Reinforcement Learning: analisi di politiche di path-planning per agenti singoli e cooperativi

Fabio Proietti*,
* DISI, University of Bologna, Italy

Email: fabio.proietti@studio.unibo.it

***Abstract*—L'apprendimento da parte di umani intelligenti è facilitato nel caso in cui esso è condiviso tra gli individui. Se però questa cooperazione avviene con l'utilizzo delle applicazioni robotiche, è importante capire quali informazioni comportano un maggior livello di conoscenza per ciascuno di loro e se la loro condivisione in termini di tempo è accettabile. Questo progetto cerca di dar luce a questi interrogativi simulando l'azione di agenti che hanno lo scopo di pianificare un cammino ottimo per raggiungere un obiettivo. E' stato quindi implemento l'algoritmo di Reinforcement Learning Qlearning: esso, utilizzando un meccanismo basato su ricompense, è in grado di allenare l'agente ad apprendere le caratteristiche dell'ambiente circostante.**

I. Introduction

Il Reinforcement Learning [1] è una tecnica che permette a degli agenti autonomi di apprendere le caratteristiche di un ambiente, che in molti casi è mutevole, attraverso la distribuzione di una ricompensa in grado di valutarne le prestazioni. Questo progetto ha lo scopo di implementare un ambiente simulato dell'algoritmo Q-learning, al fine di analizzarne le prestazioni sia nel caso di un agente indipendente che cooperativo.
Le sezione successiva descrive la teoria riguardo il RL, dell'algoritmo Qlearning e l'ambito delle sue applicazioni; si parlerà poi dell'architettura, delle scelte implementative riguardo il progetto. Infine verranno discussi i risultati ottenuti.

II. Related works

Il Reinforcement Learning [1] è una particolare branchia del Machine Learning con lo scopo di progettare agenti che apprendono e di conseguenza agiscono. Le possibili applicazioni variano dalla robotica, all'industria manifattoriera, o alla soluzioni di problemi di natura combinatoria come nei *game playing*. La filosofia del RL si basa sulla programmazione di agenti che hanno come obiettivo di raggiungere un task attraverso l'utilizzo di ricompense (positive o negative) a seguito delle azioni svolte; ciò comporta che il focus della progettazione va al di là del *come* l'obiettivo deve essere raggiunto. Le principali strategie di risoluzione utilizzate dalle tecniche di Reinforcement Learning sono due: cercare nell'insieme dei comportamenti quello che ottimizza la soluzione di un determinato ambiente (approccio utilizzato soprattutto negli algoritmi genetici), o quella di usare tecniche statistiche o metodi di programmazione dinamica al fine di stimare l'utilità di una azione intrapresa rispetto la posizione assunta dall'agente nella mappa.
Q-learning è un algoritmo di Reinforcement Learning la cui politica di apprendimento è basata sul valore della funzione stato/azione Q che viene influenzata dalla ricompensa ottenuta ad ogni scelta della mossa da effettuare. Quindi se un agente si trova in uno stato x seleziona con una data probabilità un'azione $a_i \in A$, esso ottiene una ricompensa r e si sposta nella posizione s'. Ad ogni passo esso aggiorna la funzione $Q(s,a)$ secondo la seguente equazione:

$$Q(s,a) \leftarrow Q(s,a) + \beta(r + \gamma(V(s') - Q(s,a)) \qquad (1)$$

, dove γ è un valore di discount compreso tra 0 e 1 e $V(s')$ è un valore dato da $V(s') = \max_{a' \in A} Q(s',a')$, con a' scelto dal set delle possibili mosse attuabili da quella corrente e β è un valore positivo compreso tra 0 e 1 con lo scopo di "pesare" il learning rate; in questo caso quindi durante l'esplorazione un agente aggiorna la propria funzione Q in modo graduale. E' stato inoltre dimostrato che Q-learning per un Markov Decision Process finito converge ad una ottima politica decisionale. Q-Learning è il più importante algoritmo di tipo *model-free* per l'apprendimento da un *reinforcement* ritardato: ciò significa che esso prima o poi convergerà ad un valore ottimo indipendentemente dal comportamento dell'agente durante il collezionamento dei dati. Quindi anche se l'*exploitation/exploration* devono essere indirizzate, i particolari della strategia di esplorazione non avranno effetto sulla convergenza dell'algoritmo di learning. Una volta conclusa la fase di learning, l'algoritmo nella fase di pratica utilizza varie politiche; non ne esistono di computazionalmente ottimali o meno, ma molto spesso vengono progettate ad-hoc in base all'ambiente che si va ad esplorare. Le politiche più conosciute sono:

