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

Tecniche algoritmiche e strumenti software per lo studio di patrolling security games

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:
Pietro Testa, matricola 720861

Anno Accademico 2009-2010

In ricordo di mio nonno Domenico.

Sommario

Lo studio di tecniche per il pattugliamento di un ambiente mediante l'utilizzo di robot mobili autonomi è un'area dell'Intelligenza Artificiale oggetto di crescente interesse nell'ambito della ricerca scientifica per le molteplici applicazioni reali. Lo scopo della presente tesi è quello di formulare in primis dei modelli e dei metodi matematici che siano computazionalmente trattabili e allo stesso tempo abbastanza evoluti da descrivere situazioni reali, in secundis implementare una soluzione software completa che permetta, oltre che la creazione e risoluzione dei giochi, anche la valutazione, comparazione e condivisione dei risultati strategici ottenuti.

L'inapplicabilità computazionale dei metodi proposti in letteratura ci ha spinto a formulare delle tecniche che permettessero di semplificare il problema. Il primo approccio considerato è quello di ridurre la grandezza del gioco attraverso la rimozione delle *strategie dominate* di entrambi i giocatori. Malgrado tale metodo diminuisca drasticamente la grandezza del problema, la risoluzione di situazioni realistiche resta ancora complessa. Una tecnica efficace per la riduzione di giochi di grandi dimensioni è la *strategia astratta*. L'idea di base dietro il concetto di astrazione è quella di raggruppare insieme più azioni in una singola *macro azione*, permettendoci di ridurre la dimensione del gioco. La tipologia più interessante di astrazione è quella *senza perdita di informazione*, in quanto ci permette di trovare soluzioni ottimali; tuttavia, per problemi molto grandi, tali astrazioni continuano a produrre giochi troppo complessi per poter essere facilmente risolti; per tutti questi casi, rilassiamo i vincoli necessari a preservare l'informazione, formulando astrazioni del gioco definite come *astrazioni con perdita di informazione*, la cui soluzione tuttavia non è garantito essere ottimale.

La possibilità di risolvere istanze del problema di pattugliamento sempre più complesse e comparare le prestazioni delle strategie stesse al variare di parametri ha reso necessario lo sviluppo di uno strumento software che facilitasse, oltre che la creazione stessa dei patrolling setting, anche l'interpretazione delle strategie prodotte, altrimenti di difficile comprensione.

Ringraziamenti

Finalmente è arrivato il momento più atteso: la stesura dei ringraziamenti. Cercherò di essere breve e conciso, come sono sempre stato del resto.
Prima di tutto non posso che ringraziare il mio relatore Nicola Gatti ed il mio correlatore Nicola Basilico, grazie per tutto il tempo che mi avete dedicato, partecipare al vostro lavoro è stato per me un onore oltre che un piacere.
Ringrazio i miei genitori per avermi permesso di arrivare fino a questo momento e per essersi sempre interessati ai miei studi senza mai farmi sentire il fiato sul collo.
Ringrazio Francesca per essermi stata vicino in questi anni e per avermi sopportato durante le sessioni d'esame con i miei studi matti e disperatissimi, sei stata davvero un tesoro.
Ringrazio la mia cara nonna Tina per tutto l'interesse che ha sempre dimostrato per la mia carriera universitaria, tutte le preghiere e i ceri accesi sono davvero serviti a qualcosa, grazie di cuore.
Ringrazio mio fratello Matteo per avermi sempre aiutato.
Ringrazio Domenico per avermi accompagnato durante tutto questo cammino, speriamo che la strada sia ancora lunga per noi.
Ringrazio i miei compagni di banco, Fede, Antonio, Nicolas, Marco, Nicola e Adriano, compiere insieme a voi quest'impresa è stato divertente.
Ringrazio gli amici di Biella, per aver reso indimenticabili questi anni universitari, grazie a tutti voi per le belle serate passate insieme, non abbiatemene ma elencarvi tutti sarebbe stato impossibile.

Ad maiora!

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

"Tutti possono vedere le mie tattiche, nessuno può conoscere la mia strategia."

Sun Tzu - Arte della guerra

Lo studio di tecniche per il pattugliamento di un ambiente mediante l'utilizzo di robot mobili autonomi è un'area dell'Intelligenza Artificiale oggetto di crescente interesse nell'ambito della ricerca scientifica per le molteplici applicazioni reali in cui può essere impiegata: difesa di uffici, banche, accampamenti militari, sorveglianza di zone in condizioni estreme altrimenti difficili da raggiungere da un personale umano, o più in generale, ovunque possa rendersi necessario difendere una risorsa da un ipotetico ladro.

Gli approcci proposti sono molteplici e diverse sono le dimensioni in cui possiamo declinarli: dalla *funzione obiettivo* impiegata, alla tipologia di *ambiente* considerato, alla presenza all'interno del modello degli agenti intrusi e loro caratterizzazione.

La rappresentazione dell'ambiente può essere continua, come in [1], o basata su *grafi* attraverso una discretizzazione dell'ambiente come in [2], considerando ambienti perimetrali, completamenti connessi o arbitrari con specifici obiettivi situati al suo interno.

La funzione obiettivo può assumere diverse forme, in alcuni lavori come in [2] ci si pone come scopo quello di massimizzare la frequenza tra due successivi passaggi su di una determinata zona; altri [1] suggeriscono tecniche di copertura in cui lo scopo è di occupare al meglio una determinata area; altri ancora, come in [3, 4], introducono il concetto di *tempo di penetrazione*: il periodo necessario ad un intruso per violare determinate aree, in tale tipo di formulazione risulta quindi fondamentale che il tempo tra due successivi

passaggi sulla medesima zona, da parte del pattugliatore, sia inferiore al suo tempo di penetrazione, così da garantirne la totale sicurezza.

Il criterio di maggior rilievo tuttavia è senz'altro quello di considerare nel modello la presenza degli agenti intrusi e il modo in cui viene definito il loro comportamento, se *razionale* [5] o *non razionale*. Qualora l'intruso sia razionale, le tecniche appropriate da utilizzare sono quelle della *teoria dei giochi*. Il concetto di soluzione che ne deriva, noto anche come *equilibrio leader-follower* [6], è quello di un gioco (*security game*) tra due agenti: dove il pattugliatore è il leader e il ladro, che osserva le mosse del leader e sceglie la sua strategia in base ad esse, il follower; in questo modo si definisce il concetto di *pattugliamento strategico*, il cui fine è quello di formulare una soluzione che tenga in conto il fatto di avere un rivale razionale, che come tale, può agire in base all'osservazione del comportamento del pattugliatore.

Il pattugliamento è un problema che coinvolge diverse aree scientifiche, in questa tesi ci focalizziamo sui modelli e gli algoritmi atti a produrre soluzioni per il pattugliamento di tipo *strategico*, in ambienti di topologia arbitraria con obiettivi posti al suo interno. La maggior parte dei contributi spesso riutilizza metodi derivati da altre discipline, quali la *ricerca operativa*, che mal si adattano al problema del pattugliamento, specialmente se di tipo strategico, che necessita lo sviluppo di tecniche *ad-hoc*. La più grande lacuna rimane comunque la scarsa applicabilità dei metodi proposti, che non sono in grado di risolvere problemi di grandi dimensioni e che possano pertanto essere considerati verosimili.

L'approccio *Basilico-Gatti-Amigoni* (BGA) [7], sviluppato presso il Politecnico di Milano, è un modello valido che considera ambienti di topologia arbitraria, in un gioco non cooperativo a turni di tipo leader-follower; tale metodo inoltre considera, oltre che il già citato tempo di penetrazione, anche l'ipotesi che esista una priorità tra l'insieme degli obiettivi, che può anche essere diversa tra i due agenti rivali. Malgrado le solide basi che questo approccio ha posto per la formulazione del problema di pattugliamento, definendo un modello il più generico possibile, facilmente estensibile e che consideri la questione da un punto di vista strategico, soffre tuttavia anch'esso di scarsa applicabilità a problemi di grandi dimensioni, necessitando di ampie risorse per la computazione delle strategie.

Lo scopo della presente tesi è quello di formulare in primis dei modelli e dei metodi matematici che siano computazionalmente trattabili e allo stesso tempo abbastanza evoluti da descrivere situazioni reali, in secundis implementare una soluzione software completa che permetta, oltre che la creazione e risoluzione dei giochi, anche la valutazione, comparazione e condivisione dei risultati strategici ottenuti. L'obiettivo è quello di creare un

repository che possa identificarsi col tempo, come il punto di riferimento nei *patrolling security games*.

L'inapplicabilità computazionale dei metodi precedentemente proposti in letteratura ci ha spinto a formulare delle tecniche che permettessero di semplificare il problema. Il primo approccio considerato, allo scopo di migliorare l'efficienza, è quello di ridurre la grandezza del gioco attraverso la rimozione delle *strategie dominate* di entrambi i giocatori. Una strategia può considerarsi dominata quando assegna ad un'azione dominata una probabilità non nulla di essere scelta: un'azione a *domina* un'azione b quando l'utilità attesa nel giocare l'azione a è maggiore di b, indipendentemente dalle azioni del rivale.

Malgrado la rimozione delle strategie dominate diminuisca drasticamente la grandezza del problema in termini di vertici e numero delle best-response da parte dell'intruso, la risoluzione di situazioni realistiche resta ancora complessa. Una tecnica efficace per la riduzione di giochi di grandi dimensioni e che ha ricevuto molta attenzione nella letteratura è la *strategia astratta* [8, 9]. L'idea di base dietro il concetto di astrazione è quella di raggruppare insieme più azioni in una singola *macro azione*, permettendoci di ridurre la dimensione del gioco. La tipologia più interessante di astrazione è quella *senza perdita di informazione*, in quanto ci permette di trovare soluzioni ottimali risolvendo il gioco astratto. Tuttavia, l'applicazione dei risultati in [8] ai patrolling security games produce un gioco non a somma zero che non è possibile ridurre in alcun modo, spingendoci a definire delle astrazioni *ad-hoc*.

L'applicazione delle astrazioni senza perdita di informazione ha il potenziale di ridurre drasticamente la grandezza della maggior parte dei giochi, rendendoli computazionalmente trattabili. Tuttavia, per problemi molto grandi, questo tipo di astrazione continua a produrre giochi troppo complessi per poter essere facilmente risolti; per tutti questi casi, rilassiamo i vincoli necessari a preservare l'informazione, formulando astrazioni del gioco ridotte definite come *astrazioni con perdita di informazione*, la cui soluzione tuttavia non è garantito essere ottimale.

La possibilità di risolvere istanze del problema di pattugliamento sempre più complesse e simili a situazioni reali, ha reso necessario lo sviluppo di uno strumento software che facilitasse, oltre che la creazione stessa dei patrolling setting, anche l'interpretazione delle strategie prodotte, altrimenti di difficile comprensione. Tale strumento software è stato sviluppato anche con l'intento di permettere la comparazione delle prestazioni di diverse strategie di pattugliamento al variare di parametri.

La tesi è strutturata come segue. Nel Capitolo 2, oltre ad alcune no-

zioni generali sulla *teoria dei giochi* e i *modelli di Markov* necessari alla comprensione, è riassunto l'attuale stato dell'arte per il problema di pattugliamento, presentando alcuni degli approcci e delle metodologie impiegati per la sua risoluzione, sia da un punto di vista specifico al pattugliamento, esposto minuziosamente in quanto oggetto della presente tesi, che non specifico, presentato per completezza.

Nel Capitolo 3 è descritto in dettaglio il modello BGA enunciandone le assunzioni, il formalismo matematico impiegato e il concetto di soluzione, definendo il *patrolling security game*.

Nel Capitolo 4 affrontiamo il problema di ridurre la complessità computazionale per la risoluzione dei *patrolling setting*, esponendo alcune tecniche algoritmiche per il restringimento delle possibili strategie attuabili dagli agenti, mediante il concetto di *strategia dominata*; viene qui esposta anche l'idea di *astrazione* con e senza perdita di informazione per la semplificazione delle possibili azioni attraverso il loro raggruppamento in macro azioni.

Nel Capitolo 5 descriviamo in dettaglio la soluzione software implementata, il cui scopo è quello di facilitare, oltre che la creazione stessa dei patrolling setting, anche l'interpretazione delle strategie generate mediante l'ausilio di un'interfaccia grafica semplice e intuitiva.

Nel Capitolo 6 valutiamo l'efficacia delle tecniche proposte, presentando i risultati sperimentali ottenuti su più ambienti, considerandoli a risoluzione diverse e con differenti percentuali di obiettivi presenti sulla totalità dei nodi che compongono la mappa.

Nel Capitolo 7 concludiamo il nostro elaborato proponendo alcuni possibili sviluppi futuri ed estensioni al modello qui formulato, al fine di renderlo sempre più in grado di descrivere e risolvere situazioni reali.

Capitolo 2

Stato dell'arte

L'obiettivo del capitolo che segue è quello di introdurre il problema del pattugliamento strategico attraverso l'analisi degli approcci e delle metodologie finora impiegati per la sua risoluzione. A tal fine presentiamo alcune nozioni generali della *teoria dei giochi* e dei *modelli di Markov*, elencando le formulazioni dei giochi più impiegate, in quanto elementi fondamentali alla comprensione del lavoro svolto. Successivamente illustriamo *il problema del pattugliamento*, introducendo l'argomento attraverso l'esposizione di alcuni lavori specifici, seguono alcuni lavori affini che permetteranno di fornire un quadro più completo sull'attuale stato dell'arte. Se agli approcci non strategici dedichiamo relativamente poco spazio, i metodi strategici sono invece approfonditi in maniera più dettagliata in quanto oggetto della presente tesi.

2.1 La teoria dei giochi

Gli approcci che presentiamo più avanti sono formulazioni relativamente nuove che, per essere comprese, necessitano di una buona conoscenza della materia. A tal fine nelle prossime sezioni introduciamo i concetti fondamentali e le definizioni necessarie a comprendere non solo le soluzioni proposte in altri lavori, ma anche il lavoro svolto in questa tesi.

2.1.1 Nozioni fondamentali

Le basi della moderna teoria dei giochi sono state formulate per primi da John von Neumann e Oskar Morgensten [10] nel 1944: un matematico ed un economista cercano di descrivere matematicamente il comportamento umano in quelle situazioni in cui l'interazione fra diversi soggetti comporta la vincita o lo spartirsi di un qualche tipo di risorsa.

