POLITECNICO DI MILANO
Corso di Laurea in Ingegneria Informatica
Dipartimento di Elettronica e Informazione

Pattugliamento strategico multi-robot in ambienti di topologia arbitraria

AI & R Lab
Laboratorio di Intelligenza Artificiale
e Robotica del Politecnico di Milano

Relatore: Ing. Nicola Gatti
Correlatore: Ing. Nicola Basilico

Tesi di Laurea di:
Federico Villa, matricola 720492

Anno Accademico 2009-2010

A Papà

Indice

Capitolo 1

Introduzione

Il nostro obiettivo è presentare un valido modello per il pattugliamento multiagente basato sulla #teoriadeigiochi che estenda il BGA

Lo studio di tecniche per il pattugliamento di un ambiente da parte di robot autonomi è un'area dell'Intelligenza Artificiale che è oggetto di un crescente interesse per via delle molteplici situazioni in cui può essere utilizzata.

Il pattugliamento è un problema che può essere declinato in numerosi modi, a seconda della *funzione obiettivo* utilizzata: alcuni modelli [11], ad esempio, hanno come scopo il massimizzare la frequenza tra due successivi passaggi di un robot nello stesso punto, mentre altri [15] cercano di minimizzare il tempo che intercorre tra due successive visite di una porzione di ambiente; altri ancora cercano di massimizzare l'utilità del pattugliatore.

Per trovare soluzioni accettabili ai primi due tipi di *funzione obiettivo* possiamo sfruttare approcci tradizionali, basati su tecniche mutuate da altre discipline come la *ricerca operativa* ma, qualora il nostro intento sia il massimizzare l'*utilità* di un agente, dobbiamo ricondurci alla *teoria dei giochi*. L'idea di base risiede nel fatto che sia possibile modellizzare il pattugliamento di un ambiente come un gioco non-cooperativo tra una squadra di robot e un ipotetico intruso che osserva la strategia dei pattugliatori e in base ad essa effettua le sue mosse cercando di massimizzare la propria utilità.

Esistono numerosi modelli basati su questo approccio, ma tutti falliscono nel rappresentare una valida soluzione in ambito multiagente. In alcuni casi, come in [16], non è possibile estendere il lavoro svolto ad un generico ambiente; in altri casi [1] l'approccio multiagente è una diretta derivazione di

quello singolo agente, ed impone vincoli così stringenti sulla sincronizzazione degli agenti da essere applicabile solo ad ambienti particolari.

L'approccio *Basilico-Gatti-Amigoni* [5], sviluppato al Politecnico di Milano, è un modello ben più che valido per il problema del pattugliamento singolo agente. In particolare, la difesa di un ambiente di topologia arbitraria viene rappresentato come un gioco non-cooperativo a turni di tipo *leader-follower* dove il pattugliatore è il leader e il ladro, che osserva le mosse del leader e in base ad esse sceglie la strategia, il follower. Inoltre, a differenza di altri modelli, contempla l'ipotesi che esista una gerarchia tra gli obiettivi da proteggere e che all'intruso occorra del tempo per penetrare in un obiettivo.

Lo scopo di questa tesi è estendere il modello BGA da singolo agente a multiagente, senza incorrere nelle limitazioni presenti in modelli precedenti. In particolare, accanto alla possibilità di utilizzare agenti con strategie dipendenti che si muovono sull'intero ambiente, esploriamo approcci al partizionamento che ci permettono di utilizzare robot con strategie disgiunte su porzioni separate di ambiente.

Quando lavoriamo con più pattugliatori dobbiamo necessariamente occuparci di come coordinare il loro movimento e di quanti ne abbiamo bisogno per essere certi di poter sorvegliare l'intero ambiente. Indagare in questa direzione ci consente di stabilire un *lower bound* sul numero di robot necessari per proteggere un ambiente di topologia arbitraria. Inoltre, una volta calcolato tale numero, individuiamo due *dimensioni di coordinamento* (decomposizione strategica e decomposizione topologica) tra i robot, le cui combinazioni ci permettono di definire differenti modelli di strategie.

Forniamo, poi, risultati sperimentali di ogni combinazione e la scalabilità di ogni strategia all'aumentare delle dimensioni dell'ambiente da pattugliare, in relazione anche a considerazioni sul coordinamento presenti nella letteratura precedente.

La tesi è strutturata come segue. Nel capitolo 2 è presente l'attuale stato dell'arte sul pattugliamento e vengono trattati sia approcci non strategici che, in dettaglio, approcci strategici. Contiene, inoltre, un utile compendio di teoria dei giochi.

Nel capitolo 3 presentiamo l'estensione in ambito multiagente dell'algoritmo BGA, estendendo il formalismo generale e stabilendo un *upper bound* alla complessità del problema.

Nel capitolo 4 affrontiamo la ricerca del numero minimo di robot necessari a proteggere un ambiente e presentiamo l'algoritmo *Basilico-Gatti-Villa* per la decomposizione dell'ambiente in aree pattugliabili da un solo robot.

Il capitolo 5 contiene la presentazioni delle dimensioni individuate, le loro combinazioni e i modelli associati ad ogni combinazione.

Il capitolo 6 riporta alcuni dei test che abbiamo effettuato e il capitolo 7 contiene le conclusioni.

Sono poi presenti due appendici: la A contiene la primissima formulazione data del problema presente nel capitolo 4, ed è utile per comprendere i passaggi che ci hanno portato alla soluzione; la B contiene una sommaria documentazione del programma JAVA utilizzato per preparare gli ambienti e calcolare il BGV per il capitolo 6.

Il lettore più attento può notare che ogni capitolo è aperto da una frase *Twitter-style* di massimo 140 caratteri che ne riassume il contenuto.

brevitas sapientiae anima est.

Capitolo 2

Stato dell'arte

Gli approcci esistenti al #patrolling sono delundenti; la #teoriadeigiochi offre una visione inedita del problema, indicando nuove soluzioni

Il seguente capitolo ha l'obiettivo di introdurre il problema del pattugliamento singolo agente e multiagente e gli approcci finora utilizzati per la sua soluzione. Mentre agli approcci non strategici dedichiamo relativamente poco spazio, gli approcci strategici e le basi di teoria dei giochi sono maggiormente approfonditi, in quanto fondamentali alla comprensione del lavoro svolto. Riportiamo poi l'approccio Basilico-Gatti-Amigoni (BGA) di cui questa tesi costituisce la naturale espansione in ambito multiagente.

2.1 Teoria dei giochi

Gli approcci strategici di cui parleremo nel seguito del capitolo sono relativamente nuovi, e richiedono per la loro comprensione un'approfondita conoscenza del campo. Nelle prossime sezioni sono riportati i principali concetti e le principali definizioni necessarie a comprendere non solo le soluzioni trovate in altri lavori, ma anche e soprattutto il lavoro svolto in questa tesi.

2.1.1 Nozioni fondamentali

La teoria dei giochi è lo studio dei modi in cui l'interazione strategica tra agenti razionali produce risultati relativi alle utilità di quegli agenti, nessuno dei quali potrebbe essere stato voluto dagli agenti stessi [17]. Tale definizione, vagamente criptica, verrà chiarita dal seguito del paragrafo. Tuttavia,

in modo informale, possiamo descriverla come la costruzione di un modello che rappresenti il comportamento di un'entità nei casi in cui l'interazione tra diverse entità comporta una ricompensa.

Benchè la teoria matematica alla base della teoria dei giochi sia stata sviluppata da John von Neumann e Oskar Morgensten nel 1944, e che lo studio sistematico di tale campo sia recentissimo, è possibile trovare esempi di studi simili fin dall'antica Grecia. In due scritti di Platone, *Laches* e *Symposium*, Socrate riporta un episodio della battaglia di Delio (tra Ateniesi e Beoti, nel 424 a.C.) che può essere considerato il primo esempio di problema strategico. La situazione è la seguente: si consideri un oplita al fronte, che aspetta che i suoi compagni respingano un attacco nemico. Può succedere che, qualora la difesa dei compagni abbia successo, il suo contributo personale non sia essenziale ma, se rimane in linea, rischia di essere ucciso o ferito apparentemente per niente perchè la battaglia sarà vinta senza il suo aiuto. D'altra parte, se i compagni dovessero fallire e il nemico dovesse sfondare, le possibilità di essere ucciso sarebbero ancora più alte e la sua morte sarebbe ancora più inutile poichè la battaglia è persa. Ovviamente, qualora tutti i soldati impostassero tale ragionamento (e dovrebbero farlo poichè sono tutti nella medesima situazione), fuggire incrementerebbe la loro utilità, indipendentemente da chi vincerà la battaglia, generando di fatto come esito una bruciante sconfitta. Maggiore è la paura di perdere la battaglia, maggiore sarà la voglia di fuggire ma anche se il soldato dovesse pensare che la battaglia sarà vinta senza particolari contributi individuali la strategia migliore per lui sarebbe ancora darsi alla fuga per evitare una morte inutile.

Basandoci sull'esempio di Socrate, possiamo fare altre considerazioni. L'oplita non è spinto a ritirarsi solo in base al suo ragionamento, ma anche dal fatto che ciò che è conveniente per lui dipende da ciò che è conveniente per gli altri, e che questo deve essere chiaro anche ai suoi compagni. Anche il soldato più valoroso preferirà ritirarsi piuttosto che cercare di fermare da solo l'assalto nemico. Immaginiamo una falange composta solo da soldati coraggiosissimi; seguendo la logica sopra riportata deciderebbero di gettare le sarisse e darsi alla fuga ma, essendo molto valorosi, preferirebbero rimanere, combattere e vincere. Dunque, l'interazione tra agenti ha prodotto un esito che in realtà nessuno di loro ha voluto (la fuga).

Esistono altri casi di teoria dei giochi *ante-litteram*, i più famosi dei quali si trovano nell'*Enrico V* di Shakespeare e nel *Leviatano* di Hobbes, ma l'esempio precedente è sufficiente per permettere di introdurre i concetti base della teoria dei giochi.

Un agente, per definizione, ha delle preferenze. Per modellizzare tali preferenze si sfrutta il concetto di *utilità*. La funzione di *utilità* mette in

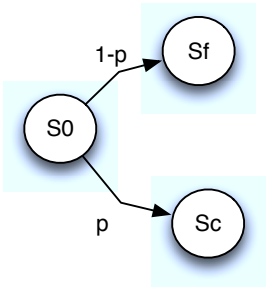

Figura 2.1: MDP dell'oplita socratico

relazione uno stato o un esito del gioco con un numero reale: più alto è il numero, maggiore è la felicità che l'agente prova nello stare in quello stato. Più formalmente, chiamato S l'insieme degli stati del mondo che l'agente i può percepire, definiamo la funzione di utilità u_i come

$$U_i : S \rightarrow \Re$$

Tornando all'esempio, possiamo semplificare dicendo che il soldato può scegliere tra due stati, il primo S_f in cui si da alla fuga e il secondo S_c in cui rimane a combattere.

L'oplita è quello che viene chiamato *agente razionale*, cioè un giocatore che cerca di massimizzare la sua utilità e dunque sceglierà S_f perchè la sua utilità in quello stato è maggiore.

Immaginiamo che il nostro oplita prima di una battaglia interroghi un aruspice per sapere come agire durante lo scontro. Tale oracolo risponde di combattere con probabilità p e di gettare lo scudo con probabilità $1 - p$. Usando tale probabilità l'oplita può calcolare la sua *utilità attesa*, ovvero l'utilità calcolata in uno stato di incertezza come la media pesata degli stati in cui può andare (fuggire o non fuggire), usando come pesi le probabilità (l'oracolo) di andare in tali stati.

Formalmente, chiamiamo A l'insieme delle possibili azioni, S l'insieme degli stati e $u_i(s)$ l'utilità dell'agente i nello stato s e possiamo scrivere

$$E[u_i, A, s] = \sum_{s' \in S} T(s, a, s')u_i(s')$$

dove $T(s, a, s')$ è chiamata *funzione di transizione* e restituisce la probabilità di andare dallo stato s allo stato s' compiendo l'azione $a \in A$.

2.1.2 Processi decisionali di Markov

Assumiamo che lo stato prossimo in cui andrà l'agente sia scelto solo in base allo stato corrente in cui si trova e all'azione che compie. Tale concetto viene efficacemente catturato dai Processi Decisionali di Markov (MDP).

Un MDP è composto da uno stato iniziale S_1 preso da un insieme di stati S, una funzione di transizione $T(s, a, s')$ e una funzione di utilità $U_i : S \to \Re$.

Chiamiamo π la *politica* dell'agente, che è una relazione tra stati ed azioni. L'obiettivo dell'agente è trovare una politica ottima che, ad esempio, gli permetta di massimizzare l'utilità attesa.

2.1.3 Giochi

Tutti i sistemi multi agente (MAS) sono composti da agenti che eseguono azioni in base alle informazioni che hanno a disposizione. Poichè anche altri agenti stanno compiendo le azioni di cui dispongono, ogni agente deve valutare anche le azioni di tutti gli altri. In sintesi, quello che fa uno dipende da quello che fanno gli altri, e viceversa.

Assumiamo di avere due agenti, che la conoscenza sia comune (entrambi hanno gli stessi dati) e che entrambi effettuino la loro azione nello stesso istante. Questo tipo di giochi è chiamato *gioco in forma normale*. Un gioco di questo tipo può essere facilmente rappresentato tramite una matrice contenente in ogni cella (i, j) i payoff ottenuti dagli agenti qualora il primo compia l'azione i e il secondo l'azione j.

Giochi a somma zero. Qualora, in ogni cella, la somma delle utilità sia 0, ovvero se un agente vince l'altro perde in proporzione alla vincita del primo agente, chiamiamo il gioco *a somma zero* (zero-sum game). Dato un gioco qualsiasi in forma normale, è naturale chiederci quale sia la strategia migliore per gli agenti, e quindi la soluzione del gioco. La soluzione a questo quesito non è semplice, poichè ciò che è bene per un agente può danneggiare l'altro. Per rispondere a questa domanda, però, dobbiamo prima chiarire il concetto di strategia.

La strategia di un giocatore è un completo piano d'azione per tutto il gioco; specifica un'azione per ogni circostanza in cui l'agente è costretto ad agire. Può essere pura, ovvero quando definisce in modo deterministico il modo in cui l'agente affronterà il gioco, o mista, quando l'agente sceglie tra diverse strategie pure basandosi su una determinata distribuzione di probabilità. La strategia di un gioco è un insieme che contiene una e una sola strategia per ogni giocatore.

Maxmin. Il primo a cercare di risolvere il problema fu, ancora una volta, Von Neumann. La sua soluzione, chiamata *maxmin*, assume che in ogni

gioco ogni agente scelga sempre l'azione che gli permette di massimizzare la minore l'utilità che può ottenere. Formalmente, in un gioco in cui agiscono due agenti j e i, la strategia maxmin dell'agente i è data da:

$$S_i^* = \max_{s_i} \min_{s_j} u_i(s_i, s_j)$$

Tale strategia può essere sempre trovata in giochi a somma-zero con due agenti.

Social welfare. Un altro comune *solution concept* è il *social welfare*, in cui si cerca di massimizzare la somma dei payoff di tutti gli agenti. Tuttavia, tale soluzione non è stabile: poichè ogni agente razionale considera solo la sua utilità, potrebbe volersi allontanare dalla soluzione per giocare una strategia che gli consenta di ottenere un payoff maggiore, danneggiando gli altri agenti.

Pareto. Per risolvere tale problema si ricorre alla *Pareto Ottimalità*: una strategia s è detta Pareto-ottimale se non esiste un'altra strategia s' tale che almeno un agente ha un'utilità maggiore in s' e nessuno ha un'utilità minore in s'; questo equivale a dire che non esiste una strategia che permetta a degli agenti di migliorare senza danneggiare gli altri. Tuttavia, benchè altamente desiderabile, la pareto ottimalità non garantisce la stabilità: un agente potrebbe avere interesse a giocare una strategia differente per ottenere una più alta utilità pur sapendo di danneggiare gli altri.

Equilibrio di Nash. La mancanza di stabilità fu risolta da John Nash, che propose una nuova definizione di equilibrio. Una strategia s è un equilibrio di Nash per tutti gli agenti i se s_i è la migliore strategia di i sotto la condizione che tutti gli agenti abbiano giocato le loro rispettive strategie in s. Più semplicemente, quando tutti gli avversari hanno giocato l'equilibrio di Nash, per l'agente la strategia migliore è giocare a sua volta l'equilibrio di Nash.

Nash ha dimostrato che tutti i giochi in forma normale hanno almeno un equilibrio di Nash ma che tale equilibrio può essere in strategie miste.

Accanto ai giochi in forma normale, esistono anche giochi in *forma estesa*. Tali giochi sono rappresentati tramite un albero, i cui rami ad ogni livello corrispondono alle diverse azioni possibili di un agente, e i payoff sono presenti solo sulle foglie.

Tale formalismo è molto utile, e piuttosto comodo, per rappresentare giochi multiagente molto complessi.

Giochi leader-follower. Nei giochi leader-follower, noti anche come giochi di Stackelberg, un agente (il leader) sceglie una strategia che viene osservata da un altro agente (il follower) prima di scegliere a sua volta la propria strategia. Il follower sceglierà, tra le sue strategie possibili, quella

che massimizza la sua utilità. In realtà per raggiungere un equilibrio, il follower non è solo un best-reponder ma, qualora sia indeciso tra due strategie, attuerà quella che massimizzerà l'utilità del leader.

Un'analisi approfondita dell'equilibrio di Stackelberg è fatta in [18] dove è dimostrato che, in giochi a somma zero, il leader-follower equilibrium coincide con l'equilibrio di Nash. Inoltre la strategia di Stackelberg, nei giochi a somma non-zero è ottima per il leader quando il follower risponde giocando la sua strategia ottimale; è evidente che il leader riuscirà ad ottenere un risultato buono almeno quanto quello ottenuto all'equilibrio di Nash, perchè di fatto sta imponendo una strategia a lui favorevole. Tuttavia, le esistenze dei due equilibri sono indipendenti, e non possiamo dedurne uno avendo trovato l'altro.

In [20] viene dimostrato che il peggior equilibrio leader-follower non è peggiore del miglior equilibrio di Nash del gioco.

In [10], viene presentato un algoritmo per il calcolo di equilibri in giochi leader-follower. In giochi con più di due giocatori, dopo che il leader ha giocato la sua strategia, il gioco può essere ridotto ad un gioco leader-follower con gli agenti rimanenti. Dunque, in un gioco *strettamente competitivo,* ovvero un gioco equivalente ad un gioco a somma zero con due giocatori [4], l'equilibrio leader-follower è indicato dal *minimax*; in giochi non strettamente competivi, invece, l'equilibrio leader-follower deve essere calcolato risolvendo un problema di programmazione lineare multi-livello che risolva tutti i sotto-giochi.

2.2 Il problema del pattugliamento

Negli ultimi anni la ricerca ha affrontato con sempre maggiori sforzi il problema del pattugliamento di un'area e di specifici obiettivi in essa contenuti.

Letteralmente, con pattugliamento intendiamo il muoversi attorno ad un'area, ad intervalli regolari, per proteggerla e sorvegliarla. Come vedremo in seguito, tale definizione è riduttiva. Le applicazioni sono notevoli, e in parte già sperimentate: difesa di uffici, di banche, pattugliamento di zone difficili da raggiungere per personale umano o in condizioni estreme, ambiti militari, difesa di accampamenti; in generale, ovunque sia necessario difendere obiettivi dall'intrusione di un ipotetico ladro.

Ad alto livello, possiamo individuare i seguenti elementi che compongono il problema:

- uno o più robot pattugliatori.
- un'area in cui tali robot si muovono.

