Effetti dell'Inganno nella Cooperazione: Il Dilemma del Cacciatore

Giovanni Spaventa
Nicola Pacella

11 ottobre 2014

1 Introduzione

L'obiettivo di questo elaborato è lo studio del famoso Dilemma del Prigioniero, introducendo una terza possibile mossa per i giocatori: *Inganna*. Il Dilemma del Prigioniero non permette una modellizzazione dell'inganno, a meno di supporre che i prigionieri abbiano avuto la possibilità di accordarsi precedentemente sulla strategia da seguire, rendendo così il Dilemma un *gioco cooperativo* e quindi l'inganno privo di significato (infatti in un gioco cooperativo l'accordo tra i giocatori per la strategia è vincolante, d'altra parte non ci può essere accordo sull'inganno perché un giocatore non potrà mai accettare una strategia che lo veda vittima di un imbroglio). Si pensi invece ad una situazione in cui due individui debbano scegliere se cooperare o non cooperare in vista di una ricompensa, e che ci sia la possibilità di far credere al proprio compagno di star cooperando quando in realtà si mira a tutta la ricompensa, anche a quella conquistata da quest'ultimo.

2 Il Dilemma del Cacciatore

Si immagini una coppia di leoni in agguato, un gruppo di gazzelle ed un elefante. Si assuma che le gazzelle e l'elefante si siano appena accorti della presenza dei leoni e che quindi per quest'ultimi ci sia pochissimo tempo per decidere quale preda inseguire. L'elefante è un bel bottino ma un leone da solo non può abbatterlo e rischia di farsi male, mentre una gazzella è una preda molto più facile ma bisogna inseguirla ed è un costo non trascurabile. Ogni leone ha a disposizione tre mosse:

- *Cooperate* [C]: corrisponde a lanciarsi contro l'elefante
- *Go Alone* [A]: corrisponde ad inseguire una gazzella
- *Deceive* [D]: corrisponde ad una illusione di cooperazione: il leone si lancia contro l'elefante ma non lo spartisce se quest'ultimo viene abbattuto.

Possiamo assegnare un valore alle prede in base a quanto vantaggioso sia averle abbattute: una gazzella potrebbe valere 2, mentre un elefante potrebbe valere 8. Se entrambi cooperano, abbattono l'elefante e lo spartiscono, guadagnando

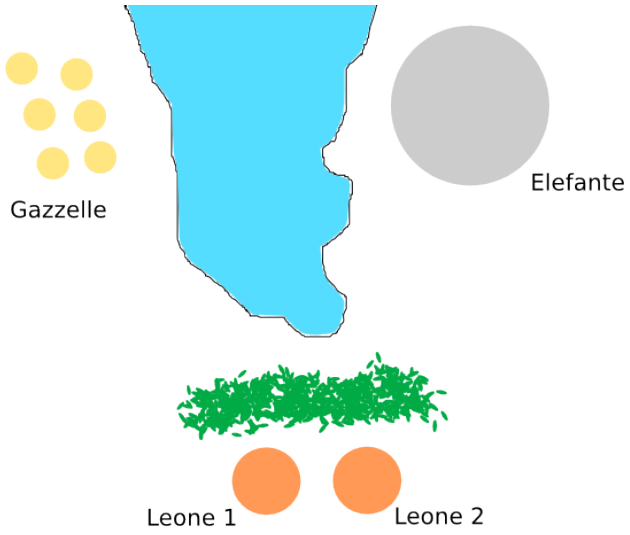

Figura 1: Rappresentazione grafica del problema.

4 punti a testa. Se un leone decide di andare da solo, guadagna 2 punti indipendentemente dalla scelta dell'altro, che però rimane senza niente se aveva cooperato o ingannato. Infine se un leone inganna, guadagna 7 degli 8 punti totali relativi all'elefante se l'altro aveva cooperato (lasciando a quest'ultimo soltanto 1 punto), 0 punti se l'altro aveva scelto di andare da solo o se aveva deciso di ingannare a sua volta (perché il guadagno derivato dall'aver abbattuto l'elefante è compensato dal costo necessario a cercare di conquistarne la maggior parte, ad esempio una lotta tra i due leoni) Abbiamo quindi la seguente *tabella dei pay-off*:

	C	A	D
C	4	0	1
A	2	2	2
D	7	0	0

Si vede subito che scegliere A corrisponde a 2 punti certi, che però sono molto pochi rispetto ai 7 che si potrebbero guadagnare con l'inganno, a loro volta troppo rischiosi rispetto ai 4 della cooperazione.