- *greedy*: ossia una serie di tecniche che hanno come punto fermo la scelta dell'azione migliore rispetto allo stato corrente. Questo tipo di politica però potrebbe però non raggiungere mai casi di ottimalità poiché essi potrebbero non essere mai scoperti.
- *randomized*: Le strategie di tipo random si basato sempre sulla scelta dell'azione ottima, ma a volte, data una certa probabilità, questa scelta viene sostituita con una mossa random tra quelle attualmente disponibili nello stato corrente. Esistono politiche più complesse appartenenti a questa categoria, ed una di esse è la cosiddetta esplorazione di Boltzmann.
- *Interval-based*: questa famiglia di tecniche invece si basa sul calcolo della stima di successo nel caso in cui

un agente scelga una determinata mossa. Una azione viene scelta calcolando l'upper-bound dell'intervallo di confidenza che si basa sulla probabilità di successo che ha quella determinata mossa. Maggiore è questo limite superiore, maggiore è la probabilità di successo; d'altra parte una bassa probabilità incoraggia una maggiore esplorazione.

III. Architecture

Il progetto si basa sulla simulazione di agenti con lo scopo di risolvere un problema di *path-planning* utilizzando l'algoritmo di Reinforcement Learning Q-learning [2]. Queste componenti sono in grado di applicare politiche di esplorazione basate sia su un agente singolo che sulla cooperazione di questi ultimi. Lo scopo è stato quello di analizzare il comportamento degli agenti, sia nel caso di cooperazione che non, comparandone le tecniche di esplorazione volte al raggiungimento del/dei obiettivi preposti. Le politiche implementate sono [3]:

- greedy ed ε-greedy per nel caso di agenti non cooperanti;
- greedy, ε-greedy e una tecnica basata su un "grado di fiducia" rispetto alle informazioni ambientali ricevute dagli altri agenti.

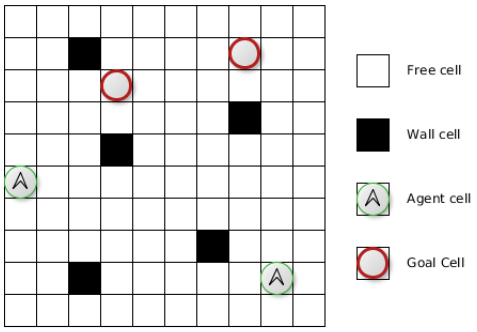

Fig. 1: $n \times n$ grid

A. Descrizione delle componenti

1) Ambiente: L'ambiente è di tipo griglia di grandezza $n \times n$. Ogni riquadro della matrice corrisponde ad uno stato. Ogni stato può essere occupato da uno o più agenti, da un "nodo-obiettivo" o da un ostacolo.

2) Agente: Ogni agente occupa uno stato dell'ambiente, più agenti possono condividere lo stesso stato. Le azioni possibili sono: spostamento in alto, in basso, a destra o a sinistra. Un'agente non può occupare uno stato in cui è presente un'ostacolo, ma può occupare una casella della griglia dove è presente un "nodo-obiettivo". Ogni agente è equipaggiato con una antenna che gli permette di scambiare informazioni rispetto l'ambiente nel caso in cui uno o più di essi si trovino all'interno del suo raggio di trasmissione. Gli ostacoli non interferiscono con la potenza del segnale.

3) Nodo-obiettivo: Un nodo-obiettivo, è progettato in modo che può essere sia mobile nello spazio che fisso; esso quindi una volta definito il suo stato nell'ambiente rimane sempre nella stessa posizione. In uno stato in cui è presente un nodo-obiettivo possono essere presenti uno o più agenti.

4) Wall: In un ambiente è possibile posizionare uno o più ostacoli. In ogni stato dove è presente un *wall* non è possibile che vi siano altri tipi di componenti.

B. Conoscenze a priori degli agenti

La conoscenza preliminare di ogni agente all'inizio della fase di training è la stessa. Ognuno di essi inizialmente è al corrente del proprio stato iniziale, del numero e della posizione dei nodi-obiettivo e dei confini dell'ambiente stesso. Con quest'ultima assunzione si intende che nessun agente è in grado di effettuare una mossa che lo porti fuori dalle coordinate della matrice. Allo stesso tempo però essi non sono a conoscenza del numero e della posizione degli ostacoli.

