Relazione Hanabi

Mihail Bida, Francesco Pandolfi

December 2020

1 Introduzione

1.1 Regolamento

- Il mazzo di Hanabi è formato da 50 carte suddivise in 5 colori (10 per ogni colore). Ogni colore è composto: 3 carte di valore 1, 2 di valore 2,3,4 e 1 di valore 5.
- In campo ci sono 8 gettoni informazione e 3 gettoni errore
- Ogni giocatore tiene le proprie carte rivolte verso gli altri (4 carte in 4 o 5 giocatori e 5 carte in 2 o 3 giocatori), quindi vede tutto tranne le proprie carte.
- Lo scopo del gioco è completare le scale di colore da 1 a 5. Ogni carta giocata vale 1 punto.
- Ad ogni turno ogni giocatore può fare una delle seguenti mosse:
  - indicare ad un giocatore tutte le carte di uno stesso colore o valore nella sua mano.
  - scartare una carta dalla propria mano.
  - giocare una carta.
- Giocare o scartare una carta fa pescare una carta dal mazzo. Finite le carte da pescare, il gioco termina quando è nuovamente il turno di chi ha pescato l'ultima carta.
- Scartare una carta aumenta i gettoni informazione di 1 (fino ad un massimo di 8).
- Giocare una carta sbagliata fa perdere un gettone errore. Finiti i gettoni errore la partita è conclusa con punteggio 0.

1.2 Stato dell'arte

Negli ultimi anni è stata prodotta un'ampia varietà di agenti. Come riporta il paper The Hanabi Challenge: A New Frontier for AI Research[1] hanabi ha destato l'interesse dei ricercatori in quanto gioco cooperativo, che quindi permette di studiare soluzioni di intelligenza artificiale in uno scenario molto diverso da quello offerto dai giochi competitivi. Tuttavia si è notato che gli agenti intelligenti sono surclassati da agenti che eseguono strategie codificate dal programmatore.

La strategia proposta dagli autori del paper è incentrata sulla comunicazione implicita, ossia lo stabilire un insieme di convenzioni che permettano di aggiungere informazioni quando si dà un suggerimento. Ciò è necessario in quanto il numero di suggerimenti che si possono dare in una partita è limitato.

2 Strategia adottata

Le strategie di gioco presentate durante il corso di Fondamenti di Intelligenza Artificiale M consistono in varianti di ricerca nello spazio degli stati.
Abbiamo poi visto come l'algoritmo min-max sia molto efficace nei giochi competitivi e come sia possibile attraverso pruning diminuire considerevolmente il numero di stati che è necessario esplorare.

Nostro malgrado Hanabi è un gioco cooperativo e questo ci preclude la possibilità di sfruttare sia min-max sia il pruning.
Lo spazio degli stati di una partita di Hanabi è molto vasto: si consideri la seguente tabella che mostra un'approssimazione del numero di stati da esplorare per conoscere completamente tutti i possibili stati che si possono raggiungere con una mossa a partire da uno stato dato.

Numero di giocatori	**Numero di stati**
2	$[10(\mathrm{m})(\mathrm{c})+10)^d$
3	$[10(\mathrm{m})(\mathrm{c})+20]^d$
4	$[8(\mathrm{m})(\mathrm{c})+12]^d$
5	$[8(\mathrm{m})(\mathrm{c})+16]^d$

Table 1: Tabella che mostra un'approsimazione del numero di stati da esplorare in funzione del numero di carte nel mazzo (m), il numero di carte che quella che si vuole giocare/scartare potrebbe essere (c) e del numero di turni di profondità di esplorazione (d)

Possiamo vedere che la dimensione dello spazio degli stati varia in modo esponenziale in funzione della profondità di esplorazione richiesta (d), come i

[1] https://arxiv.org/pdf/1902.00506.pdf

giochi tradizionali, stile scacchi.
Ma a differenza di questi giochi, il giocatore ha conoscenza parziale degli stati e inoltre è presente una componente randomica dovuta alla pesca delle carte. Di conseguenza il numero di stati cresce secondo una funzione polinomiale di due fattori: il numero di carte rimaste nel mazzo (m) e il numero di carte che quella che si vuole giocare/scartare potrebbe essere (c).

Finchè ci sono carte da pescare, gli stati esplorati hanno una natura probabilistica, nel senso che scelta una mossa (giocata o scartata) non abbiamo mai un singolo stato futuro ma una distribuzione di probabilità su un set di stati futuri. Ciò rende difficile la definizione di una funzione euristica per l'algoritmo di ricerca degli stati.

