

# Programmazione logica e Mastermind

Christian Barbato, Dario Busetto

4 febbraio 2002

# 1 Introduzione

Nel gioco del MASTERMIND un giocatore costruisce segretamente una sequenza di  $k$  colori, con ripetizione ammessa, scelti da un insieme di  $n$  colori. Il secondo giocatore cerca di indovinare il codice. Ad ogni tentativo il primo giocatore fornisce al secondo due indicazioni:

- il numero di colori che si trovano nella corretta posizione rispetto al codice segreto;
- il numero di colori che si trovano nel codice segreto ma non sono nella posizione corretta.

Nella versione disponibile in commercio  $k = 4$  e  $n = 6$ , ma esistono degli studi che si riferiscono ad una versione *potenziata* in cui si aumentano sia il numero di colori disponibili, sia la lunghezza del codice. In questa relazione si farà riferimento solo alla versione base.

Da tener presente anche il fatto che il gioco è stato convertito utilizzando una sequenza di numeri dall'1 al 6 al posto dei colori.

## 2 Gli algoritmi

Sono stati scritti molti articoli riguardanti algoritmi per la soluzione del mastermind. Tutti comunque si possono ricondurre a due categorie principali:

- *stepwise optimal*, ossia i cosiddetti algoritmi greedy. Ogni tentativo che viene fatto è ottimo nel senso che punta ad essere la possibile soluzione. Questi algoritmi non hanno un limite massimo di prove per cui si può dire di arrivare certamente alla soluzione, ma danno la sicurezza di arrivarci.
- *strategically optimal*, in cui si giocano volutamente combinazioni di cui si conosce in precedenza l'inesattezza per avere il maggior numero di informazioni possibili sulla soluzione. Questo tipo di algoritmi generalmente arrivano alla soluzione entro un numero massimo di tentativi ma quasi sempre sono di più di quelli richiesti dagli algoritmi dell'altro tipo.

### 2.1 Algoritmo classico

Si tratta di un algoritmo appartenente alla categoria *stepwise optimal* con una strategia di tipo *exhaustive search*, ossia che non si limita a delle scelte casuali con qualche euristica di supporto.

Viene dapprima generato tutto lo spazio delle 1296 possibili soluzioni ( $6^4$ ). Da questo spazio viene scelto casualmente un elemento rappresentante il primo tentativo. Se é la soluzione esatta (!) l'algoritmo termina altrimenti si filtra lo spazio delle soluzioni in modo da salvare solamente le combinazioni che sono coerenti con il risultato fornito dal tentativo. In pratica sapendo che l'ultima combinazione giocata ha dato come risposta  $b$  bianchi e  $n$  neri, si toglieranno dallo spazio tutte le combinazioni che danno un risultato diverso da  $b, n$  non rispetto alla soluzione ma rispetto all'ultimo tentativo. A questo punto si sceglierá un altro elemento dallo spazio rimasto e si ripeterá l'algoritmo.

Ad ogni passo lo spazio delle soluzioni viene ridotto e la soluzione viene generalmente trovata con un massimo di 7 tentativi.

## 2.2 Algoritmo genetico

Oltre agli approci sopra descritti ultimamente si sta cercando di creare degli algoritmi genetici per la soluzione del mastermind.

Dopo aver creato una popolazione casuale di sequenze ne viene scelta una casualmente e la si propone come primo tentativo ottenendo un risultato  $b, n$ . Di seguito viene filtrata la popolazione con lo stesso criterio utilizzato per l'algoritmo classico, mantenendo cioè solamente le sequenze coerenti con l'ultimo risultato ottenuto. Vengono quindi applicati gli operatori genetici sulla rimanente popolazione:

- *mutation*: ogni elemento della sequenza in esame viene cambiato con probabilità  $m$  prendendo il valore successivo (ad esempio 1 va in 2, 6 in 1);
- *crossover*: l'incrocio tra due sequenze con probabilità  $c$ . Ad esempio prese le sequenze 1,1,2,3 e 4,5,6,1 si ottiene la sequenza 1,1,6,1;
- *transpose*: si effettua una permutazione della sequenza con probabilità  $t$ ;

ottenendo una nuova popolazione su cui, previa rimozione degli elementi duplicati, verrà di nuovo applicato l'algoritmo. Qualora dopo il filtraggio della lista la popolazione si esaurisse ne verrà creata una nuova.