C. Strategie di esplorazione

Le strategie di esplorazione implementate ed utilizzate sono di due tipi: greedy e randomizzata. La progettazione di queste politiche inoltre è applicabile sia nel caso di agenti singoli che cooperativi.

1) Random: Politica di tipo randomized che sceglie random una mossa $a_i \in A$ con A l'insieme delle mosse disponibili rispetto allo stato corrente dell'agente nello spazio.

2) Boltzmann Distribution: Politica di tipo randomized basata sulla distribuzione di Boltzmann ossia:

$$\rho(a_i|s) = \frac{e^{Q(s,a_i)/T}}{\sum_{k \in A} e^{Q(s,a_k)/T}} \tag{2}$$

dove $\rho(a_i|s)$ è la probabilità che l'azione a_i venga scelta come stato successivo dell'agente e $0 < T \leq 1$ proporzionalmente crescente rispetto al numero di prove di esplorazione dell'ambiente da parte dell'agente.

3) ϵ-greedy: E' una politica di tipo greedy, dove un agente sceglie la miglior mossa disponibile rispetto al proprio stato (ossia quella con $Q(s,a)$ maggiore) con una probabilità pari a ρ; con probabilità $1-\rho$ invece il nuovo stato dell'agente sarà random. Nella versione cooperativa, nel caso in cui due o più agenti si trovano nello stato range di trasmissione, la probabilità ρ consiste nello scegliere la miglior mossa dell'agente corrente rispetto il suo stato, in base anche a quella che è la conoscenza dell'ambiente da parte degli altri agenti con cui è in grado di comunicare. Dati quindi due agenti X e Y, l'agente X opterà per l'azione $a_i \in A$ comparando il suo $Q_X(s,a)$ maggiore con il corrispondete valore massimo $Q_Y(s,a)$. Con probabilità $1-\rho$ l'agente X sceglie una mossa random $a_k \in A$.

4) Value Max: Questa strategia può essere applicata sia in caso di agenti singoli che cooperanti e sceglie sempre l'azione con $Q(s,a)$ maggiore tra quelli presenti nel set delle azioni disponibili. Nel caso cooperativo la mossa scelta corrisponde al valore $Q(s,a)$ maggiore tra tutti gli agenti che in quel momento comunicano tra di loro.

5) ad-hoc policy: E' una politica applicabile solo nel caso cooperativo e si comporta in modo simile a ϵ-greedy. Dato che non sempre ad una mossa con valore massimo corrisponde il miglior path, l'idea è di dare una "percentuale di verità" alle informazioni ricevute da altri agenti. Quindi con probabilità ρ l'agente sceglierà la miglior mossa comunicata dagli altri agenti e con probabilità $1 - \rho$ l'azione data dalla sua esplorazione.

D. Workflow di una esecuzione

Dopo che è stato definito l'ambiente da esplorare, vengono generati i nodi-obiettivo ed uno o più agenti. In questa fase si crea anche la matrice delle ricompense: un agente riceve una ricompensa nulla nel caso si sposta da una cella libera ad un'altra, una ricompensa negativa in caso volesse muoversi in una parte della griglia dove è presente un ostacolo e positiva nel caso in cui si muove in uno stato in cui è presente un obiettivo. Ogni agente è composto da diverse strutture dati utili a mantenere le informazioni che raccoglie durante la fase di esplorazione.
A questo punto inizia la vera e propria fase di esplorazione: ad ogni agente, posizionato in modo random nella mappa, gli viene "impartito" quale politica utilizzare. Un ciclo di learning si conclude quando un agente raggiunge l'obiettivo; quindi esso viene riposizionato in un altro stato random e inizia una nuova esplorazione. Il numero delle esplorazioni è definito a priori. Terminata questa fase, inizia quella pratica: in base alla conoscenza appresa ed alla politica adottata dagli agenti (anch'essa scelta a priori) viene impartito l'ordine di raggiungere un nodo obiettivo e verificata l'ottimalità del path-planning rispetto la conoscenza appresa nella fase di learning, le azioni svolte in quella di testing e le strategie adottate per la scelta delle azioni.