Le strategie seguite dai bot prodotti consistono nel valutare se fare un tipo di mossa in funzione di 3 parametri calcolati per le carte coinvolte dalla mossa:

- entropia: incertezza sul tipo di carta. Se la carta è completamente conosciuta la sua entropia è 0. Abbiamo notato empiricamente che una mano completamente sconosciuta ha un'entropia di circa 22 bit (dipende dal numero di carte in mano e dal numero di giocatori).
- giocabilità: probabilità che giocando una carta non si commetta errore
- inutilità: probabilità che scartando una carta non si comprometta il punteggio finale.

Giocare una carta sarà una buona mossa se la giocabilità di quella carta è alta, situazione analoga per lo scartare una carta con alta inutilità. Un buon consiglio invece cercherà di alzare la giocabilità delle carte del compagno. Il parametro di entropia viene per lo più usato per selezionare il miglior consiglio nei casi in cui più consigli fanno raggiungere ad una carta lo stesso valore di giocabilità o inutilità.

L'effetto di una mossa è modificare i parametri delle carte ma si verifica solo tra lo stato attuale e quello generato. Negli stati successivi la mossa non produrrà altre variazioni.
Per questo motivo riteniamo che un'esplorazione profonda non aggiunga informazioni rilevanti che aiutino a prendere una decisione sulla mossa da effettuare rispetto alla sola esplorazione degli stati più prossimi a quello attuale.

Giocabilità e inutilità di una data carta sono calcolate sommando per ogni tipo di carta giocabile/inutile le probabilità che la carta data sia proprio di quel tipo.
Sebbene la definizione di carta giocabile sia semplice (una carta è giocabile se la scala del suo colore ha valore precedente a quello della carta data), altrettanto non vale per la definizione di carta inutile. Una carta è inutile se è vera almeno una delle seguenti condizioni:

- La scala dello stesso colore ha valore maggiore o uguale della carta.
- Tra gli scarti sono presenti tutti gli esemplari di carte di stesso colore e uno qualsiasi dei valori inferiori.
- Almeno una copia della carta è ancora in gioco.

L'entropia invece è calcolata secondo la formula classica.

3 Bot prodotti

Abbiamo sviluppato due bot, Bot1 e Bot2 che seguono due strategie differenti ma con alcuni punti in comune. Il più importante di questi è che durante la valutazione di una mossa il bot si immedesima negli altri giocatori e valuta calcolando i parametri dal loro punto di vista, introducendo un errore sistematico nelle valutazioni: un giocatore che si immedesima in un altro non conosce le proprie carte, mentre gli altri le vedono. Questo è un motivo in più per rinunciare ad un'esplorazione profonda dello spazio degli stati.

La strategia di Bot1 potrebbe essere definita "altruista": all'inizio del suo turno il bot osserva la situazione dei suoi compagni e li classifica in fasi:

- Fase 1: il giocatore ha una carta con giocabilità 1 (quindi può giocare sicuro).
- Fase 2: il giocatore ha una carta con inutilità 1 (quindi può scartare sicuro).
- Fase 3: il numero di gettoni rimasti è maggiore del numero di turni che devono passare affinché tocchi al giocatore (quindi può sicuramente suggerire)
- Fase 4 (o critica): nessuna delle condizioni precedenti è soddisfatta (il giocatore non può suggerire e qualsiasi mossa faccia rischia di sbagliare)

Se Bot1 ha compagni in fase critica cercherà il miglior suggerimento da dare al primo di loro (in ordine di gioco) in modo che raggiunga la fase di indice più basso possibile. In caso di parità darà il suggerimento che abbassa di più l'entropia della mano.
Se nessuno dei compagni di Bot1 è in fase critica il bot cercherà di compiere la prima azione possibile di questa lista:

1. gioca una carta sicura.
2. se i gettoni suggerimento sono superiori ad un certo limite (numero di giocatori) allora dà il miglior suggerimento al giocatore in fase peggiore (in caso di parità si suggerisce al giocatore cui tocca prima).
3. scarta la carta con inutilità più alta.