Questo algoritmo non fornisce un limite massimo di tentativi entro cui verrà trovata la soluzione e nemmeno ci assicura di trovare la combinazione in un tempo finito. D'altra parte ha la caratteristica di essere molto flessibile e facilmente applicabile anche alle versioni del mastermind potenziate senza perdere efficienza e senza richiedere molte risorse. Si noti ad esempio che per

quanto riguarda il mastermind classico il numero di possibili combinazioni è di 1296 consentendone la totale rappresentazione in memoria, mentre per quanto riguarda una versione potenziata in cui la sequenza è lunga 8 e i colori sono 10 ci vorrebbe un terabyte di memoria per rappresentare tutte le possibili combinazioni; se invece si utilizza un algoritmo genetico viene comunque richiesta solamente la memorizzazione della popolazione che è di gran lunga inferiore dello spazio delle soluzioni.

### 3 Il programma

Per lo sviluppo degli algoritmi è stato utilizzato un linguaggio logico. L'interprete utilizzato è stato principalmente SWIProlog ma è stato utilizzato anche Sicstus.

Vengono di seguito riportati degli stralci del programma ritenuti particolarmente importanti e significativi.

#### 3.1 Bianchi Neri

```
% Uso :      neri(Sol,Inp,N)
% Risultato : Restituisce in N il numero di neri dell'input Inp rispetto
%             alla soluzione Sol

neri([A|[]],[A|[]],1):-!.

neri([_|[]],[_|[]],0).

neri([A|B],[A|C],M):-!,neri(B,C,N),M is N+1.

neri([_|B],[_|C],M):-neri(B,C,M).

% Uso :      bianchi(Sol,Inp,B,N)
% Risultato : Restituisce il numero di bianchi dell'input Inp rispetto alla
%             soluzione Sol sapendo che il numero di neri calcolati e' N

bianchi(Sol,Inp,B,N) :-
    bianchi(Sol,Inp,Inp,C,N),
    B is C - N.

bianchi(_,_,[],0,_).
```

```
bianchi(Sol,Inp,[I1|I1s],B,N):-
    conta(Sol,I1,H),
    conta([I1|I1s],I1,K),
    min(K,H,X),
    togli(I1,[I1|I1s],I2s),
    bianchi(Sol,Inp,I2s,Z,N),
    B is (X + Z).
```

### 3.2 Filtralista

```
% Uso :          filtralista(SS,B,N,Inp,_,Ris)
% Risultato : Restituisce in Ris il risultato del filtraggio dello spazio delle
%              soluzioni SS tramite l'input Inp, che con la soluzione iniziale
%              ha un valore di bianchi e neri di B ed N rispettivamente.
%              Nota : SS non deve contenere Inp (nel codice viene tolto prima)
```

```
filtralista([],_,_,_,Nuovalista,Nuovalista).
```

```
filtralista([H|T],B,N,C,Nuovalista,X):-
    bianchineri(C,H,BB,MN),
    NN==N,
    BB==B,
    append(Nuovalista,[H],Newlist),
    !,filtralista(T,B,N,C,Newlist,X).
```

```
filtralista([_|T],B,N,C,Nuovalista,X):-filtralista(T,B,N,C,Nuovalista,X).
```

### 3.3 Risolvi

```
% Uso :          risolvi(SS,Partenza,T,Sol)
% Risultato : Risolve in maniera non interattiva il gioco del mastermind
%              tenendo conto del numero T di tentativi e mettendo
%              la soluzione in Sol. La combinazione utilizzata per iniziare
%              la scrematura dei risultati e' Partenza
```

```
risolvi(_,S,T,S):-write('Ho trovato la soluzione : '),write(S),
    write(' in '),write(T),
    write(' tentativi'),nl.
```

```
risolvi(X, Soluzione, T, E):-
    NT is T + 1,
```

```

write('Combinazione selezionata: '),
write(E), listlength(X,L),
write(' Soluzioni rimaste: '),write(L), nl,
bianchineri(Soluzione, E, B,N),
% write('Neri      : '),write(N),nl,
% write('Bianchi  : '),write(B),nl,
filtralista(X,B,N,E,_,NuovoSpazio),
listlength(NuovoSpazio,NL),
randomselect(NuovoSpazio,NE,NX,NL),
risolvi(NX, Soluzione, NT, NE).

