Relazione di Intelligenza Artificiale e Laboratorio

Modulo di Planning e Sistemi a regole

Matteo Brunello (matr. 858867)
Lorenzo Caresio (matr. 836021)

Introduzione

Questa relazione ha lo scopo di illustrare le varie scelte implementative intraprese per lo sviluppo del progetto di laboratorio inerente al modulo di planning e di sistemi a regole. Il risultato del nostro progetto consiste in 3 versioni differenti di un agente con con capacità crescenti nel gioco della battaglia navale proposto.

Ciclo di controllo

Ogni agente interagisce con l'ambiente creando diverse ipotesi che vengono eventualmente rinforzate o rivalutate sia in base allo stato del gioco che in base allo stato interno dell'agente. Più precisamente, ogni agente mantiene una propria rappresentazione delle ipotesi sullo stato del gioco su cui operano diverse regole necessarie a inferire eventualmente nuova informazione (nuove ipotesi). Poiché in questa fase di ragionamento l'agente necessita di ritrarre e asserire diverse ipotesi abbiamo ritenuto opportuno evitare di far eseguire all'agente le azioni di `guess` e `unguess` all'interno di queste regole, in modo da minimizzare le interazioni necessarie con l'ambiente (e sprecare di conseguenza azioni). Durante lo svolgimento del gioco, quindi, l'agente ragiona solamente sulle proprie ipotesi prima di *materializzarle* in delle *guess* nella fase finale. Il ciclo di ogni agente consiste perciò nelle seguenti fasi:

1. **Fase di inizializzazione**: tutti i fatti necessari al funzionamento dell'agente vengono inizializzati
2. **Fase di fire**: l'agente interagisce con l'ambiente eseguendo azioni di fire sui punti ritenuti più opportuni
3. **Fase di guess**: una volta finite le azioni di fire, l'agente sceglie le 20 ipotesi che ritiene più verosimili e le *materializza* opportunamente in `guess` nell'ambiente di gioco
4. **Terminazione**: l'agente riconosce di aver finito le azioni disponibili ed emette un'azione di `finish`

Per poter mantenere il più possibile isolata la fase di inizializzazione dalle altre fasi abbiamo creato un modulo apposito `INIT`, in cui l'agente rimane fino a quando non ha esaurito tutte le regole eseguibili in esso. Una volta terminata

l'inizializzazione, viene impostato un apposito *flag* che sposta il focus sul modulo `AGENT` in cui sono implementate le altre fasi.

Strategia di Fire

Nonostante alcuni tra gli agenti differiscano leggermente nella strategia di fire, tutti condividono però lo stesso principio di base. Per rendere ogni agente in grado di effettuare delle fire anche in mancanza di informazione iniziale, abbiamo deciso di sfruttare le informazioni riguardo ai conteggi di pezzi di nave presenti per riga e per colonna. A tal proposito, introduciamo della semplice notazione. Sia $X_{r,c}$ la seguente variabile aleatoria:

$$X_{r,c} = \begin{cases} 1 & \text{se la casella } (r,c) \text{ contiene un pezzo di nave} \\ 0 & \text{altrimenti} \end{cases}$$

Siccome la probabilità $P(X_{r,c} = 1)$ esatta è difficile da calcolare poiché bisogna tenere conto di tutte le possibili disposizioni delle navi in una griglia 10×10, abbiamo deciso di utilizzare uno stimatore, cioè un calcolo approssimato della stessa

$$P(X_{r,c} = 1) \approx \frac{k_r + k_c}{u_r + u_c}$$

dove:

- k_r è il numero di pezzi di nave presenti nella riga r (`k-rows`)
- k_c è il numero di pezzi di nave presenti nella colonna c (`k-cols`)
- u_r è il numero di caselle incerte presenti nella riga r
- u_c è il numero di caselle incerte presenti nella colonna c

Ovviamente si tratta di una stima molto rudimentale perché introduce implicitamente diverse ipotesi forti, ma è risultata comunque sufficiente a ottenere risultati particolarmente soddisfacenti.