Bot2 è stato pensato per implementare una strategia più aggressiva: si tende a giocare immediatamente le carte sicure e a suggerire in modo da aumentare la giocabilità delle carte che possono essere giocate. Le mosse scelte inoltre tendono ad aumentare il numero di carte giocabili tra quelle possedute dai giocatori. Come prima, il bot cercherà di effettuare la prima azione possibile della seguente lista:

1. gioca una carta sicura. Nel caso in cui avesse più carte giocabili gioca quella che rende giocabili il maggior numero di carte possedute dai giocatori.
2. se ha gettoni suggerimento, controlla gli altri giocatori: se uno ha una carta giocabile non sicura gli dà il suggerimento più significativo (in termini di diminuzione dell'entropia della mano) che coinvolge quella carta. Nel caso in cui il giocatore avesse più carte giocabili non sicure gli indica quella che, se giocata, rende giocabili il maggior numero di carte possedute dai giocatori. Questo consiglio viene dato solo se la carta è resa sicuramente giocabile.
3. se non c'è un gettone suggerimento per giocatore e ha una carta sicura da scartare la scarta.
4. se ha almeno un gettone suggerimento prova a suggerire come nel punto 2 anche se la carta non è resa sicuramente giocabile.
5. se ha almeno un gettone suggerimento dà il suggerimento (a qualsiasi giocatore) che diminuisce del massimo l'entropia delle carte.
6. scarta la carta con uselessness maggiore.

Segue la tabella che mostra i risultati medi raggiunti dai bot calcolati su 100 partite per tipo.

Numero di giocatori	**Bot1**	**Bot2**	**Bot1/Bot2**
2	11.57	14.53	12.52
3	**11.97**	**17.42**	**15.11**
4	9.28	17.09	13.38
5	9.09	16.30	13.07

Table 2: Tabella che mostra i punteggi medi su 100 partite raggiunti dai bot

4 Possibili sviluppi

Bot1 e Bot2 non raggiungono punteggi elevati per motivi differenti: Bot1 tende a scartare molte più carte di quante ne gioca, arrivando quindi in fretta alla fine della partita. Bot2 invece perde punti perchè scarta tutti gli esemplari di carte

che erano ancora da giocare.
L'adozione di convenzioni dovrebbe permettere di superare questi limiti.

Un esempio di convenzione che dovrebbe migliorare le prestazioni si basa sul fatto che non ci sono limitazioni su come ordinare le carte nella mano dei giocatori. La convenzione impone di posizionare l'ultima carta pescata nella posizione più a destra nella propria mano e di suggerire le carte giocabili o da non scartare dando la precedenza a quelle a destra. In questo modo si lasciano implicitamente le carte da scartare nella parte sinistra della mano.

Un altro esempio può essere lo stabilire che se un giocatore riceve un suggerimento che coinvolge una sola carta deve giocarla.

Dal punto di vista dell'implementazione degli agenti, le convenzioni hanno il costo di dover tenere traccia di tutti i suggerimenti, dati e ricevuti, e delle informazioni implicite che essi trasportano per convenzione. Inoltre è possibile che la ricerca della mossa migliore richieda un'esplorazione degli stati più profonda di quella effettuata dai bot presentati, a seconda di come la convenzione è formulata.

5 Framework hanabi-unibo

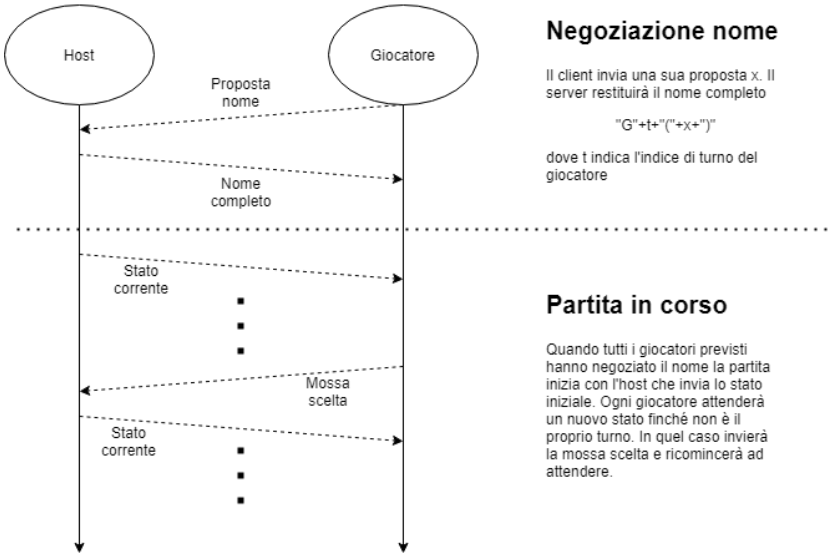
Il framework hanabi-unibo è stato prodotto ex novo come soluzione portabile ed estendibile. Scritto completamente in java è composto da più moduli:

- hanabi-api: contiene classi di supporto agli altri moduli
- hanabi-game: eseguibile, implementa il server host della partita.
- hanabi-human-player: eseguibile, offre una GUI di gioco per utente umano.
- hanabi-bot1: eseguibile, implementa Bot1.
- hanabi-bot2: eseguibile, implementa Bot2.