La teoria dei giochi è la scienza matematica che studia situazioni di conflitto o accordo tra diversi soggetti e ne ricerca soluzioni competitive o cooperative attraverso l'uso di modelli: ovvero lo studio delle decisioni individuali finalizzate al massimo guadagno, in situazioni in cui vi sono interazioni tra due o più agenti, tali per cui le decisioni prese da un soggetto possono influire sui risultati conseguibili da parte di un rivale o alleato.

In un gioco dunque ogni agente ha delle preferenze che vengono espresse mediante il concetto di *utilità*. La funzione di utilità mette in relazione un determinato stato o esito del gioco con un numero reale: maggiore è tale valore, maggiore sarà la soddisfazione raggiunta dall'agente. Più formalmente, chiamiamo S l'insieme degli stati del mondo che l'agente può percepire e definiamo la funzione di utilità u_i come

$$u_i : S \to \mathbb{R}$$

L'agente dunque, a partire da un determinato stato e conoscendo le possibili azioni attuabili, sceglierà come agire in base alla sua *speranza matematica*. Più formalmente parlando, chiamato A l'insieme delle possibili azioni, S l'insieme degli stati e $u_i(s)$ l'utilità dell'agente i nello stato s, possiamo definire *l'utilità attesa* del giocatore come

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

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

La mossa o l'insieme di azioni che l'agente intende attuare viene chiamata *strategia o politica*: una relazione tra stati ed azioni il cui obiettivo è, ad esempio, quello di massimizzare l'utilità attesa. La strategia di un agente è dunque un piano d'azione completo per tutta la durata del gioco, specifica un'azione per ogni circostanza in cui si è costretti ad agire: essa può essere *pura* 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.

2.1.2 Modelli decisionali di Markov

Un Processo Decisionale di Markov (MDP) [11] è un modello probabilistico stato-azione, con la proprietà aggiunta che la scelta dello stato successivo dipende unicamente dalle condizioni attuali e non da *come* si è giunti a tale stato. Si potrebbe dire che le proprietà del modello di Markov pongono l'agente in una condizione di "assenza di memoria", in cui ogni decisione viene presa in base a ciò che è possibile rilevare adesso e non in base a ciò

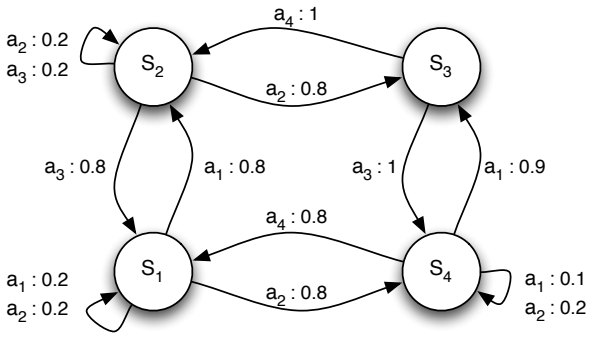


Figura 2.1: Un modello decisionale di Markov rappresentato graficamente con le azioni e le probabilità ad esse associate.

che è stato deciso in precedenza.

Un MDP è caratterizzato 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 $u_\mathbf{i} : S \to \mathbb{R}$. Volendo generalizzare, in un ambiente deterministico si avrebbe una funzione di transizione pari a zero per tutti gli stati s' a partire da una determinata coppia (s, a), eccetto che una, per la quale tale valore è pari a 1. Il che riprodurrebbe un effetto di assoluta prevedibilità, e dunque di scarso interesse strategico per il nostro lavoro. La Figura 2.1 mostra un esempio di rappresentazione grafica di un MDP con quattro stati: si noti come le azioni dell'agente siano limitate in base allo stato in cui si trova, per esempio nello stato s_1 è possibile solamente optare per le azioni a_1 o a_2. Inoltre si tenga presente che quando l'agente si trova nello stato s_1 e compie l'azione a_1 c'è una probabilità dello 0.2 di rimanere nello stesso stato.

Il comportamento dell'agente è descritto dalla sua *politica*, utilizzeremo il carattere π per identificarla. Lo scopo dell'agente sarà dunque quello di trovare la *strategia ottimale*, che massimizza la sua utilità attesa. Più formalmente possiamo definirla come:

$$\pi_i^*(s) = \arg\max_{a \in A} \; E[u_i, a, s].$$

Allo scopo di determinare il valore atteso mediante questa equazione, dobbiamo prima specificare come trattare le future utilità. Per questo motivo viene generalmente preferito considerare un'utilità ridotta, che ci permette di diminuire man mano l'impatto dei rinforzi futuri attraverso un *fattore di sconto*, generalmente indicato con γ, un numero tra zero e uno.

2.1.3 Tipologie di giochi

La premessa indispensabile per la teoria dei giochi è che tutti i soggetti devono essere a conoscenza delle regole del gioco, ed essere consapevoli delle conseguenze di ogni singola mossa, sia dal loro punto di vista, che da quello degli avversari o degli alleati. Quando la conoscenza tra agenti è uguale il gioco viene definito *in forma normale*. Possiamo facilmente rappresentare questo tipo di giochi mediante l'ausilio di una matrice contenente in ogni cella (i, j) i rinforzi ottenuti dagli agenti qualora il primo compia l'azione i e il secondo l'azione j.

Vengono invece definiti in *forma estesa* tutti quei giochi il cui processo decisionale può essere espresso mediante un albero, in cui ogni nodo rappresenta una scelta presa dall'agente, i rami sono le azioni stesse e i payoff sono presenti solamente sulle foglie. Questa forma consente di modellare esplicitamente le interazioni in cui un giocatore fa più di una mossa durante il gioco, e la sua mossa può dipendere da diversi stati.

Esistono vari concetti di soluzione attraverso i quali è possibile classificare un gioco, di seguito riportiamo i principali.

Maxmin.

Questo tipo di soluzione assume che in un gioco l'agente scelga sempre l'azione che gli permette di massimizzare la minore utilità che può ottenere. Più formalmente, in un gioco in cui agiscono due agenti i e j, la strategia *maxmin* dell'agente i è data da

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

Nei giochi a somma zero, quando in ogni cella la somma delle utilità dei due giocatori è 0, ovvero quando un agente vince e l'altro perde proporzionalmente, la strategia maxmin coincide con la miglior soluzione del gioco.

Social welfare.

Un altro comune tipo di soluzione è il social welfare, in cui si cerca di massimizzare la somma dei payoff di tutti gli agenti. Tuttavia questo tipo di approccio non permette di trovare soluzioni stabili, poiché ogni agente razionale che considera solamente la propria utilità, potrebbe decidere di cambiare la propria strategia con un'altra che gli consente di ottenere un payoff maggiore, danneggiando di conseguenza gli altri agenti.

Pareto ottimalità.

Le soluzioni di questo tipo derivano direttamente dal concetto di *Pareto ottimalità*: una strategia s è detta Pareto-ottimale se non esiste un'altra strategia s' tale per cui almeno un agente ha un'utilità maggiore in s' e nessuno ha un'utilità minore in s'. Equivalentemente possiamo dire che non esiste una politica che permetta a degli agenti di migliorare senza danneggiare gli altri. Appare evidente come anche questo tipo di soluzione non possa definirsi stabile: un agente infatti potrebbe sempre avere interesse a giocare la strategia migliore per se, pur sapendo di danneggiare gli altri.

Equilibrio di Nash.

La mancanza di stabilità venne risolta da *John Nash* che propose la propria definizione di equilibrio. Una strategia s è quindi 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. Equivalentemente, quando tutti gli avversari hanno giocato l'equilibrio di Nash, per l'agente la strategia migliore è a sua volta quella di scegliere l'equilibrio di Nash. Egli ha dimostrato che tutti i giochi in forma normale hanno almeno un equilibrio di Nash ma che esso può essere in strategie miste.

Equilibrio leader-follower.

Esistono particolari giochi, noti come giochi di Stackelberg, dove un agente, il leader, sceglie una strategia, il follower invece, prima di scegliere, osserva il comportamento del leader e preferirà, tra tutte le strategie attuabili, quella che massimizza la sua utilità. In realtà, qualora il follower fosse indeciso tra due strategie, potrebbe anche scegliere di massimizzare l'utilità del leader a patto che questa scelta porti a un equilibrio. In [6] è stato dimostrato che nei giochi a somma zero, l'equilibrio leader-follower coincide con l'equilibrio di Nash. Nei giochi non a somma zero invece, la strategia di Stackelberg è ottima per il leader quando il follower risponde giocando la sua strategia ottimale. Risulta infatti evidente che il leader può ottenere un risultato buono almeno quanto quello ottenuto dall'equilibrio di Nash, perché di fatto sta imponendo una strategia a lui favorevole. Tuttavia, l'esistenza dei due equilibri è indipendente e non è possibile dedurne uno avendo prima ricavato l'altro.

2.2 Il problema del pattugliamento

Letteralmente per pattugliamento s'intende "l'atto di muoversi attorno ad un'area ad intervalli regolari per proteggerla e sorvegliarla da eventuali intrusi". Negli ultimi anni la ricerca ha affrontato con sforzo crescente il problema della sorveglianza di un'area definita o di specifici obiettivi in essa contenuti. Le applicazioni pratiche sono ovviamente notevoli: difesa di uffici, banche, sorveglianza di zone altrimenti difficili da raggiungere per un personale umano o in condizioni estreme, difesa di accampamenti in ambito militare, o più in generale, ovunque possa rendersi necessario difendere un obiettivo dall'intrusione di un ipotetico ladro. Diverse sono le aree scientifiche coinvolte, ma tra tutte noi ci focalizziamo sugli algoritmi per produrre strategie di pattugliamento.

Ad alto livello il problema può essere ricondotto ai seguenti elementi che lo compongono:

- uno o più robot pattugliatori;
- un'area in cui tali robot possono muoversi;
- un insieme di obiettivi, posti all'interno dell'area, che devono essere protetti;
- un intruso che tenta di violare un determinato obiettivo e deve essere fermato.

Possiamo invece distinguere due differenti tipologie di pattugliamento, come evidenziato in Figura 2.2:

- *pattugliamento perimetrale*, uno o più robot impediscono l'accesso ad una determinata area muovendosi lungo il perimetro della stessa;
- *pattugliamento ad obiettivi*, lo scopo dei robot è quello di difendere determinati obiettivi posti all'interno dell'area e potrebbero essere definiti dei livelli di priorità differenti per ciascuno di essi.

Va sottolineato come sia possibile ricondurre una tipologia all'altra in determinate situazioni, immaginando ad esempio di avere degli obiettivi posizionati in modo da poterli sorvegliare in un ciclo, oppure avendo dei target così concentrati da poter essere difesi girando loro intorno; tuttavia le due tipologie di pattugliamento presentano problematiche differenti e diverse difficoltà da affrontare, specialmente nel caso multi-agente.

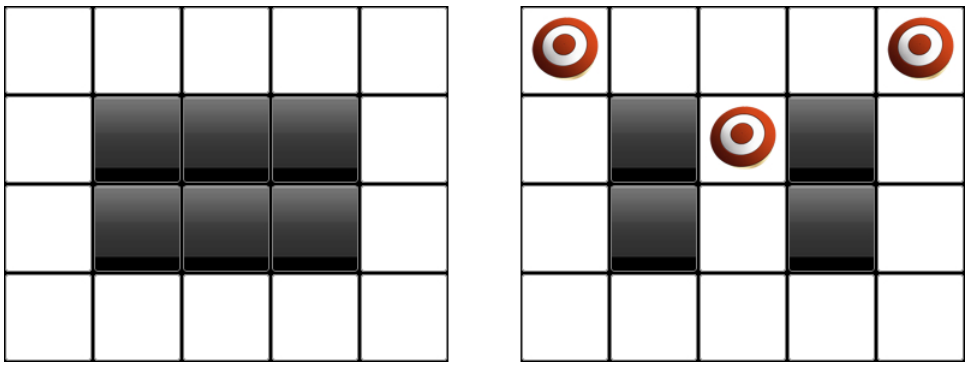


Figura 2.2: Un esempio di pattugliamento perimetrale (sinistra) e ad obiettivi (destra).

2.3 Approcci al pattugliamento

Una buona strategia può essere definita, in maniera del tutto informale, come quella che permette di minimizzare il tempo tra due passaggi successivi su uno stesso nodo, per tutti i nodi di cui abbiamo un certo interesse.

La sezione seguente riporta i principali metodi usati per risolvere il problema, distinguendo tra approcci specifici al *pattugliamento*, dove si cerca di modellare il comportamento dell'intruso mediante la teoria dei giochi, e approcci affini al problema di pattugliamento, basati spesso su *information covering* e tecniche mutate da altre discipline.

2.3.1 Lavori specifici al pattugliamento

Gli algoritmi per il pattugliamento possono essere raggruppati secondo tre dimensioni principali. La prima è inerente alla *rappresentazione* dell'ambiente da pattugliare, che può essere basata su *grafi* o *continua*, nel primo caso possiamo a sua volta distinguere quattro sotto categorie:

- *perimetrale*, chiusa o aperta;
- *completamente connessa*, in cui ogni vertice è connesso agli altri;
- *arbitraria*;
- *arbitraria con obiettivi*, dove questi ultimi costituiscono un sottoinsieme dei vertici di maggior interesse.

La seconda dimensione che possiamo considerare è la *funzione obiettivo* del pattugliatore. Essa può considerare esplicitamente la presenza di *avversari* o meno. In assenza di rivali, vengono impiegate diverse funzioni

obiettivo, quali ad esempio quelle basate su copertura, dove lo scopo è di coprire al meglio una determinata area, quelle basate su frequenza, il cui scopo è di sorvegliare un'area con una determinata frequenza che soddisfi determinate proprietà e altri tipi di funzioni ad-hoc o combinazioni multiple. In particolare possiamo identificare quattro approcci differenti basanti sulla frequenza di visita:

- *uniforme:* (anche chiamata *blanket time*), dove ogni vertice deve essere visitato con la stessa frequenza data;
- *media massima* (o *average idleness*), in cui lo scopo è di massimizzare appunto la frequenza media di visita per ciascuna cella;
- *maxmin*, (o *worst idleness*), nella quale è massimizzata la minima frequenza di visita;
- *vincoli di posizione specifica*, in cui ogni singola cella ha un limite inferiore di frequenza di visita che deve essere rispettato.