E. Cooperazione e aggiornamento dello stato globale dell'ambiente

Per garantire che la cooperazione si basi su uno stato globale identico per tutti gli agenti, subito prima dell'inizio della simulazione viene creata una copia dello spazio. Ogni agente presente nel set di agenti quindi effettua sequenzialmente una mossa basandosi sullo stato "reale" della matrice e aggiornando poi la copia. A questo punto alla fine del ciclo, ossia dopo che tutti gli agenti hanno effettuato una azione, viene aggiornato lo stato globale della matrice in base alle mosse presenti nella copia.

IV. Implementation

La simulazione dell'algoritmo Q-learning è stata completamente realizzata in Java. In allegato vi è il diagramma UML delle classi [1]

A. Descrizione delle classi

1) Model.java: la classe Model si occupa della inizializzazione dell'ambiente e di tutte le altre componenti grazie al metodo runModel()1 . La seguente porzione di codice riguarda il nucleo dell'esecuzione della simulazione. Con il metodo runMultiAgent presente nella classe Qlearning, ogni agente esegue autonomamente una fase di apprendimento dello spazio circostante. A questo punto le istruzioni contenute all'interno del ciclo while sono eseguite finché ogni agente ha raggiunto l'obiettivo stabilito. L'esecuzione di questa seconda fase da parte degli agenti può essere eseguita mantenendo la stessa politica utilizzata nella fase di learning o utilizzarne un'altra tra quelle spiegate precedentemente.

Algorithm 1: Pseudocodice del metodo runModel().

```
initialize environment;
foreach agent do
    explore environment;
while true do
    if all agents are in goal states then
        break;
    make a copy of environment;
    foreach agent do
        if agent not in goals.state then
            make an action using a policy;
            update copy environment;
    update environment;
```

2) GridWorld.java: GridWorld contiene i metodi utili ad inizializzare, gestire ed aggiornare le strutture dati dell'ambiente di simulazione. Sono qui presenti quindi le matrici che contengono le coordinate delle componenti, quelle adibite al mantenimento del raggio di propagazione dell'antenna di ogni agente, ecc.

3) Agents.java, Walls.java, States.java, Nodes.java: Queste classi contengo tutte le strutture utili alla gestione degli stati delle singole componenti; nella classe Agents sono presenti liste che mantengono l'elenco delle azioni svolte, l'inizializzazione e l'aggiornamento della matrice $Q(s, a)$ la gestione dello stato corrente e di quello iniziale, ecc.

4) Antenna.java: Associata alla classe Agents.java, Antenna.java ha lo scopo di verificare e gestire la presenza di eventuali agenti nel range di propagazione dell'agente corrente. Perciò ogni agente si mantiene una matrice identica a quella dell'ambiente in cui mostra il suo stato corrente più la potenza di propagazione del segnale. In questo modo prima di eseguire una possibile azione, nel caso di politiche cooperative, ognuno effettua uno scan sulla $i-esima$ e $j-esima$ posizione di ogni matrice degli altri agenti raggiunte dal segnale.

[1] ./UMLDiagram.png

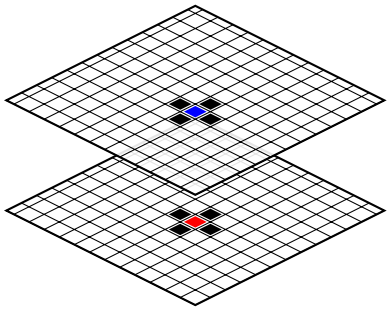

Fig. 2: Range di ricezione delle antenne presenti in ogni agente

5) Qlearning: La classe Qlearning.java infine implementa sia i metodi che hanno lo scopo aggiornare tutte le strutture utilizzate dall'algoritmo che le strategie di scelta delle successive mosse utili a raggiungere l'obiettivo. Il metodo runMultiagent() 2 esegue le istruzioni per il learning dell'ambiente.

Algorithm 2: runMultiAgent()

```
for n° of Trials do
    while true do
        if agent is in a goal state then
            define new agent state;
            break;
        choose an action using a policy from current
          state;
        take an action, observe r and next state;
        Q(s,a) ← Q(s,a) + β(r + γ(V(s') − Q(s,a)));
        state ← next state;
```

V. Performance evaluation

Il calcolo delle prestazioni dell'algoritmo sono basate sul numero di mosse medio e sul tempo di esecuzione che gli agenti nella fase testing impiegano nel raggiungere gli obiettivi prefissati dopo che sono state svolte n esplorazioni di training. Tutti i risultati delle simulazioni sono valori calcolati sulla media delle 20 run. Il numero di esplorazioni nella fase di training varia da 50 a 525 con un passo di 25. L'ambiente invece ha le seguenti proprietà:

- una matrice 30×30 per quanto riguarda lo spazio;
- 30 ostacoli posti casualmente;
- 8 agenti posti in stati random;
- un solo nodo-obiettivo comune per tutti gli agenti;
- il raggio di copertura di ogni agente è impostato a 2 celle.