Per poter tenere conto del numero di pezzi di nave presenti per riga/colonna e il numero di caselle incerte per riga/colonna abbiamo aggiunto 2 fatti, chiamati rispettivamente `local-k-per-row/col` e `uncertain-per-row/col`. Tali fatti sono poi modificati nel corso dello svolgimento del gioco da diverse regole per mantenere i conteggi coerenti sia con lo stato mentale dell'agente che con lo stato dell'ambiente.

Durante la fase di inizializzazione, l'agente calcola $\forall r, c \in [0, 9]$ la stima $P(X_{r,c} = 1)$ tramite il calcolo precedente e salva sia questo risultato che le coordinate r e c in un fatto chiamato `fire-hypothesis`. Nella fase successiva (di *fire*), l'agente estrae ad ogni step del gioco la `fire-hypothesis` con il valore della stima massimo e ne esegue una fire sulla cella associata.

Si noti inoltre che per essere efficaci le stime non devono essere solamente calcolate inizialmente ma è necessario ri-calcolarle ogni qual volta che i valori di k_r, k_c, u_r e u_c cambiano durante lo svolgimento del gioco. Per far ciò, abbiamo introdotto una regola di *update* `update-estimate` che ricalcola la stima per tutte i fatti `fire-hypothesis` che sono incongruenti con lo stato attuale. Inoltre per evitare di incorrere in ricorsioni infinite, è stato introdotto un fatto

`considered` in modo da poter tenere conto quali ipotesi sono state aggiornate e lo *step* in cui ciò è avvenuto.

Agente 1

La prima versione dell'agente utilizza un'ipotesi che codifica il fatto che una determinata casella con coordinate x, y contenga o meno un pezzo di nave. Il grado di certezza con cui l'agente *crede* a questa ipotesi è data dal suo *Certainty Factor*. Dal punto di vista implementativo:

```
(deftemplate hypothesis
  (slot x)                                    ; row
  (slot y)                                    ; column
  (slot CF (type FLOAT) (range -1.0 +1.0))    ; certainty factor value
)
```

Le ipotesi sulle stesse celle sono combinate tra di loro per mezzo di opportune regole che ne combinano i CF secondo quanto visto a lezione. La definizione del template e tutte le regole necessarie a combinare i CF sono state definite nel modulo `UNCERTAINTY` (file `3_Uncertainty.clp`).

Tali ipotesi vengono poi create per mezzo di diverse regole di inferenza che le generano in base ai pezzi conosciuti sulla griglia. In particolar modo, si consideri il seguente esempio in cui un pezzo di nave `top` è conosciuto. La regola andrà a generare specificatamente ipotesi sia sulla casella contenente il pezzo di nave, che nelle 3 successive caselle sottostanti. Ciò è ragionevole siccome tale pezzo potrebbe essere il pezzo finale di una nave da 4.

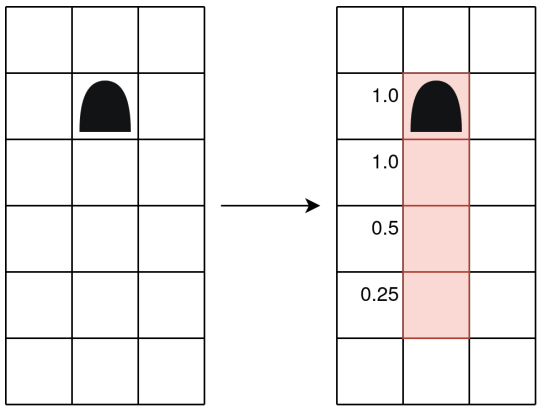


Figura 1: Situazione in cui è conosciuto un pezzo top e relativa generazione di ipotesi dalla regola di inferenza. Le ipotesi sono rappresentate mediante le caselle in rosso. Il loro Certainty Factor è rappresentato a sinistra delle ipotesi.