Nel caso in cui venga esplicitamente considerata la presenza di rivali, possiamo distinguere due casi differenti:

- *utilità attesa con avversario fisso*, in cui i rinforzi previsti per il pattugliatore sono massimizzati partendo da un modello dell'avversario fisso e pertanto non razionale;
- *utilità attesa con avversario razionale*, nel quale il rivale viene modellato come un agente razionale che prende decisioni strategiche per massimizzare la propria utilità.

La terza dimensione che possiamo considerare è il numero di pattugliatori adottato, singolo o multi agente. La Tabella 2.1 mostra una classificazione dei maggiori lavori nell'ambito della ricerca scientifica che andremo ora ad analizzare in breve.

			basati su grafi				continui
			perimetrali	completamente connessi	arbitrari	arbitrari con obiettivi	
senza avversari	copertura		singolo agente				[1]
			multi-agente				[12], [13], [14]
		basati su frequenza					
		uniforme (blanket time)	singolo agente	[2]	[15], [16], [17], [18], [19], [20]		
			multi-agente				
		media massima (average idleness)	singolo agente	[2]	[16], [17], [18], [19]		
			multi-agente				
		maxmin (worst idleness)	singolo agente	[2]	[21], [16], [17], [18], [19]		
			multi-agente				
	altri		singolo agente		[22], [23]		
			multi-agente				
con avversari	utilità attesa con avversario fisso		singolo agente	[24]	[25]		
			multi-agente				
	utilità attesa con avversario razionale		singolo agente	[3], [28] [29], [30]	[26]	[27]	
			multi-agente				

Tabella 2.1: Tabella riassuntiva sullo stato dell'arte del problema di pattugliamento.

In [14] e [13], sono presentati sistemi composti da più veicoli aerei che pattugliano una zona di confine. L'ambiente è rappresentato come una regione continua a due dimensioni, a sua volta suddivisa in sotto regioni, a ognuna di esse è assegnato un veicolo che la pattuglia con una traiettoria a spirale.

Anche [12] considera il pattugliamento multi-robot in ambienti continui. In questo caso l'ambiente viene suddiviso impiegando le regioni di Voronoi, ogni agente è assegnato a un'area da pattugliare, in modo da ottenere una copertura completa con la minima sovrapposizione tra i diversi agenti. I movimenti dei robot sono modellati da una rete neurale che permette di interagire con ambienti dinamici.

In [1], allo scopo di coprire un determinato ambiente, vengono tracciate imprevedibili traiettorie caotiche.

Il lavoro in [18] si occupa del pattugliamento multi-agente di vertici su grafi i cui archi hanno lunghezza unitaria. Diverse architetture per gli agenti sono confrontate sperimentalmente in base alla loro efficacia nel ridurre al minimo l'idleness.

L'approccio è stato generalizzato in [16], dove sono considerati archi di lunghezza arbitraria, e analizzato da un punto di vista teorico in [17].

Anche l'apprendimento per rinforzo è stato proposto come un metodo per coordinare gli agenti di pattuglia e guidarli all'interno dell'ambiente [19].

Il lavoro in [2] fornisce algoritmi efficienti per trovare strategie di pattugliamento multi-agente in perimetri (chiusi o aperti), che minimizzano diverse nozioni di idleness, mentre il lavoro in [21] calcola in modo efficiente le strategie minimizzando la peggior idleness su grafi arbitrari.

Anche l'approccio in [15] considera il pattugliamento multi-agente su grafi, ma l'obiettivo è quello di pattugliare i bordi perimetrali di un'area e non determinati vertici; la funzione obiettivo usata è quella uniforme. L'algoritmo proposto è dimostrato convergere ad un ciclo Euleriano in un numero finito di passi e di visitare i nodi in un periodo di tempo finito. Un lavoro simile è mostrato in [20].

In [22] e in [23] gli autori considerano il pattugliamento multi-agente nel caso in cui la funzione obiettivo tenga in considerazione diversi criteri sfruttando tecniche derivate dai MDP.

Finora abbiamo considerato, come criteri di valutazione, la frequenza di visita di una cella di particolare interesse, e eventuali rinforzi positivi o negativi che l'agente può ricevere. Il lavoro esposto in [3] invece introduce il concetto di *tempo di penetrazione*: un possibile intruso può accedere ad un qualsiasi vertice, ma è necessaria una determinata quantità di tempo per riuscirvi, misurata in turni. Con questa formulazione risulta fondamentale

che il tempo tra due successivi passaggi sulla medesima cella, da parte del pattugliatore, sia inferiore al tempo di penetrazione di quella stessa cella, così da garantirne la sua totale sicurezza. L'elaborato illustra un efficiente algoritmo in tempo polinomiale per risolvere strategie di pattugliamento perimetrali in un modello di gioco teorico multi-agente. Gli agenti non hanno tuttavia alcun tipo di preferenza rispetto ai vertici e l'intruso preferirà quindi introdursi in quei nodi in cui la probabilità di essere catturato è minima. Il problema è essenzialmente un gioco a somma zero la cui soluzione è una strategia maxmin.

In [28] e in [30] il modello viene esteso considerando l'incertezza sull'attività sensoriale dei pattugliatori, mentre in [24] vengono utilizzati agenti non razionali (fissi).

In [26] l'autore considera un ambiente descritto mediante un grafo completamente connesso, in cui intruso e sorvegliante possono avere preferenze diverse tra i vertici ed è proposto un algoritmo efficiente per calcolare l'equilibrio di Nash del gioco.

Il lavoro in [25] assume tre comportamenti diversi per l'intruso: casuale, con avversario che sceglie sempre di penetrare un nodo visitato di recente da un sorvegliante, e con avversario che usa metodi statistici per predire le probabilità che un nodo verrà visitato in futuro. Le strategie di pattugliamento sono state valutate sperimentalmente attraverso la simulazione, dimostrando che nessuna politica è ottimale per tutti i possibili avversari. In [27] gli autori si focalizzano su ambienti di topologia arbitraria fornendo un approccio euristico per calcolare le strategie ottimali degli agenti.

2.3.2 Lavori affini al pattugliamento

Nella ricerca operativa si possono trovare un ampio numero di lavori strettamente correlati al problema di pattugliamento, la letteratura li presenta per lo più come varianti del problema del commesso viaggiatore (TSP). Tecnicamente parlando, queste opere sono simili ai problemi di pattugliamento basati su grafi e frequenze, ma la funzione obiettivo che adottano non è impiegabile per lo scopo di sorveglianza, come evidenziamo in questa sezione presentando i principali lavori.

La prima estensione del problema TSP che consideriamo è chiamata *deadline-TSP*, nella quale vengono presi in considerazione dei vincoli temporali [31]. In questo tipo di problemi i vertici hanno una scadenza temporale a partire dalla loro visita e occorre del tempo per attraversare gli archi che li uniscono. Quando un vertice viene visitato prima della sua scadenza viene assegnato un rinforzo positivo, mentre ne viene ricevuto uno negativo quan-

do un vertice viene attraversato dopo la sua scadenza o quando non viene visitato del tutto. L'obiettivo è quello di trovare un percorso che massimizzi l'utilità, visitando più vertici possibile. Tuttavia, differentemente da come accade nel pattugliamento, il rinforzo viene ricevuto solamente alla prima visita.

Una variante più generale è quella del *Vehicle Routing Problem with Time Windows* [32], dove le scadenze sono rimpiazzate da finestre temporali fisse, durante le quali deve avvenire la visita dei vertici del grafo. In questo tipo di problema la finestra temporale non dipende dalle visite del pattugliatore, come accade invece nella sorveglianza strategica.

Sequenze cicliche di visita sono attribuite nelle tecniche di *Period Routing Problem* [33, 34], in cui gli itinerari dei veicoli vengono costruiti per funzionare in un periodo di tempo limitato, durante il quale ogni vertice deve essere visitato almeno un numero predefinito di volte.

Nel *Cyclic Inventory Routing Problem* [35] i vertici rappresentano dei clienti con una determinata capacità di stoccaggio e uno specifico tasso di domanda. L'obiettivo è quello di trovare un percorso tramite il quale il distributore può ripetutamente rifornire i clienti sotto alcuni vincoli di frequenza della visita.

Infine, possiamo citare che problemi simili al pattugliamento sono stati studiati in *pursuit-evasion* e in *hide-and-seek* [36]. Praticamente, in entrambi i casi, il fuggitivo può nascondersi in un qualsiasi vertice di un grafo arbitrario, mentre la guardia può muoversi all'interno del grafo per cercarlo entro un tempo finito. In ogni caso, alcune assunzioni, incluso il fatto che l'obiettivo del fuggitivo è solamente quello di evitare la cattura e non di violare una determinata area di interesse, rendono entrambi i problemi non direttamente comparabili con il problema di pattugliamento strategico che stiamo considerando.

Nella prossima sezione descriviamo in maniera più approfondita due modelli di recente formulazione evidenziandone le novità principali da essi introdotte.

2.3.3 Il modello AK

Il primo modello a prendere ispirazione dalla teoria dei giochi per risolvere il problema del pattugliamento di tipo strategico, è stato presentato da Agmon e Kaminka in [3] e si basa su [2]. Gli autori affrontano il problema del pattugliamento multi-agente in un ambiente perimetrale, dove gli agenti si muovono attorno ad un'area chiusa. Pur presentandosi con qualche limitazione, questo lavoro introduce numerose e interessanti novità rispetto

alle precedenti formulazioni, le più importanti sono senz'altro l'utilizzo di un modello per la strategia dell'intruso e l'introduzione del suddetto *tempo di penetrazione*.

Vengono abbandonate le strategie deterministiche: è infatti facile dimostrare che in moltissimi ambienti un intruso può violare una determinata cella con probabilità 1, nel momento in cui conosce il percorso seguito dal pattugliatore. Immaginiamo di avere k robot coordinati tra loro, omogenei dal punto di vista del modello, che si muovono lungo una linea spezzata aperta. Tale linea è divisa in N segmenti, non necessariamente di uguale lunghezza, che vengono percorsi dai robot in un *time cycle*, il che significa che i segmenti possono essere considerati uniformi dal punto di vista del tempo, ma non necessariamente nella lunghezza. A loro volta dividiamo gli N segmenti in $d = N/K$ sezioni. Ad ogni ciclo, i pattugliatori, sincronizzati tra loro, devono decidere dove andare: se proseguire lungo il percorso o voltarsi e tornare indietro, perdendo alcuni turni extra ($\tau \geq 1$) nella medesima cella per effettuare l'inversione.

L'algoritmo proposto si basa sulla probabilità p che i robot continuino per la loro strada e $q = 1 - p$ che decidano di voltarsi, descrivendo quindi un ambiente non deterministico. L'intruso invece, deve decidere all'istante 0 quale cella violare impiegando un numero di turni t. Rispetto agli studi precedenti, in cui i pattugliatori si muovono su percorsi ciclici, come ad esempio l'approccio basato su TSP, la scelta di studiare il pattugliamento su linee spezzate aperte complica notevolmente il problema. Il tempo che intercorre tra due successivi passaggi su un determinato obiettivo può addirittura raddoppiarsi rispetto a quello necessario in un ciclo a causa, come già citato, del tempo dovuto all'inversione di marcia.

L'obiettivo è quello di 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 significa che il sistema ha la certezza di visitare ogni segmento ogni $r \leq t$ turni, rilevando sempre l'intruso.

Una volta calcolato il PPD di tutti i segmenti possiamo scegliere quale modello utilizzare per l'avversario. Scegliamo di massimizzare la minima PPD, quando decidiamo di utilizzare un modello *forte* a *conoscenza completa* da parte dell'avversario. Nel modello *debole* invece l'avversario sceglie casualmente e con probabilità uniforme, in quale segmento entrare, assumendo quindi di trovarci in un gioco a *conoscenza incompleta*.

Le limitazioni del modello sono evidenti: non vi è alcuna priorità tra i diversi obiettivi in quanto il tempo di penetrazione è univoco; inoltre la soluzione multi-agente è una banale estensione del modello singolo, con robot

equidistanti che si muovono lungo un perimetro.

2.3.4 Il modello DOBSS

Un altro interessante lavoro che vogliamo approfondire è il *Decomposed Optimal Bayesian Stackelberg Solver* (DOBSS) [4] che ha trovato una sua evoluzione e applicazione reale in ARMOR [37]: un sistema di sicurezza per l'aereoporto di Los Angeles il cui scopo è di assistere la polizia nel posizionamento delle risorse, quali cani poliziotto e punti di controllo, all'interno dei diversi terminal.

Nella teoria dei giochi un *gioco Bayesiano* è un modello in cui le informazioni dei giocatori sulle caratteristiche degli avversari risultano incomplete. Lo scopo degli autori è quello di formulare una strategia mista per il pattugliatore (il leader), che può essere osservata da un altro agente (il follower) prima di scegliere la propria strategia. In questo tipo di giochi, in cui il leader è incerto su quale tipo di avversario può incontrare, trovare la sua strategia ottimale è un problema NP-complesso, per questo motivo viene fornita una funzione euristica per limitare le possibili strategie del leader, selezionando tutte quelle azioni le cui probabilità sono multiplo di $1/k$, con k un numero intero fissato.

L'introduzione di un modello leader-follower, in cui viene descritta l'idea che l'intruso possa osservare il comportamento dei sorveglianti e agire di conseguenza, rappresenta un'interessante novità; tuttavia il sistema si limita a considerare la disposizione di risorse del tutto *statiche* all'interno di un ambiente, il che rende l'approccio abbastanza limitativo.

2.4 Considerazioni

Nonostante l'indubbio interesse che la sorveglianza ricopre e nonostante il problema sia stato affrontato da numerosi gruppi di ricerca in tutto il mondo, ognuno dei quali ha utilizzato approcci spesso molto diversi tra loro, il più delle volte sono stati riutilizzati risultati ottenuti da studi in campi limitrofi, che mal si adattano al pattugliamento, che necessita di soluzioni *ad-hoc*. Abbiamo infatti evidenziato come ognuno dei modelli proposti dalla letteratura risulti lacunoso sotto determinati aspetti.

Non si è mai cercato, ad esempio, di valutare le strategie di pattugliamento sotto prospettive diverse, come se fosse un problema multi-obiettivo; non è mai stata valutata la *robustezza* del metodo considerato a fronte di piccole variazioni dei parametri di ingresso e, per la maggior parte dei casi presentati, non si è mai cercato di applicare queste tecniche a problemi di

grandi dimensioni, che possano descrivere in questo modo situazioni il più reali possibile.