I test eseguiti sono stati di 4 tipi:

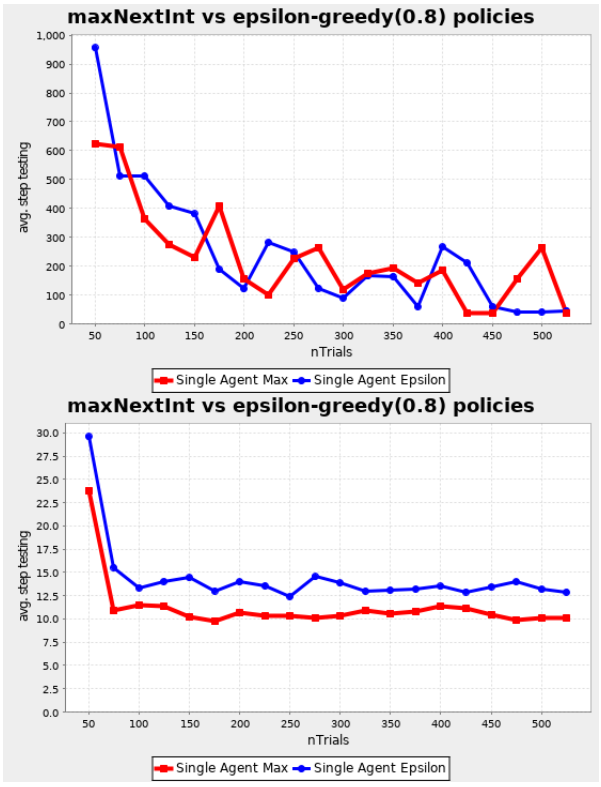

Fig. 3: Numero di passi medi dopo la fase di testing: (a) nella fase di training è stata applicata la politica ϵ-greedy(0.8) e in (b) una politica random.

1) Test 1: Applicazione della politica ϵ-greedy con valore 0.8 durante la fase di training e le politiche Max Value e di nuovo ϵ-greedy senza cooperazione nella fase di testing. Il risultato è dato dal numero di passi medio per raggiungere l'obiettivo durante la pratica. 3(a)
Nella Fig. 3(b) invece durante la fase di training viene impiegata una politica random rispetto la scelta della mossa da eseguire.

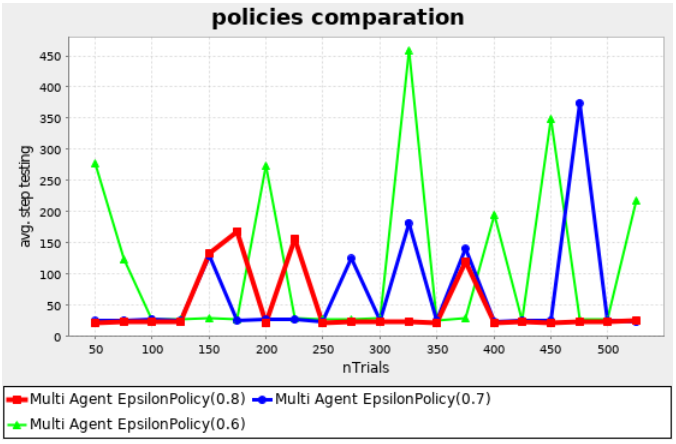

Fig. 4: Comparazione della politica ϵ-greedy() con valori ϵ pari a 0.8, 0.7, 0.6 durante la fase di testing.

