** è il numero di uni. La probabilità di avere una generica stringa di **z** zeri e **k** uni è:

$$p(z,u) = \binom{u+z}{z} p^z q^u$$

Poiché l'ordine in cui arrivano gli zeri e gli uni (prima dell'ultima volta) non conta, la probabilità che vinca il cavallo 2 è quindi la seguente:

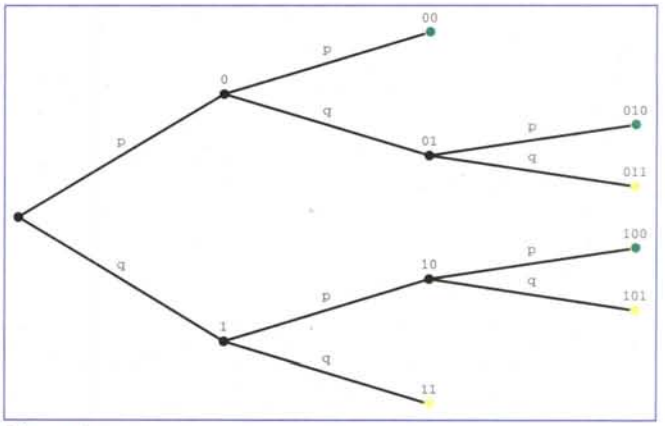

Figura 1

$$v_2 = q\left[\binom{2}{0}q + \binom{2}{1}pq\right] = q^2[1+2p]$$

Rimaniamo sempre con due cavalli ma aumentiamo **N**. La formula di v_2 diviene:

$$v_2 = q\sum_{i=0}^{N-1}\binom{N-1+i}{i}p^i q^{N-1}$$

Vediamo il grafico di v_2 al variare di **q** tra **0** e **1** per **N=2**, in rosso, e per **N=15**, in blu, (**p** vale **1-q** e varia di conseguenza).

Vedi Figura 2

Si vede che la probabilità di vincere per il secondo cavallo è nulla se esce sempre **0** e **1** se esce sempre **1**. Per i valori intermedi si ha una curva a gradino che diviene sempre più ripida al crescere di **N**. È facile inferire che per **N** molto grande la curva diviene uno scalino.

Attenzione: se provate a fare questo grafico per **N** elevato sorge il problema di disegnare un polinomio di grado alto (circa **2N**) il cui valore, per **q=1**, è esattamente 1. Gli errori di calcolo possono

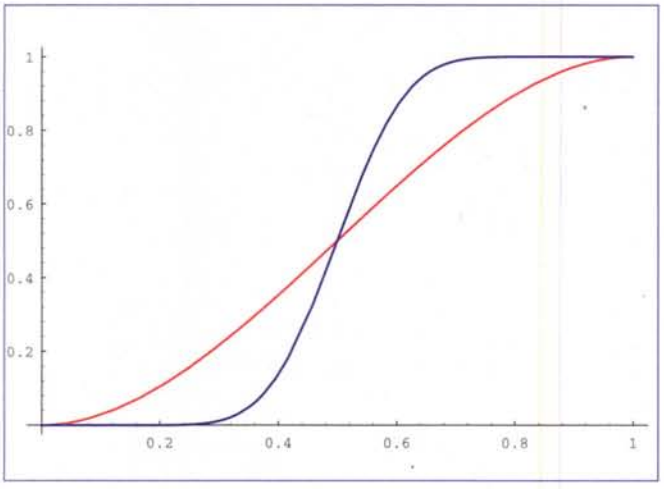

Figura 2

essere molto elevati e si può ottenere un grafico molto strampalato (provare per credere).