Nel prossimo capitolo presentiamo in dettaglio il modello BGA, il cui obiettivo è di andare incontro a diverse carenze fin qui esposte, presentando un modello il più generale possibile, facilmente estensibile e che affronti il problema del pattugliamento in maniera strategica.

Capitolo 3

Il modello BGA

In questo capitolo descriviamo in maniera dettagliata il modello *Basilico-Gatti-Amigoni* (BGA) [7], sviluppato presso il Politecnico di Milano, il quale ha dimostrato di essere il modello più generale nell'ambito della ricerca scientifica, per il problema di pattugliamento con avversari. Iniziamo presentando in dettaglio il concetto di *patrolling security game*: le assunzioni previste, il *patrolling setting* e il modello del gioco. In seguito descriviamo il *concetto di soluzione* e una formulazione di programmazione matematica per la risoluzione del gioco.

3.1 Patrolling security games

Qui riportiamo le *assunzioni* previste dal modello nella Sezione 3.1.1, il *patrolling setting* nella Sezione 3.1.2 e il modello del gioco nella Sezione 3.1.3.

3.1.1 Assunzioni del modello BGA

Sono qui elencate tutte le ipotesi teoriche necessarie alla formulazione del modello:

- il tempo è *discreto* e misurato in turni;
- l'agente pattugliatore in grado di percepire un eventuale intruso è singolo;
- l'ambiente è *discretizzato* in celle, la sua topologia è rappresentata mediante un grafo orientato;
- per ciascun obiettivo all'interno della mappa è definito un *tempo di penetrazione*, perfettamente conosciuto da entrambi i giocatori. In

una situazione reale potrebbe al massimo essere stimato, ad esempio, analizzando l'integrità di una porta o di una finestra in un determinato nodo;

- i movimenti e la localizzazione del robot pattugliatore sono supposti *privi di errori*, il che non è chiaramente vero in una situazione reale, anche se tale assunzione non è limitativa in quanto è possibile estendere il formalismo in modo da considerare situazioni più complesse;

- l'intruso è supposto essere un *best-responder*, vale a dire un agente razionale che massimizza la sua utilità data la strategia del pattugliatore, ciò significa che è il rivale più forte possibile. In una situazione reale potrebbe quindi essere anche più debole;

- l'intruso conosce *esattamente* la strategia del patroller. In una situazione reale potrebbe al più derivarne un'approssimazione data dall'osservazione, altrimenti sarebbe necessario un tempo infinito;[1]

- l'intruso può apparire direttamente su un determinato obiettivo, anche se vedremo un'estensione che prevede invece di definire degli specifici *punti di ingresso*.

3.1.2 Patrolling setting

Un *patrolling setting* è composto da un ambiente che deve essere sorvegliato, dalle caratteristiche sensoriali e dalla capacità di azione del pattugliatore e dell'intruso. Nel modello BGA l'ambiente è composto da un insieme di n celle, la cui topologia è descritta mediante l'utilizzo di un grafo orientato $G = (V, A, T, v, d)$, dove V è l'insieme dei vertici che costituiscono l'ambiente, T è il set di celle che devono essere sorvegliate, A rappresenta gli archi che uniscono i nodi e che definiscono quindi la topologia dell'ambiente attraverso la funzione $a : V \times V \to \{0, 1\}$; ad esempio se $a(i, j) = 1$ allora i nodi i e j risultano essere adiacenti sulla mappa in quanto è presente un arco che li unisce, gli agenti quindi possono muoversi direttamente da i a j in un turno.

Ad ogni cella è associato una coppia di valori non negativi che rappresentano il livello di priorità di quel nodo per il pattugliatore e per l'intruso. Le strategie messe in atto nell'ambiente dai due agenti vengono definite sulla

[1] Si tenga presente che conoscere la strategia del pattugliatore non significa conoscere quale sarà la sua prossima azione, bensì solamente quale sia la distribuzione di probabilità con cui un'azione potrebbe essere scelta.

base di questi due valori $v_{\mathbf{p}}(t)$ e $v_{\mathbf{i}}(t)$, che denotano i *payoff* del pattugliatore e dell'intruso rispettivamente. Tutti questi valori possono essere impostati liberamente, con i vincoli che:

$$0 \leq v_{\mathbf{p}}(i) \leq v_{\mathbf{p}}(0) \quad \forall i \in T \tag{3.1}$$
$$v_{\mathbf{i}}(0) \leq 0 \leq v_{\mathbf{i}}(i) \quad \forall i \in T \tag{3.2}$$
$$v_{\mathbf{p}}(i) = v_{\mathbf{p}}(0) \quad \forall i \in C/T \tag{3.3}$$
$$v_{\mathbf{i}}(i) = 0 \quad \forall i \in C/T \tag{3.4}$$

Tutti i nodi a valori positivi per entrambi i giocatori vengono chiamati *obiettivi* (o *target*), $T \subseteq V$. Tutti gli altri vertici del grafo (in $V \setminus T$) fanno parte dei percorsi che il pattugliatore attraversa per muoversi tra i diversi target.

Ogni tentativo di intrusione in uno specifico target necessita di un determinato intervallo di tempo (misurato in turni), definito come il *tempo di penetrazione*. La funzione $d : T \to \mathbb{N} \setminus \{0\}$ assegna ad ogni target tale valore.

In Figura 3.1 possiamo vedere un esempio di patrolling setting in cui l'insieme dei target è $T = \{06, 08, 12, 14, 18\}$; in ogni target viene inoltre riportato il valore per $d(t)$ e la coppia $(v_{\mathbf{p}}(t), v_{\mathbf{i}}(t))$.

Il *sensore* del robot è modellato tramite la funzione $S : V \times T \to [0, 1]$ dove $S(i, t)$ indica la probabilità con cui il pattugliatore, trovandosi nel vertice i, è in grado di percepire un intruso nel target t.

3.1.3 Modello del security game

Un gioco viene definito formalmente attraverso il meccanismo che determina le regole del gioco e le preferenze degli agenti, specificando quali siano le strategie messe in atto.

Il modello BGA sfrutta un gioco in forma estesa, più precisamente un *gioco ripetuto*, in cui ad ogni turno entrambi i giocatori, pattugliatore e intruso, agiscono contemporaneamente. Il patroller sceglie la successiva cella in cui muoversi tra quelle adiacenti tramite l'azione $move(j)$ dove $j \in V$ è un nodo adiacente a quello corrente. L'intruso, qualora non abbia ancora deciso di entrare nell'ambiente, può scegliere se penetrare in una cella tramite l'azione $enter(t)$ con $t \in T$ o se aspettare un turno mediante l'azione *wait*. Nel momento in cui l'intruso decide di giocare l'azione $enter(t)$ al turno k non gli è permesso di prendere ulteriori decisioni per i successivi $d(t) - 1$ turni.

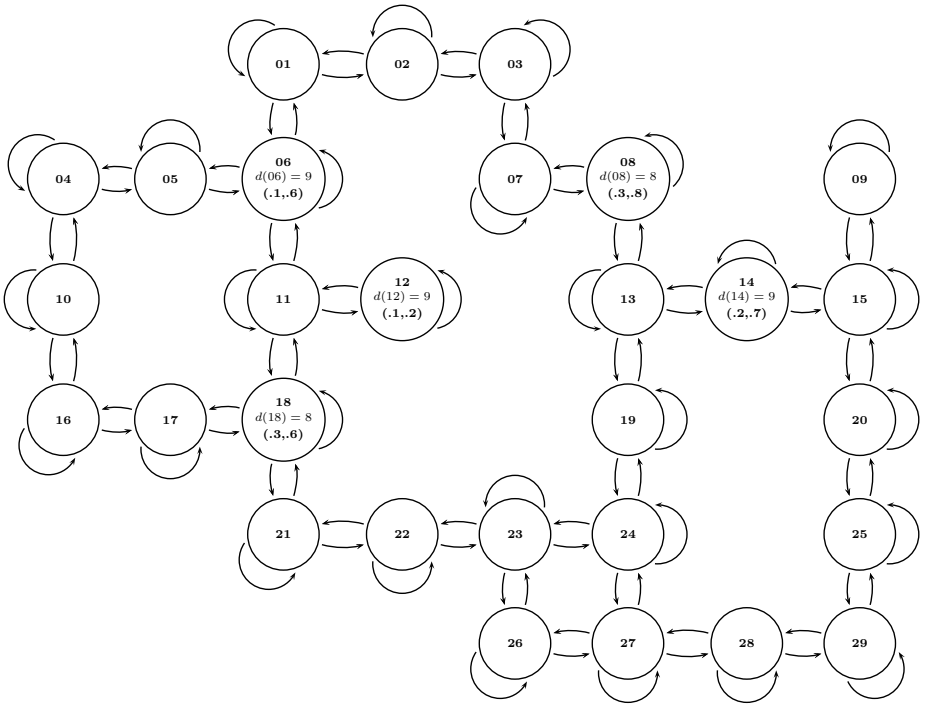


Figura 3.1: Un esempio di grafo orientato che descrive un patrolling setting.

Il gioco possiede altre due caratteristiche: è a *informazione imperfetta*, poiché il patroller non può sapere se l'intruso è in attesa di entrare o se sta già forzando un obiettivo, ed è a *orizzonte infinito* poiché, teoricamente, l'intruso può decidere di attendere indefinitamente al di fuori dell'ambiente. I risultati possibili del gioco sono tre:

- *no-attack:* quando l'intruso sceglie di aspettare ad ogni turno k mediante l'azione *wait*, evitando di entrare nell'ambiente;
- *intruder-capture:* quando l'intruso gioca l'azione *enter(t)* al turno k e il pattugliatore riesce a rilevarlo nella finestra temporale $I = \{k, k + 1, \ldots, k + d(t) - 1\}$;
- *penetration-t:* quando l'intruso non viene rilevato nella finestra temporale I appena citata e l'intruso riesce nel suo intento di violare un obiettivo.

Le utilità dei due agenti sono calcolate come segue: l'utilità del pattugliatore, chiamata $u_{\mathbf{p}}$, viene definita come la sommatoria dei rinforzi previsti per tutti i target preservati. Assumendo che l'agente sia neutrale al rischio abbiamo che:

$$u_{\mathbf{p}}(x) = \begin{cases} \sum_{i \in T} v_{\mathbf{p}}(i) & x = \textit{intruder-capture o no-attack} \\ \sum_{i \in T \setminus \{t\}} v_{\mathbf{p}}(i) & x = \textit{penetration-t} \end{cases}.$$

Si noti che il pattugliatore riceve la stessa utilità sia nel caso in cui l'intruso venga catturato, sia nel caso in cui non vi sia alcun ingresso. Questo perché se venisse data un'utilità maggiore in caso di cattura, l'agente preferirebbe una scelta casuale tra *intruder-capture* e *penetration-t* rispetto a *no-attack*. Ovviamente tale comportamento non sembra essere ragionevole, in quanto lo scopo primario del pattugliatore è quello di preservare il maggior valore possibile, nei termini di obiettivi, e non catturare necessariamente l'intruso.

L'utilità dell'intruso è denotata con $u_{\mathbf{i}}$: nel caso in cui venga catturato riceve una penalità, altrimenti riceve un rinforzo pari al valore del target attaccato. Assumendo che l'intruso sia neutrale al rischio, abbiamo che:

$$u_{\mathbf{i}}(x) = \begin{cases} 0 & x = \textit{no-attack} \\ v_{\mathbf{i}}(t) & x = \textit{penetration-t} \\ -\epsilon & x = \textit{intruder-capture} \end{cases},$$

dove $\epsilon \in \mathbb{R}^+$ è la penalità dovuta alla cattura. Questo significa che l'agente preferirà non attaccare piuttosto che essere catturato.

Gli autori propongono l'uso di un vettore H, definito come la sequenza di tutte le possibili successioni h di vertici visitati da parte del pattugliatore. Per esempio in Figura 3.1, partendo dal presupposto che il pattugliatore sia nel nodo 01, una possibile storia è $h = \langle 01, 02, 03, 07, 08 \rangle$.

Possiamo definire la strategia del pattugliatore come $\sigma_{\mathbf{p}} : H \to \Delta(V)$ dove $\Delta(V)$ è una distribuzione di probabilità sui vertici V. Data una storia $h \in H$, la strategia $\sigma_{\mathbf{p}}$ ci fornisce le probabilità, rispetto ai vertici adiacenti, con cui l'agente si sposterà al prossimo turno. Si noti che la strategia del pattugliatore non dipende dalle azioni prese dall'intruso, che come già detto, risultano non osservabili da parte dell'agente.

Possiamo distinguere tra strategie *deterministiche* e *non-deterministiche*. Quando $\sigma_{\mathbf{p}}$ è in strategie *pure*, assegnando una probabilità pari a 1 ad un singolo vertice per tutte le storie h possibili, diciamo che la strategia è deterministica; altrimenti diciamo che è non-deterministica.

Definiamo la strategia dell'intruso come $\sigma_{\mathbf{i}} : H \to \Delta(T \cup \{wait\})$ dove $\Delta(T \cup \{wait\})$ è la distribuzione di probabilità tra i vertici T e l'azione *wait*.

Esempio 3.1.1 *In Figura 3.1, una strategia deterministica potrebbe essere quella di seguire il ciclo* $\langle 04, 05, 06, 11, 18, 17, 16, 10, 04 \rangle$*, mentre una non-deterministica, nel caso in cui il pattugliatore sia nel vertice* 01*, data una*

storia h potrebbe essere:

$$\sigma_{\mathbf{P}}(h) = \begin{cases} 01 & \text{con una probabilità di } 0.25 \\ 02 & \text{con una probabilità di } 0.25 \\ 06 & \text{con una probabilità di } 0.5 \end{cases}.$$

Una strategia di esempio per l'intruso potrebbe essere: giocare wait per tutte le storie per cui l'ultimo vertice non è 04 *e giocare enter*(18) *altrimenti.*

3.2 Concetto di soluzione

Immaginando che un intruso abbia la possibilità di aspettare indefinitamente, osservando il comportamento del pattugliatore, e scelga la sua strategia in base a quella del robot sorvegliante, possiamo ricondurre il problema alla definizione leader-follower vista in precedenza, dove il patroller è il leader, e l'intruso è il follower. L'obiettivo è quello di trovare una strategia di pattugliamento ottimale che induca l'intruso a scegliere di non penetrare mai nell'ambiente.