2) Test 2: Comparazione delle politiche, durante la fase di testing, ϵ-greedy nel caso di un agente singolo con valori di ϵ pari a 0.8, 0.7, 0.6. Il training è stato eseguito con una politica random. 4

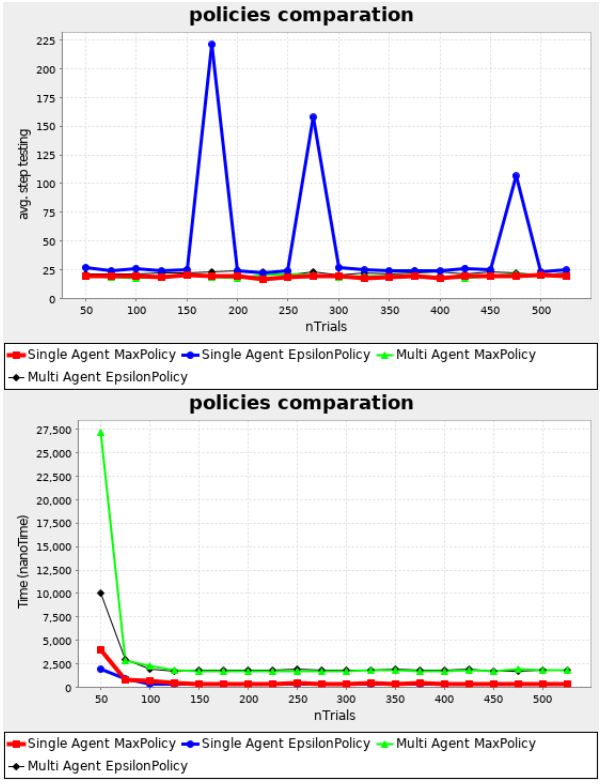

Fig. 5: Comparazione delle politiche sia singole che cooperative:(a) mostra il numero medio di mosse e (b) il tempo medio in nanoTime per raggiungere l'obiettivo.

3) Test 3: Comparazione delle politiche Max Val e ϵ-greedy(0.8) nel caso di agenti singoli e cooperativi. La fase di training è stata eseguita applicando la politica random. 5

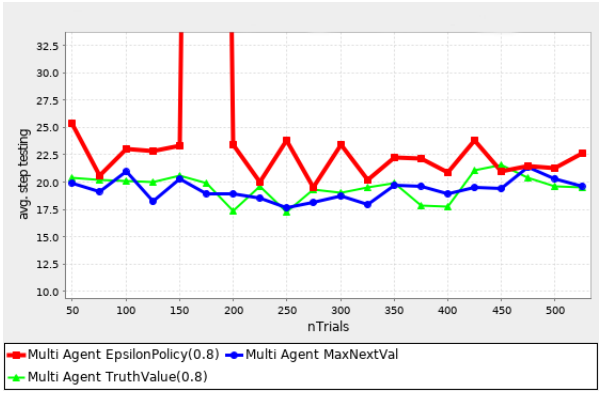

Fig. 6: Comparazione delle politiche multi Agent Max Value, ϵ-greedy(0.8) e la politica ad-hoc truthValue(0.8)

4) Test 4: Come ultimo test di valutazione delle prestazioni dell'algoritmo sono stati comparati i valori ottenuti dalla esplorazione nella fase di testing applicando le politiche Max Value, ϵ-greedy(0.8) e TruthPolicy(0.8). 6

A. Discussione sui risultati ottenuti

Prima di ricavare questi risultati sono state effettuate delle analisi preliminari volte a giustificare le scelte adottate. I parametri interessati sono:

- la scelta della politica utilizzata durante il training;
- il numero di agenti e obiettivi presenti nello spazio;
- il raggio di copertura dell'antenna di ogni agente;
- la quantità di esplorazioni nella fase di apprendimento;
- in base alla grandezza della griglia (comparata anche al numero di agenti) o alla quantità di ostacoli definiti.