In generale per **k>2** si ha la relazione trovata da Ferrari nell'articolo citato:

$$v_k = p_k^N \sum_{n_1+n_2+\ldots+n_{k-1}<N} \frac{(n_1+n_2+\ldots+n_{k-1})!}{n_1!\,n_2!\ldots n_{k-1}!(k-1)!}\, p_1^{n_1} p_2^{n_2} \ldots p_{k-1}^{n_{k-1}}$$

Questa relazione è il legame diretto (ovvero la formula per trovare le **v** a partire dalle **p**), si tratta di un sistema di **k** polinomi nelle variabili $p_1, p_2, \ldots, p_k$ con termini di grado compreso tra **N** e **2N-1**.

Risolvere un tale sistema in forma chiusa è in generale impossibile, esistono complicati algoritmi di algebra computazionale che permettono di ridurre il sistema ad una equazione polinomiale, di grado molto elevato, in una sola variabile che comunque andrebbe risolta per via numerica. L'approccio più ragionevole è quello numerico o attraverso una simulazione o attraverso un metodo iterativo (entrambe le soluzioni sono state implementate da Ferrari). Vediamo con *Mathematica* cosa si può fare con **k=3** cavalli e **N=2**. Cominciamo col calcolare v_3.

In[1]:=
```
n=2;
v3 = p3^n Sum[Sum[
   (n1+n2+(n-1))!/(n1! n2! (n-1)!) p1^n1 p2^n2,
      {n1,0,n-1}],{n2,0,n-1}]
```

Out[1]=
```
(1 + 2 p1 + 2 p2 + 6 p1 p2) p3^2
```

Ora calcoliamo v_1 e v_2 per simmetria e verifichiamo che se (p_1, p_2, p_3) è un vettore di probabilità lo è anche (v_1, v_2, v_3). Eliminiamo poi p_3 ottenendo l'espressione esplicita di v_1, v_2 in funzione di p_1, p_2.

In[2]:=
```
v2 = v3/.{p1->p4, p2->p1, p3->p2}/. p4->p3;
v1 = v2/.{p1->p4, p2->p1, p3->p2}/.p4->p3;
v1+v2+v3/.{p3->1-p1-p2}//Simplify
```

Out[2]=
```
1
```

In[3]:=
```
v1 = v1/.{p3->1-p1-p2}//Expand
v2 = v2/.{p3->1-p1-p2}//Expand
```

Out[3]=
```
3 p1^2 - 2 p1^3 + 6 p1^2 p2 - 6 p1^3 p2 - 6 p1^2 p2^2
3 p2^2  + 6 p1 p2^2  - 6 p1^2  p2^2  - 2 p2^3  - 6 p1
p2^3
```

Risolviamo il sistema con **FindRoot** partendo da **p1=1/2**, **p2=1/3**.

In[4]:=
```
FindRoot[{v1-0.5==0,                    v2-1/3==0},
{p1,1/2},{p2,1/3}]
```

Out[4]=
```
{p1 -> 0.441699, p2 -> 0.338788}
```

Vediamo quanto si può andare avanti con **N** e come variano le probabilità risultanti. Si nota come per **N** grande piccole perturbazioni nelle **p** generano lo sbilanciamento desiderato nelle **v**.

N	p1	p2	p3	secondi
1	0.5	0.333333	0.166667	0.01
2	0.441699	0.338788	0.219513	0.02
5	0.397394	0.338989	0.263617	0.17
10	0.377313	0.337986	0.284701	2.02
15	0.368815	0.337354	0.293831	10.77
20	0.363855	0.336928	0.299216	45.12
25	0.361...	0.335...	0.303...	127.23

Dopodiché sorgono problemi di precisione che renderebbero necessario un trattamento più accurato.

Il problema del Topo

Anche questo problema è stato posto da D. Ferrari nel solito articolo: un topolino è posto in una scatola cubica divisa in **9x9x9** stanze cubiche. Il formaggio si trova in una stanza particolare ed il topo lo cerca muovendosi a caso, ma senza ripassare dalle stanze già toccate. Se non ha mosse possibili rinuncia. Quante sono le probabilità di successo, e quante le mosse in media per arrivarci?