La presenza di un orizzonte infinito complica notevolmente lo studio del problema. Introducendo simmetrie di gioco, l'azione da compiere è decisa in base alle ultime $|H|$ azioni, con $|H|$ costante durante tutto il gioco e non superiore ad un valore fissato l. Ad esempio quando $l = 0$, la cella in cui muoversi non dipende dalla cella in cui si trova il pattugliatore; con $l = 1$ il modello è Markoviano. Maggiore è la cardinalità dell'insieme H, maggiore è la complessità computazionale necessaria per trovare una strategia. Fissato $|H|$, tuttavia, il gioco si riduce ad una forma strategica che si ripete ogni $|H|$ turni. Non sorprende il fatto che, aumentando il valore di l, l'utilità attesa del pattugliatore non diminuisce mai: questo perché dispone di maggiori informazioni per scegliere la sua prossima azione.

In questo tipo di gioco le azioni per il patroller possono essere scritte nella forma $\{\alpha_{h,i}\}$, dove $\alpha_{h,i}$ è la probabilità di eseguire l'azione $move(i)$ data la storia h. Le azioni dell'intruso invece, possono essere ridotte a *enter-when*(t, h), con $t \in T$ ed $h \in H$: ovvero penetrare nella cella t dopo che il pattugliatore ha compiuto la serie di azioni h; e *stay-out*: eseguire l'azione *wait* all'infinito.

3.3 Calcolo dell'equilibrio

Al fine di trovare un equilibrio leader-follower del patrolling security game, gli autori introducono dei vincoli che forzino la strategia del pattugliatore a ripetersi ogni l turni. Per semplicità viene riportata la formulazione matematica nel caso più semplice, con $l = 1$, ovvero, come già sottolineato, sotto ipotesi Markoviane. Tali vincoli permettono di calcolare le probabilità con cui l'intruso verrà catturato e la corrispondente utilità attesa per ogni possibile azione. Viene definita con $P_c(t, h)$ la probabilità di cattura relativamente all'azione *enter-when*(t, h): ovvero la probabilità che il pattugliatore raggiunga l'obiettivo t partendo da h nei prossimi $d(t)$ turni. Tale probabilità dipende non linearmente da $\alpha_{h,i}$. Viene invece definita con $\gamma_{i,j}^{w,t}$ la probabilità che il pattugliatore raggiunga il vertice j in w turni, partendo dal nodo i ma senza passare attraverso l'obiettivo t.

Il problema ha la seguente formulazione matematica:

$$\alpha_{i,j} \geq 0 \qquad \forall i, j \in V \tag{3.5}$$

$$\sum_{j \in V} \alpha_{i,j} = 1 \qquad \forall i \in V \tag{3.6}$$

$$\alpha_{i,j} \leq a(i,j) \qquad \forall i, j \in V \tag{3.7}$$

$$\gamma_{i,j}^{1,t} = \alpha_{i,j} \qquad \forall t \in T, i, j \in V \setminus \{t\} \tag{3.8}$$

$$\gamma_{i,j}^{w,t} = \sum_{\substack{x \in V \setminus \{t\}, \\ w \leq \rho(x,t)}} \left(\gamma_{i,x}^{w-1,t} \alpha_{x,j}\right) \qquad \begin{array}{c} \forall w \in \{2, \ldots, \rho(j,t)\}, \\ t \in T, i, j \in V, j \neq t \end{array} \tag{3.9}$$

$$P_c(t,i) = 1 - \left(\sum_{j \in V \setminus \{t\}} \gamma_{i,j}^{\rho(j,t),t} + \sum_{j \in V \setminus \{t\}} \sum_{w \leq \rho(j,t)-1} \gamma_{i,j}^{w,t} \sum_{\substack{x \in V \setminus \{t\}, \\ w \geq \rho(x,t)}} \alpha_{j,x} \right) \qquad \begin{array}{c} \forall t \in T, \\ i \in V \end{array} \tag{3.10}$$

I Vincoli (3.5) e (3.6) impongono che le probabilità $\alpha_{i,j}$ siano ben formate; il Vincolo (3.7) impone che il pattugliatore possa muoversi solo tra nodi adiacenti; i Vincoli (3.8) e (3.9) esprimono le ipotesi di Markovianità sulle decisioni strategiche del pattugliatore; il Vincolo (3.10) definisce la probabilità $P_c(t, h)$. Si noti che nei Vincoli (3.9) e (3.10) è stato introdotto un limite superiore $\rho(j, t)$ sull'indice w di ogni variabile $\gamma_{i,j}^{w,t}$. Il termine nel Vincolo (3.10) racchiuso tra parentesi rappresenta la probabilità di successo dell'intruso quando gioca l'azione *enter-when*(t, i); il primo addendo considera tutti i possibili percorsi che iniziano da i e finiscono in j dopo

esattamente $\rho(j,t)$ turni; il secondo addendo invece considera tutti i percorsi che finiscono in un nodo x ad un turno $w+1 > \rho(x,t)$, dopo che al turno w è stato visitato il vertice j senza aver raggiunto il corrispondente limite superiore $\rho(j,t)$.

3.3.1 Giochi strettamente competitivi

Quando per tutti gli obiettivi $s,t \in T$ abbiamo che, se $v_{\mathbf{p}}(s) \geq v_{\mathbf{p}}(t)$, allora $v_{\mathbf{i}}(s) \geq v_{\mathbf{i}}(t)$ e *viceversa*, il gioco può essere definito strettamente competitivo (*stricly-competitive*). Ciò significa che pattugliatore e intruso hanno lo stesso ordine di priorità sugli obiettivi, anche se assegnano valori differenti ad ogni target; un'assunzione che risulta essere ragionevole nella maggior parte dei casi reali.

La formulazione del problema di programmazione matematica è definita come segue:

Formulazione 3.3.1 *L'equlibrio leader-follower di un gioco strettamente competitivo è la soluzione di :*

$$\max u$$

t.c.

$$\text{vincoli (3.5), (3.6), (3.7), (3.8), (3.9), (3.10)}$$

$$u \leq u_{\mathbf{p}}(\textit{intruder-capture})P_c(t,h) + u_{\mathbf{p}}(\textit{penetration-t})(1-P_c(t,h)) \quad \forall t \in T, h \in V \qquad (3.11)$$

Il Vincolo (3.11) definisce u come il limite inferiore dell'utilità attesa del pattugliatore. Risolvendo questo problema otteniamo il massimo limite inferiore u^* (maxmin). Il valore delle variabili $\{\alpha_{i,j}\}$ corrispondenti a u^* rappresentano la strategia di pattugliamento ottimale.

3.3.2 Giochi non strettamente competitivi

Nel caso in cui il gioco non sia strettamente competitivo, l'algoritmo è sviluppato in due passi. Il primo permette di controllare l'esistenza di almeno una strategia per la quale la best-response dell'intruso è *stay-out*; se tale strategia esiste, l'utilità attesa del pattugliatore è massima.

Formulazione 3.3.2 *L'equilibrio leader-follower per cui la miglior strategia dell'intruso è stay-out esiste se il seguente problema di programmazione matematica ammette una soluzione:*

$$\text{vincoli (3.5), (3.6), (3.7), (3.8), (3.9), (3.10)}$$

$$u_{\mathbf{i}}(\textit{intruder-capture})P_c(t,h) + u_{\mathbf{i}}(\textit{penetration-t})(1 - P_c(t,h)) \leq 0 \quad \forall t \in T, h \in V \tag{3.12}$$

Nel caso in cui il problema ammetta una soluzione si ottiene un vettore di probabilità $\alpha_{i,j}$ che definiscono una possibile strategia per il patroller. Quando non è possibile trovare una soluzione al problema, l'algoritmo procede con il secondo passo, il cui obiettivo è quello di trovare la *best-response* dell'intruso che massimizzi l'utilità attesa del pattugliatore. Il problema è formulato nel seguente modo:

Formulazione 3.3.3 *La maggior utilità attesa per il pattugliatore quando la best response dell'intruso è enter-when*(s,q) *e la soluzione di:*

$$\max \quad u_{\mathbf{p}}(\textit{penetration-s})(1 - P_c(s,q)) + u_{\mathbf{p}}(\textit{intruder-capture})P_c(s,q)$$

t.c.

$$\text{vincoli (3.5), (3.6), (3.7), (3.8), (3.9), (3.10)}$$

$$\begin{aligned} u_{\mathbf{i}}(\textit{intruder-capture})(P_c(s,q) - P_c(t,h)) + u_{\mathbf{i}}(\textit{penetration-s})(1 - P_c(s,q)) - \quad & \forall t \in T, \\ u_{\mathbf{i}}(\textit{penetration-t})(1 - P_c(t,h)) \geq 0 \quad & h \in V \end{aligned} \tag{3.13}$$

Il problema appena formulato è riferito ad una determinata coppia di vertici (s,q), dunque per massimizzare l'utilità attesa del pattugliatore il calcolo deve essere eseguito per ogni azione pura dell'intruso. Il vincolo (3.13) dice che nessuna azione *enter-when*(t,h) assicura all'intruso un'utilità maggiore dell'azione *enter-when*(s,q).

Esempio 3.3.4 *Riportiamo in Figura 3.2 la strategia corrispondente all'equilibrio leader-follower per il patrolling-setting di Figura 3.1. I vertici che non vengono mai visitati sono stati omessi; la best-response per l'intruso è enter-when*$(08,12)$*.*

Nel capitolo che segue presenteremo in dettaglio le tecniche da noi proposte allo scopo di semplificare la risoluzione dei problemi di pattugliamento, in taluni casi mantenendo l'ottimalità della soluzione trovata, in altri massimizzando l'utilità attesa ma senza mantenerne l'ottimalità.

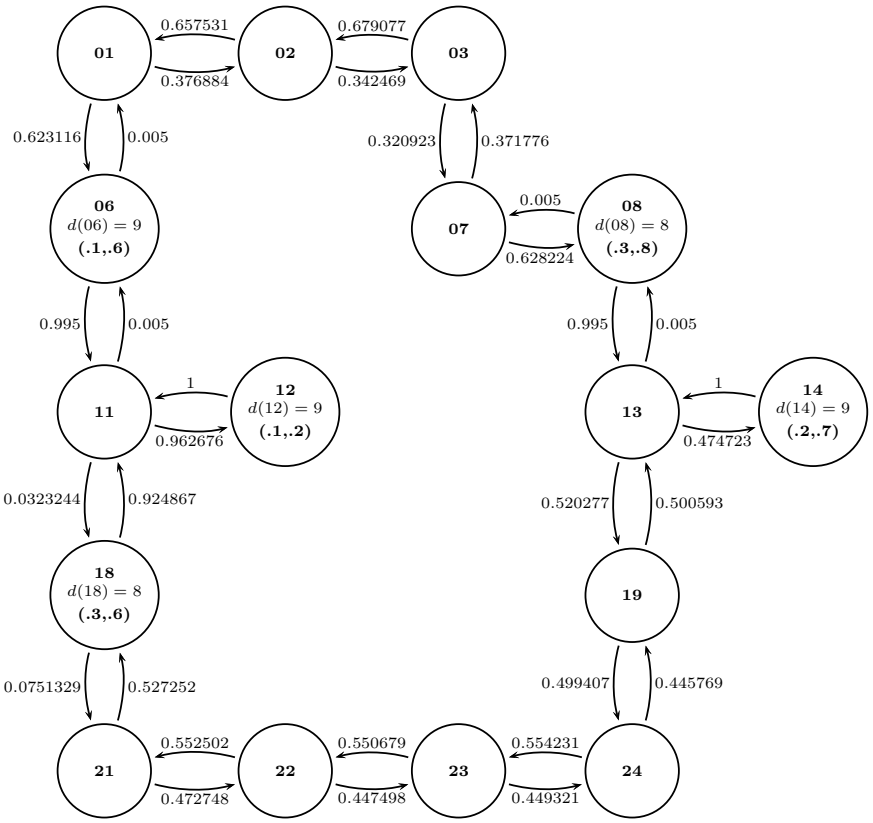


Figura 3.2: Strategia di pattugliamento ottimale per la Figura 3.1.

Capitolo 4

Tecniche di riduzione

Dato che istanze di una certa rilevanza sono computazionalmente gravose da risolvere, vengono proposte in questo capitolo alcune tecniche di riduzione allo scopo di rendere il problema più trattabile. Nella Sezione 4.1 viene introdotto il concetto di *strategia dominata* e la conseguente riduzione del problema per il pattugliatore e l'intruso. Nelle Sezioni 4.2 e 4.3 vengono presentate le *astrazioni* del gioco, rispettivamente, senza e con perdita di informazione.

4.1 Eliminazione delle strategie dominate

Il primo approccio considerato per migliorare l'efficienza è quello di ridurre la grandezza del gioco attraverso la rimozione delle *strategie dominate* di entrambi i giocatori. Una strategia può considerarsi dominata quando assegna ad un'azione dominata una probabilità non nulla di essere scelta: un'azione a *domina* un'azione b quando l'utilità attesa nel giocare l'azione a è maggiore di b, indipendentemente dalle azioni del rivale. Al fine di rimuovere le strategie dominate, vanno quindi identificate e a loro volta rimosse, tutte le azioni dominate, ottenendo un gioco equivalente ma di dimensioni minori, con una conseguente riduzione della complessità computazionale necessaria a risolverlo.

Qui di seguito riportiamo nella Sezione 4.1.2 e Sezione 4.1.1, due tecniche per rimuovere le azioni dominate per il pattugliatore e per l'intruso rispettivamente, nella Sezione 4.1.3 discutiamo invece la possibilità di iterare il processo di rimozione per il pattugliatore.

4.1.1 Semplificazioni per il pattugliatore

Perché un'azione *move(j)* del pattugliatore possa essere considerata dominata e pertanto rimossa, deve soddisfare la seguente condizione: nel caso in cui venga eliminata dalle azioni attuabili, l'utilità attesa del pattugliatore non diminuisce. Ciò accade quando, dopo la rimozione di *move(j)*, nessuna probabilità di cattura $P_c(t,i)\ \forall t \in T, i \in V \setminus \{j\}$ diminuisce per tutte le possibili strategie dell'intruso. Più formalmente, la rimozione dell'azione *move(j)* significa ridurre il grafo G mediante l'eliminazione del nodo j e di tutti gli archi ad esso associati.