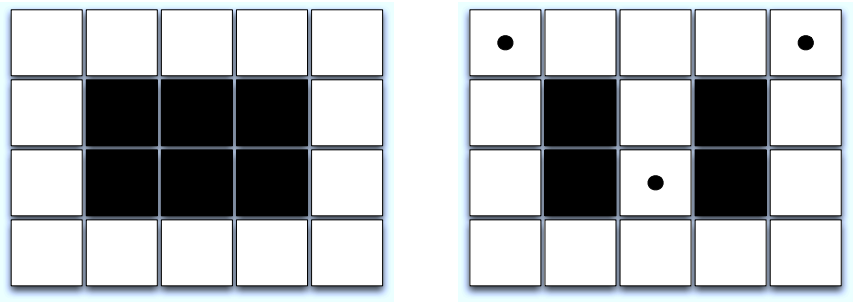

Figura 2.2: Tipi di pattugliamento

- un insieme di obiettivi che i robot devono proteggere.

La presenza degli obiettivi può fuorviare il lettore che abbia l'idea preconcetta di pattugliamento di un perimetro. E' infatti necessario distinguere due tipi di pattugliamento:

- *pattugliamento perimetrale*: uno o più robot devono impedire l'accesso ad un'area e lo fanno muovendosi sul perimetro di tale zona.

- *pattugliamento ad obiettivi*: i robot devono proteggere diversi obiettivi all'interno di un'area e può esistere una gerarchia tra questi ultimi (può essere più importante difendere un obiettivo piuttosto che un altro, qualora ci sia la necessità di scegliere).

Benchè sia possibile ricondurre il secondo tipo al primo immaginando di avere gli obiettivi posizionati in modo da poterli pattugliare in un ciclo (e il primo al secondo, immaginando che gli obiettivi siano così concentrati da poter essere difesi girando loro intorno), le due tipologie di pattugliamento presentano problematiche differenti e diverse difficoltà da affrontare, soprattutto nella costruzione del modello multiagente. Tali problematiche sono state affrontate da gruppi di ricerca, in varie parti del mondo, che hanno fornito soluzioni sfruttando approcci, spesso diversissimi tra loro, fornendo però risultati quasi sempre non all'altezza delle aspettative.

Nonostante l'indubbio interesse che il pattugliamento ricopre, il problema non è stato affrontato in modo rigoroso; spesso, si è preferito riutilizzare risultati ottenuti da studi in campi vicini che hanno trovato soluzioni safe and sound a problemi simili, che però mal si adattano al pattugliamento, che necessita di soluzioni *ad-hoc*.

2.2.1 Architettura del Problema

Dal punto di vista architetturale, ci sono vari elementi in base ai quali è possibile classificare le soluzioni.

Il primo criterio distingue i tipi di agenti in:

- *Agenti reattivi*: agiscono in base alla loro condizione istantanea, cioè alla loro percezione corrente.
- *Agenti cognitivi*: non solo agiscono in base alla loro percezione, ma possono anche perseguire uno scopo (*Goal*).

Dunque, gli agenti reattivi non sono in grado di pianificare le mosse in anticipo, non percepiscono la profondità della scelta, mentre gli agenti cognitivi sono in grado, ad esempio, di pianificare un percorso per un nodo non adiacente e distante.

Il secondo criterio riguarda il tipo di conoscenza e possiamo distinguere:

- *Architettura a conoscenza globale e scelta centralizzata*: un robot è consapevole della presenza di altri robot ed effettuano scelte concordate
- *Architettura a conoscenza globale e scelta locale*: un robot è consapevole della presenza di altri robot ma effettua scelte locali, in funzione di segnali lasciati sull'ambiente
- *Architettura a nessuna conoscenza*: un robot non è consapevole della presenza di altri agenti, agisce come se fosse solo.

Il terzo parametro specifica la tecnica utilizzata per la coordinazione tra gli agenti impegnati nel pattugliamento:

- *Flag*: segnali lasciati sul terreno dagli agenti sul modello delle tracce ormonali utilizzate dagli insetti.
- *Blackboard*: un Repository di Conoscenza comune, accessibile ad ogni agente.
- *Messaggi*: informazioni scambiate tra agente e ipotetico coordinatore e agente e agente.

Il quarto parametro è il *decision making*, la strategia con cui viene deciso il prossimo nodo da visitare di cui è necessario prendere in considerazione due aspetti. Il primo è la *visione* degli agenti, che può essere locale (ogni agente ha la sua) o globale (ogni agente percepisce ciò che percepiscono gli

altri); il secondo è il *criterio di scelta* che può essere randomico, o sfruttare euristiche su una qualche metrica.

Il quinto parametro è il fattore chiave quando parliamo di movimento in ambito multiagente e riguarda la scelta di uno scopo per agenti cognitivi, scelta che può essere fatta *globalmente* da un coordinatore o *localmente* dall'agente stesso.

Un altro importante parametro per la classificazione architetturale stabilisce se il sistema prenda o meno in considerazione il comportamento dell'intruso, quindi se sia *opponent-based* o meno. Qualora l'intruso venga considerato, è necessario distinguere le architetture in cui viene modellizzato come *a conoscenza completa* e quelle, invece, in cui viene presa in considerazione l'ipotesi che l'avversario abbia solo *conoscenza parziale* del problema [2].

Il primo tentativo di schematizzare rigorosamente non solo le diverse architettetture di pattugliamento, ma di definire alcune *metriche* per la valutazione della soluzione, è stato fatto in [15] nel 2003. Immaginiamo di avere dei robot che si muovono su un ambiente continuo, molto grande, che per semplicità riteniamo immutabile (non subisce cambiamenti tanto netti da mutare la strategia del robot). Possiamo modellizzare tale ambiente come un grafo i cui nodi sono celle di dimensione arbitraria e gli archi collegano nodi corrispondenti a celle adiacenti. Machado ha supposto, per semplicità modellistica, che le celle abbiano tutte la stessa priorità (i robot non hanno preferenze sulla cella da visitare).

L'autore individua le seguenti metriche di valutazione della bontà delle soluzioni trovate:

- idleness.
- worst idleness.
- exploration time.

Se definiamo *ciclo* il tempo necessario ad un agente per andare da un nodo (una cella) ad un nodo adiacente, possiamo chiamare *idleness istantanea del nodo* il numero di cicli in cui un nodo non viene visitato. La *idleness istantanea del grafo* è la IIN media di tutti i nodi in un dato ciclo. La *idleness del grafo* è la IIG media su n cicli.

Nello stesso contesto, è interessante valutare la *worst idleness*, ovvero il maggior valore di IIN misurata durante l'intera simulazione. L'*exploration time*, in fine, è il tempo necessario ad un agente per visitare tutti i nodi del grafo almeno una volta.

Il lavoro di Machado, che non considerava alcun tipo di decomposizione del problema, ha portato a due risultati significativi: in primis, in assenza

di coordinazione, l'introduzione di nuovi agenti non determina un aumento delle performance del sistema, ma spesso anzi peggiora la soluzione trovata; in secundis, agenti senza alcuna capacità di comunicare e che effettuano scelte puramente locali hanno performance migliori di sistemi che adottano strategie molto più complesse.

Considerazioni multiagente. E' spesso possibile immaginare il caso multiagente come robot singoli equidistanziati che si muovono sullo stesso percorso, con la medesima velocità e nello stesso senso. Maggiore è il numero dei robot, minore è la idleness di un nodo e maggiore è la frequenza con cui ogni cella viene visitata. Questo è particolarmente chiaro su un ambiente perimetrale. Come vedremo in seguito, l'equidistanza e la sincronizzazione non sono sufficienti in numerosi casi a garantire un buon pattugliamento.

Un modello multiagente come estensione di un modello singolo-agente è stata analizzato da Elmailach in [11], che sfrutta la *frequenza* come metrica: l'ambiente è suddiviso in poligoni aperti, e ogni segmento viene assegnato ad uno o più robot secondo i tre criteri *synchronized* (un robot per segmento, robot sincronizzati nel movimento) *synchronized overlap* (un robot per segmento, ma i segmenti possono avere sovrapposizioni, robot sincronizzati nel movimento) e *unsynchronized* (un robot per segmento, nessuna sincronizzazione). L'approccio sincronizzato con overlapping, che di fatto comporta una sorta di comunicazione implicita, si rivela, naturalmente il migliore, benchè un'errata scelta delle linee spezzate ne pregiudichi le performance.

Le prossime sezioni riporteranno i principali approcci al problema del pattugliamento suddividendoli in base alle tecniche utilizzate, mostrandone i risultati relazionati agli obiettivi dichiarati.

2.3 Approcci al pattugliamento

Informalmente, una buona strategia è quella che minimizza il tempo tra due successivi passaggi su uno stesso nodo per tutti i nodi di cui riteniamo composto l'ambiente. La seguente sezione riporta i principali approcci al problema, distinguendo tra approcci di tipo *Non Strategico*, basati spesso su information covering e tecniche mutuate da altre discipline, e approcci di tipo *Strategico*, dove si cerca di modellizzare l'intruso utilizzando la Teoria dei Giochi.

2.3.1 Approcci non strategici

Gli approcci strategici si rifanno pesantemente a tecniche prese da diversi ambiti di ricerca, tra cui:

- *Ricerca operativa*, in particolar modo il problema del Commesso Viaggiatore (TSP).
- *NL-MAS*: sistemi multiagente non apprendenti (non learning MAS) che sfruttano euristiche.
- *L-MAS con reinforcement learning*, che modificano continuamente la loro strategia in base ad input sensoriali, modellizzati tramite catene di Markov
- *Bargaining*: la teoria delle aste viene sfruttata per cercare di dividere il grafo in sottografi.
- *Bio-Inspired*: basati sull'emulazione di comportamenti naturali.

Sfruttando, in parte, la classificazione e i parametri introdotta nella sezione precedente, nel 2004 Almeida [3] ha proposto quattro possibili classi in base alle tecniche utilizzate.

La prima classe è quella composta da agenti che seguono un approccio basato sul problema del commesso viaggiatore, noto problema di ricerca operativa. Dato un grafo completamente connesso in cui ogni nodo rappresenta una citta e ogni arco pesato la distanza tra le due città, il problema consiste nel trovare il percorso che porti a visitare una città una e una sola volta. I *TSP-based single cycle approach* cercano un percorso che permetta ad un singolo agente di visitare ogni nodo almeno una volta e di tornare al nodo di partenza. Tale percorso è trovato risolvendo all'ottimo, in termini di *worst idleness* il TSP. Nel caso di più agenti, il problema è risolto facendo muovere i robot in sequenza, sincronizzati, in modo da mantenere tra loro una distanza costante. Tale approccio non è però scalabile, per via della complessità del TSP e non può essere utilizzato qualora gli obiettivi abbiano priorità differenti. Tuttavia, in ambienti piccoli, ha ottime prestazioni. Nell'ambito di questa classe possiamo collocare [9], che non solo ha studiato analiticamente la complessità del problema, ma ha anche messo a confronto la soluzione trovata risolvendo il TSP con una basata sul partizionamento del grafo scoprendo che, tranne in casi di cammini molto lunghi, la risoluzione del TSP è preferibile.

La seconda classe è composta da agenti che utilizzano una qualche euristica nella scelta del nodo da visitare e vengono chiamati *agenti euristici*. Ad esempio, la scelta del nodo può essere guidata da una funzione di utilità che prenda in considerazione un costo (la distanza di un nodo dal nodo in cui è l'agente) e una ricompensa (la idleness). La qualità della soluzione è

influenzata dalla qualità dell'euristica utilizzata, e in generale non è molto scalabile per via della complessità dell'algoritmo di ricerca del percorso migliore (generalmente Floyd-Warshall).

La terza classe proposta è quella di agenti che sfruttano la teoria delle aste per scegliere come muoversi. Ogni agente riceve un set casuale di nodi del grafo, tendenzialmente molto vicini tra di loro. Quando un agente individua nel set un nodo che non può visitare in un tempo ragionevole, lo mette all'asta. Tutti gli altri agenti cercano nei rispettivi set un nodo che può essere scambiato con il nodo all'asta e fanno l'eventuale offerta; la migliore offerta è scelta, e viene effettuato lo scambio. Vengono proposti diversi algoritmi, in base al tipo d'asta utilizzato, ma la loro analisi esula dall'obiettivo di questa tesi. Le performace di agenti che sfruttano questo approccio sono molto scarse, se non in casi molto particolari.

La quarta classe è composta da sistemi che utilizzano tecniche di apprendimento con rinforzo (RL). Per permettere agli agenti di modificare le loro strategie tramite RL, vengono utilizzati i processi di Markov (MDP), formalismo già approfondito nelle sezioni precedenti. Tale approccio prevede lo sviluppo di un appropriato sistema di *reward* che possa portare ad un'adeguata ricompensa sul lungo termine. Gli agenti vengono poi istruiti tramite algoritmi standard, come il Q-Learning. Tale approccio garantisce ottime performance e una grande scalabilità; tuttavia, in caso di ambienti molto grandi (e quindi molti nodi), il costo per l'addestramento può essere considerato troppo elevato.

E' possibile, ormai, aggiungere una nuova classe, quella composta da algoritmi *bio inspired*,che si rifanno in genere al comportamento delle formiche. Un'analisi esaustiva viene proposta in [13] dove vengono riportati due algoritmi ant-based che utilizzano feromone digitale, ovvero un valore assegnato ad una cella che decresce con il tempo e viene riportato al massimo da un agente che visita il nodo. E' dimostrato che tali algoritmi portano sistematicamente al partizionamento dell'ambiente in cicli (anche sovrapposti), ma rimangono insolute molte domande sul reale funzionamento dell'algoritmo, in particolar modo su grandi ambienti.

2.4 Approcci strategici

Finora abbiamo considerato come buoni criteri la idleness e la frequenza con cui una cella viene visitata. Ma l'approccio strategico e il conseguente studio di un agente intenzionato a penetrare nella zona pattugliata rendono

Approccio	Campo	Vantaggi	Svantaggi
TSP-SC	Ricerca Operativa	Ottime prestazioni in ambienti piccoli	Non è scalabile
Euristica	NL-MAS	Buone prestazioni, ma influenzate dalla scelta dell'euristica	Non è scalabile
Aste	Bargaining	Ottimi risultati in ambienti composti da aree poco connesse tra loro	Performance in genere molto scarse
Apprendenti	RL-MAS	Ottime performance e grande scalabilità	Costo eccessivo per il Training
Ant-Based	Bio-Inspired	Divisione dell'ambiente in cicli. Validi su ambienti dinamici	In fase di studio

Tabella 2.1: Tabella riassuntiva degli approcci

molto meno significative tali metriche. Questo è dovuto all'introduzione di *tempi di penetrazione*, ovvero del tempo che impiega l'intrusore a penetrare un obiettivo (o un nodo del grafo, o un poligono). Dunque, è più importante che tra due successivi passaggi sulla medesima cella intercorra meno tempo del penetration time proprio di quella cella.

2.4.1 AK

Il primo modello basato sulla teoria dei giochi è stato presentato in [1] da Agmon e Kaminka e si basa su [11]. Agmon affronta il problema del pattugliamento muli-agente in un ambiente perimetrale, dove i robot si muovono attorno ad un'area chiusa. Pur con qualche limite, il lavoro introduce numerose novità rispetto alle strategie viste nei precedenti capitoli, prime tra tutti la modellizzazione dell'avversario e l'utilizzo del *penetration time*.

Agmon abbandona le strategie deterministiche: è facile provare che in moltissimi ambienti (da ora chiamati anche *setting*)un intruso può espugnare una cella con probabilità 1 qualora conosca il percorso che il pattugliatore seguirà.

Immaginiamo di avere k robot, omogenei quel tanto che basta per essere considerati uguali tra di loro dal punto di vista modellistico, che si muovono su una linea spezzata aperta. Tale linea è divisa in N segmenti, di lunghezze anche diverse, che possono essere percorsi da un robot in un *time cicle*. Poi

dividiamo gli N segmenti in sezioni, ognuna contenente $d = N/k$ segmenti. Ad ogni ciclo, i pattugliatori, che sono sincronizzati (si muovono nello stesso senso, contemporaneamente), devono decidere dove andare, se proseguire oppure se voltarsi e tornare indietro.

L'algoritmo proposto si basa sulla probabilità p che i robot continuino per la loro strada e $q = 1 - p$ che decidano di voltarsi. L'intrusore, invece, deve decidere all'istante 0 in quale cella penetrare, e impiegherà t turni per riuscire a scassinare l'obiettivo. La scelta di studiare il pattugliamento su linee spezzate aperte complica notevolmente il problema rispetto agli studi classici in cui i pattugliatori si muovono su percorsi circolari (come nel TSP-based approach). Infatti, il tempo intercorso tra due successivi passaggi sullo stesso obiettivo può essere anche doppio su una spezzata aperta rispetto a quello necessario su un ciclo, questo perchè il robot deve fermarsi, girarsi e ripercorrere il tratto percorso in precedenza, mentre in un ambiente circolare potrebbe semplicemente proseguire. Inoltre, in un ambiente circolare, considerata la sincronizzazione dei robot, è sufficiente disporli simmetricamente per assicurare la copertura.

L'obiettivo dichiarato è massimizzare la Probability of Penetration Detection (PPD) definita come la probabilità di visitare un segmento s_i per la prima volta in t turni, dove t è il tempo di penetrazione. Quando questa probabilità è 1, il sistema ha la certezza di visitare ogni segmento ogni $r \leq t$ turni, individuando sempre l'intruso.

Una volta calcolato il PPD di tutti i segmenti, possiamo scegliere quale modello utilizzare per l'avversario. Quando decidiamo di utilizzare un modello *forte* massimizzando la mimina ppd, ci troviamo in un gioco *a conoscenza completa*, in cui l'avversario ha tutte le informazioni necessarie per fare la sua mossa. Quando, invece, decidiamo di massimizzare la PPD attesa (modello *debole*), ci troviamo in un gioco *a conoscenza incompleta* e l'avversario sceglierà casualmente, con probabilità uniforme, in quale segmento fare breccia.

Le limitazioni del modello di Agmon sono evidenti: in primis, tutti i segmenti hanno lo stesso penetration time, di fatto ignorando qualsiasi gerarchia tra obiettivi; poi la soluzione multiagente è, banalmente, un'estensione del modello singolo agente con robot sincronizzati ed equidistanti che si muovono su un perimetro.

Procediamo per punti: immaginiamo che i penetration time siano differenti, e che quindi il patroller abbia delle preferenze sugli obiettivi. In questo caso, l'equidistanza è limitativa poichè il mantenimento di tale condizione spesso impedisce il raggiungimento di un obiettivo entro il tempo massimo concesso. In figura 2.3, se R_1 si muove verso t_1, R_2 è costretto a muoversi

verso t_2 ma non potrà mai raggiungerlo in tempo. Un discorso analogo è va-

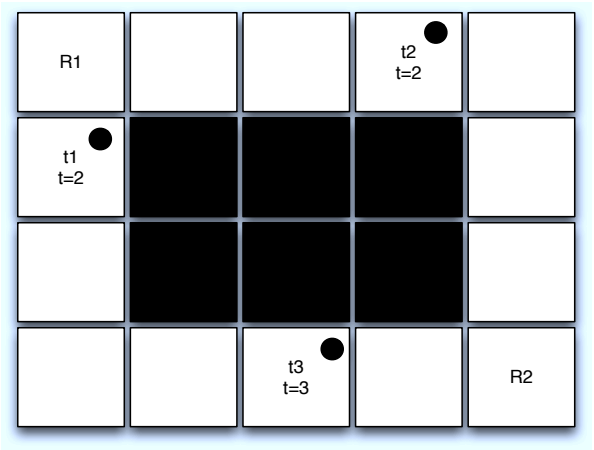

Figura 2.3: Limitazioni introdotte dall'equidistanza

lido anche per la sincronizzazione del movimento, che impedisce ad un robot di stare fermo mentre gli altri si muovono. In figura 2.4, R_1, spostandosi su t_2 costringe R_2 a lasciare scoperto t_3, che in più non potrebbe raggiungere nessuno dei due obiettivi, che altrimenti verrebbero lasciati incustoditi.