Concordo con Ferrari che il problema originario sia intrattabile, se non con una estensiva simulazione. Vediamo invece come risolvere con Mathematica lo stesso problema, ma di dimensioni molto ridotte. Innanzitutto lavoriamo sul piano con un reticolo a maglia quadrata con **nxn** nodi. il topo si trova nel nodo **(1,1)** e deve trovare il formaggio nel nodo **(k,k)**.

L'idea consiste nel generare tutti i possibili percorsi e calcolarne le probabilità. Anche in questo caso il modello adatto è un albero di dimensioni finite in cui ogni nodo ha al più **3** figli (le **3** direzioni possibili, eliminando quella da cui si proviene). Un cammino lungo n può essere rappresentato come un numero di **n** cifre in base **3**. Nel nostro caso il cammino è lungo al più **n^2** passi e quindi il nostro albero ha al più **3^(n^2)** foglie (per esempio se **n=3** vi sono al più **19.683** cammini, se **n=5** vi sono al più **847.288.609.443** cammini).

In pratica i cammini sono molti di meno, perché ci si ferma se si arriva al formaggio oppure in un vicolo cieco.

Rappresentiamo un cammino come una lista di coppie (i nodi toccati), preceduta dalla probabilità corrispondente. Per esempio il cammino di **Figura 3** ha probabilità **0.125** e si rappresenta come:

In[1]:=
```
c={0.125, {1,1}, {2,1}, {3,1},
   {3,2}, {3,3}, {2,3}, {2,2}};
```

Vediamo come generare automaticamente tutti i cammini possibili e le relative probabilità.

I predicati **in** e **ok** dicono, rispettivamente, se una stanza è dentro la scatola o se contiene il formaggio, **norip** dice se un cammino è privo di ripetizioni.

In[4]:=
```
in[x_]     := 1<=x<=n;
ok[_]      := False;
ok[{k,k}] := True;
norip[x_] := Union[x]==Sort[x];
```

La funzione **vic** calcola i vicini ammissibili e **succ** genera tutti i cammini ottenibili facendo un passo in più a partire da un cammino dato.

In[5]:=
```
vic[{x_,y_}] := Select[
    {{x+1,y},{x-1,y},{x,y+1},{x,y-1}},
    in[#[[1]]]&&in[#[[2]]]&]
succ[x_List] :=
 Select[Append[x,#]&/@vic[Last[x]],norip]
```

Teniamo tre liste globali: **aa** che contiene tutti i cammini ancora vivi, **gg** quelli che hanno portato al formaggio e **ff** quelli finiti in un vicolo cieco. La funzione **elab** applicata ad un cammino **x** calcola tutte le possibili stanze in cui si può andare, aggiorna le liste **ff** e **gg** e rende la lista dei cammini vivi non terminali generati dal cammino **x**.

In[7]:=
```
elab[x_]:= If[(ss = succ[x]) == {}, AppendTo[ff,x],
    q = 1./Length[ss];
    ss=Join[{q First[#]},Rest[#]]&/@ss;
    gg1=Select[ss,ok[Last[#]]&];
    gg=Join[gg,gg1];
    bb=Join[bb,Complement[ss,gg1]]];
```

Il programma principale inizializza **aa** con il cammino iniziale (il topo sta nell'angolo con probabilità 1), inizializza **ff** e **gg** con la lista vuota e va avanti a generare cammini finché possibile. Alla fine calcola le probabilità e la media dei tentativi.