3 La Simulazione

Per riuscire a studiare il gioco, abbiamo scritto un programma in Python che simulasse un *Dilemma del Cacciatore* iterato, implementando diverse strategie ed analizzando il punteggio medio per strategia.
Il programma (*huntersdilemma.py*) è strutturato in modo da gestire N iterazioni del gioco (nel programma, N=30, modificabile all'interno di *battle()*) con due giocatori scelti ogni volta da un *pool* settato precedentemente, insieme alla distribuzione di strategie voluta.

Abbiamo modellizzato un giocatore come una *classe*, contenente il metodo *move()* che gestisce le mosse. La funzione *battle()* gestisce le iterazioni del gioco tra i due giocatori selezionati ed infine i risultati vengono salvati in tre files:

- *battles.dat*: contiene la cronologia delle mosse per le singole battaglie
- *partscores.dat*: contiene la cronologia dei punteggi per le singole battaglie;
- *totscores.dat*: contiene il punteggio totale medio di ogni strategia

3.1 Le Strategie

Le strategie implementate nel programma sono:

- *Cooperator*: coopera sempre;
- *Loner*: va sempre da solo;
- *Deceiver*: inganna sempre;
- *Random*: compie una mossa casuale;
- *Good Random*: compie una mossa casuale ma non inganna mai;
- *Tit For Tat*: al primo turno coopera, poi copia le mosse precedenti dell'altro;

con l'aggiunta di un'ultima semplice strategia inventata da noi:

- *Evil Tit For Tat*: al primo turno coopera. Poi, se la mossa precedente dell'altro è coopera allora inganna, altrimenti si comporta come *Tit For Tat*.

Il programma è strutturato in modo da poter aggiungere e/o rimuovere facilmente nuove strategie: basta infatti inserire una *label* nell'array *Vstrats*, settare il numero di giocatori in *poolDistr* ed infine definire la strategia all'interno del metodo *move()*.

4 L'Esperimento

Come primo esperimento con questo modello, vogliamo provare a capire come la presenza di un *Deceiver* cambi i punteggi in un pool uniforme di altre strategie che non ingannano. Successivamente vogliamo confrontare i punteggi ottenuti sostituendo il *Deceiver* con un *Evil Tit For Tat*. Tali punteggi verranno forniti nella forma *valor medio* $\pm$ *deviazione standard*, ottenuti dal set di dati proveniente da 10 iterazioni del programma ed arrotondati all'intero più vicino.

4.1 I Risultati

4.2 Pool di *good guys*

Inizializziamo un pool di giocatori in cui non siano presenti strategie che utilizzano la mossa *inganna*:

2 *Cooperator*, 2 *Loner*, 2 *Good Random*, 2 *Tit For Tat*.

I punteggi medi per strategia sono risultati essere:

Cooperator : 483 ± 11

Tit For Tat : 592 ± 9

Loner : 420 ± 0 [1]

Good Random : 419 ± 15

Come succede nel Dilemma del Prigioniero, *Tit For Tat* è la strategia che totalizza più punti. Si vede anche che *Loner* è praticamente alla pari con *Random* (che possiamo prendere come una sorta di strategia media di riferimento).

4.3 Pool di *good guys* con l'aggiunta di un *Deceiver*

vediamo ora cosa succede se nel pool del paragrafo precedene sostituiamo un *Good Random* con un *Deceiver*.
I punteggi medi per strategia sono risultati essere:

Cooperator : 451 ± 7

Tit For Tat : 538 ± 5

Loner : 420 ± 0

Good Random : 412 ± 14

Deceiver : 538 ± 15

Si noti come il *Deceiver* abbia totalizzato tanti punti quanti *Tit For Tat* (sebbene la distribuzione sia 3 volte più larga), nonostante sia una strategia fin troppo semplice per una mossa come *Deceive*, che paga soltanto quando l'altro sceglie *Cooperate*. Ci si potrebbe aspettare che un ingannatore più sofisticato raggiunga punteggi più grandi.