L'identificazione delle azioni dominate per il pattugliatore è eseguita in due passaggi: nel primo ci si focalizza sui vertici e i corrispondenti archi incidenti e si basa sul seguente teorema.

Teorema 4.1.1 *Visitare un vertice che non appartiene ad un qualunque cammino minimo tra una coppia qualsiasi di obiettivi è un'azione dominata.*

Quando ci sono più cammini minimi che uniscono la stessa coppia di obiettivi (t_1, t_2), la visita di ciascun vertice può rappresentare un'azione dominata in accordo con il seguente teorema.

Teorema 4.1.2 *Dati due obiettivi t_1 e t_2 e due cammini minimi che li collegano $P = \langle t_1, \ldots, p_i \ldots, t_2 \rangle$ e $Q = \langle t_1, \ldots, q_i \ldots, t_2 \rangle$ di lunghezza L, se $\forall k \in \{2, \ldots, L-1\}$ e $t \in T \setminus \{t_1, t_2\}$ abbiamo che $dist(p_k, t) \geq dist(q_k, t)$, allora la visita di ciascun vertice interno a P (tutti i nodi esclusi t_1 e t_2) è un'azione dominata.*

Il primo passo identifica le azioni che possono essere considerate dominate, indipendentemente dal nodo corrente del pattugliatore. Se *move(j)* è dominata, allora il pattugliatore non visiterà mai j da un qualsiasi vertice ad esso adiacente.

Nel secondo passo invece teniamo in considerazione l'attuale posizione occupata dal pattugliatore all'interno dell'ambiente, considerando tutte le possibili azioni *move(j)* a partire dalle quali il vertice corrente è i. Possiamo formulare il seguente teorema.

Teorema 4.1.3 *Ogni azione che prevede per il turno successivo di rimanere nello stesso vertice i, con $i \in V \setminus T$, è un'azione dominata.*

L'applicazione del Teorema 4.1.3 ci permette di rimuovere tutti gli autoanelli di G.

Non ci sono ulteriori strategie del pattugliatore che possono essere rimosse indipendentemente dalla strategia dell'intruso. Più precisamente, non possono essere rimossi ulteriori vertici ed archi senza ridurre le probabilità di cattura, diminuendo di conseguenza l'utilità attesa. Pertanto, i teoremi sopra riportati ci permettono di rimuovere tutte le possibili strategie dominate per il pattugliatore.

Chiamiamo $G_r = (V_r, A_r, T, v, d)$ il grafo ridotto rimuovendo tutti i vertici e gli archi in accordo con il Teorema 4.1.1 e il Teorema 4.1.3. Possiamo constatare che, se la distanza di un vertice V_r da un obiettivo t è maggiore di $d(t)$, allora nessuna strategia di pattugliamento può coprire tutti i target.

Da qui in avanti considereremo solo il grafo ridotto G_r, anziché quello completo G.

Esempio 4.1.4 *Riportiamo in Figura 4.1 il grafo ridotto G_r per l'esempio di Figura 3.1 dopo aver rimosso i vertici e gli archi corrispondenti alle strategie dominate del pattugliatore.*

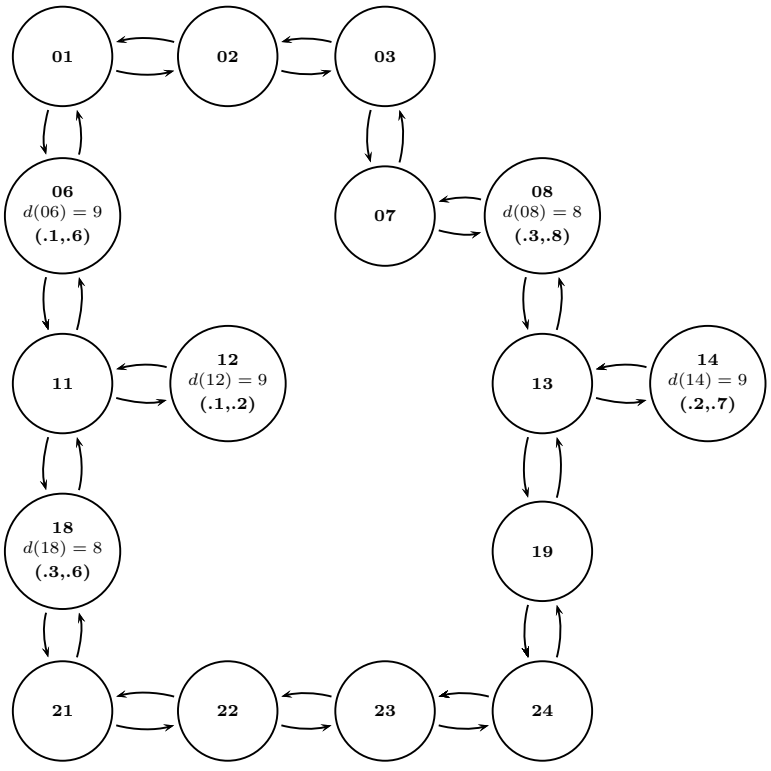


Figura 4.1: Il grafo ridotto G_r per il patrolling setting di Figura 3.1, ottenuto rimuovendo le strategie dominate per il pattugliatore.

4.1.2 Semplificazioni per l'intruso

Equivalentemente possiamo dire che per l'intruso un'azione *enter-when*$(\overline{t}, \overline{i})$ è dominata da un'azione *enter-when*$(\overline{s}, \overline{j})$ se l'utilità attesa della prima è minore o uguale alla seconda, per tutte le strategie miste $\sigma_{\mathbf{p}}$. Applicando questa definizione di dominanza, il controllo se un'azione è dominata da un'altra o meno può essere condotto risolvendo un problema di ottimizzazione.

Formulazione 4.1.5 *L'azione enter-when*$(\overline{t}, \overline{i})$ *è dominata da enter-when*$(\overline{s}, \overline{j})$ *se il risultato del seguente problema di ottimizzazione matematica è strettamente non positivo:*

$$\max \mu$$

t.c.

$$\text{vincoli (3.5), (3.6), (3.7), (3.8), (3.9), (3.10)}$$

$$\begin{aligned} u_{\mathbf{i}}(\textit{penetration-}\overline{t})(1 - P_c(\overline{t}, \overline{i})) - u_{\mathbf{i}}(\textit{penetration-}\overline{s})(1 - P_c(\overline{s}, \overline{j}))+ \\ u_{\mathbf{i}}(\textit{intruder-capture}) \left(P_c(\overline{t}, \overline{i}) - P_c(\overline{s}, \overline{j})\right) \geq \mu \end{aligned} \tag{4.1}$$

Il vincolo (4.1) definisce μ come il limite inferiore della differenza tra l'utilità attesa delle azioni *enter-when*$(\overline{t}, \overline{i})$ e *enter-when*$(\overline{s}, \overline{j})$. Il μ ottimale corrisponde alla massima differenza ottenibile e quindi, se non positivo, *enter-when*$(\overline{t}, \overline{i})$ è dominata da *enter-when*$(\overline{s}, \overline{j})$. Il problema sopra riportato presenta asintoticamente lo stesso numero di vincoli della Formulazione 3.3.1. La non linearità, la grandezza del problema e l'ampio numero di possibili istanze, rende la rimozione delle azioni dominate per l'intruso computazionalmente molto complessa. Tuttavia, sfruttando la struttura del problema, può essere derivato un algoritmo che rimuove le azioni dominate senza ricorrere alla programmazione matematica. Il seguente teorema fornisce due condizioni necessarie e sufficienti.

Teorema 4.1.6 *L'azione enter-when*$(\overline{t}, \overline{i})$ *è dominata da enter-when*$(\overline{s}, \overline{j})$ *se e solo se per una strategia mista completa* $\sigma_{\mathbf{p}}$ *si ha che:*

- (i) $u_{\mathbf{i}}(\textit{penetration-}\overline{t}) \leq u_{\mathbf{i}}(\textit{penetration-}\overline{s})$*, e*
- (ii) $P_c(\overline{t}, \overline{i}) > P_c(\overline{s}, \overline{j})$.

Riportiamo qui di seguito un metodo efficiente che sfrutti le condizioni (i) e (ii) del Teorema 4.1.6. Denotiamo la procedura come Algoritmo 1.

Algoritmo 1: INTRUDER_DOMINATION

```
for each t ∈ T do
    tabu(t) = {}
    for each v ∈ V do
        domination(t, v) = V
    EXPAND(t, t, {t}, 0)
for each t ∈ T do
    tabu(t) = {v ∈ V | ∀t ∃t', t ∈ domination(t', v), u_i(penetration-t) ≤
    u_i(penetration-t')}
    nondominated(t) = V \ {∪_{v∈V\{t}} domination(t, v) ∪ tabu(t)}
```

Algoritmo 2: EXPAND$(v, t, B, depth)$

```
N = {f | η(f) ≠ v, a(η(f), v) = 1}
for each f ∈ N do
    domination(t, η(f)) = domination(t, η(f)) ∩ η(B)
if depth < d(t) then
    for each f ∈ N do
        EXPAND(f, t, {B ∪ f}, depth + 1)
```

L'algoritmo ricorre a degli alberi di ricerca dove ogni nodo q contiene un vertice $\eta(q)$ del grafo originale. Per ogni target t, costruiamo un albero di profondità $d(t)$, dove la radice è t e i successori di un nodo q sono tutti i nodi q' tali per cui: $\eta(q')$ è adiacente a $\eta(q)$, $\eta(q')$ è diverso sia da $\eta(q)$ che dal vertice del nodo padre di q.

Introduciamo il vettore $domination(t, v)$ che contiene tutti i vertici i tali per cui *enter-when*(t, i) è dominata da *enter-when*(t, v). Tale vettore viene costruito in maniera iterativa, inizializzandolo come $domination(t, v) = V$ per tutti $t \in T$ e $v \in V$ e aggiornandolo ogni volta che un nodo q viene esplorato, come:

$$domination(t, \eta(q)) = domination(t, \eta(q)) \cap predecessors(q)$$

dove *predecessors(q)* è il vettore dei predecessori di q. Una volta completata la costruzione dell'albero di radice t, $domination(t, v)$ contiene tutti e soli i vertici v' tali per cui $P_c(t, v) < P_c(t, v')$. Questo perché, per raggiungere t da v in $d(t)$ turni, il pattugliatore deve sempre attraversare $v' \in$ *domination*(t, v), pertanto, dalle catene di Markov con alterazione, abbiamo che $P_c(t, v) = P_c(t, v') \cdot \phi < P_c(t, v')$ con $\phi < 1$. Vengono così soddisfatte le condizioni (i) e (ii) del Teorema 4.1.6 e l'azione *enter-when*(t, v') è (debolmente) dominata da *enter-when*(t, v).

Utilizzando gli alberi dei possibili percorsi identifichiamo le dominanze

relativamente ad un singolo target; ad ogni modo, le dominanze possono esistere anche su azioni che coinvolgono target diversi. Allo scopo di trovarle, definiamo:

$$tabu(t) = \{v \in V \text{ t.c. } \exists t', t \in domination(t', v), \\ u_{\mathbf{i}}(penetration\text{-}t) \leq u_{\mathbf{i}}(penetration\text{-}t')\} \quad \forall t \in T;$$

tabu(t) contiene tutti e soli i vertici di v per i quali esiste una coppia $t' \in T$ con $t' \neq t$ e $v' \in V$ con $u_{\mathbf{i}}(penetration\text{-}t) \leq u_{\mathbf{i}}(penetration\text{-}t')$ e $P_c(t, v) > P_c(t', v')$; pertanto sono soddisfate le condizioni (i) e (ii) del Teorema 4.1.6 e l'azione *enter-when*(t, v) è (debolmente) dominata da *enter-when*(t', v'). Per concludere denotiamo:

$$nondominated(t) = V \setminus \{\cup_{v \in V} domination(t, v) \cup tabu(t)\} \quad \forall t \in T.$$

Tutte e sole le azioni *enter-when(t,i)* dove $i \in$ *nondominated(t)* risultano essere dominate.

Esempio 4.1.7 *In Figura 4.2 i nodi in nero denotano i vertici i per i quali l'azione enter-when*$(06, i)$ *è dominata; per esempio, l'azione enter-when*$(06, 13)$ *è dominata dato che ogni occorrenza del vertice 13 all'interno dell'albero di ricerca ha come nodo figlio il nodo 14.*

4.1.3 Dominanze iterate

In seguito alla rimozione delle strategie dominate per il pattugliatore e per l'intruso (in questo stesso ordine), possiamo solamente rimuovere alcune ulteriori strategie per il pattugliatore. Possiamo formulare il seguente teorema.

Teorema 4.1.8 *L'assegnazione di una probabilità positiva ad* $\alpha_{t,t}$ *con* $t \in T$ *è un'azione dominata se l'azione dell'intruso enter-when*(t, t) *è a sua volta dominata.*

Non sono possibili ulteriori semplificazioni, dopo la rimozione degli archi prevista dal Teorema 4.1.8, le strategie dominate dell'intruso non possono essere ulteriormente ridotte.

4.2 Astrazioni senza perdita di informazione

Malgrado la rimozione delle strategie dominate diminuisca drasticamente la grandezza del problema in termini di vertici e numero delle best-response

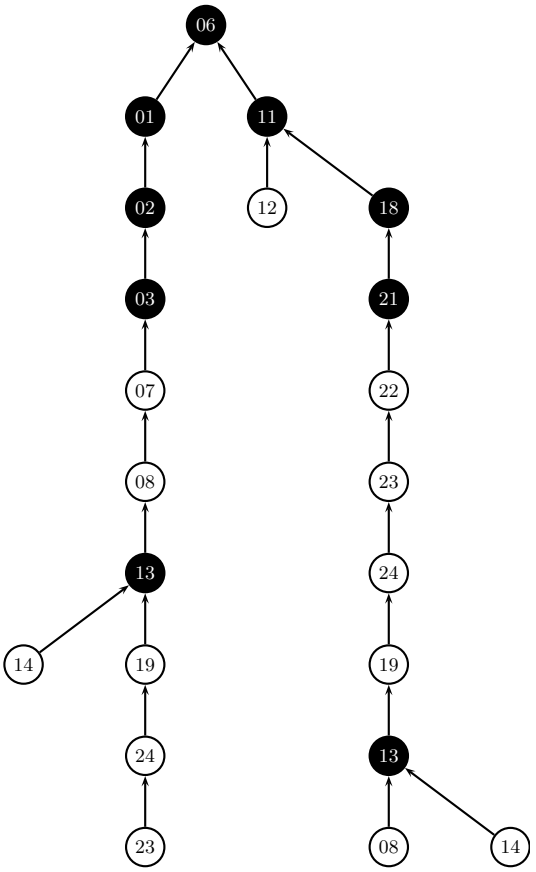


Figura 4.2: L'albero di ricerca per trovare le azioni dominate del target 06 di Figura 4.1, i nodi in bianco costituiscono il vettore $nondominated(06)$.

da parte dell'intruso, riducendo in media del 95% il tempo di esecuzione necessario a risolvere il problema, la risoluzione di situazioni realistiche resta ancora complessa. Una tecnica efficace per la riduzione di giochi di grandi dimensioni, che ha ricevuto molta attenzione nella letteratura è la *strategia astratta* [8, 9]. In questa sezione applichiamo le astrazioni senza perdita di informazione ai patrolling security games: introduciamo la sua definizione nella Sezione 4.2.1, formuliamo una generica classe di astrazione senza perdita di informazione nella Sezione 4.2.2 e discutiamo come possano essere applicate ai patrolling security games nella Sezione 4.2.3.