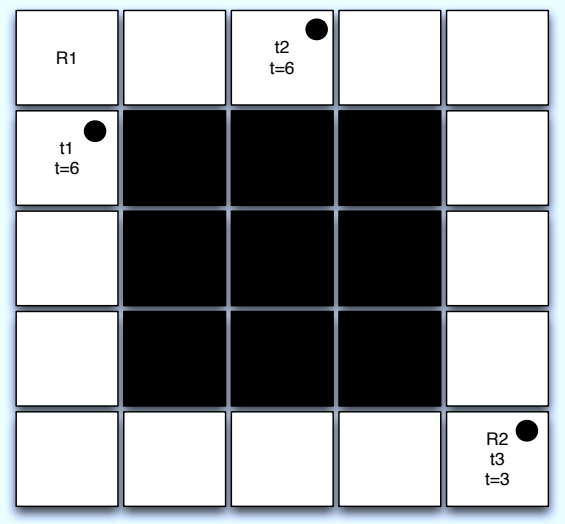

Figura 2.4: Limitazioni introdotte dalla sincronizzazione del movimento

2.4.2 BGA

Il secondo modello strategico trattato è stato sviluppato al Politecnico di Milano [5], è esclusivamente singolo agente ma, a differenza del modello precedente, può essere applicato ad ambienti di topologia arbitraria.

Immaginiamo che l'agente che voglia introdursi nell'ambiente (da ora chiamato intrusore, o intruder) abbia la possibilità di aspettare indefinitamente, nascosto, osservando il movimento del pattugliatore e che scelga la sua strategia in base a quella del patroller. Tale situazione riporta alla definizione di leader-follower vista in precedenza, dove il patroller è il leader, e l'intruso è il follower. L'obiettivo dichiarato dagli autori è trovare una strategia ottima di pattugliamento che induca l'intruso a non penetrare mai nell'ambiente

Assunzioni. Il modello proposto ha le seguenti caratteristiche:

- *tempo discreto*, misurato in turni.
- *Agente singolo* in grado di percepire un intruso.
- *Ambiente* discretizzato in celle, e la sua topologia è rappresentata tramite un grafo orientato.
- *Penetration time*, dipendenti da ogni singolo obiettivo.

Modello. L'ambiente è composto da un insieme C di n celle, la cui topologia è data da un grafo orientato G. Rappresentiamo tale grafo com con una matrice $T(n \times n)$ dove $t_{i,j} = 1$ indica l'adiacenza tra le cella i e la cella j. Per ogni cella i vale $d_i \geq 0$ dove d_i è il tempo necessario all'intruso per penetrare nella cella i.

Il movimento del pattugliatore si basa sull'assunzione che in un singolo turno possa passare solo da una cella ad una adiacente; tale assunzione non è limitativa poichè è comunque possibile estendere il formalismo in modo da considerare situazioni più complesse, come ad esempio il pattinamento su superfici scivolose o il tempo necessario ad effettuare l'inversione del movimento.

Il sensore del robot è modellizzato tramite una matrice di vista $V(n \times n)$ dove $v_{i,j} = 1$ indica che il patroller è in grado di percepire l'intruso in j quando il robot si trova in i. Il formalismo è facilmente estensibile, e può rappresentare l'incertezza del sensore immagazzinando in V le probabilità di sensing.

Il modello sfrutta un gioco in forma estesa, più precisamente un *gioco ripetuto*: ad ogni turno, si ripete un gioco in cui entrambi i giocatori agiscono contemporaneamente. Il pattugliatore sceglie la cella in cui muoversi

tra quelle direttamente connesse alla cella in cui si trova tramite l'azione *move(j)*. L'intruder, qualora non abbia ancora deciso se tentare di entrare o meno, può scegliere tra penetrare in una cella (*enter(j)*) o aspettare (*wait*).

Inoltre, il gioco possiede altre due caratteristiche: è a *informazione imperfetta* poichè il patroller non può sapere se l'intruso è in attesa di entrare o se sta forzando una cella, ed è ad *orizzone infinito*, poichè, virtualmente, l'intruso può decidere di aspettare indefinitamente.

Esistono due possibili esiti:

- *intruder_capture* quando l'intruso cerca di penetrare in una cella e il patroller percepisce quella cella in un numero di turni inferiore al tempo di penetrazione dell'obiettivo dal momento dell'attacco
- *no-attack* quando l'avversario decide di non entrare mai

Per quanto riguarda i *payoff*, indichiamo con X_0 e Y_0 le utilità rispettivamente del pattugliatore e dell'intruso qualora il gioco termini con un *intruder_capture*. Qualora, invece, il gioco termini con un *no-attack*, l'utilità dell'intruder è 0.

Solution concept. Il *solution concept* più adatto per un gioco in forma estesa ad informazione imperfetta è l' *equilibrio sequenziale*: una variante dell'equilibrio di Nash che specifica non solo una strategia per ogni agente, ma anche un *belief* (traducibile con *credenza*) per ogni giocatore. Il *belief* assegna ad ogni insieme di azioni di quel giocatore una probabilità e l'insieme di una strategia e di una credenza formano un assessment. In un equilibrio sequenziale, le strategie sono sicuramente razionali (massimizzano l'utilità) e le credenze sono consistenti con le strategie ottime degli agenti.

La presenza dell'orizzonte infinito complica lo studio del gioco. Gli autori propongono l'uso di una storia H, definita come la sequenza delle ultime $|H|$ azioni del pattugliatore. Definiamo H anche come l'insieme delle ultime $|H|$ celle visitate dal robot. Ovviamente, poichè il gioco è a orizzonte infinito, tale insieme può avere cardinalità infinita.

Introducendo simmetrie di gioco, l'azione da compiere è decisa in base alle ultime $|H|$ azioni, con $|H|$ costante durante tutto il gioco. Ad esempio, quando $|H| = 0$, la cella in cui muoversi non dipende dalla cella in cui si trova il pattugliatore; con $|H| = 1$, la strategia è Markoviana.

Maggiore è la cardinalità dell'insieme H, maggiore è la complessità computazionale per trovare una strategia. Fissato $|H|$, tuttavia, il gioco si riduce ad un gioco in forma strategica che si ripete ogni $|H|$ turni. Le azioni per il patroller, in questo nuovo gioco, possono essere scritte nella forma $\alpha_{H,move(j)}$, cioè la probabilità di compiere l'azione *move(j)* data una storia $|H|$.

Le azioni dell'intruder, invece, possono essere ridotte a *enter_when(H,j)*, ovvero penetrare nella cella j dopo che il patroller ha compiuto una serie di azioni H, e *stay_out*.

In tale formulazione, il *solution concept* più appropriato è il *leader follower equilibrium* e dunque il leader (il pattugliator), non può ottenere un'utilità attesa inferiore a quella che otterrebbe da un equilibrio sequenziale.

Formulazione Matematica

Gli autori forniscono la formulazione matematica nel caso in cui $|H| = 1$, quindi sotto ipotesi Markoviane, e denotano con $\{\alpha_{i,j}\}$ la probabilità che il pattugliatore si sposti dalla cella i alla cella j e con $\gamma_{i,j}^{h,w}$ la probabilità che il pattugliatore raggiunga la cella j partendo dalla cella i in h passi senza passare per la cella w. Tutti i penetration time vengono rappresentati con d.

L'algoritmo si sviluppa in due passi. Il primo step cerca se esiste almeno una strategia del pattugliatore (il leader) a cui l'intruso (il follower) risponderebbe con *stay_out*. Qualora esista, l'utilità del robot è massima. Il problema ha la seguente formulazione:

$$\alpha_{i,j} \geq 0 \qquad \forall i,j \in C \tag{2.1}$$

$$\sum_{j \in C} \alpha_{i,j} = 1 \qquad \forall i \in C \tag{2.2}$$

$$\alpha_{i,j} \leq t_{i,j} \qquad \forall i,j \in C \tag{2.3}$$

$$\gamma_{i,j}^{1,w} = \alpha_{i,j} \qquad \forall w,i,j \in C, j \neq w \tag{2.4}$$

$$\gamma_{i,j}^{h,w} = \sum_{x \in C \setminus w} \left(\gamma_{i,x}^{h-1,w} \alpha_{x,j} \right) \qquad \begin{array}{r} \forall h \in \{2,\dots,d\}, \\ \forall w,i,j \in C, j \neq w \end{array} \tag{2.5}$$

$$Y_0 \left(1 - \sum_{i \in C \setminus w} \gamma_{z,i}^{d,w} \right) + Y_w \sum_{i \in C \setminus w} \gamma_{z,i}^{d,w} \leq 0 \qquad \forall z,w \in C \tag{2.6}$$

I vincoli (1.1)-(1.2) impongono che le probabilità $\alpha_{i,j}$ siano ben formate; il vincolo (1.3) prescrive che il pattugliatore possa muoversi solo tra celle adiacenti; i vincoli (1.4)-(1.5) impongono la Markovianità sulle decisioni del patroller; il vincolo (1.6) dice nessuna azione *enter-when*(z,w) assicura all'intruder un'utilità attesa migliore di quella di una *stay-out*. Se il problema ammette soluzione, la sua soluzione è un set di probabilità $\alpha_{i,j}$ che definisce una possibile strategia del patroller.

Quando il problema è *unfeasible*, ovvero non ha soluzione, si procede al secondo passo che ha l'obiettivo di trovare la *best response* dell'intruder che

massimizzi l'utilità attesa del pattugliatore. Il problema di programmazione matematica è formulata come segue.

$$\max \qquad X_q \sum_{i \in C \setminus q} \gamma_{s,i}^{d,q} + X_0 \left(1 - \sum_{i \in C \setminus q} \gamma_{s,i}^{d,q} \right)$$

s.t.

$$\text{constraints (1)-(5)}$$

$$\begin{aligned} & Y_0 \left(1 - \sum_{i \in C \setminus q} \gamma_{s,i}^{d,q} \right) + Y_q \sum_{i \in C \setminus q} \gamma_{s,i}^{d,q} \geq \\ & \geq Y_0 \left(1 - \sum_{i \in C \setminus w} \gamma_{z,i}^{d,w} \right) + Y_w \sum_{i \in C \setminus w} \gamma_{z,i}^{d,w} \end{aligned} \qquad \forall z, w \in C \qquad (2.7)$$

La funzione obiettivo massimizza l'utilità attesa del patroller. Il vincolo (1.7) dice che nessuna azione *enter-when*(z, w) assicura un valore migliore all'intruder di quello di un'azione *enter-when*(s, q).

Estensioni e miglioramenti. Gli autori propongono anche un metodo semplificato che si basa sull'idea che, in un ambiente realistico, il numero di obiettivo (quindi di celle con una qualche importanza per il patroller) sia molto più piccolo del numero totale di celle. Tali celle più importanti compongono l'insieme T dei obiettivo veri e propri.

Il primo step dell'algoritmo prevede la ricerca di una strategia che permetta al robot di intercettare l'intruso con probabilità 1, dunque una strategia deterministica. Il calcolo di tale strategia avviene ricercando tutti i cammini minimi che coprono tutti i obiettivo. Poi l'algoritmo controlla che nessun obiettivo sia lasciato incustodito per più di d turni. Se esiste un cammino che passa per tutti i obiettivo e che non ne lascia esposto nessuno per più di d turni, allora esiste una strategia deterministica che assicura al pattugliatore la cattura dell'avversario. Tale strategia può essere efficacemente rappresentata come un insieme di cammini minimi, ognuno dei quali congiunge due soli obiettivo e individuati utilizzando l'algoritmo di Dijkstra.

E' facile intuire che alcune celle non verranno mai pattugliate all'equilibrio (perchè non appartenenti a nessuno dei cammini scelti dal patroller) e quindi possono essere eliminate dal problema che viene ridotto.

Dominanze. Tale riduzione del gioco, tuttavia, non è sufficiente ad ottenere risultati soddisfacenti in ambienti molto grandi. Gli autori, dunque, introducono il concetto di *Strategia Dominata*. Entrambi i giocatori, patroller e intruder, possono avere strategia dominate.

Rimuovere una strategia dominata del pattugliatore vuol dire, sostanzialmente, imporre il valore 0 ad alcune probabilità $\alpha_{i,j}$ se, indipendentemente dalla strategia del robot, la sua utilità attesa non peggiora. Questo equivale a rimuovere dal grafo dell'ambiente singoli archi o singoli vertici con gli archi correlati.

Gli autori propongo due teoremi per individuare un'azione dominata: visitare un vertice che non è sullo *shortest path* tra due obiettivo è un'azione dominata e assegnare una probabilità $\alpha_{i,i}$ con $i \in V \backslash T$ è un'azione dominata. E' possibile reperire la dimostrazione formale in [6].

Una azione *enter_when(H_1, i)*, dal punto di vista dell'intrusore, è dominata quando esiste un'altra azione *enter_when(H_2, j)* tale che $EU_i($ *enter_when(H_i, i)* $\leq EU_i($ *enter_when(H_2, j)*$)$. Informalmente, questo equivale a dire che, data una storia H, l'intrusore non sceglierà mai di attaccare una cella quando sa che ne esiste un'altra cella che gli garantisce un'utilità attesa maggiore. La ricerca di tali azioni dominate richiede la risoluzione di un problema di ottimizzazione, e si rimanda sempre a [6] per la formalizzazione del problema.

I prossimi capitoli hanno come obiettivo il presentare una soluzione multiagente basata sul modello BGA.

Capitolo 3

Il problema multiagente

Il modello BGA multiagente prevede il concetto di #configurazione che consente di indicare l'upperbound della dimensione del problema

Il seguente capitolo ha il compito di introdurre la naturale estensione del modello BGA in ambito multiagente, estensione che prevede una modifica sostanziale dei formalismi visti alla fine del capitolo precedente.

3.1 Modello multiagente

Immaginiamo di avere un numero $|R|$ di robot, tutti con le medesime caratteristiche. Possiamo caratterizzare tale generico problema multiagente attraverso un grafo orientato $G = (V, a, T, v, d, R)$. Vediamo gli elementi costituenti in dettaglio:

- V è l'insieme delle celle, i vertici, che devono essere pattugliate.
- a è la funzione *Successore* di una cella, ed enumera tutte le celle adiacenti ad un dato vertice. Formalmente $a : V \times V \rightarrow \{0, 1\}$ dove $a(i, j) = 1$ indica che le due celle sono adiacenti, $a(i, j) = 0$ indica che non lo sono. Una cella è considerata adiacente a se stessa, per catturare la possibilità che un robot rimanga fermo.
- $T \subseteq V$ è l'insieme contenente gli obiettivi.
- $v : T \rightarrow \Re$ indica le utilità del patroller sui singoli obiettivi. Assumiamo che intruder e patroller compiano le medesime valutazioni.
- $d : T \rightarrow \mathbb{N}\backslash\{0\}$: è l'insieme dei penetration time dei target

- R: è l'insieme dei robot.

3.1.1 Sensing dei robot

Nel modello BGA il sensore del robot è modellizzato tramite una matrice di vista $V(n \times n)$ dove $v_{i,j} = 1$ indica che il patroller è in grado di percepire l'intruso in j quando il robot si trova in i. In ambito multiagente, dobbiamo fare qualche considerazione in più.

Chiamiamo $P_{ij,m}$ la probabilità che il robot m veda la cella j dalla posizione i. Possiamo rappresentare l'insieme di tali probabilità come una matrice tridimensionale $n \times n \times |R|$, con n numero di celle del set.

L'obiettivo è trovare la probabilità che almeno uno degli $|R|$ robot individui l'intruso. La probabilità che nessuno dei patroller veda la cella (e identifichi l'intruso) è dunque:

$$\psi = \prod_{k=1}^{|R|} 1 - P_{ij,k} \tag{3.1}$$

Quindi, la probabilità che almeno uno dei robot veda l'intruso è:

$$\Delta = 1 - \prod_{k=1}^{|R|} 1 - P_{ij,k} \tag{3.2}$$

Tale nuova formulazione permette di rappresentare facilmente l'incertezza del sensore, esattamente come il modello originale.

3.2 Configurazioni

Dati $|R|$ robot, definiamo *configurazione* un vettore di cardinalità $|R|$ i cui elementi sono le celle in cui si trovano i robot. Più formalmente:

$$Sc =< c_k, ..., c_{|R|} > \forall c_k \in V, 1 \leq k < |R| \tag{3.3}$$

E' inoltre necessario che le celle di una configurazione siano diverse tra di loro al fine di catturare il fatto che due robot non possano stare contemporaneamente nella stessa cella.

3.2.1 Generazione delle configurazioni

Data una configurazione iniziale $c =< c_k ... c_{|R|} > \forall c_k \in V$ e $1 \leq k < |R|$, le configurazioni seguenti sono generate dal prodotto cartesiano di tutti gli insiemi A_k generati dalle funzioni $a(c_k)$. Se immaginiamo c come radice di

un generico sottoalbero, i suoi figli sono tutte le possibili m-uple (insiemi) formate dagli elementi contenuti negli insiemi A_k forniti dalla funzione a invocata sulle diverse celle c_k appartenenti a c.

Più formalmente, l'insieme delle configurazioni successive alla configurazione c è definito come:

$$A_1 \times ... \times A_m = \{< c_k ... c_{|R|} >: c_k \in A_k, ..., c_{|R|} \in A_{|R|}, 1 \leq k < |R|\} \quad (3.4)$$

Indichiamo con C l'insieme di tutte le configurazioni possibili di un gioco e, analogamente a quanto avviene con i vertici, $a(c, c') = 1$ indica l'adiacenza tra due configurazioni.

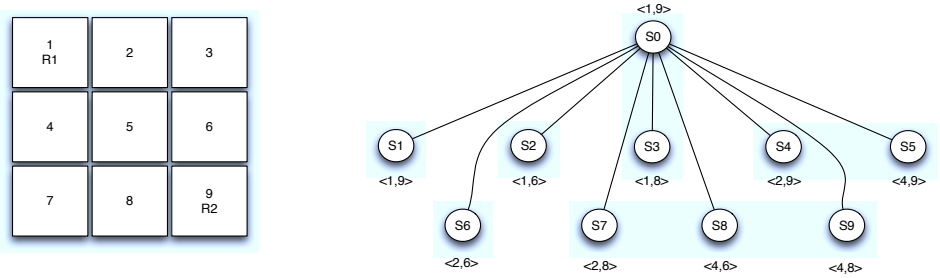

Figura 3.1: configurazioni successive a 1,9

In Figura 3.1, a destra, possiamo vedere tutte le configurazioni successive alla configurazione $S_0 =< 1, 9 >$ generate attraverso il prodotto cartesiano tra gli insiemi $N(1) = \{1, 2, 4\}$ e $N(9) = \{6, 8, 9\}$.

3.2.2 Lo spazio delle configurazioni

Lo spazio delle configurazioni varia in base alla rappresentazione scelta:

- *Vettore ordinato*: l'insieme delle possibili combinazioni ha cardinalità $n^{|R|}$.
- *Insieme*: le possibili combinazioni, prive di ripetizioni, sono $\binom{n}{|R|}$.

La rappresentazione ad insieme, che esclude le ripetizioni, è valida solo sotto l'ipotesi che i robot siano identici (anche dal punto di vista del sensing).