Tutti gli eseguibili offrono un'interfaccia grafica swing e possono essere avviati con doppio click. In alternativa, l'avvio da terminale permetterà di vedere il log di esecuzione.

Per iniziare una partita si deve avviare l'host hanabi-game e impostare le configurazioni della partita.
Una volta confermata la configurazione, l'host attenderà una connessione tcp alla porta indicata per ogni giocatore indicato come non chiuso. Se il giocatore non è indicato come aperto allora il programma avvierà automaticamente l'eseguibile indicato.

Durante la partita i giocatori comunicano con l'host attraverso il seguente protocollo



I nomi sono inviati come stringhe seguite da un carattere terminatore di riga, gli stati e le mosse invece sono modellati come oggetti json, alla pari degli altri concetti chiave del gioco, come le carte, le mani e le scale di colore.

Segue un esempio di stato inviato dall'host a G2(Human). Rappresenta il secondo stato di una partita a due giocatori, tra un umano e un Bot1. Bot1 ha eseguito la prima mossa che è stata un suggerimento delle carte di valore 3.

```
{
   "discarded":[],
   "green": 0,
   "hints": 7,
   "deck": 40,
   "yellow": 0,
   "red": 0,
   "fuse": 3,
   "current":"G2 (Human)",
   "white": 0,
   "blue": 0,
   "round": 2,
   "G2 (Human)":[
      {"color":null,"possible_values":[1,2,4,5],"value":0,"
          possible_colors":["red","green","white","blue","
          yellow"]},
      {"color":null,"possible_values":[3],"value":3,"
          possible_colors":["red","green","white","blue","
          yellow"]},
      {"color":null,"possible_values":[3],"value":3,"
```

```
            possible_colors":["red","green","white","blue","
            yellow"]},
17        {"color":null,"possible_values":[1,2,4,5],"value":0,"
            possible_colors":["red","green","white","blue","
            yellow"]},
18        {"color":null,"possible_values":[3],"value":3,"
            possible_colors":["red","green","white","blue","
            yellow"]}
19      ],
20    "final": -1,
21    "G1 (Bot1)":[
22        {"color":"white","possible_values":[1,2,4,5,3],"value"
            :4,"possible_colors":["red","green","white","blue",
            "yellow"]},
23        {"color":"yellow","possible_values":[1,2,4,5,3],"value
            ":1,"possible_colors":["red","green","white","blue"
            ,"yellow"]},
24        {"color":"green","possible_values":[1,2,4,5,3],"value"
            :1,"possible_colors":["red","green","white","blue",
            "yellow"]},
25        {"color":"yellow","possible_values":[1,2,4,5,3],"value
            ":2,"possible_colors":["red","green","white","blue"
            ,"yellow"]},
26        {"color":"red","possible_values":[1,2,4,5,3],"value":2
            ,"possible_colors":["red","green","white","blue","
            yellow"]}
27      ],
28    "lastaction":{"hinted":"G2(Human)","type":"hintvalue","
        value":3,"player":"G1(Bot1)"}
29  }
```

Il campo final indica l'ultimo turno di gioco, o vale -1 se ancora ci sono carte nel mazzo. Il campo round indica il turno corrente. Quando il campo final ha valore uguale a quello di round si ha l'ultimo stato della partita. Man mano che i giocatori ricevono gli stati dall'host controllano questi due campi: se sono uguali smettono di seguire il protocollo di gioco e la partita finisce.

Il modulo hanabi-api implementa i concetti chiave sfruttando la libreria di supporto json-v2. Inoltre implementa i componenti grafici e offre classi utili allo sviluppo di nuovi bot: GameClient, che implementa il protocollo di gioco, e Analitics, che offre metodi per l'analisi di uno stato della partita. I bot prodotti sono stati costruiti usando queste classi e i loro sorgenti costituiscono degli ottimi esempi d'uso.

Ad ogni modo, l'host hanabi-game è compatibile con qualsiasi programma (scritto in qualsiasi linguaggio di programmazione) che implementi correttamente il protocollo di gioco.