Si noti inoltre che nella fig. 1 i valori dei CF associati a ogni ipotesi vengono assegnati approssimativamente in modo proporzionale al conteggio delle navi che rappresentano (*es.* l'ultima ipotesi ha CF più basso dal momento che può esserci solo una nave da 4).

Oltre alle regole che creano ipotesi con CF positivo, sono state aggiunte anche regole che creano ipotesi *negative*, cioè con evidenza a favore della presenza di acqua. Queste regole sfruttano il fatto che tutte le navi devono essere distanziate di una posizione tra loro, per cui creano ipotesi negative opportunamente intorno ai pezzi conosciuti.

L'idea alla base di questo agente è che diverse regole di inferenza generino indipendentemente ipotesi che combinandosi tra loro generino ipotesi sempre più verosimili all'aumentare dell'informazione disponibile a disposizione dell'agente.

Una volta finita la fase di *fire*, l'agente prende le prime 20 ipotesi con CF positivo e le rende effettive asserendo una *guess* per ognuna nell'ambiente. In caso le regole di inferenza non abbiano generato un numero sufficiente di ipotesi, l'agente genera le rimanenti prendendo le ipotesi di fire il cui valore della stima è il più alto. In questo modo si evita di andare a sprecare delle *guess*, asserendole di fatto in punti verosimili. È bene notare che questo comportamento non è esclusivo dell'Agente 1, ma è condiviso da tutti gli agenti.

Agente 2

Siccome si è constatato che dal punto di vista pratico il primo agente non generasse abbastanza ipotesi per far si che si combinassero, è stato deciso di prendere una direzione differente per lo sviluppo del secondo agente. Dal punto di vista qualitativo, infatti, le informazioni inerenti ai valori dei CF non venivano sfruttate in nessun modo siccome nella fase finale di *guess* venivano considerate solamente quelle positive e scartate quelle negative a prescindere. Inoltre, siccome un'ipotesi conteneva solamente un valore numerico, non si poteva fare ragionamento in base alla tipologia di pezzo di nave che rappresentava una determinata ipotesi. Per questa ragione, il secondo agente utilizza una rappresentazione delle ipotesi che esplicita la tipologia di contenuto che si crede contenga la casella.

```
(deftemplate hypothesis
  (slot x)
  (slot y)
  (slot content
    (allowed-values
      ver           ; middle piece vertical orientation
      hor           ; middle piece vertical orientation
      top           ; top piece
      bot           ; bot piece
      right         ; right piece
      left          ; left piece
      sub           ; submarine
      water         ; contains water
      unsure        ; it may either be a piece of boat or water
    )
  )
)
```

Anche in questo caso, dopo la fase di *fire*, l'agente crea opportunamente una *guess* per ogni ipotesi che non contenga acqua.

Utilizzare questa rappresentazione alternativa ci ha permesso di riscrivere le regole di inferenza *inverse*, riducendo particolarmente il numero di regole e aumentando la mantenibilità del codice. Inoltre in questa versione abbiamo aggiunto delle regole per inferire la tipologia di nave colpita. A tal proposito si consideri la situazione seguente: un pezzo `middle` è stato colpito e 2 caselle sotto è presente un pezzo bot.

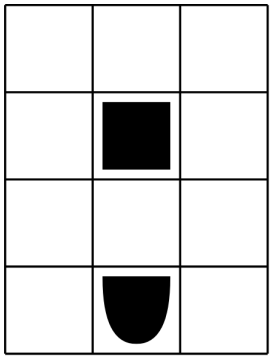


Figura 2: Situazione in cui è possibile determinare con certezza la tipologia di nave colpita

Da questa situazione risulta evidente che la nave in questione è una *corazzata*, per cui è possibile utilizzare questa informazione per creare nuove ipotesi più raffinate. Per esempio, se l'agente sa di aver trovato una corazzata, allora tutti i successivi pezzi `middle` di nave trovati saranno sicuramente di un incrociatore.