Tali cardinalità teoriche sono solo una stima delle cardinalità effettive, che dipendono dallo specifico ambiente e dagli obiettivi, e possono essere considerate come *upper bound*. Il numero delle configurazioni reale sarà molto probabilmente inferiore, una volta eliminate tutte le configurazioni *non ammissibili* (*unfeasible*). Una configurazione non ammissibile è uno stato in cui almeno uno degli obiettivi è irraggiungibile da qualunque robot in

un numero di turni inferiore o al più uguale al suo penetration time. Questo vuol dire che, per ogni obiettivo t con penetration time $d(t)$ deve esistere almeno un cammino di lunghezza $l \leq d(t)$ che lo colleghi ad almeno un robot. Il capitolo seguente tratterà in dettaglio come sia possibile sfruttare questa proprietà per stabilire un *lower bound* sul numero di pattugliatori da utilizzare.

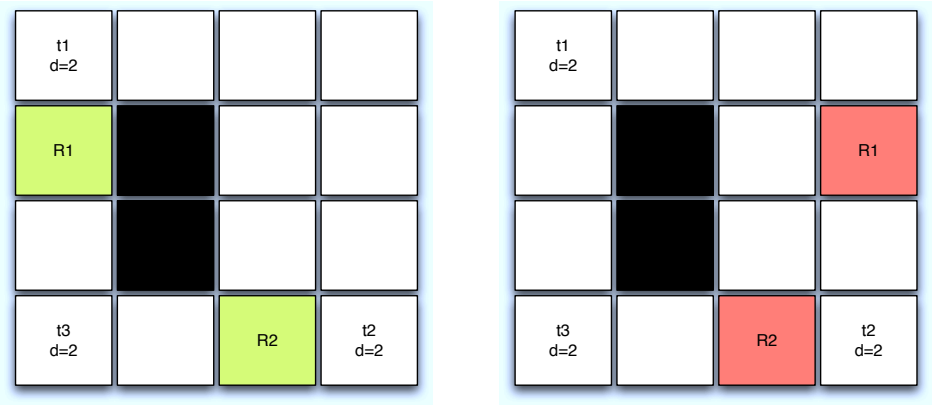

Figura 3.2: configurazione ammissibile e non ammissibile

In Figura 3.2 riportiamo un esempio di configurazione ammissibile (a sinistra) e uno di configurazione non ammissibile (a destra). E' immediato notare che, nel secondo caso, non esiste un cammino in grado di collegare t_1 ad almeno uno dei pattugliatori in al più $d(t_1) = 2$.

3.3 Il gioco

Il gioco rimane sostanzialmente uguale ma, come conseguenza dell'introduzione delle configurazioni, lo modifichiamo nelle azioni e nei possibili esiti.

3.3.1 Azioni

Assumendo che il gioco sia Markoviano, i robot R hanno l'unica azione possibile *move*(c, c') con $c, c' \in C$ e $a(c, c') = 1$. Possono dunque muoversi da una configurazione a una adiacente analogamente a quanto avviene nel modello singolo agente con le celle. Indichiamo con $\alpha\{c, c'\}$ la probabilità di passare da una configurazione c alla sua successiva c'. Tali probabilità sono le variabili del nostro problema.

L'intruso, ha a disposizione due azioni: *wait()* che gli permette di aspettare indefinitamente fuori dal set e *enter_when(t,c)* che gli permette di aspettare fuori dal set che i robot assumano la configurazione c e per poi attaccare l'obiettivo t.

3.3.2 Esiti

Il gioco ha due possibili esiti:

- *intruder_capture*: l'intruso tenta di entrare in un obiettivo t al turno k e almeno uno dei robot controlla la cella t entro il turno $k + d(t) - 1$. Dunque, l'intruso non riesce a forzare l'obiettivo prima di essere individuato
- *penetration_t*: l'intruso penetra in t al turno k e nessun robot controlla l'obiettivo al turno $k + d(t) - 1$. L'intruso, dunque, vince.

Il modello presentato non contempla alcun tipo di scomposizione del problema e si basa sull'assunzione di avere a disposizione un numero $|R|$ di robot che può non essere ottimale. Nel prossimo capitolo illustreremo come sia possibile, partendo dalle considerazioni appena viste, calcolare il numero minimo di robot necessari per pattugliare un ambiente di topologia arbitraria e come tale ambiente possa essere diviso.

Capitolo 4

La scomposizione del problema

La divisione dell'ambiente in clique etichettate massime (set di obiettivi difesi da 1 solo agente) determina il LB sul numero di robot

In questo capitolo vedremo come sia possibile stabilire a priori quale sia il numero minimo di robot non coordinati necessari a garantire di catturare l'intruso con probabilità 1. A tale risultato, accademicamente rilevante, arriveremo scomponendo il problema in sottoproblemi indipendenti. Il capitolo è strutturato facendo sì che, ad una esposizione informale ma chiara di un concetto, faccia seguito la sua formalizzazione matematica.

4.1 Dalle configurazioni non ammissibili alle cliques

Nel capitolo precedente abbiamo visto come sia possibile ridurre il numero delle configurazioni di un gioco, escludendo quelle *non ammissibili*. Facendo considerazioni simili, siamo in grado di individuare l'insieme dei cammini validi che uniscono due generici obiettivi e conseguentemente un insieme di obiettivi (e dei cammini che li uniscono) con caratteristiche tali da poter essere pattugliati da un solo robot.

Immaginiamo di avere due obiettivi, t_1 e t_2, con penetration time arbitrari $d()$ e che esista almeno un cammino, non necessariamente minimo, di lunghezza $s \leq min(d(t_i), d(t_2))$, che collega i due target. Informalmente, se tale insieme di cammini esiste, il pattugliatore può raggiungere entrambi

gli obiettivi da qualunque punto del percorso in un tempo inferiore al penetration time dell'obiettivo destinazione. Possiamo dunque dire che tale percorso ha *etichetta* $l_p = \{t_1, t_2\}$, e l'etichetta è l'insieme degli obiettivi *coperti* dal percorso p.

Immaginiamo di inserire nel nostro setting un nuovo obiettivo, t_3, e che sia possibile raggiungere t_3 in un numero di passi inferiore a $d(t_3)$ da ogni cella appartenente al percorso p individuato precedentemente. Equivalentemente, possiamo dire che t_3 non è mai indifeso mentre un robot si muove da t_1 a t_2 e che $l_p = \{t_1, t_2, t_3\}$.

Ma non possiamo ancora dire se, muovendoci da t_3 a t_2, sia possibile coprire t_1 e se, muovendoci da t_3 a t_1, sia possibile coprire t_2. In Figura 4.1, a sinistra, non è possibile andare da t_1 a t_3 senza lasciare indifeso t_2 passando nelle celle in rosso (e analogamente, non è possibile andare da t_2 a t_3 senza lasciare indifeso t_1 nel passare nelle celle in giallo), e possiamo pattugliare al massimo due obiettivi con un singolo robot. Nell'immagine a destra, invece, il pattugliatore può andare da un obiettivo ad un altro senza lasciare mai indifeso il terzo, e quindi l'insieme cercato comprende tutti e 3 gli obiettivi.

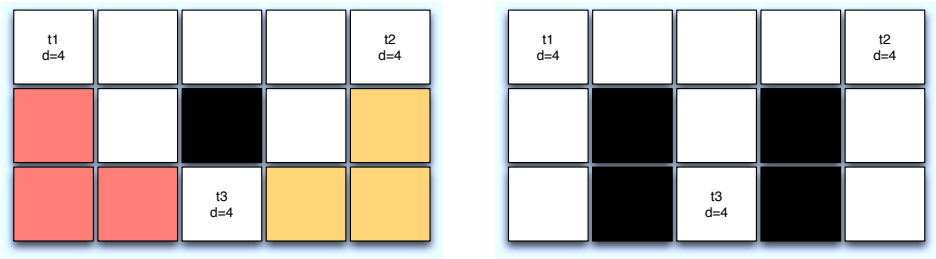

Figura 4.1: Esempio introduttivo I - Cammini validi

4.1.1 La formalizzazione del problema

Condizioni così complesse, sia in termini espressivi che computazionali, hanno richiesto l'elaborazione di un formalismo semplice ma sufficientemente potente da non trascurare dettagli importanti. Abbiamo deciso di utilizzare un multigrafo non orientato $Q = (T, E, l)$ dove T è l'insieme degli obiettivi, E è un insieme di archi e l è la funzione incaricata di generare le etichette su E.

Dati due obiettivi t_i e t_j, chiamiamo $p^k_{i,j} \in V$ il k-esimo percorso tra t_i e t_j. Nessuna assunzione è fatta sulla lunghezza del cammino. Generalmente, tra due obiettivi ci sarà più di un percorso, e ogni cammino permetterà

di avere etichette differenti; non è quindi possibile limitare l'insieme dei cammini a quelli minimi benchè, come vedremo in seguito, sia possibile ridurre il set eliminando cammini che sappiamo per certo essere peggiori di altri.

Chiamato $e = (t_i, t_j, k)$ il k-esimo arco che connette gli obiettivi t_i e t_j, definiamo E come l'insieme $E = \{(t_i, t_j, k)\}$ tale che $|p^k_{i,j}| \leq min\{d(t_i), (d(t_j)\}$. Questo equivale ad effettuare una prima selezione escludendo, come visto nell'esempio introduttivo, i percorsi più lunghi del minore dei penetration time dei due obiettivi.

La funzione $l : E \rightarrow \wp(T)$ dove $\wp$ è l'insieme delle parti ha la seguente definizione: dato un arco $e = (t_i, t_j, k)$, $l(e) = \{t \in T\}$ contiene tutti gli obiettivi tali che per ogni $v \in p^k_{i,j}$ esiste un percorso $p_{v,t}$ con $|p_{v,t}| \leq d(t_v)$. Tale condizione, in linea con quanto detto informalmente in precedenza, permette di individuare gli obiettivi che non vengono lasciati incustoditi muovendosi su un dato percorso.

L'insieme E può essere ulteriormente ridimensionato eliminando gli archi che sono Pareto-dominati in termini di lunghezza ed etichetta: un arco $e = (t_i, t_j, k)$viene rimosso se esiste almeno un altro arco $e\prime = (t_i, t_j, k\prime)$ per cui $l(e) \subseteq l(e\prime)$ e $|p^k_{t_i,t_j}| \geq |p^k_{t_i,t_j}\prime|$. Questo equivale a dire che il pattugliatore, a parità di lunghezza, preferirà muoversi sul percorso con l'etichetta più grande, e a parità di etichetta sceglierà il percorso più breve.

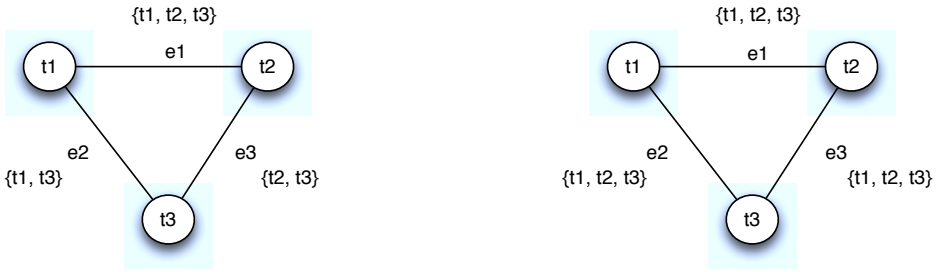

Figura 4.2: Esempio introduttivo II - Grafo Etichettato

In Figura 4.2 riportiamo il formalismo, chiamato *multigrafo etichettato*, applicato ai setting in figura 4.1. Nell'immagine a sinistra e_2 ed e_3 sono, in realtà, archi multipli. Esistono, infatti, 3 percorsi che collegano t_1 a t_3 (t_2 a t_3) ma hanno tutti lunghezza e etichetta uguale e vengono dunque considerati come uno. Accanto ad ogni arco, tra parentesi graffe, riportiamo l'insieme l.

4.1.2 La clique o sottografo completo

Per comprendere la parte seguente, è necessaria una breve digressione di teoria dei grafi. Dato un grafo non orientato $G = (V, E)$ dove V è l'insieme dei vertici e E è l'insieme dei nodi, definiamo *clique*, o cricca, un set di vertici $C \subseteq V$ per cui, per ogni possibile coppia di nodi, esiste un arco $e \in E$ che li collega. In altre parole, il grafo indotto dall'insieme C è completo.

Definiamo, inoltre, *clique massima*, una clique non contenuta in altre clique, o equivalentemente una clique che non può essere estesa con l'aggiunta di altri vertici.

Analogamente, orientando il grafo tramite doppie frecce e imponendo la riflessività sul nostro grafo, una clique è una partizione di equivalenza di una delle possibili relazioni di equivalenza (gode di proprietà *riflessiva*, *simmetrica*, *transitiva*) che è possibile individuare nel grafo. Tale rappresentazione, dettagliatamente riportate nell'appendice A, può essere utile per rendere più chiaro come operi il partizionamento dell'ambiente che abbiamo deciso di imporre.

Trovare tutte le possibili clique di un grafo (o specifiche clique di dimensioni prefissate) è un problema *NP-completo* (è possibile verificare la correttezza della soluzione in tempo polinomiale).

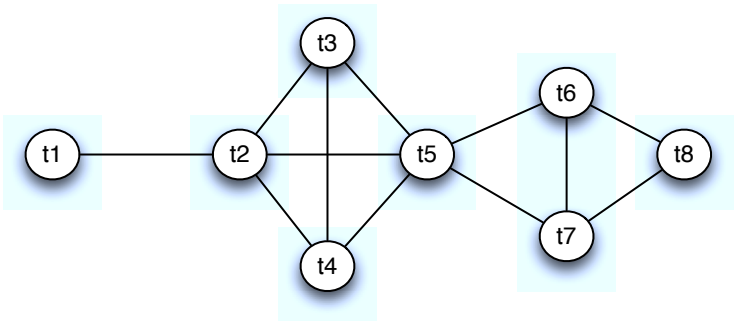

Figura 4.3: Esempio di clique

Il grafo in Figura 4.3 contiene 4 clique massime: $\{t_1, t_2\}$, $\{t_2, t_3, t_4, t_5\}$, $\{t_5, t_6, t_7\}$, $\{t_6, t_7.t_8\}$. Qualunque sottoinsieme di nodi di una clique è a sua volta una clique.

4.1.3 Clique etichettata

Abbiamo elaborato, dunque, un concetto più forte di clique, che chiamiamo *clique etichettata*. Una clique etichettata è un sottografo $W = (T', E', l')$ di Q per cui valgono le seguenti condizioni:

$$\forall t_i, t_j \in T', t_i \neq t_j, \text{ esiste } k \text{ tale che } (t_i, t_j, k) \in E' \tag{4.1}$$

$$T' \subseteq \bigcap_{e \in E'} \{l(e)\} \tag{4.2}$$

La prima condizione impone che, per ogni coppia di obiettivi, esista un arco (e quindi un solo percorso) che li colleghi. La seconda condizione, invece, richiede che l'insieme degli obiettivi sia contenuto o al più uguale all'intersezione delle etichette degli archi che formano la clique. L'intesezione delle etichette ha un preciso significato topologico: è l'insieme degli obiettivi che possono essere raggiunti da una cella qualsiasi tra quelle che appartengono ai cammini associati agli archi della clique. Se coincide, o è più ampio, dell'insieme degli obiettivi appartenenti alla clique, è possibile muoversi tra tali obiettivi senza lasciarne mai esposto nessuno.

Definiamo *copertura* di un grafo H l'insieme S delle clique tali per cui ogni vertice del grafo H compare in almeno una clique.

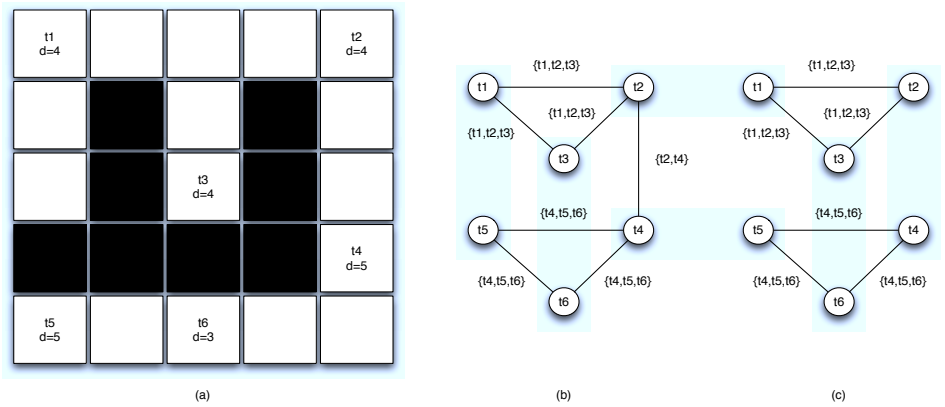

Figura 4.4: Dal setting alle clique

In Figura 4.4 mostriamo il passaggio da una generica topologia (a) al grafo etichettato equivalente (b) e poi alle clique etichettate che determinano la copertura e il partizionamento dell'ambiente. E' interessante notare che le astrazioni introdotte ci permettono di rendere esplicite informazioni che, dalla topologia, non sono evidenti. Setting molto differenti, sia come forma che come dimensioni, possono essere ricondotti a grafi etichettati simili (a meno, ovviamente, dei percorsi associati agli archi) e dunque tale astrazione può rappresentare un criterio di classificazione dei problemi trattati.

4.1.4 Lower bound sul numero di robot

Basandoci sulle sezioni precedenti, possiamo formulare il seguente teorema:

Teorema 1. *Dato un problema di pattugliamento definito da* $G = (V, a, T, v, d, R)$ *e la sua astrazione* $Q = (T, E, l)$*, se esiste una copertura composta da* $|R|$ *clique etichettate, allora è possibile trovare una strategia di pattugliamento che non lasci esposto alcun obiettivo.*

Tale teorema è equivalente ad indicare come lower bound sul numero di robot la cardinalità dell'insieme S che ne rappresenta la copertura mediante clique etichettate.

E' facile verificare che, data una copertura composta da $|R|$ clique, è possibile produrre almeno una strategia in cui, ad ogni turno, in ogni clique è presente almeno un robot. Quando ciò avviene, i robot si troveranno solo in *configurazioni possibili*.

Le considerazioni fatte fino a questo momento valgono nel caso in cui i robot non abbiano alcun tipo di coordinazione. In caso di robot coordinati, esistono casi in cui è possibile ridurre il numero $|R|$ di agenti necessari a coprire l'intero ambiente.

4.2 Algoritmo di enumerazione delle clique etichettate

Poichè il nostro lavoro si basa su un concetto (la clique etichettata) nuovo ma comunque estensione della più generale idea di clique, è stato necessario sviluppare un algoritmo *ad-hoc* in grado, partendo da un multigrafo etichettato, di estrapolarne le clique.

Nel 1973, Bron e Kerbosch [8] hanno presentato l'algoritmo più efficiente disponibile per calcolare l'insieme delle clique in un grafo. Poichè una clique etichettata è una estensione più restrittiva di una clique, è possibile modificare l'algoritmo in modo da adattarlo al nostro problema.

4.2.1 L'algoritmo originale

Dato un grafo $G = \{V, E\}$, l'algortimo BK sfrutta il *backtracking ricorsivo* per trovare tutte le clique massime in un grafo. Dati gli insiemi R, P, X trova tutte le clique massime che hanno tutti i vertici in R, alcuni vertici in P e nessun vertice in X. Tramite chiamate ricorsive, P e X sono ridotti ai vertici che formano una clique quando sono aggiunti ai vertici già in R poichè sono gli unici vertici ancora utilizzabili.