4.2.1 Definizione di astrazione

L'idea di base dietro il concetto di astrazione è quella di raggruppare insieme più azioni in una singola macro azione, permettendoci di ridurre la dimensione del gioco. La tipologia più interessante di astrazione è quella senza

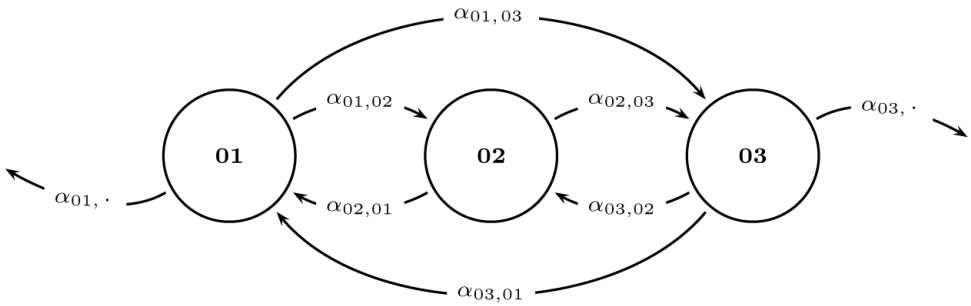


Figura 4.3: Astrazioni sui vertici 01*,* 03*.*

perdita di informazione, in quanto ci permette di trovare soluzioni ottimali risolvendo il gioco astratto.

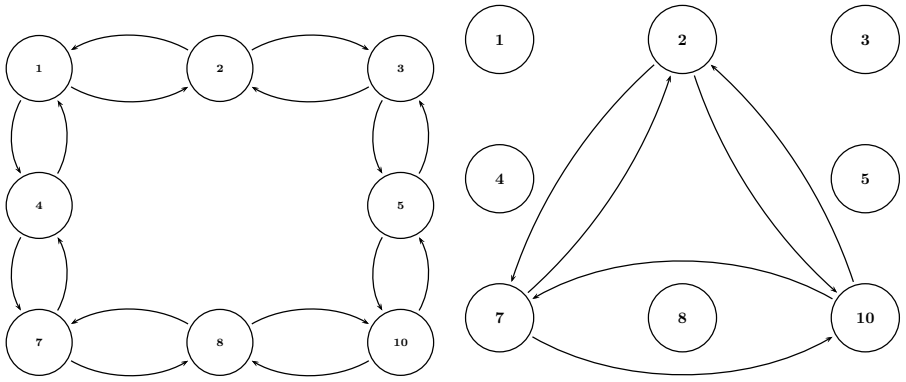
Sono state sviluppate diverse soluzioni che fanno uso delle astrazioni nei giochi in forma normale con informazione imperfetta, in particolare per il gioco del poker [8, 9]. Tuttavia, l'applicazione dei risultati in [8] ai patrolling security games produce un gioco non a somma zero che non è possibile ridurre in alcun modo. Questo ci spinge a definire delle astrazioni *ad-hoc*.

Definizione 4.2.1 *Un'astrazione di una coppia di vertici non adiacenti* i, j *è una coppia di macro azioni del pattugliatore move-along*(i, j) *e move-along*(j, i) *con le seguenti proprietà:*[1]

- *Quando il pattugliatore compie la macro azione move-along*(i, j) *(move-along*(j, i)*), si muove dal vertice corrente* i *(*j*) verso il vertice* j *(*i*) lungo il cammino minimo, visitando, turno dopo turno, i nodi che compongono il percorso;*
- *il completamento di una macro azione richiede un numero di turni pari alla lunghezza del cammino minimo;*
- *durante il compimento di una macro azione il pattugliatore non può prendere ulteriori decisioni;*
- *l'intruso può violare un obiettivo durante l'esecuzione di una macro azione da parte del pattugliatore.*

Esempio 4.2.2 *Consideriamo la Figura 4.3. Applicando un'astrazione sui vertici* $01, 03$ *rimuoviamo gli archi etichettati con* $\alpha_{01,02}, \alpha_{02,01}, \alpha_{02,03}, \alpha_{03,02}$ *(dove* $\alpha_{i,j}$ *corrisponde all'azione move-along*(i, j)*) e introduciamo gli archi*

[1]Per semplicità consideriamo il caso in cui i due vertici siano connessi da un unico cammino minimo.



(a) grafo originale (b) grafo astratto

Figura 4.4: Grafo originale e la sua astrazione.

etichettati con $\alpha_{01,03}, \alpha_{03,01}$. *Quando il pattugliatore è in* 01 *e decide di andare in* 03*, impiegherà due turni, durante i quali si sposterà da* 01 *a* 02 *(primo turno) e da* 02 *a* 03 *(secondo turno). Quando il pattugliatore è in* 02*, non può fermare l'azione corrente e cambiarla.*

Definizione 4.2.3 *Un'astrazione su di un intero grafo* G *è il risultato dell'applicazione di più astrazioni su diverse coppie di vertici. Possiamo ottenerla rimuovendo da* G *alcuni sotto grafi connessi disgiunti* $G' \subset G$ *e introducendo in* G *per ogni* G'*:*

- *degli archi* $\{(i, j)\}$*, dove* i, j *sono dei vertici in* $G \setminus G'$ *e sia* i *che* j *sono adiacenti ai vertici in* G' *(ogni arco* (i, j) *corrisponde ad una macro azione move-along*(i, j)*);*

- *una funzione* $e : V \times V \to \mathbb{N}$ *assegna ad ogni arco i turni necessari al pattugliatore per attraversarlo.*

Esempio 4.2.4 *Riportiamo un esempio di grafo astratto* G *nella Figura 4.4.*

Il principale problema è la selezione dei vertici che devono essere rimossi, facendo in modo che l'astrazione ottenuta preservi le strategie di equilibrio.

4.2.2 Definizione di astrazione senza perdita di informazione

Quando il pattugliatore si muove lungo gli archi astratti (i, j), l'intruso può ricavarne un vantaggio, perché conosce con certezza alcune delle sue prossime mosse.

Esempio 4.2.5 *Consideriamo la Figura 4.3: nel caso in cui il pattugliatore decida di muoversi da 01 a 03, quando l'intruso lo vede transitare in 02 sa che al prossimo turno si sposterà in 03.*

Siamo in grado di produrre astrazioni senza perdita di informazione facendo in modo che l'insieme delle strategie dominate dell'intruso sia un invariante. Come risultato, l'intruso non penetrerà mai un target (in maniera ottimale) mentre il pattugliatore esegue una macro azione *move-along*(i, j) e pertanto non può prendere vantaggio dalla conoscenza di alcune delle prossime azioni del pattugliatore.

Non sono attuabili ulteriori astrazioni (prodotte *ex-ante* la risoluzione del gioco) che siano computazionalmente efficienti per la semplificazione dei patrolling security games. A dimostrazione di ciò formuliamo il seguente teorema.

Teorema 4.2.6 *Condizione necessaria a formulare un'astrazione (*ex-ante*) che sia senza perdita di informazione, è che l'insieme delle strategie dominate dell'intruso sia un invariante.*

Riportiamo alcune condizioni necessarie affinché la rimozione di un vertice, durante l'applicazione di un'astrazione, mantenga invariato l'insieme delle strategie dominate.

Corollario 4.2.7 *La rimozione di un vertice i durante l'applicazione di un'astrazione può considerarsi senza perdita di informazione se e soltanto se:*

- *quando $i \notin T$, $\forall t \in T$ le azioni enter-when(t, i) sono dominate;*
- *quando $i \in T$, $\forall t \in T$ le azioni enter-when(t, i) sono dominate e $\forall j \in V$ le azioni enter-when(i, j) sono dominate.*

Assicurare che l'insieme delle strategie dominate non venga modificato non è tuttavia sufficiente, questo perché dobbiamo assicurarci che risolvendo l'astrazione del gioco siamo ugualmente in grado di trovare una strategia ottimale quanto nel caso originale. A tal fine denotiamo con $dom(i, t)$ l'insieme dei vertici i' per cui *enter-when*(t, i') non è dominata e domina *enter-when*(t, i) (come abbiamo visto nella Sezione 4.1.2). Possiamo formulare il seguente teorema.

Teorema 4.2.8 *Data un'astrazione, se, per ogni coppia di vertici i, j su cui sono definite le astrazioni, per tutti i vertici k che compongono la coppia di archi "astratti" che connettono i e j, vale che:*

- $dist(i, dom(k,t)) \geq dist(i,k)$ *e*
- $dist(j, dom(k,t)) \geq dist(j,k)$ *per tutti gli obiettivi* $t \in T$*,*

allora l'insieme delle strategie dominate è invariante e risolvere la versione astratta del gioco ci permette di trovare delle soluzioni valide quanto quelle ottimali del caso originale.

Si noti che, dopo aver astratto il gioco impiegando la tecnica qui esposta, possiamo direttamente risolverlo senza rimuovere ulteriormente le strategie dominate, che sono invarianti rispetto alla versione non astratta del gioco.

Astrazioni più *forti* (nei termini di rimozione del numero dei vertici) di quelle descritte dal precedente teorema possono essere prodotte in casi specifici, ma la loro derivazione non è del tutto efficiente.

4.2.3 Calcolo delle astrazioni

Chiamiamo $C \subset V$ l'insieme dei vertici che soddisfano il Corollario 4.2.7; introduciamo la variabile binaria $x_i \in \{0,1\}$ con $i \in C$, dove $x_i = 1$ significa che il vertice i può essere rimosso da un'astrazione e $x_i = 0$ che non può essere tolto. Definiamo la variabile intera $s_{i,t} \in \{0,\ldots,n\}$ con $i \in V$ e $t \in T$, come la distanza tra un vertice i e un target t quando viene applicata un'astrazione. Chiamiamo $succ(i,j)$ l'insieme dei vertici adiacenti ad i che appartengono ai cammini minimi che connettono i e j.

Formulazione 4.2.9 *Un'astrazione è senza perdita di informazione se il seguente sistema lineare intero di programmazione matematica associato all'astrazione ammette soluzione:*

$$s_{i,t} = dist(i,t) \qquad \forall i \notin C, t \in T \tag{4.2}$$

$$s_{i,t} \geq dist(i,t) \qquad \forall i \in C, t \in T \tag{4.3}$$

$$s_{i,t} \leq dist(i,t) + nx_i \qquad \forall i \in C, t \in T \tag{4.4}$$

$$s_{i,t} \leq s_{j,t} + 1 - n(1-x_i) \qquad \forall i \in C, t \in T, j \in succ(i,k), k \in dom(i,t) \tag{4.5}$$

$$s_{i,t} \geq s_{j,t} + 1 + n(1-x_i) \qquad \forall i \in C, t \in T, j \in succ(i,k), k \in dom(i,t) \tag{4.6}$$

$$s_{i,t} \leq dist(j,t) \qquad \forall i \in C, t \in T, j \in dom(i,t) \tag{4.7}$$

Il Vincolo (4.2) obbliga $s_{i,t}$, per tutti i vertici i che non possono essere rimossi, ad essere uguale alla distanza tra i e t; il Vincolo (4.3) impone che $s_{i,t}$ sia uguale o maggiore della distanza tra i e t per tutti i vertici che possono essere rimossi; il Vincolo (4.4) obbliga $s_{i,t}$ ad essere uguale alla distanza tra i e t se $x_i = 0$ per tutti i vertici eliminabili; i vincoli (4.5) e (4.6) forzano $s_{i,t}$, per tutti i vertici che possono essere rimossi, ad essere uguale a $s_{j,t} + 1$ con $j \in succ(i,k)$ dove $k \in dom(i,t)$ se $x_i = 1$; il Vincolo (4.7) impone che

$s_{i,t}$ non sia più grande della distanza $dist(j,t)$ con $j \in dom(i,t)$.

La formulazione di cui sopra ci permette di controllare quando un'astrazione è senza perdita di informazione o meno. Naturalmente siamo interessati a trovare l'astrazione che riproduce il gioco e che richiede le minime risorse per essere risolta, vale a dire il problema con il minimo numero di variabili (archi) α.

Chiamiamo δ_i il grado esterno di un vertice i, rimuovendo tale vertice dal grafo, eliminiamo $2\delta_i$ archi (corrispondenti a $2\delta_i$ variabili α) e introduciamo $\delta_i(\delta_i - 1)$ nuovi archi (corrispondenti a $\delta_i(\delta_i - 1)$ nuove variabili α).[2] In pratica possiamo limitarci a rimuovere i vertici i con $\delta_i \leq 3$.