Da ciò è risultato che le politiche di tipo greedy applicate durante la fase di esplorazione riflettono scarse prestazioni nella fase di testing. Questo comporta la necessità di un maggior numero di esplorazioni da parte dell'agente; è stato provato che nel caso si utilizzi una politica greedy anche durante la fase di testing, la probabilità di incorrere in zone dello spazio poco esplorate è maggiore. In questi casi si creano un loop di mosse che non permettono all'agente di raggiungere l'obiettivo.
La scelta di eseguire test in un ambiente in cui è presente un solo obiettivo è dettata dal fatto che in questo modo è possibile analizzare in maniera più chiara la dinamica rispetto la cooperazione degli agenti.
Infine è stato testato che l'intervallo di esplorazioni $[50, 525]$ è sufficiente per apprendere una conoscenza che, con molte probabilità, evita la formazione di massimi e minimi locali nella mappa.
Le supposizioni avute prima della fase di testing del progetto, sono state quasi totalmente confermate dai risultati prodotti dalle analisi. Come si può notare nel caso del primo test (Fig.3), l'applicazione di una politica random rispetto ad una greedy durante l'esplorazione, comporta poi nella successiva fase di pratica, una drastica diminuzione nel numero medio di passi utili al raggiungimento dell'obiettivo.
La scelta del valore 0.8 per quanto riguarda l'epsilon delle politiche ϵ-greedy, è dettato dai risultati del test numero due (Fig.4): come ci si poteva aspettare ad una maggiore probabilità di scegliere la miglior mossa nella fase di testing corrisponde una maggiore linearità per quanto riguarda il numero di mosse medio utile al raggiungimento dello stato-obiettivo. Analizzando ancora questa politica è interessante notare come la non linearità che si presenta spesso nel caso di un agente singolo, è riscontrata in proporzione minore nel caso della cooperazione tra agenti (Fig. 5, Fig. 6). Inoltre la figura 5 conferma come il confronto tra le politiche greedy e ϵ-greedy(0.8) applicate nei casi di agenti singoli e cooperativi non comporta nessun significativo miglioramento delle prestazioni dell'algoritmo. Da notare è invece un lieve aumento di tempo, dato dallo scambio di informazioni durante la cooperazione. I motivi di questo risultato possono essere molteplici:

- è possibile che la grandezza dell'ambiente non permetta un continuo scambio di informazioni tale da determinare un maggiore differenza di prestazioni;
- la scelta di stabilire l'azione successiva solo in base al $Q(s, a)$ massimo potrebbe non essere sufficiente;
- man mano che un nodo si avvicina all'unico obiettivo potrebbe non essere più fondamentale recepire informazioni dagli altri.

Infine l'ultimo test (Fig.6) valuta le prestazioni della politica TruthValue implementata ad-hoc per questa simulazione: i risultati mostrano che anche se in questo caso le prestazioni sono in linea con le altre politiche cooperative, in alcune situazioni si possono notare dei lievi miglioramenti rispetto al numero di mosse medio utili al raggiungimento del goal. E' quindi lecito assumere le stesse supposizioni del caso precedente, ma è altrettanto auspicabile che alcune modifiche possano comportare risultati migliori rispetto alle altre strategie.

VI. Conclusioni

Lo scopo del progetto è stato quello di implementare l'algoritmo Q-learning in un ambiente simulato, che fosse in grado di risolvere il problema del path-planning. Le strategie utilizzate per questo scopo sono appartenenti alle classi di politiche greedy e random e applicate sia in ambito di agenti individuali che cooperativi. Le analisi effettuate a seguito della simulazione hanno evidenziato come le politiche di tipo greedy siano migliori per un calcolo del percorso ottimale rispetto l'obiettivo, ma che nel caso di una fase di esplorazione debole non assicurano la risoluzione del problema. Strategie pseudo-random, anche se meno performanti, possono sopperire a questa lacuna. Per quanto riguarda il caso cooperativo invece, è stato interessante verificare che lo scambio di messaggi riguardanti solo la miglior azione possibile da intraprendere, non è sufficiente ad aumentare le prestazioni dell'algoritmo. Possibili sviluppi futuri potrebbero quindi concentrarsi a migliorare questo aspetto introducendo politiche più specifiche, aumentando il numero di informazioni scambiate o particolarizzare il processo di esplorazione con tecniche ad-hoc rispetto l'ambiente circostante.

References

[1] Kaelbling, Leslie Pack, Michael L. Littman, and Andrew W. Moore. "Reinforcement learning: A survey." Journal of artificial intelligence research 4 (1996): 237-285.

[2] Tan, Ming. "Multi-agent reinforcement learning: Independent vs. cooperative agents." Proceedings of the tenth international conference on machine learning. 1993.

[3] https://www.google.it/url?sa=t&rct=j&q=&esrc=s&source=web&cd=2&ved=0ahUKEwiYzpHUhqvPAhWDbRQKHQ21C6EQFggoMAE&url=http%3A%2F%2Fweb.engr.oregonstate.edu%2F~afern%2Fclasses%2Fcs533%2Fnotes%2Frl.ppt&usg=AFQjCNF20kqx0YciDr61v2gJjUI6RGNjzA&sig2=YVOElpZvjncb_HKBlf0iYw