La ricorsione inizializza R e X all'insieme vuoto, e P all'insieme dei vertici del grafo. Ad ogni chiamata ricorsiva, l'algoritmo considera i vertici presenti nell'insieme P; se P è vuoto e X è vuoto, l'algoritmo segnala una clique massima. Se P è vuoto, ma X no, l'algortimo fa backtracking.

Per ogni vertice v in P, viene effettuata una chiamata ricorsiva in cui v è aggiunto ad R e P e X sono limitati ai vicini di v, permettendo di trovare tutte le clique generate come estensione di R che contengono v. Poi v viene aggiunto a X (per evitare di rivisitarlo) e rimosso da P.

Riportiamo, per completezza, la routine principale dell'algoritmo, dove con $N(v)$ intendiamo i nodi *vicini* al vertice v.

Algorithm 1 Function BK(R,P,X)

```
1: if P è vuoto then
2:    if X è vuoto then
3:       segnalo la clique massima
4:    end if
5: end if
6: for all v ∈ P do do
7:    BK(R ∪ {v}, P ∩ N(v), X ∩ N(v)
8:    P ← P\{v}
9:    X ← X ∪ {v}
10: end for
```

4.2.2 L'estensione Basilico-Gatti-Villa

L'algoritmo originale inserisce in una clique un vertice alla volta, e non si preoccupa degli archi che vengono aggiunti. Questo perchè, essendo stato progettato per lavorare su grafi, non è in grado di funzionare su un multigrafo, in cui non è solo importante quale vertice viene aggiunto alla clique ma anche con quali archi viene aggiunto.

Inoltre, l'estensione da noi progettata permette di distinguere clique massime (quindi clique non contenute in altre clique) da clique non massime. In più, sulle foglie dell'albero generato ci saranno solo clique (come anche nei nodi intermedi, perchè come detto, un sottoinsieme di nodi di una clique è a sua volta una clique).

Dato un multigrafo etichettato, chiamiamo *candidata* la coppia $\langle t, E_t \rangle$ dove $t \in T$ e $E_T \subseteq E$ è un insieme di archi che connettono t. Chiamiamo CLI, ovvero la soluzione parziale, un insieme di candidate che può venire esteso aggiungendo candidate ammissibili. Se l'insieme CLI è vuoto, le candidate ammissibili sono nella forma $\langle t, \emptyset \rangle$, quindi possiamo aggiungere qualunque obiettivo senza preoccuparci di quali archi utilizzeremo per connetterlo alla soluzione parziale. Se invece CLI non è vuota, definiamo $T(CLI)$ come l'insieme dei vertici presenti nella soluzione parziale e $E(CLI)$

come l'insieme degli archi inseriti. Una candidata $\langle t, E_t \rangle$ è ammissibili se rispetta le condizioni seguenti:

- $t \notin T(CLI)$: aggiungiamo un obiettivo che non è ancora presente nella soluzione parziale.
- E_t contiene esattamente un arco per ogni coppia di obiettivi (t, t'), $t' \in T(CLI)$: il vertice inserito deve essere collegato a tutti i vertici già presenti con uno e un solo arco. Questo assicura che il nuovo obiettivo sia completamente connesso.
- chiamata $CLI' = CLI \cup \{\langle t, E_t \rangle\}$, il grafo i cui vertici sono $T(CLI')$ e gli archi $E(CLI')$ è una clique etichettata: deve dunque valere la condizione 4.2 che impone che l'insieme dei vertici sia contenuto o al più uguale all'intersezione delle etichette degli archi appartenenti alla clique.

L'algoritmo. Data una soluzione parziale CLI, introduciamo i seguenti insiemi:

- D: è l'insieme delle candidate ammissibili, generato dalla funzione $candidates(CLI)$ secondo le condizioni di cui sopra.
- N: contienele candidate che sono già state sfruttate per generare una clique al medesimo livello.

Inoltre, indichiamo con $T(D')$ l'insieme dei vertici contenuti nell'insieme D'.

Algorithm 2 find_solution(T, E, l)

$D \leftarrow \{\}$
$CLI \leftarrow \{\}$
$N \leftarrow \{\}$
for all all t in T **do**
  $D \leftarrow D \cup \{\langle t, \emptyset \rangle\}$
end for
$S \leftarrow$ CLIQUES(CLI, D, N)

Algorithm 3 cliques(CLI, D, N)

if D is empty **then**
  if N is empty **then**
    CLI è una clique massima
  else
    CLI è una clique non massima
  end if
else
  for all all $t \notin N$ appartenente ad almeno una candidata in D **do**
    for all all $\langle t', E_{t'} \rangle$ in D with $t' = t$ **do**
      $CLI' = CLI \cup \{\langle t', E_{t'} \rangle\}$
      $D' = \text{CANDIDATES}(CLI')$
      $S \leftarrow S \cup \text{CLIQUES}(CLI', D', N \cap T(D'))$
    end for
    $N = N \cup \{t\}$
  end for
end if
return S

Algorithm 4 candidates(CLI)

$D \leftarrow \{\}$
for all all t in $T \setminus T(CLI)$ **do**
  $T' \leftarrow T(CLI) \cup t$
  for all all E_t in $\text{EDGES}(CLI, t)$ **do**
    if le condizioni (4.1) e (4.2) sono valide su $(T', \{E(CLI) \cup E_t\}, l)$ **then**
      $D \leftarrow D \cup \langle t, E_t \rangle$
    end if
  end for
end for
return D

Altre soluzioni. Prima di procedere al design dell'algoritmo, ci siamo chiesti se non esistessero metodi più veloci per l'individuazione di tutte le clique etichettate. Una soluzione alternativa può essere la ricerca di tutte le clique di un grafo tramite algoritmi noti (ed eventualmente librerie già implementate) e in seguito il controllo della condizione sulle etichette, per ogni clique individuata. Tale soluzione potrebbe risultare più veloce in alcuni casi, ma sicuramente meno performante in altri.

Abbiamo deciso di non indagare ulteriormente in questa direzione, ritenendo accademicamente più interessante lo sviluppo di un algorimo *ad hoc*. Rimangono dunque insoluti eventuali dubbi riguardo la reale validità di tale idea.

4.2.3 Copertura del grafo

Una volta individuate tutte le clique del grafo etichettato, è possibile individuare la più piccola copertura risolvendo un problema di programmazione matematica. Sia $S_{\max} \subseteq S$ l'insieme delle clique etichettate massime. Possiamo limitare la nostra ricerca alle clique massime poichè il loro numero, benchè nel caso pessimo sia $3^{\frac{|T|}{3}}$, è trascurabile se paragonato al numero totale delle clique.

Definiamo una matrice $q_{i,j}$ di copertura clique/obiettivo dove $q_{i,j} = 1$ se l'obiettivo t_j appare nella clique W_i, $q_{i,j} = 0$ altrimenti. Introduciamo la variabile decisionale x_i e sia $x_i = 1$ se la clique W_i viene scelta per fare parte della copertura, $x_i = 0$ altrimenti. Possiamo scrivere:

$$\min \sum_{i:W_i \in S_{\max}} x_i \tag{4.3}$$

sotto il vincolo:

$$\sum_{i:W_i \in S_{\max}} (x_i q_{i,j}) \geq 1 \ \forall t_j \in T, x_i \in \{0,1\} \ \forall i : W_i \in S \tag{4.4}$$

Può essere interessante notare che è necessario risolvere il problema di programmazione matematica solo se il multigrafo etichettato è effettivamente un multigrafo (e quindi sono presenti doppi archi tra due nodi). Qualora fossero presenti solo archi singoli, l'insieme di copertura coinciderà con l'insieme delle clique massime, poichè non esisteranno clique che coprono esattamente il medesimo insieme di obiettivi.

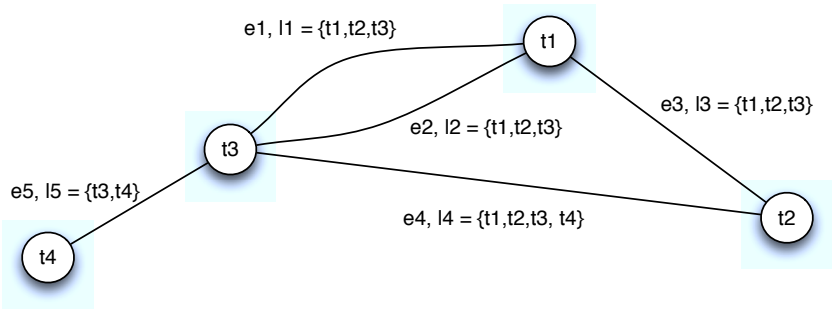

Figura 4.5: Multigrafo etichettato

4.2.4 Esempio

La Figura 4.5 presenta un multigrafo etichettato composto da tre clique massime, e con archi doppi. Nella sua semplicità, permette di vedere come l'algoritmo operi nell'individuare le clique.

In Figura 4.6, riportiamo l'albero prodotto dall'algoritmo BGV. Nei nodi sono presenti gli obiettivi aggiunti, sugli archi sono riportati l'insieme di archi con cui l'obiettivo viene aggiunto. Il quadratino nei pressi di un nodo segnala l'ordine di espansione (in profondità). Il raggiungimento di una clique massima è segnalata dall'etichetta *MAX*.

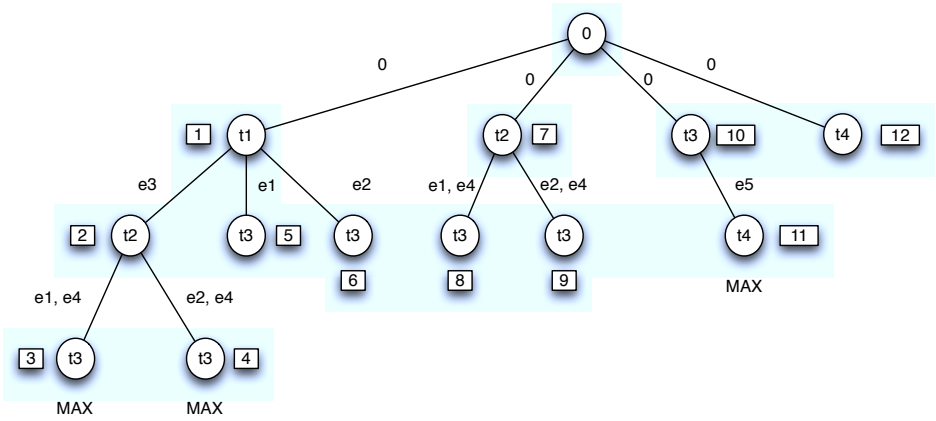

Figura 4.6: Albero prodotto da BGV

Più nel dettaglio, è possibile notare che la clique composta da $\{t_1, t_2, t_3\}$ è massima perchè al passo 3 l'insieme D delle candidate ancora utilizzabili per quel nodo è vuoto, come lo è l'insieme N delle candidate già utilizzate. Analogamente, allo step 8, la clique $\{t_2, t_3\}$ non è massima in quanto l'insieme D è vuoto, ma l'insieme N contiene le candidate sfruttate a quel livello

dai fratelli espansi fino all'instante 6, candidate che non possono più essere aggiunte impedendo di generare clique contenenti t_1 (per completezza, le candidate in N sono $\langle t_1, 0\rangle$, $\langle t_3, e_1\rangle$, $\langle t_3, e_2\rangle$).

L'algoritmo produce quindi le seguenti clique etichettate massime: $W_1 = \{t_1, t_2, t_3\}$ con archi $\{e_1, e_3, e_4\}$, $W_2 = \{t_1, t_2, t_3\}$ con archi $\{e_2, e_3, e_4\}$ e $W_3 = \{t_3, t_4\}$ con archi $\{e_5\}$.

Possiamo dunque scrivere la matrice di copertura q:

$$q_{i,j} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

Esistono due possibili coperture in termini di clique etichettate: $\{W_1, W_3\}$ e $\{W_2, W_3\}$. Sono dunque sufficienti 2 robot non coordinati per coprire l'ambiente associato al multigrafo etichettato in Figura 4.5.

Capitolo 5

Dimensioni di coordinamento

Le due dimensioni - decomposizione strategica e topologica - permettono di definire differenti modelli in base al livello di coordinamento

L'obiettivo di questo capitolo è indicare come diversi gradi di coordinamento dei robot possano influire sulla soluzione al problema del pattugliamento alla luce di quanto detto nel capitolo dedicato allo stato dell'arte. Le varie possibilità vengono mostrate con l'ausilio di un semplice esempio.

5.1 Livelli di Coordinamento

Come abbiamo visto nel capitolo dedicato allo stato dell'arte, esistono studi che dimostrano che agenti semplici ma in grado di coordinarsi hanno performance molto migliori di quelle di agenti complessi ma non in grado di scegliere strategie comuni. Basandosi sui risultati del capitolo precedente, identifichiamo 2 diverse dimensioni di coordinamento: *decomposizione strategica* e *decomposizione topologica*.

5.1.1 Decomposizione strategica

La decomposizione strategica è la dimensione relativa al calcolo delle probabilità $\{\alpha_{c,c'}\}$ ovvero la probabilità che i robot, da una configurazione c, decidano di andare nella configurazione c'. Indichiamo con C l'insieme delle configurazioni, con R l'insieme dei robot e con V l'insieme delle celle dell'ambiente.

Possiamo individuare tre gradi di decomposizione strategica, dal più forte al più debole: *strategia unica*, *strategia disaccoppiata* e *strategie separate*.

Strategia unica. Il sistema è assimilabile ad un'*architettura a conoscenza globale e scelta centralizzata* dove le strategie sono calcolate in modo centralizzato, come lo sono i movimenti degli agenti ad ogni turno. La strategia di ogni robot dipende dalla sua posizione e dalla posizione di tutti gli altri. La soluzione del problema è rappresentata dall'insieme delle $\{\alpha_{c,c'}\}$ con $c, c' \in C$ e contiene $O^{|R|}$ variabili.

Strategia disaccoppiata. Il sistema è assimilabile ad un'*architettura a conoscenza globale e scelta locale* dove ogni robot è consapevole della presenza di altri robot ma la sua strategia dipende solo dalla sua posizione. La soluzione è un insieme di $\{\alpha^i_{j,k}\}$, una per ogni robot r_i, dove $j, k \in V$ e contiene $O(|R| \cdot n^2)$ variabili. Le probabilità $\alpha_{c,c'}$ sono definite come

$$\alpha_{c,c'} = \prod_{i \in R} \alpha^i_{c_i,c'_i} \tag{5.1}$$

Le strategie di ogni robot sono calcolate in modo centralizzato, tenendo conto delle strategie di tutti gli altri. Tale livello può portare ad una qualità della soluzione inferiore alla soluzione in strategia unica ma è computazionalmente molto più leggero.

Strategie separate. Il sistema è assimilabile ad un'*architettura a nessuna conoscenza* dove ogni robot agisce come se fosse solo. Il calcolo di ogni $\{\alpha^i_{j,k}\}$ è svolto in modo locale su ogni robot e $\alpha_{c,c'}$ è definito come nel caso precedente. La qualità della soluzione trovata in questo modo è potenzialmente molto inferiore a quelle dei punti precedenti, questo perchè stiamo riducendo il problema a $|R|$ problemi singolo agente privi di comunicazione.

5.1.2 Decomposizione topologica

La decomposizione topologica è la dimensione relativa alla coordinazione nel partizionare l'ambiente, basandosi su quanto visto al capitolo precedente. Ad ogni robot è assegnata una porzione del multigrafo etichettato e gli viene impedito di uscire da tale area.

Possiamo individuare tre gradi di decomposizione topologica, dal più forte al più debole: *assegnamento completo*, *assegnamento di clique massima* e *assegnamenti separati*.

Assegnamento completo. Ogni robot può potenzialmente muoversi su ogni vertice del multigrafo purchè il sistema si mantenga in una configurazione possibile. Tutti i robot possono pattugliare ogni obiettivo.

Assegnamento di clique massima. Imponiamo che ad ogni robot venga assegnata una clique massima; in questo caso, è possibile che alcuni obiettivi (quelli condivisi da più clique) vengano difesi da più robot, mentre

altri da un robot solo. Questo permette di ridurre il numero delle possibili configurazioni e, quindi, delle possibili $\alpha_{c,c'}$. E' però molto importante notare che la qualità della soluzione può essere inferiore a quella ottenuta con un assegnamento completo.

Assegnamenti separati. Ogni robot è assegnato ad una clique che non ha sovrapposizioni con altre clique. Dunque, facciamo cadere l'ipotesi di lavorare su clique massime per trovare una copertura composta anche da clique non massime. La scelta di come effettuare la scelta della copertura migliore esula dall'obiettivo di questa tesi ed è parte degli sviluppi futuri.

Esempio

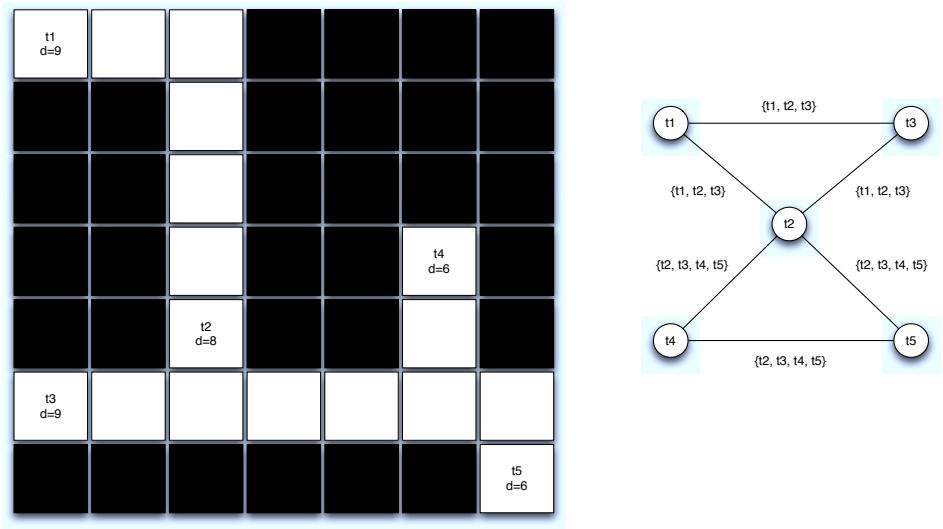

Figura 5.1: Esempio

In Figura 5.1 riportiamo la topologia di un ambiente di esempio e il suo multigrafo etichettato. Possiamo notare che è partizionabile in due clique massime, $W_1 = \{t_1, t_2, t_3\}$ e $W_2 = \{t_2, t_4, t_5\}$.