4.4 Pool di *good guys* con l'aggiunta di un *Evil Tit For Tat*

vediamo ora cosa succede se nel pool del paragrafo precedene sostituiamo un *Good Random* con un *Evil Tit For Tat*.
I punteggi medi per strategia sono risultati essere:

Cooperator : 453 ± 7

[1] *Loner* prende sempre 420, a causa del fatto che i suoi punteggi sono indipendenti dalla scelta dell'altro giocatore.

Tit For Tat : 541 ± 7

Loner : 420 ± 0

Good Random : 401 ± 18

Evil Tit For Tat : 634 ± 15

Evil Tit For Tat ha raggiunto un punteggio molto più alto degli altri, arrivando, in una particolare iterazione, a totalizzare ben 672 punti contro un *Tit For Tat* secondo classificato con 539 punti.

5 Conclusioni

Come potevamo aspettarci, la presenza dell'inganno modifica in modo non trascurabile la dinamica del gioco. Si è trovato, in particolare, che esso risulta una mossa molto efficiente nonostante paghi soltanto contro *Cooperate*. Confrontando il pool al paragrafo 4.2 con i successivi, si può notare che **tutte** le strategie hanno sofferto la presenza dell'ingannatore, con conseguente calo dei loro punteggi medi.
Probabilmente una così alta efficienza dell'inganno è dovuta alla piccola memoria delle strategie implementate: la massima memoria di una strategia in questo elaborato è di una mossa soltanto (di *Tit For Tat* e *Evil Tit For Tat*).
Un esperimento più sofisticato (e più realistico), potrebbe estendere le memorie delle strategie e modificarle in modo da far agire un individuo in base al numero di volte in cui un altro individuo ha provato ad ingannarlo, cosa che dovrebbe limitare di molto la cooperazione con ingannatori recidivi.
Per fare ciò bisognerebbe modificare il modo in cui viene gestito l'attributo *history* della classe *Hunter*, per farlo diventare un array di mosse invece che un carattere (e quindi una sola mossa). Successivamente si dovrebbero scrivere strategie che sfruttino le mosse precedenti degli altri giocatori.

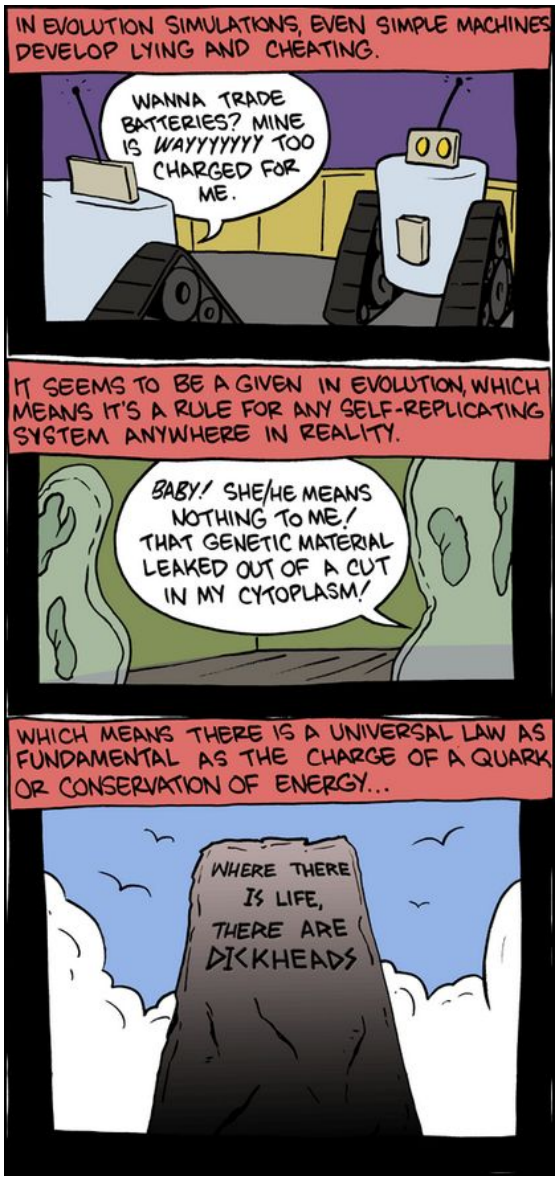

Figura 2: Da *http://www.smbc-comics.com/*.