In[8]:=
```
go:= (
  aa={{1., {1,1}};
  ff = gg ={};
  While[Length[aa]>0,
    bb={};
    Scan[elab,aa];
    aa=bb];
  Print["num. cammini no    ",Length[ff]];
  Print["num. cammini ok    ",Length[gg]];
  Print["prob. fallimento   ",Plus@@(First/@ff)];
  Print["prob.  successo    ",Plus@@(First/@gg)];
  Print["media. tentativi   ",media[gg]];
  )
```

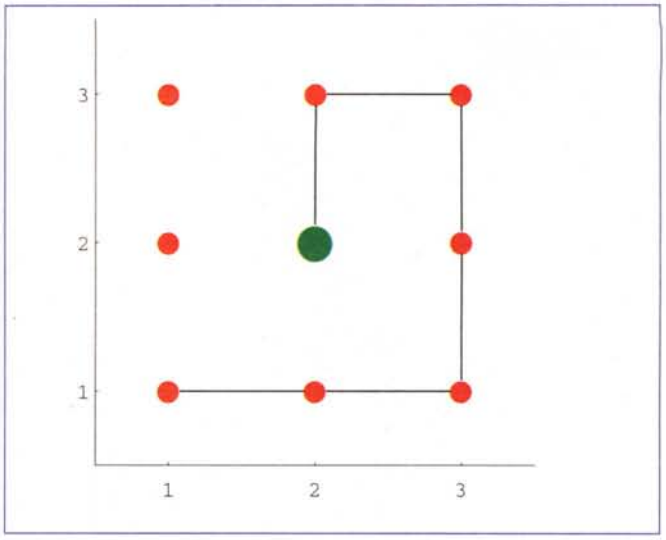

Figura 3

Vediamo il caso **n=3**, **k=2**. Il numero totale di cammini, tutti buoni, è 8 invece che 19.683!

In[9]:=
```
n=3;k=2;
go
```

Out[8]=
```
num. cammini no   0
num. cammini ok   8
prob. fallimento  0
prob.  successo   1.
media. tentativi  6.
```

Il caso **n=5**, **k=3** è ancora trattabile. Il numero totale di cammini è **5048 + 4846 = 9894** invece che **847.288.609.443**.

In[9]:=
```
n=5; k=3;
go
```

Out[8]=
```
num. cammini no   5048
num. cammini ok   4846
prob. fallimento  0.199831
prob.  successo   0.800169
media. tentativi  18.3578
```

Invece il caso **n=7** è al di fuori delle possibilità della mia macchina.

P.S. È possibile dimezzare il tempo di esecuzione ottenendo gli stessi risultati: se si parte con **aa={1.,{1,1},{1,2}}** si sfrutta la simmetria del problema generando solo la metà dei cammini.

Concludo restando in attesa di nuovi Intelligiochi di Corrado & Dani & C., così mi diverto un po' anch'io. MS

Bibliografia

Dani Ferrari, **Corri corri Topolino...**, MCmicrocomputer, n.187, settembre 1998, pp.150-153.

ISDN world

ZyXEL

Prestige Series
ISDN Routers

Navigate alla grande ...

Soluzioni complete ed a basso costo per accesso **internet e networking via rete ISDN**

- Connessione di tutta la LAN ad Internet attraverso un unico indirizzo IP
- Ottimizzazione della gestione della banda di trasmissione (Bandwitdh on demand, bundling dei canali)
- Supporto delle funzioni di sicurezza (Firewall, proxy-server, call-back)
- Software di configurazione basato su interfaccia WEB
- Funzioni aggiornabili via software

La più completa gamma di prodotti per la connessione Internet/Intranet, a Vs disposizione dal distributore italiano specializzato in **ISDN.**

CoFax®
TELEMATICA

DA 10 ANNI AL SERVIZIO DELLA COMUNICAZIONE VELOCE

Per maggiori informazioni - **http://www.cofax.it**
Roma - 00151 V.le dei Colli Portuensi, 110/a
Tel. 06/58201362 r.a. **Fax 06/58201550**
Milano - 20129 C.so Buenos Aires, 37
Tel. 02/29526100 r.a. **Fax 02/29520884**

Numero Verde **167-865108**

ISO 9002

Deutscher Akkreditierungs Rat DAR TGA-ZQ-002/92-00 Cert. N°48135

ISDN world by CoFax Telematica ®

NET working by CoFax Telematica ®