In Figura 5.2 vediamo uno stato possibile del sistema per ogni livello di decomposizione topologica. L'immagina (a) mostra una configurazione possibile nel caso di assegnamento completo. Entrambi i robot (le celle arancioni) si trovano nella medesima clique W_1 ma nessun obiettivo è lasciato scoperto. L'immagine (b) mostra il sistema in caso di assegnamento in clique massima. Ogni robot (le celle verdi) è in una clique massima differente. E' importante notare che, mentre la cella denotata con x appartiene solo alla clique W_1, la cella segnalata con la lettera y appartiene sia a W_1

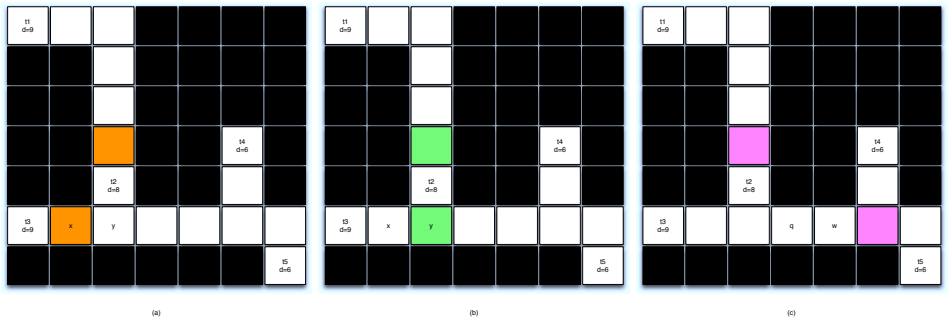

Figura 5.2: Stati possibili

che a W_2 ma in questa combinazione non è possibile che i due agenti si invertano, scambiandosi le rispettive clique. Questo porta a concludere che il disaccoppiamento topologico a livello di clique massima non implica necessariamente che le celle condivise dalle due clique siano solo gli obiettivi in comune. L'immagine (c) mostra i robot (le celle rosa) qualora stiano sfruttando gli assegnamenti separati dati dalla copertura delle clique non massime $W_1 = \{t_1, t_2, t_3\}$ e $W_2 = \{t_4, t_5\}$. I due robot non pattugliano celle in comune e dei cammini, quelli corrispondenti agli archi tagliati nello scegliere la copertura, non vengono percorsi (nell'esempio, le celle q e w non vengono più esplorate).

5.2 Combinazioni possibili delle due dimensioni

Non tutte le combinazioni {decomposizione strategica, decomposizione topopologica} precedenti sono significativi. In particolare, {strategia unica, assegnamenti separati} è equivalente a {strategia disaccoppiata, assegnamenti separati} perchè due robot che pattugliano porzioni disgiunte di ambiente non hanno alcun bisogno di coordinarsi nello scegliere la configurazione successiva a quella in cui si trovano.

La combinazione {strategia disaccoppiata, assegnamento completo} può essere ridotta a {strategia disaccoppiata, assegnamento di clique massima}. Infatti, immaginando di avere un ambiente diviso in due clique massime con alcuni obiettivi in comune, non è possibile trovare una configurazione ammissibile in cui entrambi i robot si muovono nella medesima clique. Questo perchè, essendo le strategie separate, l'essere nella medesima clique lascerebbe inevitabilmente esposti gli obiettivi coperti solo dall'altra clique massima. Possiamo fare considerazioni analoghe per la combinazione {strategie separate, assegnamento completo}.

La Tabella 5.1 riporta, con $\mathcal{D}_i$, le dimensioni di cui abbiamo riconosciuto l'importanza e che abbiamo approfondito.

	completo	clique massima	separati
unica	$\mathcal{D}_1$	$\mathcal{D}_2$	–
disaccoppiata	–	$\mathcal{D}_3$	–
separate	–	$\mathcal{D}_4$	$\mathcal{D}_5$

Tabella 5.1: Combinazioni possibili strategia-assegnamento

5.2.1 Strategia unica - assegnamento completo ($\mathcal{D}_1$)

Il sistema multiagente può essere ricondotto ad un sistema singolo agente in cui il robot (che rappresenta tutti i nostri $|R|$ robot) si muove sull'insieme delle configurazioni e non più sull'insieme delle celle come avviene nel BGA classico. Conseguentemente, il problema può essere formulato come una diretta estensione del modello singolo agente visto nel capitolo dedicato allo stato dell'arte. Indichiamo con $\gamma_{c,c'}^{h,w}$ la probabilità che i robot raggiungano la configurazione c' in h turni, partendo dalla configurazione c senza passare per l'obiettivo w. C_{-w} è l'insieme di tutte le configurazioni in cui w non è occupato da nessun robot. Indichiamo, poi, con $X = U_{\mathbf{r}}(\textit{intruder-capture})$ e $Y_i = U_{\mathbf{r}}(\textit{penetration-}i)$. La formulazione matematica, non lineare, è la seguente:

$$\max u$$

s.t.

$$\alpha_{c,c'} \geq 0 \qquad \forall c, c' \in C \tag{5.2}$$

$$\sum_{c' \in C} \alpha_{c,c'} = 1 \qquad \forall c \in C \tag{5.3}$$

$$\alpha_{c,c'} \leq \prod_{r \in R} a(c_i, c_i') \qquad \forall c, c' \in C \tag{5.4}$$

$$\gamma_{c,c'}^{1,w} = \alpha_{c,c'} \qquad \forall w \in T, c, \in V, c' \in C_{-w} \tag{5.5}$$

$$\gamma_{c,c'}^{h,w} = \sum_{x \in C_{-w}} \left(\gamma_{c,x}^{h-1,w} \alpha_{x,c'}\right) \qquad \begin{array}{r} \forall h \in \{2, \ldots, d(w)\}, \\ \forall w \in T, c \in C, c' \in C_{-w} \end{array} \tag{5.6}$$

$$P(c,w) = 1 - \sum_{c' \in C_{-w}} \gamma_{c,c'}^{d(w),w} \qquad \forall w \in T, c \in C \tag{5.7}$$

$$u \leq P(c,w)(X - Y_w) + Y_w \qquad \forall w \in T, c \in C \tag{5.8}$$

I vincoli (5.2) e (5.3) assicurano che le probabilità $\alpha_{c,c'}$ siano positive e ben definite per ogni configurazione c. Il vincolo (5.4) garantisce che almeno una cella della configurazione c' sia adiacente ad una cella di c. Essenzialmente, si occupa di assicurare che c' sia una configurazione successiva di

c. I vincoli (5.5) e (5.6) impongono la markovianità della strategia dei robot. Il vincolo (5.7) assegna a $P(c, w)$ la probabilità che l'intruso venga catturato se sceglie di effettuare una *enter-when*(w, c). Il membro destro del vincolo (5.8) rappresenta l'utilità attesa dei robot quando l'intruso sceglie di effettuare una *enter-when*(w, c), mentre u è l'utilità attesa effettiva del robot. L'obiettivo è massimizzare u.

5.2.2 Strategia unica - assegnamento di clique massima ($\mathcal{D}_2$)

Assegnamo ad ogni robot r una clique W_r e imponiamo che ad ogni turno del gioco la posizione di r si trovi in una delle celle associate a W_r. Il modello è il medesimo della combinazione $\mathcal{D}_1$ ma l'insieme delle combinazioni valide viene ridotto, eliminando quelle che violano il nuovo vincolo e che quindi prescrivono che almeno una clique rimanga vuota.

5.2.3 Strategia disgiunta - assegnamento clique massima ($\mathcal{D}_3$)

Dato un assegnamento in termini di clique, un robot r difende un insieme di obiettivi. Dato un obiettivo w, chiamiamo $R_w \subseteq R$ l'insieme di robot da cui viene protetto. Ovviamente, ogni obiettivo viene difeso da almeno un robot. La formulazione matematica del problema è la seguente:

$$\max u$$

s.t.

$$\alpha^r_{i,j} \geq 0 \qquad \forall r \in R, i, j \in V \tag{5.9}$$

$$\sum_{j \in V} \alpha^r_{i,j} = a_r(i, i) \qquad \forall r \in R, i \in V \tag{5.10}$$

$$\alpha^r_{i,j} \leq a_r(i, j) \qquad \forall r \in R, i, j \in V \tag{5.11}$$

$$\gamma^{1,w}_{r,i,j} = \alpha^r_{i,j} \qquad \forall r \in R_w, w \in T, i, j \in V, j \neq w \tag{5.12}$$

$$\gamma^{h,w}_{r,i,j} = \sum_{x \in V \setminus w} \left(\gamma^{h-1,w}_{r,i,x} \alpha^r_{x,j} \right) \qquad \begin{array}{r} \forall h \in \{2, \ldots, d(w)\}, \\ \forall r \in R_w, w \in T, i, j \in V, j \neq w \end{array} \tag{5.13}$$

$$P(c, w) = 1 - \prod_{r \in R_w} \sum_{j \in V \setminus \{w\}} \gamma^{d(w),w}_{r,c_r,j} \qquad \forall w \in T, c \in C \tag{5.14}$$

constraints (5.8)

I vincoli (5.9)-(5.13) hanno lo stesso significato di (5.2)-(5.8). Il vincolo (5.14) assegna a $P(c, w)$ la probabilità che l'intruso venga catturato quando decide di effettuare una *enter-when*(w, c) considerando le strategie di tutti i robot che proteggono l'obiettivo w.

5.2.4 Strategie separate ($\mathcal{D}_4$, $\mathcal{D}_5$)

Il problema multiagente è ricondotto ad $|R|$ problemi singolo agente, sfruttando la formulazione base dell'algoritmo BGA. Dato un assegnamento, risolviamo tutti i problemi singolo agente associati; la minore tra le utilità dei pattugliatori è l'utilità ottenuta nella risoluzione. Poichè possono esserci diverse coperture in termini di clique (non massime), siamo costretti a risolvere tutti i problemi associati ad ogni possibile copertura e a scegliere la soluzione che massimizza la minore tra le utilità ottenute nelle clique del problema: una volta risolti tutti i problemi singolo agente, per calcolare l'utilità attesa dei pattugliatori, dobbiamo trovare la *best response* dell'intruso, e in base a quella, calcolare le utilità che cerchiamo. Infatti, i problemi singolo robot non considerano il fatto che altri robot possano pattugliare eventuali obiettivi in comune.

5.3 Dominanze

Identificare eventuali strategie dominate, ovvero strategie per cui esiste un'altra strategia sicuramente migliore dal punto di vista degli agenti, ci permette di ridurre il problema limitando considerevolmente il tempo necessario alla risoluzione del modello. Nel capitolo dedicato allo stato dell'arte, approfondendo il modello BGA, abbiamo indicato le condizioni per cui una cella è considerata dominata da un'altra cella dal punto di vista dell'intruso. Tali considerazioni possono essere estese nel caso multiagente, per ogni singola combinazione delle nostre dimensioni.

5.3.1 Dominanze in $\mathcal{D}_4$ e $\mathcal{D}_5$

E' intuitivo comprendere che la combinazione $\mathcal{D}_4$ e $\mathcal{D}_5$, poichè si basano sulla scomposizione del problema originale in sottoproblemi singolo agente, sfruttano il concetto di dominanza già noto e non necessitano di alcuna estensione. L'intruso, potendo scegliere tra due sottoproblemi strategicamente indipendenti, sceglierà di cercare di penetrare in quello che gli garantirà l'utilità attesa maggiore.

5.3.2 Dominanze in $\mathcal{D}_2$ e $\mathcal{D}_3$

Possiamo formulare le dominanze nella combinazione $\mathcal{D}_3$ ricorrendo all'esempio in Figura 5.1. Un robot r_1 pattuglia la clique $W_1 = \{t_1, t_2, t_3\}$ e l'altro, r_2, la clique $W_2 = \{t_2, t_4, t_5\}$. L'intruso potrebbe preferire attaccare obiettivi non condivisi da più clique (quindi pattugliati da un solo robot)

quando il robot assegnato a tale clique si trova in una cella non dominata nel problema singolo agente equivalente, indipendentemente da ciò che farà l'altro robot (la cui posizione è per noi un *don't care*).

Se l'intruso decide di attaccare l'obiettivo t_1, attaccherà quando il robot r_1 si trova nell'obiettivo t_3, che è una cella non dominata. Qualora, invece, l'intrusore decidesse di attaccare un obiettivo in comune a più clique, lo farebbe nel momento in cui tutti i robot che difendono l'obiettivo si trovano in una configurazione che li prescrive nell'insieme delle celle non dominate della clique a cui sono assegnati. Nel nostro esempio, l'intruso può volere attaccare t_2 quando r_1 si trova in t_1 e r_2 si trova in t_4 o t_5.

Più formalmente, dato T l'insieme degli obiettivi, t un generico obiettivo, e $W_t \in W$ l'insieme delle clique a cui appartiene l'obiettivo t (W è l'insieme delle clique massime), chiamiamo $L_{t,z} \subseteq T$ l'insieme delle celle appartenenti alle clique $z \in W_t$ che sono non dominate in singolo agente qualora l'intruso decidesse di attaccare l'obiettivo t. Chiamiamo W_{-t} l'insieme delle clique che non contengono l'obiettivo t e indichiamo con V_z l'insieme delle celle appartenenti alla clique z.

Ua configurazione $s_0 =< c_1, ..., c_{|R|} >$ è non dominata per l'obiettivo t se e solo se contiene solo celle non dominate per ogni clique a cui appartiene t.

Possiamo costruire le configurazioni non dominate introducendo il concetto di *nucleo*. Un nucleo è un insieme di celle non dominate, una per ogni clique a cui appartiene l'obiettivo t, sufficienti ad esprimere la dominanza.

Possiamo indicare l'insieme dei nuclei N_t come:

$$N_t = L_{t,z} \times ... \times L_{t,z} \forall z \in W_t \tag{5.15}$$

Dunque, possiamo definire una configurazione non dominata tramite il prodotto cartesiano dell'insieme dei nuclei e quelli delle celle appartenti alle clique che non contengono t (queste ultime esprimono i *don't care*):

$$S_c = N_t \times V_b ... \times V_b \forall b \in W_{-t} \tag{5.16}$$

Nell'esempio in Figura 5.3, vediamo tutte le configurazioni non dominate per ogni obiettivo (in verde l'obiettivo considerato, in arancio le celle in cui può essere presente un robot).

Nel caso dell'obiettivo t_1 l'insieme dei nuclei contiene l'unico nucleo $s_{t_1} =< t_3 >$ e quindi le configurazioni non dominate avranno il robot assegnato a W_1 in t_3 e il robot assegnato a W_2 in una cella qualunque assegnata alla clique W_2.

Nel caso di t_2, l'insieme dei nuclei è $N = \{< t_1, t_4 >, < t_1, t_5 >\}$ e quindi esistono solo 2 configurazioni non dominate poichè non esistono altre clique

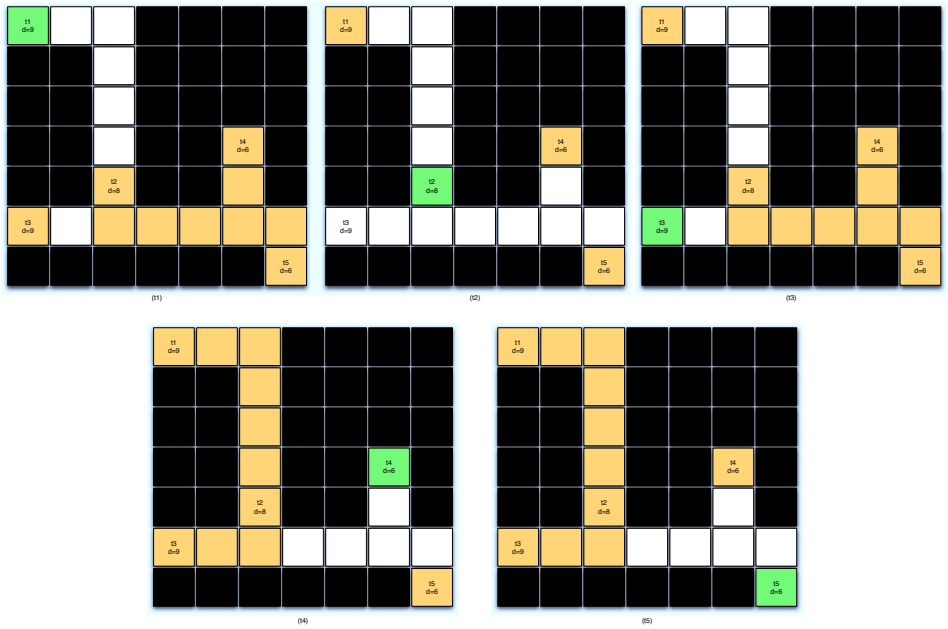

Figura 5.3: Dominanze di ogni obiettivo

da considerare. Gli obiettivi rimanenti sono del tutto simili a t_1 e vengono riportati per completezza.

Le azioni dell'intrusore non dominate saranno nella forma *enter_when(t,c)* con il significato di cominciare l'attacco all'obiettivo t quando i pattugliatori sono in una configurazione c non dominata.

Considerazioni analoghe possono essere fatte per $\mathcal{D}_2$, poichè, benchè i robot agiscano seguendo una strategia calcolata in modo globale ed effettuando scelte globali, il partizionamento dell'ambiente impone la presenza di un robot in ogni clique rendendo il calcolo delle dominanze del tutto analogo a quallo appena visto per $\mathcal{D}_2$

5.3.3 Dominanze in $\mathcal{D}_1$

Per quanto riguarda $\mathcal{D}_1$, non possiamo sfruttare il partizionamento dell'ambiente in clique massime poichè il modello si basa sulle configurazioni ammissibili e non sulle clique. Inoltre, la strategia di ogni robot dipende da quella di tutti gli altri. Un possibile approccio è la costruzione dell'albero delle configurazioni, in modo simile alla costruzione dell'albero dei cammini nel BGA singolo agente.

In figura 5.4 vediamo un setting di esempio, la sua clique etichettata e l'albero delle configurazioni per t_1 (parziale).

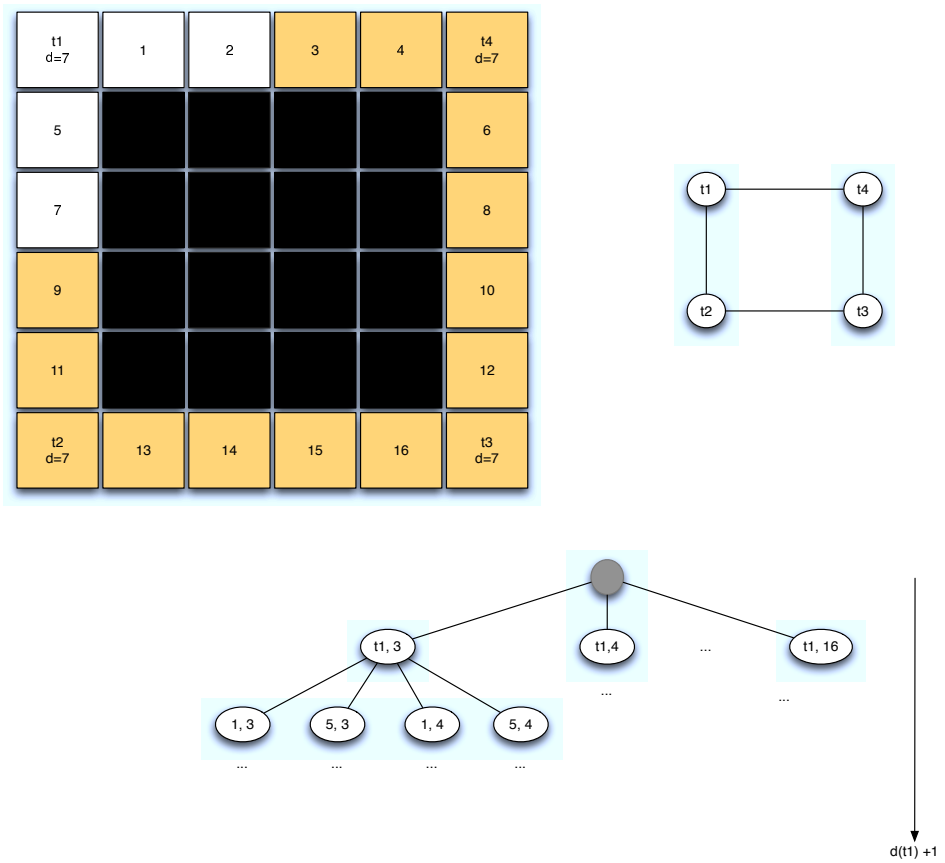

Figura 5.4: Esempio di dominanze $\mathcal{D}_1$

L'albero è costruito nel modo seguente: assumiamo che un robot sia in t_1 e che il secondo robot (è evidente dal grafo etichettato che l'ambiente sia difendibile da soli 2 robot) sia in una delle celle color arancio, quindi in una configurazione ammissibile. Il nodo in grigio ha un preciso significato: da esso immaginiamo discendere tutte le configurazioni con un robot in t_1. Ogni nodo ha come figli tutte le possibili configurazioni successive a quella associata al nodo stesso e l'altezza dell'albero è esattamente il penetration time dell'obiettivo t_1 più il nodo radice in grigio.