Per mantenere queste informazioni è stato creato un fatto apposito, modificato opportunamente dalle regole adibite a riconoscere queste situazioni.

```
(deftemplate boat-count
  (slot boat (allowed-values battleship cruiser destroyer submarine))
  (slot num)
)
```

Con l'aggiunta di queste ulteriori regole, l'agente risultante ha ottenuto performance significativamente migliori rispetto alla versione precedente.

Agente 3

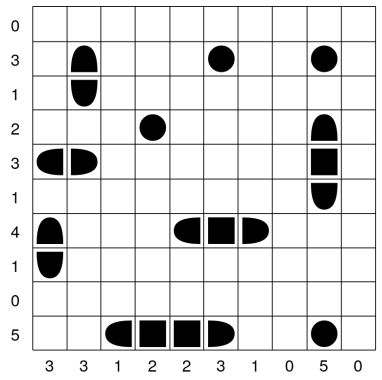
L'ultima versione dell'agente introduce una strategia di *fire* differente. Questa strategia si basa sull'osservazione che ogni pezzo di nave scoperto durante lo svolgimento del gioco può generare un alto numero di ipotesi incerte. Come caso limite, si pensi per esempio a un pezzo `middle` scoperto nel mezzo della griglia in cui non è possibile in nessun modo disambiguare l'orientamento possibile della nave sottostante. In questo caso, per tenere conto di tutti gli orientamenti, le lunghezze e le disposizioni possibili, sarebbe necessario generare 12 ipotesi, tra cui nel caso migliore solo 3 di esse saranno *guess* valide. Questa situazione migliorerebbe però se si conoscesse il contenuto di una delle celle adiacenti.

La strategia di *fire* accennata in precedenza sfrutta questo fatto eseguendo delle *fire* nelle celle adiacenti ai pezzi incerti in modo da disambiguare queste situazioni. Per implementare questa strategia è stato sufficiente introdurre delle regole per riconoscere queste situazioni che se soddisfatte asseriscono delle `fire-hypothesis` con stima pari a `1`. Così facendo, l'agente selezionerà tale cella per la prossima azione di *fire* da eseguire nello step successivo.

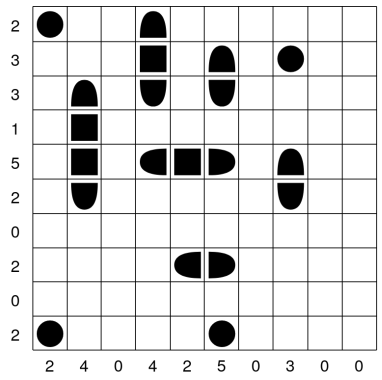
Ovviamente, in caso non si attivassero mai queste regole, l'agente ricadrebbe nella sua strategia usuale di firing.

Risultati della sperimentazione

Tutti gli agenti sono stati testati su 2 mappe differenti. Ogni mappa è stata a sua volta testata partendo da 0 pezzi conosciuti iniziali e incrementandone il numero fino a 5. Le mappe utilizzate sono riportate di seguito



(a) Mappa 1



(b) Mappa 2

I risultati dei test nella prima mappa sono riassunti nella seguente tabella:

Agente	Nr.Pezzi Conosciuti	Punteggio
1	0	110
1	1	120
1	2	70
1	3	110
1	4	150
1	5	150
2	0	245
2	1	225
2	2	225
2	3	255
2	4	285
2	5	270
3	0	190
3	1	225
3	2	210

Agente	Nr.Pezzi Conosciuti	Punteggio
3	3	295
3	4	325
3	5	310

Di seguito, invece i risultati dei test relativi alla seconda mappa:

Agente	Nr.Pezzi Conosciuti	Punteggio
1	0	170
1	1	200
1	2	210
1	3	195
1	4	270
1	5	245
2	0	140
2	1	315
2	2	300
2	3	340
2	4	325
2	5	310
3	0	190
3	1	315
3	2	300
3	3	340
3	4	325
3	5	310