```

### 3.4 Operatori genetici

```

% Uso :      permuta(L,Ln)
% Risultato : Ln e' una permutazione di L

```

```

permuta([], []).
permuta(Xs, [X|Ys]) :-
    append(X1s, [X|X2s], Xs),
    append(X1s, X2s, Zs),
    permuta(Zs, Ys).

```

```

% Uso :      transpose(L,Ln,P)
% Risultato : Ln e' una permutazione di L, ma diversa da L. P e' un valore
%             casuale compreso tra 1 e 100 che deve essere passato in ingresso

```

```

transpose(L,Ln,P) :-
    P>10,
    permuta(L,Ln),
    L\=Ln.

```

```

transpose(L,Ln,_) :- Ln = L.

```

```

% Uso :      crossover(L1,L2,Ln,P)
% Risultato : Ln e' il risultato dell'operazione di crossover tra L1 e L2. P e'
%             un valore casuale compreso tra 1 e 100 che deve essere passato
%             in ingresso

```

```

crossover(L1,L2,LN,P) :-
    P>40,

```

```

nth1(1,L1,A),
nth1(2,L1,B),
nth1(3,L2,C),
nth1(4,L2,D),
LN=[A,B,C,D].

crossover(L1,_,LN,_) :- LN=L1.

% Uso :      mutation(L,Ln)
% Risultato : Ln e' il risultato dell'operazione di mutazione su L

mutation([],[]).
mutation([T|C],[NT|NC]) :-
    random(1,100,P),
    altera(T,NT,P),
    mutation(C,NC).

% Uso :      altera(E,NE,P)
% Risultato : NE e' l'elemento E aumentato di 1 con una certa probabilita'.
%            Qualora E sia 6 NE diventa 1. P e' un valore casuale compreso
%            tra 1 e 100 che deve essere passato in ingresso.

altera(E,NE,P) :-
    P>95,
    X is E+1,
    shifta(X,NE).

altera(E,E,P) :- P =< 95.

shifta(7,1).
shifta(X,X).

```

## 4 Risultati

Dei molti algoritmi scritti per la soluzione del mastermind nessuno ha dimostrato di essere l'ottimo per cui con l'andare avanti delle ricerche il limite si abbassa. Il primo fu Knuth che con un algoritmo simile al nostro ottenne una media di 4.478 tentativi per arrivare alla soluzione. Irving e Neurwith lo portarono a 4.364 e Koyama e Lai a 4.340.

## 4.1 Caso classico

Il nostro algoritmo classico su di un totale di 300 prove ha avuto una media di 4.643 tentativi necessari per arrivare alla soluzione, variando da un minimo di 1 ad un massimo di 7 tentativi per concludere il gioco.

## 4.2 Caso genetico

L'algoritmo genetico da noi implementato è stato posto a una serie di prove per testarne il funzionamento e l'efficacia. L'algoritmo è sempre arrivato alla soluzione (tranne qualche volta in cui l'interprete prolog si bloccava), ma i risultati non sono molto performanti.

I risultati ottenuti sono riportati nella seguente tabella in cui sono riportati i valori della popolazione iniziale, della probabilità  $m$  dell'operatore mutation, della probabilità  $c$  dell'operatore di crossover, della probabilità  $t$  dell'operatore di transpose, il numero minimo di tentativi necessari per arrivare alla soluzione, quello massimo e alla media. Tutte le prove sono state ripetute 100 volte per condizione di partenza.

Popolazione	$m$	$c$	$t$	Minimo	Massimo	Media
1000	0.05	0.75	0.85	2	42	10.68
1000	0.05	0.60	0.90	2	80	16.36
1000	0.10	0.70	0.90	2	53	12.64
800	0.05	0.75	0.85	2	47	11.51
800	0.05	0.60	0.90	2	46	12.96
800	0.10	0.70	0.90	3	89	15.34
600	0.05	0.75	0.85	1	53	13.77
600	0.05	0.60	0.90	3	51	16.83
600	0.10	0.70	0.90	2	77	15.85
300	0.05	0.75	0.85	3	62	16.85
300	0.05	0.60	0.90	2	101	22
300	0.10	0.70	0.90	3	117	20.34

I risultati sembrano indicare che con l'aumentare della popolazione iniziale l'algoritmo migliori la sua efficienza. Per quanto riguarda i parametri sembra che  $m$  influisca negativamente se troppo alto; per gli altri due le prove effettuate non danno grosse indicazioni.

Da tener inoltre presente che le 100 prove effettuate per ogni condizione non sono un numero molto elevato tale da fornire delle statistiche molto precise.