Se una configurazione è non dominata in tutti i sottoalberi che hanno come radice i figli del nodo marcato in grigio, allora è una configurazione non dominata per l'obiettivo t_1.

Capitolo 6

Valutazioni sperimentali

I test dimostrano che per ogni ambiente esiste almeno una combinazione in grado di calcolare una soluzione in tempi accettabili

Questa sezione contiene alcuni esperimenti svolti utilizzando i modelli matematici presentati nel capitolo precedente. Per ogni esperimento, sono riportate le informazioni sull'ambiente (dimensione, massimo penetration time, possibili assegnamenti in clique etichettate massime) e la mappa associata alla topologia. Il capitolo termina con le considerazioni tratte dalle sperimentazioni effettuate.

6.1 Ambiente di test

Abbiamo implementato l'algoritmo di enumerazione delle clique massime in JAVA. Il programma, di cui è disponibile una sommaria documentazione nell'Appendice B, converte un file di input contenente informazioni su ambiente e obiettivi in file (uno o più per ogni dimensione) sfruttabili per essere utilizzati da AMPL [12] come file DAT.

Abbiamo, poi, risolto i vari modelli utilizzando CPLEX [14] per risolvere i problemi di programmazione matematica lineari e SNOPT [19] per risolvere i problemi non lineari. La macchina di test un server UNIX con 2 processori quad-core a 2.33Ghz e 8GB di memoria RAM.

E' importante notare come la preparazione dei file DAT da utilizzare con AMPL, inclusa la ricerca delle clique etichettate massime, sia computazionalmente trascurabile rispetto al tempo necessario a risolvere il modello nonostante la natura prototipale del programma sviluppato, e dunque non venga riportato nelle tabelle.

6.2 Test

Riportiamo 5 ambienti che riteniamo significativi e su cui abbiamo effettuato la maggior parte degli esperimenti. Le topologie degli ambienti differiscono per numero di celle e di obiettivi e le loro caratteristiche sono riportate in Tabella 6.1. Riportiamo il numero di assegnamenti possibili in clique massime e in clique non massime senza sovrapposizioni (assegnamenti separati). E' utile notare che ogni ambiente può essere scomposto in un unico assegnamento in clique massime, e può essere difeso da esattamente 2 pattugliatori.

In Figura 6.1 sono riportate le topologie per ambiente1 (a), ambiente2 (b) e ambiente3 (c). In Figura 6.2, invece, sono mostrati ambiente3 (d) e ambiente4 (e).

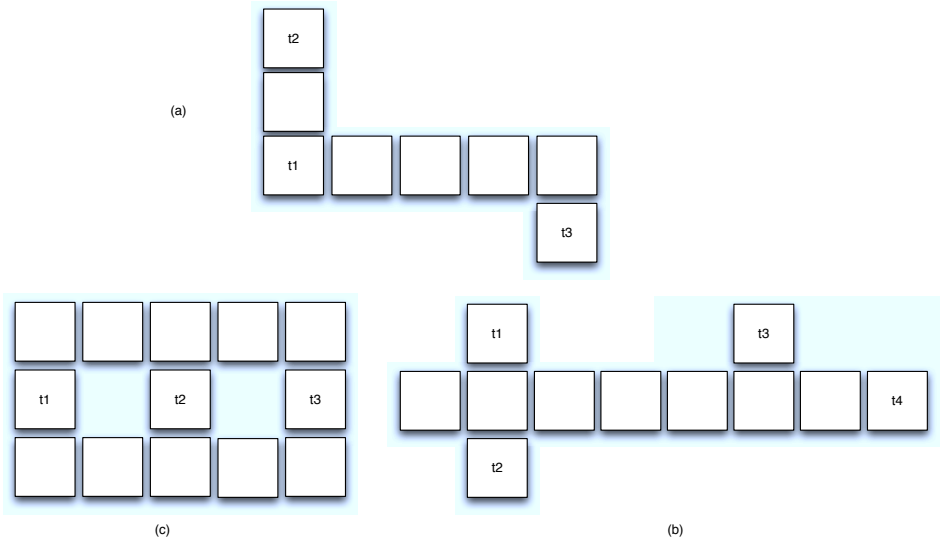

Figura 6.1: Ambienti di test - I

Abbiamo creato 10 istanze per ogni ambiente, facendo in modo che i penetration time fossero uniformemente generati all'interno dei rispettivi intervalli (range) e imponendo che il numero minimo di robot fosse 2. Le utilità dei robot sono state normalizzate generando le preferenze di ogni agente in modo fossero uniformemente distribuite sull'intervallo $[0, \frac{1}{|T|}]$.

Le istanze sono state risolte per ogni possibile combinazione e il risultati sono riportati in Tabella 6.3. Per comodità, in Tabella 6.2 riportiamo le possibili *combinazioni di coordinamento*.

Nel caso della combinazione $\mathcal{D}_5$, enumeriamo tutti i possibili assegnamenti e in Tabella 6.3 riportiamo la somma di tutti i tempi di computazione e tra parentesi il più lungo mentre in Tabella 6.4 riportiamo l'utilità nel caso

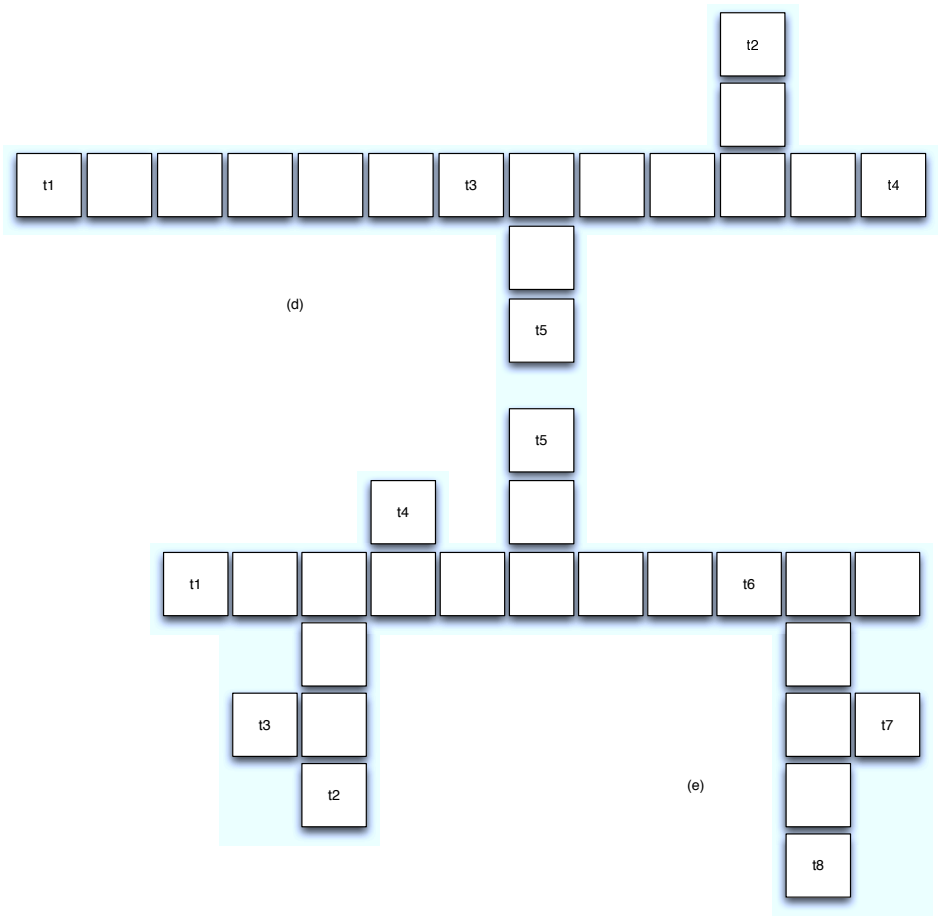

Figura 6.2: Ambienti di test - II

dell'assegnamento migliore. Le percentuali sono calcolate in base a $\mathcal{D}_5$, che possiede il minor grado di coordinamento.

6.3 Valutazione dei risultati

I risultati da noi ottenuti confermano un risultato già presente in letteratura: al diminuire del livello di coordinamento tra agenti, diminuisce anche la qualità della strategia. Nel nostro caso, quando il grado di coordinamento tra agenti si indebolisce, l'utilità attesa dei robot decresce ma decresce anche il tempo necessario a calcolare la soluzione.

Il trade-off tra grado di coordinamento e complessità del calcolo della soluzione (in termini di memoria e di tempo) è particolarmente evidente in $\mathcal{D}_1$ e $\mathcal{D}_2$. Qualora, infatti, i robot agiscano con una strategia calcolata

	celle	obiettivi	range di $d()$	# ass. max. clique	#sep. ass.
ambiente1	8	3	$[2, 3]$	1	2
ambiente2	11	4	$[4, 5]$	1	4
ambiente3	13	3	$[4, 6]$	1	4
ambiente4	17	4	$[6, 9]$	1	4
ambiente5	23	8	$[6, 9]$	1	4

Tabella 6.1: Ambienti

	completo	clique massima	separati
unica	$\mathcal{D}_1$	$\mathcal{D}_2$	–
disaccoppiata	–	$\mathcal{D}_3$	–
separate	–	$\mathcal{D}_4$	$\mathcal{D}_5$

Tabella 6.2: Combinazioni possibili strategia-assegnamento

globalmente ed effettuando scelte globali, non siamo in grado di risolvere problemi con grandi ambienti e dunque tali combinazioni sono sfruttabili solo su topologie composte da un numero esiguo di celle. $\mathcal{D}_1$, inoltre, come visto nel capitolo precedente quando abbiamo presentato gli algoritmi di calcolo delle dominanza, non può essere ridotto (o per essere ridotto necessita di un tempo di calcolo forse superiore a euello necessario alla sua risoluzione) e dunque, pur garantendo l'utilità attesa maggiore, non rappresenta una soluzione percorribile nella grande maggioranza degli ambienti significativi.

$\mathcal{D}_3$ si comporta meglio in termini di utilità di attesa di $\mathcal{D}_5$ ma è computazionalmente più oneroso; è però importante notare che, qualora il numero di possibili assegnamenti sia grande, il tempo di calcolo di $\mathcal{D}_5$ può essere molto più lungo di quello di $\mathcal{D}_3$.

Abbiamo modificato gli ambienti 1-2-3 in modo che il numero minimo di robot necessari fosse 3 e li abbiamo risolti in $\mathcal{D}_3$ e $\mathcal{D}_5$. Le soluzioni vengono calcolate in un tempo inferiore al computo effettuato con soli 2 robot. Questo è dovuto alla porzione di grafo associata ad ogni robot, che è più piccola in termini di celle alle porzioni associate a soli 2 agenti.

Dunque, $\mathcal{D}_3$ e $\mathcal{D}_5$ sono modelli in grado di scalare egregiamente al crescere dei robot, ma il tempo di computazione è dipendente dalle dimensioni delle porzioni di grafo assegnate ad ogni robot; rappresentano, dunque, le combinazioni più adatte per risolvere ambienti di medie (e grandi) dimensioni. Per scegliere tra le due la soluzione migliore è necessario ispezionare la topologia.

Qualora $\mathcal{D}_3$ e $\mathcal{D}_5$ fossero entrambi inutilizzabili dal punto di vista compu-

	$\mathcal{D}_1$	$\mathcal{D}_2$	$\mathcal{D}_3$	$\mathcal{D}_4$	$\mathcal{D}_5$
ambiente1	18.80 s	0.38 s	0.01 s	< 0.01 s	< 0.01 s
ambiente2	1 h 1 m	29 m	2.95 s	0.03 s	0.06 s (0.01 s)
ambiente3	1 h 20 m	2 m 48.71 s	0.28 s	0.12 s	0.30 s (0.04 s)
ambiente4	memory over.	memory over.	8.54 s	1.50 s	4.21 s (0.66 s)
ambiente5	memory over.	memory over.	284.60 s	12.42 s	36.64 s (8.12 s)
media	1 h ($+10^4$%)	22 m ($+10^3$%)	59.01 (+600%)	2.81 s (-65%)	8.25 s (8.12 s)

Tabella 6.3: Tempi di elaborazione medi

	$\mathcal{D}_1$	$\mathcal{D}_2$	$\mathcal{D}_3$	$\mathcal{D}_4$	$\mathcal{D}_5$
ambiente1	1.00 (+51%)	1.00 (+51%)	0.75 (+13%)	0.53 (-20%)	0.66
ambiente2	1.00 (+127%)	0.65 (+52%)	0.49 (+9%)	0.40 (-10%)	0.44
ambiente3	1.00 (+121%)	1.00 (+121%)	0.52 (+21%)	0.35 (-19%)	0.43
ambiente4	memory over.	memory over.	0.41 (+36%)	0.21 (-30%)	0.30
ambiente5	memory over.	memory over.	0.35 (+10%)	0.15 (-53%)	0.32
media	1.00 (+117%)	0.88 (+92%)	0.55 (+18%)	0.26 (-19%)	0.46

Tabella 6.4: Utilità attesa media dei robot

tazionale, una possibile soluzione può essere rappresentata dall'individuare, tramite euristica, il miglior assegnamento per $\mathcal{D}_5$. Le nostre prove, infatti, dimostrano che il peggior assegnamento in $\mathcal{D}_5$ è sempre migliore della soluzione ottenuta in $\mathcal{D}_4$. Questo perchè, come già presente in letteratura, pattugliare porzioni sovrapposte di ambiente da parte di agenti non sincronizzati si rivela controproducente nella totalità dei casi.

In Tabella 6.5 riportiamo una tabella riassuntiva, con i vantaggi e gli svantaggi di ogni combinazione.

$\mathcal{D}_i$	Vantaggi	Svantaggi
$\mathcal{D}_1$	Garantisce la massima utilità attesa calcolabile	Oneroso, non è scalabile, non è riducibile
$\mathcal{D}_2$	Garantisce la massima utilità attesa calcolabile	Non è scalabile, ma è riducibile tramite dominanze
$\mathcal{D}_3$	Miglior compromesso coordinamento/tempo di calcolo. Scala all'aumentare del numero di robot e all'aumentare delle celle	Utilità attesa inferiore a $\mathcal{D}_1$ e $\mathcal{D}_2$
$\mathcal{D}_4$	Nessuno	Qualità della soluzione strettamente inferiore a $\mathcal{D}_5$
$\mathcal{D}_5$	Scala all'aumentare delle celle	Il tempo di calcolo è dipendente dal numero di assegnamenti da risolvere

Tabella 6.5: Tabella riassuntiva

Capitolo 7

Conclusioni

Siamo stati bravi, e abbiamo pure pubblicato

Il presente lavoro di tesi ha avuto come oggetto la progettazione di un modello strategico multiagente per il pattugliamento di ambienti di topologia arbitraria.

L'estensione del modello BGA ci ha permesso di stabilire un *upper bound* in termini di configurazioni del problema, discernendo tra configurazioni ammissibili e non ammissibili in base allo stato in cui ogni configurazione lascia gli obiettivi (se protetti o indifesi).

In seguito, abbiamo ricondotto un generico ambiente ad una rappresentazione astratta, chiamata *multigrafo etichettato*, che permette di rendere evidenti relazioni tra obiettivi. Una *clique etichettata* è un insieme di obiettivi che possono essere pattugliati da un singolo robot garantendo che, in caso di intrusione, la probabilità di percepire l'intruso sia 1. Il numero di clique etichettate massime (ovvero clique non contenute in altre clique) con cui è possibile coprire il multigrafo è il numero minimo di robot necessari a difendere l'intero ambiente.

Abbiamo proposto un algoritmo per l'enumerazione delle clique etichettate massime e abbiamo individuato due *dimensioni di coordinamento*: la *decomposizione topologica* basata su clique etichettate non necessariamente massime e la *decomposizione strategica* che indica il grado in cui la strategia di ogni robot debba dipendere da quella degli altri pattugliatori.

Abbiamo indicato differenti modelli, uno per ogni combinazione delle due dimensioni, che sono stati valutati tramite appositi esperimenti.

Basandoci sui test empirici, abbiamo potuto confermare alcuni risultati già presenti in letteratura, tra cui il trade-off tra il livello di coordinamento dei robot e la qualità della soluzione elaborata.

Abbiamo anche prodotto risultati scientificamente rilevanti, e per ogni ambiente siamo capaci di proporre una combinazione delle nostre dimensioni in grado di produrre una soluzione più che accettabile. Abbiamo analizzato, inoltre, la relazione tra coordinamento dei robot e complessità computazionale del generare la soluzione.

Lo sviluppo di questa tesi ha portato alla stesura di un paper dal titolo *Asynchronous Multi-Robot Patrolling against Intrusion in Arbitrary Topologies* pubblicato alla conferenza della *Association for the Advancement of Artificial Intelligence (AAAI) 2010*.

Rimangono, tuttavia, ancora inesplorate alcune problematiche: prima di tutto, non abbiamo proposto alcun algoritmo di scelta del partizionamento migliore in termini di clique massime e non massime; poi permangono dei dubbi sul computo delle strategie dominanti in caso non si scomponga il problema nè topologicamente nè strategicamente.

Inoltre, nel prossimo futuro è prevista la codifica dell'algoritmo di enumerazione delle clique etichettate in MATLAB, in modo da essere utilizzabile nell'ambito del progetto TALOS [7].

Bibliografia

[1] Noa Agmon, Sarit Kraus, and Gal A. Kaminka. Multi-robot perimeter patrol in adversarial settings. In *ICRA*, pages 2339–2345, 2008.

[2] Noa Agmon, Vladimir Sadov, Gal A. Kaminka, and Sarit Kraus. The impact of adversarial knowledge on adversarial planning in perimeter patrol. In *AAMAS (1)*, pages 55–62, 2008.

[3] Alessandro Almeida, Geber Ramalho, Hugo Santana, Patricia Azevedo Tedesco, Talita Menezes, Vincent Corruble, and Yann Chevaleyre. Recent advances on multi-agent patrolling. In *SBIA*, pages 474–483, 2004.

[4] R.J. Aumann and S. Hart, editors. *Handbook of Game Theory with Economic Applications*, volume 1 of *Handbook of Game Theory with Economic Applications*. Elsevier, April 1992.

[5] Nicola Basilico, Nicola Gatti, and Francesco Amigoni. Leader-follower strategies for robotic patrolling in environments with arbitrary topologies. In *AAMAS (1)*, pages 57–64, 2009.

[6] Nicola Basilico, Nicola Gatti, Thomas Rossi, Sofia Ceppi, and Francesco Amigoni. Extending algorithms for mobile robot patrolling in the presence of adversaries to more realistic settings. In *IAT*, pages 557–564, 2009.

[7] Nicola Basilico, Nicola Gatti, Pietro Testa, Francesco Amigoni, and Alessandro Busi. Talos: a web application for solving mobile robot patrolling situations with adversaries http://www.youtube.com/watch?v=0zbovco9hni, 2010.

[8] Coenraad Bron and Joep Kerbosch. Finding all cliques of an undirected graph (algorithm 457). *Commun. ACM*, 16(9):575–576, 1973.

[9] Yann Chevaleyre, François Sempé, and Geber Ramalho. A theoretical analysis of multi-agent patrolling strategies. In *AAMAS*, pages 1524–1525, 2004.

[10] Vincent Conitzer and Tuomas Sandholm. Computing the optimal strategy to commit to. In *ACM Conference on Electronic Commerce*, pages 82–90, 2006.

[11] Yehuda Elmaliach, Asaf Shiloni, and Gal A. Kaminka. A realistic model of frequency-based multi-robot polyline patrolling. In *AAMAS (1)*, pages 63–70, 2008.

[12] R. Fourer, D.M. Gay, and B.W. Kernighan. A modeling language for mathematical programming. *Management Science*, 36(5):519–554, 1990.

[13] Arnaud Glad, Olivier Simonin, Olivier Buffet, and François Charpillet. Theoretical study of ant-based algorithms for multi-agent patrolling. In *ECAI*, pages 626–630, 2008.

[14] ILOG Inc. http://ilog.com.sg/products/cplex, 2010.

[15] Aydano Machado, Geber Ramalho, Jean-Daniel Zucker, and Alexis Drogoul. Multi-agent patrolling: An empirical analysis of alternative architectures. In *MABS*, pages 155–170, 2002.

[16] James Pita, Manish Jain, Fernando Ordóñez, Christopher Portway, Milind Tambe, Craig Western, Praveen Paruchuri, and Sarit Kraus. Armor security for los angeles international airport. In *AAAI*, pages 1884–1885, 2008.

[17] Ross. The stanford encyclopedia of philosophy - game theory http://plato.stanford.edu/entries/game-theory/, 2010.

[18] M. Simaan and J. B. Cruz Jr, editors. *On the Stackelberg strategy in nonzero-sum games*. Springer Netherlands, 2005.

[19] Stanford Business Software Inc. http://www.sbsi-sol-optimize.com/, 2010.

[20] Bernhard Von Stengel and Shmuel Zamir. Leadership with commitment to mixed strategies. Technical report, 2004.

Appendice A

Origine della clique etichettata

Il contenuto di questa tesi è stato sviluppato sfruttando un modello a spirale; ovviamente, nello stendere l'elaborato abbiamo riportato solo la formulazione finale, trascurando completamente i passi intermedi che ci hanno permesso di giungere a tale formulazione. La formalizzazione del concetto di clique etichettata affonda le sue radici nell'algebra relazionale. Benchè ormai sorpassato, contenente imprecisioni ed inesattezze, può aiutare il lettore a comprendere quali siano stati gli *step* logici che ci hanno portato al formalismo definitivo.

Le limitazioni che hanno reso obsoleta questa rappresentazione sono principalmente due: è basato sull'assunzione che un cammino minimo sia sempre migliore di qualunque altro cammino e che tale cammino sia unico (non contempla archi doppi).

Definizioni Definiamo *relazione di equivalenza* una relazione riflessiva (1 sulla diagonale principale - autoanelli), simmetrica (matrice simmetrica, - archi non orientati), transitiva. Tutte le righe (colonne) uguali in tale matrice formano una partizione di equivalenza. All'interno di ogni partizione ci sono elementi (gli obiettivi) con le medesime caratteristiche. Le partizioni di equivalenza sono per definizione disgiunte. Il nostro obiettivo è scoprire se il nostro ambiente è riconducibile ad una relazione di equivalenza. Nel caso non lo fosse, dobbiamo controllare se esistano sotto-relazioni di equivalenza ed enumerarne tutte le possibili partizioni.

Zona di Interesse La zona di interesse di un obiettivo è l'insieme di celle da cui è possibile raggiungere l'obiettivo in $p \leq d$ turni. Un insieme di

obiettivi è una **Clique etichettata** quando:

- ogni obiettivo cade nella zona di interesse degli altri.
- esiste un cammino minimo tra tutte le possibili coppie di obiettivi le cui celle appartengono alle zone di interesse di tutti gli obiettivi della cricca e tale ogni cella del percorso è ancora nella zona di interesse comune a tutti gli obiettivi selezionati.

Relazione Copre Definiamo la relazione algebrica *copre* R, sull'insieme T degli obiettivi. Un obiettivo t_1 copre un obiettivo t_2 quando t_2 è nella zona di interesse di t_1 ed esiste un cammino minimo che collega i due obiettivi le cui celle sono nella zona di interesse di entrambi. Per costruzione, imponiamo che tale relazione sia riflessiva, e che quindi un obiettivo copra se stesso. Dire che le zone di interesse di due obiettivi hanno un'intersezione non nulla contenente entrambi gli obiettivi equivale a dire che la relazione R possiede la proprietà simmetrica. Inoltre, estendendo la proposizione a più di due obiettivi, equivale ad imporre la proprietà transitiva.

Qualora, dunque, la relazione R possieda queste caratteristiche (sia dunque, riflessiva, simmetrica, transitiva), R è una relazione di equivalenza. Questo ci permette di partizionare l'ambiente in base alle partizioni di equivalenza della relazione, che sono per definizione disgiunte. Ad ogni classe è associato un robot, e il problema multi-robot è decomponibile in problemi singolo-robot.

Esempio 1 $R_1 = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$

La matrice R_1 è una relazione di equivalenza (è riflessiva, simmetrica, transitiva). E' scomponibile in due partizioni di equivalenza (per definizione, disgiunte) ognuna delle quali pattugliabile da un singolo robot. Il problema è quindi duplicemente scomponibile: a livello topologico e a livello strategico. Inoltre, il partizionamento, assumendo di voler utilizzare il minimo numero di robot, è unico. Il numero minimo di robot necessario è 2, quante sono le partizioni.

Qualora, invece, la relazione R non sia una relazione di equivalenza, è comunque possibile trovare delle sottomatrici che posseggano tali proprietà. La matrice della relazione R sarà quindi scomponibile in più relazioni R_n; le partizioni di equivalenza di tutte le possibili relazioni di equivalenza rap-

presentano l'insieme di tutte le possibili aree in cui è potenzialmente scomponibile il setting. Come è chiaro, un obiettivo potrebbe fare parte di più partizioni di equivalenza appartenenti a distinte relazioni di equivalenza, imponendo dunque differenti partizionamenti.

Esempio 2 $R_2 = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$

La matrice R_2 contiene due relazioni di equivalenza. La prima comprende le prime tre righe e le prime tre colonne, ed è composta da due partizioni di equivalenza ($\{t_1, t_3\}$,$\{t_2\}$); la seconda è composta dalla terza e quarta riga e dalla terza e quarta colonna e contiene una sola partizione di equivalenza ($\{t_1, t_3\}$). Poichè il nodo t_3 appartiene a due partizioni di relazioni differenti, il problema è potenzialmente scomponibile sia livello strategico che a livello di setting.

La presenza di più cammini tra due obiettivi, e l'impossibilità di rappresentare l'ambiente come sovrapposizione di classi di equivalenzas, ha portato alla nascita del *multigrafo etichettato* e delle sue *clique etichetta*.

Appendice B

Documentazione del programma

Lo sviluppo di questa tesi ha contemplato l'implementazione di un programma che, dato un file di input in sintassi *AMPL-like* contenente la topologia dell'ambiente (obiettivi, e matrice di adiacenza ambientale) genera tutti i file *DAT* di AMPL necessari all'esecuzione dei vari modelli associati alle combinazioni delle dimensioni viste nell'elaborato.

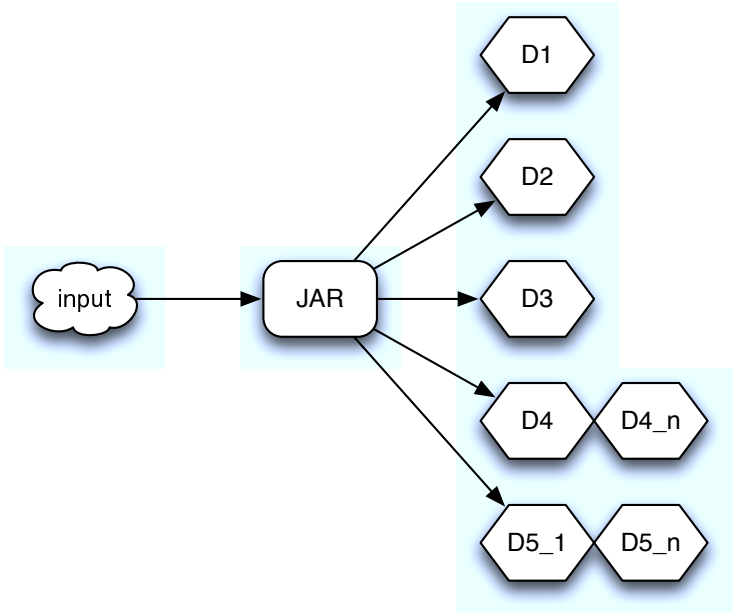

Figura B.1: Input - Output del programma

In figura B, possiamo vedere i file che vengono creati. I modelli $\mathcal{D}_4$

e $\mathcal{D}_5$ necessitano di più file (uno per ogni clique in cui viene partizionato l'ambiente) e dunque il software produce più output.

B.1 UML

La seguente sezione comprende, in breve, l'organizzazione del codice e dei package. E' importante notare che, poichè il software ha chiaramente natura di prototipo per la prova di algoritmi mai sperimentati e per i quali è già stata pianificata la codifica sotto forma di script MATLAB, potremmo non avere rispettato tutte le buone abitudini viste nei corsi di Ingegneria del Software.

B.1.1 Package

Il programma è composto dai seguenti package:

- default package: contiene i main utilizzati in fase di test.
- cliques: contiene l'implementazione dell'algoritmo BGV per la ricerca delle clique massime e tutte le strutture dati necessarie.
- config: contiene le classi che, una volta effettuato il parsing dell'input, istanziano gli oggetti relativi all'ambiente e agli obiettivi.
- dominanze: contiene algoritmi, sperimentali, per il computo delle dominanze.
- reductions: contiene le classi che si occupano di ridurre la rappresentazione in input in una rappresentazione ad oggetti delle varie combinazioni $\mathcal{D}_i$.
- shortestPath: contiene un'implementazione dell'algorimo di Dijkstra.
- tools: contiene classi di supporto principalmente adibite al parsing del file di input.

B.1.2 Class Diagram UML

Sono qui riportati i class diagram dei package più importanti

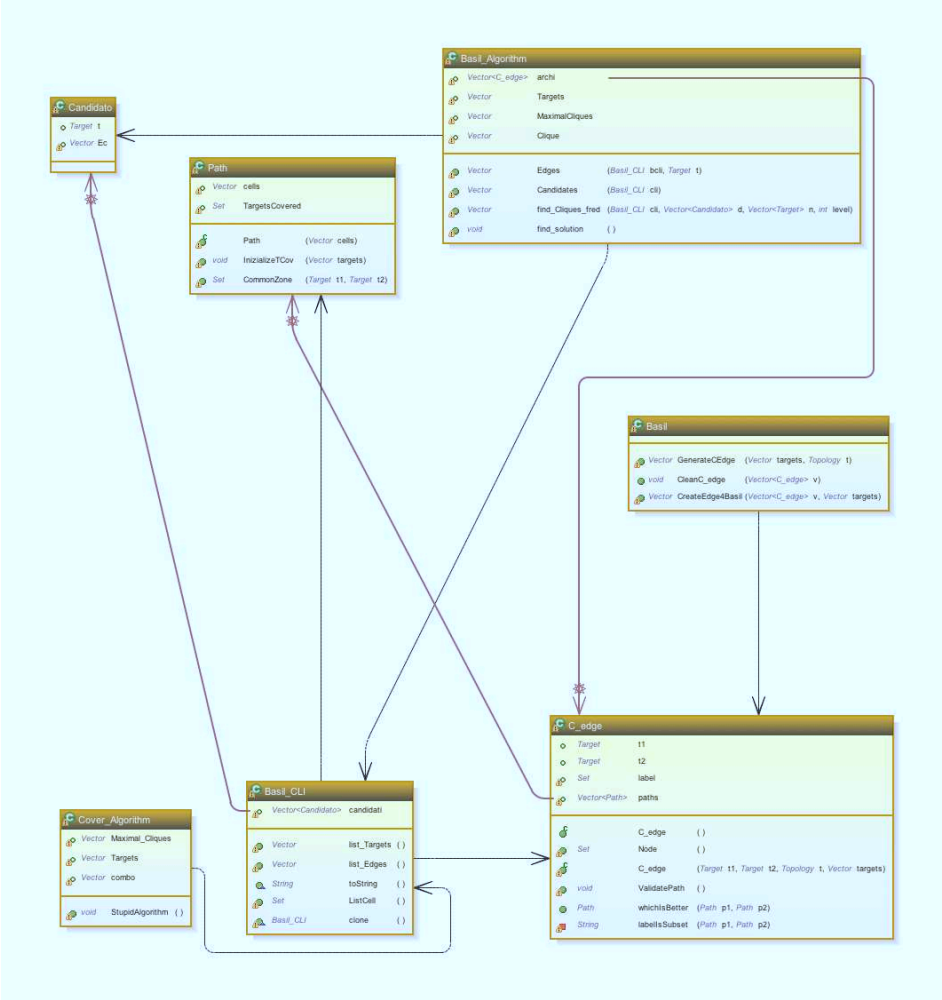

Figura B.2: clique

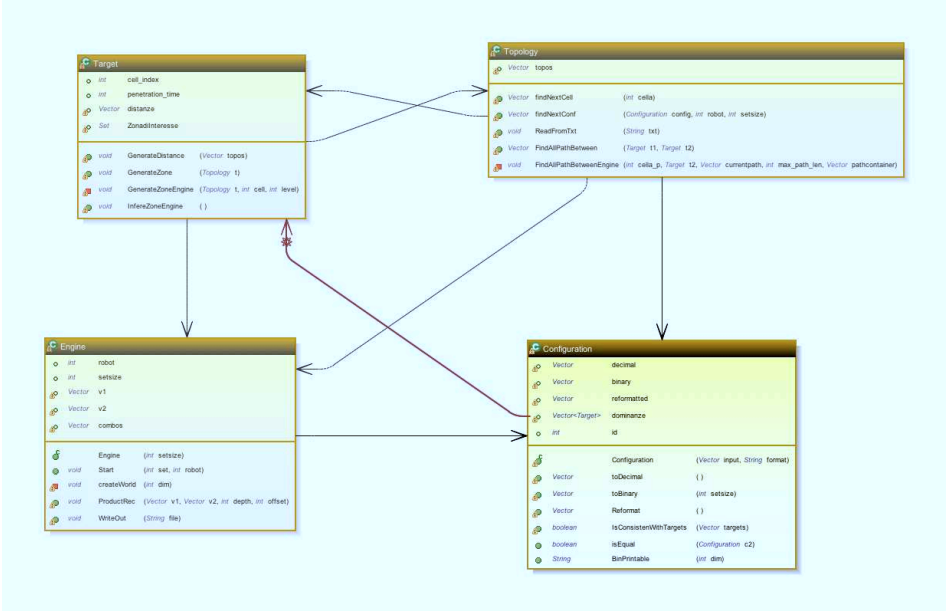

Figura B.3: config

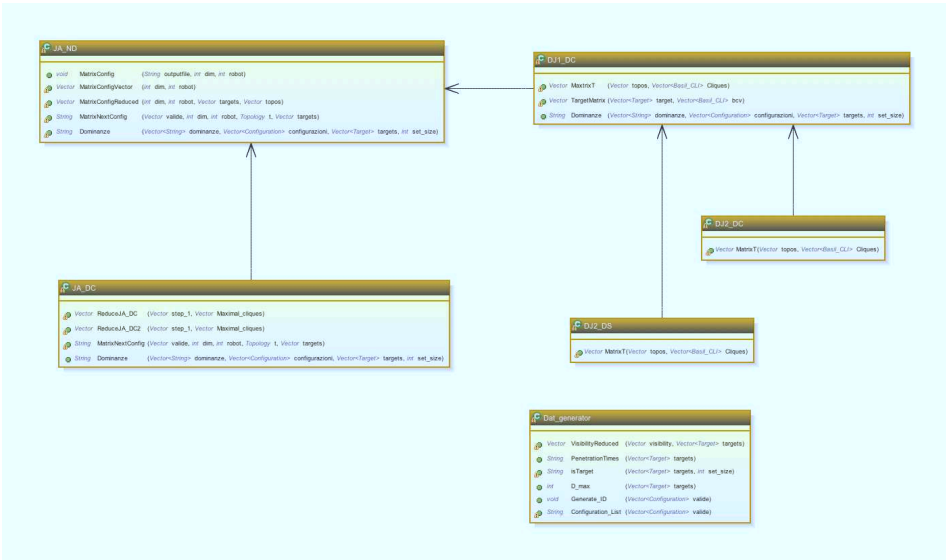

Figura B.4: reductions

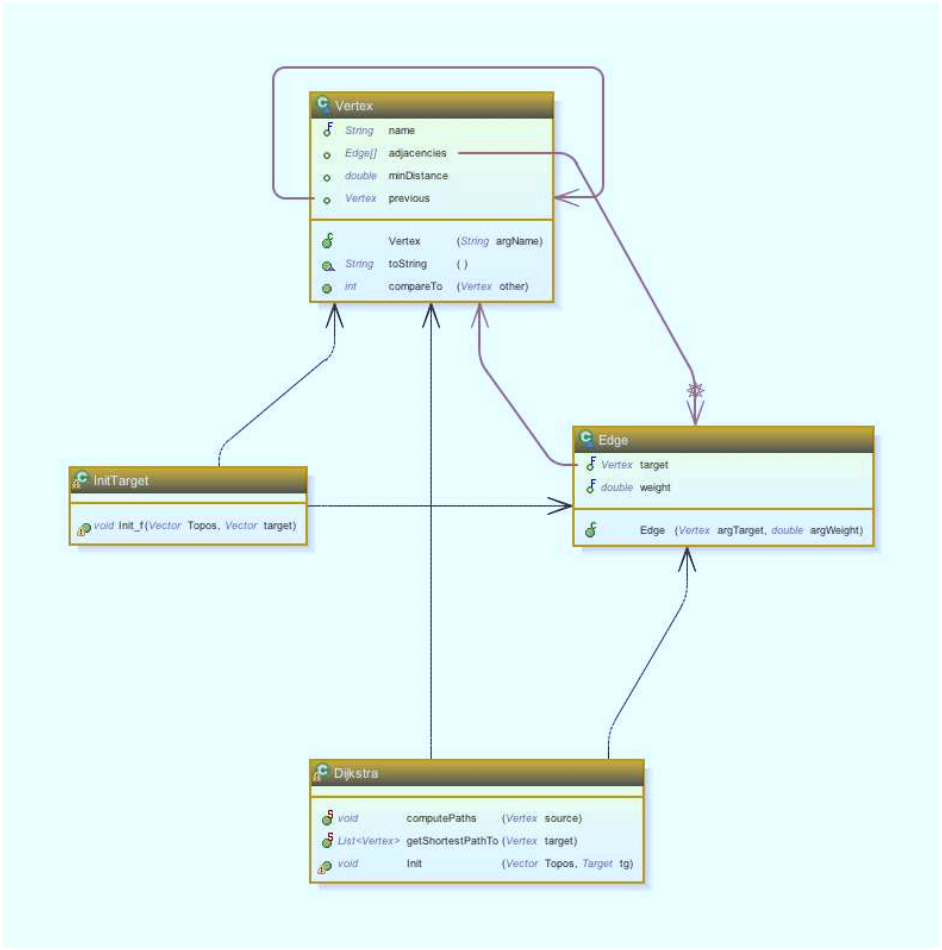

Figura B.5: shortestPath