Formulazione 4.2.10 *La più forte astrazione senza perdita di informazione è ottenuta come soluzione del seguente problema lineare intero di ottimizzazione:*

$$\max \sum_{i \in C, \delta_i \leq 3} x_i$$

$$\text{vincoli } (4.2), (4.3), (4.4), (4.5), (4.6), (4.7)$$

Chiamiamo A' l'insieme degli archi del gioco astratto e lo rappresentiamo mediante la funzione $a' : V \times V \to \{0,1\}$. Allo scopo di calcolare le probabilità di cattura per l'intruso, dobbiamo sostituire i vincoli (3.7), (3.8), (3.9), and (3.10) con i seguenti, i quali considerano la possibilità che attraversare gli archi richieda un tempo maggiore di un solo turno:

$$\alpha_{i,j} \leq a'(i,j) \qquad \forall i,j \in V \tag{4.8}$$

$$\gamma_{i,j}^{e(i,j),t} = \alpha_{i,j} \qquad \forall t \in T, i,j \in V, j \neq t \tag{4.9}$$

$$\gamma_{i,j}^{w,t} = \sum_{\substack{x \in V \setminus \{t\}, \\ w \leq \rho(x,t), \\ w \geq e(i,x) + e(x,j)}} \left(\gamma_{i,x}^{w-e(x,j),t} \alpha_{x,j}\right) \qquad \begin{array}{r} \forall w \in \{2, \ldots, \rho(j,t)\}, \\ t \in T, i,j \in V, j \neq t \end{array} \tag{4.10}$$

$$P_c(t,i) = 1 - \sum_{j \in V \setminus \{t\}} \gamma_{i,j}^{\rho(j,t),t} - \sum_{j \in V \setminus \{t\}} \sum_{w \leq \rho(j,t)-1} \gamma_{i,j}^{w,t} \sum_{\substack{x \in V \setminus \{t\}, \\ w \geq e(i,j) + \rho(x,t)}} \alpha_{j,x} \qquad \forall t \in T, i \in V \tag{4.11}$$

[2] In realtà volendo essere più rigorosi, rimuovendo un vertice i rimuoviamo anche delle variabili γ, tuttavia abbiamo sperimentalmente osservato che l'efficienza data dalle variabili α è maggiore di quella data dalle variabili γ.

4.3 Astrazioni con perdita di informazione

L'applicazione delle astrazioni senza perdita di informazione ha il potenziale di ridurre drasticamente la grandezza della maggior parte dei giochi, rendendoli computazionalmente trattabili. Tuttavia, per problemi molto grandi (specialmente quelli che contengono cicli), questo tipo di astrazione continua a produrre giochi troppo complessi per poter essere facilmente risolti; per tutti questi casi, rilassiamo i vincoli necessari a preservare l'informazione, formulando astrazioni del gioco ridotte, la cui soluzione tuttavia non è garantito essere ottimale.

Nella Sezione 4.3.1 discutiamo come possano essere dedotte le astrazioni con perdita di informazione e nella Sezione 4.3.2 presentiamo un algoritmo per rimuovere le strategie dominate dell'intruso.

4.3.1 Astrazioni automatizzate

Mentre per le astrazioni senza perdita di informazione riproduciamo un gioco in cui l'insieme delle strategie dominate è invariante, con la perdita di informazione viene prodotto un gioco in cui possiamo garantire una condizione più debole: ogni obiettivo non è *esposto*. Più precisamente, diciamo che un target t è esposto quando esiste un'azione *enter-when*(t, x) per cui la relativa probabilità di cattura è zero. Ciò accade quando esiste un vertice x appartenente al percorso di sorveglianza del pattugliatore e $dist(t, x) > d(t)$.

Essenzialmente trovare astrazioni con perdita di informazione è simile, in linea concettuale, al caso senza perdita di informazione, la differenza principale è nella definizione del limite superiore di $s_{i,t}$: quando le astrazioni sono senza perdita di informazione, abbiamo bisogno che $s_{i,t}$ non sia più grande della distanza tra i e t; quando invece le astrazioni sono con perdita di informazione, è necessario che $s_{i,t}$ non sia più grande di $d(t)$. Questo tipo di astrazione è la più forte possibile, dato che ulteriori riduzioni del gioco renderebbero un obiettivo esposto, il che equivarrebbe a rimuoverlo del tutto dal problema. Tutti i candidati C che possono essere rimossi, sono tutti quei vertici che non sono degli obiettivi.

Formulazione 4.3.1 *I vincoli sulle astrazioni con perdita di informazione per un patrolling security game sono lineari interi:*

vincoli (4.2), (4.3), (4.4)

$$s_{i,t} \leq s_{j,t} + 1 - n(1 - x_i) \qquad \forall i \in C, t \in T, j \in succ(i,t) \tag{4.12}$$

$$s_{i,t} \geq s_{j,t} + 1 + n(1 - x_i) \qquad \forall i \in C, t \in T, j \in succ(i,t) \tag{4.13}$$

$$s_{i,t} \leq dist(i,t) + 1 - n(1 - x_i) \qquad \begin{array}{c} \forall i \in C, t \in T, succ(i,t) = \emptyset, \\ \exists k, a(i,k) = 1, dist(k,t) = dist(i,t) \end{array} \tag{4.14}$$

$$s_{i,t} \geq dist(i,t) + 1 + n(1 - x_i) \qquad \begin{array}{c} \forall i \in C, t \in T, succ(i,t) = \emptyset, \\ \exists k, a(i,k) = 1, dist(k,t) = dist(i,t) \end{array} \tag{4.15}$$

$$s_{i,t} = dist(i,t) \qquad \begin{array}{c} \forall i \in C, t \in T, succ(i,t) = \emptyset, \\ \forall k, a(i,k) = 1, dist(k,t) < dist(i,t) \end{array} \tag{4.16}$$

$$s_{i,t} \leq d(t) \qquad \forall i \in C, t \in T \tag{4.17}$$

I Vincoli (4.12), (4.13) e (4.17) allentano i corrispondenti (senza perdita di informazione) Vincoli (4.2), (4.3) e (4.7), considerando direttamente un obiettivo t anziché il $dom(i,t)$. I Vincoli (4.14), (4.15) e (4.16) sono analoghi a (4.12) e (4.13), ma vengono applicati quando, dato un vertice i e un target t, non vi è alcun successore (i.e., $succ(i,t) = \emptyset$), ciò accade in presenza di cicli, più precisamente quando i è il vertice più lontano da t. I Vincoli (4.14) e (4.15) sono applicati quando esiste un vertice k che è tanto lontano da t quanto i, mentre il Vincolo (4.16) viene applicato quando x è il più lontano in senso stretto.

Come per il caso senza perdita di informazione, stiamo cercando le astrazioni che riproducono il gioco il più ridotto possibile.

Formulazione 4.3.2 *La più forte astrazione con perdita di informazione per un patrolling security game è ottenuta come soluzione del seguente problema lineare intero di ottimizzazione matematica:*

$$\max \sum_{i \in C, \delta_i \leq 3} x_i$$

$$\text{vincoli } (4.2), (4.3), (4.4), (4.12), (4.13), (4.14), (4.15), (4.16), (4.17)$$

Come nella precedente sezione, chiamiamo A' l'insieme degli archi del gioco astratto e rappresentiamo A' con la funzione $a' : V \times V \to \{0, 1\}$.

Esempio 4.3.3 *In Figura 4.5 riportiamo il patrolling setting dell'esempio di Figura 3.1 dopo l'applicazione dell'astrazione con perdita di informazione più forte possibile.*

4.3.2 Rimozione delle strategie dominate per l'intruso

L'applicazione delle astrazioni con perdita di informazione produce un gioco ridotto, le cui strategie dominate sono potenzialmente differenti da quelle

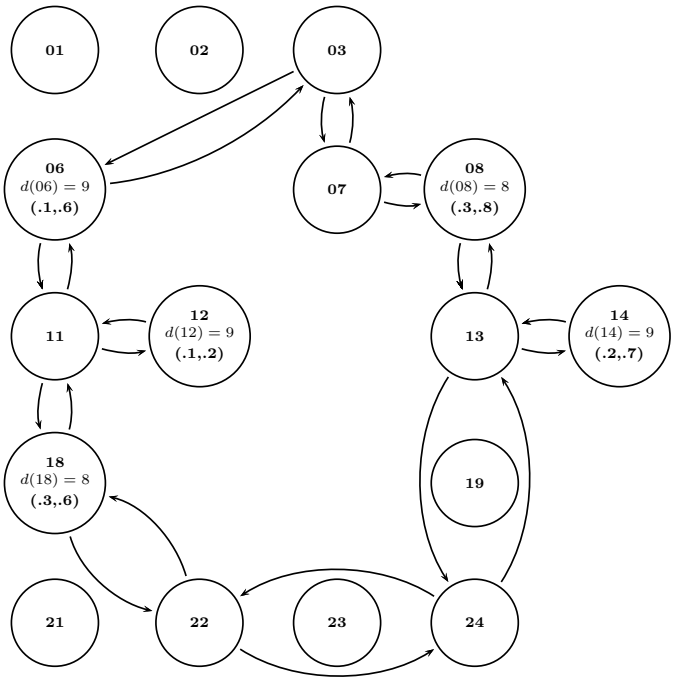


Figura 4.5: Grafo G_r per il patrolling setting di Figura 3.1, ottenuto applicando le astrazioni con perdita di informazione.

del gioco originale; per di più l'Algoritmo 1, così com'è stato formulato in precedenza, non può essere applicato alla versione astratta del gioco, poiché non considera la possibilità che la distanza tra due vertici sia maggiore di uno e che l'intruso possa entrare in un obiettivo mentre il pattugliatore sta eseguendo una macro azione, muovendosi da un vertice ad un altro. In ogni caso, l'algoritmo per la rimozione delle strategie dominate nella formulazione astratta è una sua semplice variante. Lo riportiamo qui di seguito come Algoritmo 3.
L'Algoritmo 3 funziona esattamente come l'Algoritmo 1 salvo le seguenti modifiche:

- nel Passo 2, in cui viene definita la variabile $delay(t, v) = 0$;
- nel Passo 3, dove la lunghezza dei percorsi è misurata nei termini di costo temporale;
- nel Passo 5, in cui per ogni vertice $\eta(q)$ consideriamo il ritardo più ampio (in turni), che il pattugliatore può avere muovendosi lungo gli archi per raggiungere $\eta(q)$;

Algoritmo 3: INTRUDER_DOMINATION

```
1  for each t ∈ T do
2      tabu(t) = {}
3      for each v ∈ V do
4          domination(t, v) = V
5          delay(t, v) = 0
6      EXPAND(t, t, {t}, 0)
7      for each v ∈ V do
8          for each w ∈ domination(t, v) do
9              if dist(v, w) < delay(t, w) then
10                 domination(t, v) = domination(t, v) \ {w}
11 for each t ∈ T do
12     tabu(t) = {v ∈ V | ∀t ∃t', t ∈ domination(t', v), u_i(penetration-t) ≤
       u_i(penetration-t')}
13     nondominated(t) = V \ {∪_{v∈V\{t}} domination(t, v) ∪ tabu(t)}
```

Algoritmo 4: EXPAND$(v, t, B, depth)$

```
1  N = {f | η(f) ≠ η(v), a(η(f), η(v)) = 1}
2  for each f ∈ N do
3      domination(t, η(f)) = domination(t, η(f)) ∩ η(B)
4      if dist(η(v), η(f)) > 1 then
5          delay(t, η(v)) = max{delay(t, η(v)), dist(η(v), η(f)) − 1}
6  if depth < d(t) then
7      for each f ∈ N do
8          EXPAND(f, t, {B ∪ f}, depth + 1)
```

- nel Passo 6, dove l'insieme delle dominanze è ridotto considerando anche il ritardo $delay(t, v)$.

In particolare, focalizzandoci su quest'ultimo, dato un target t e *domination*(t, v), calcolata come descritto nei Passi 3-5, un vertice v domina v' solamente se $delay(t, v) \leq dist(v, v')$. Altrimenti, se $delay(t, v) > dist(v, v')$, non possiamo avere garanzie che la probabilità di cattura di *enter-when*(t, v) sia minore di *enter-when*(t, v'), considerando un ritardo $delay(t, v)$.

Una volta rimosse le strategie dominate per l'intruso, il calcolo dell'equilibrio leader-follower è basato sulla stessa formulazione del problema di programmazione matematica usato per le astrazioni senza perdita di informazione, ad eccezione di ogni azione non dominata *enter-when*(t, v) in cui usiamo $d(t) - delay(t, v)$ invece di $delay(t, v)$.
Nel prossimo capitolo presenteremo la soluzione software che è derivata dai nostri studi, presentando in dettaglio il suo funzionamento e la sua interfaccia.

Capitolo 5

Talos

Il mitologico e invulnerabile automa di bronzo guardiano di Creta.

La possibilità di risolvere istanze del problema di pattugliamento sempre più complesse e simili a situazioni reali, ha reso necessario lo sviluppo di uno strumento software che facilitasse, oltre che la creazione stessa dei patrolling setting, anche l'interpretazione delle strategie risultanti, altrimenti di difficile comprensione, trattandosi per lo più di matrici di adiacenza pesate di grandi dimensioni.

In questo capitolo descriviamo in dettaglio le scelte di progetto effettuate allo scopo di sviluppare un'applicazione che vada incontro a queste esigenze. Iniziamo nella Sezione 5.1 con un'analisi delle specifiche necessarie ad effettuare il passaggio dal modello matematico, precedentemente descritto nel Capitolo 3, all'applicativo vero e proprio; vengono qui elencate anche alcune estensioni al modello che abbiamo deciso di prendere in considerazione per renderlo ancora più verosimile; infine viene descritta l'interfaccia utente dell'applicazione. Nella Sezione 5.2 presentiamo l'architettura della soluzione software proposta, descrivendone le tecnologie utilizzate, la struttura, i paradigmi impiegati e le funzionalità previste dall'Application Program Interface (API). Successivamente vengono delineati nella Sezione 5.3 i servizi che l'applicazione offre, descrivendo nel dettaglio le modalità di interazione con l'interfaccia utente. Il capitolo termina con la Sezione 5.4 analizzando il flusso di esecuzione ed i software coinvolti allo scopo di risolvere un patrolling security game.

5.1 Dal modello allo strumento software

Nella Sezione 5.1.1 esaminiamo in prima battuta le specifiche e le scelte progettuali effettuate allo scopo di implementare un'applicativo che sia coerente con il modello e che soddisfi determinati requisiti; nella Sezione 5.1.2 presentiamo alcune estensioni che abbiamo deciso di prendere in considerazione nello sviluppo, così da permettere di descrivere ambienti e meccanismi sempre più simili a situazioni reali; concludendo presentiamo l'interfaccia utente di Talos e le parti che la compongono nella Sezione 5.1.3.

5.1.1 Specifiche

Il software che vogliamo sviluppare deve soddisfare alcune caratteristiche che andremo qui ad analizzare, dividendole in specifiche non funzionali e funzionali.

Specifiche non funzionali

La necessità di impiegare software licenziatari per la formulazione, l'analisi, la riduzione e la conseguente risoluzione del problema di pattugliamento, quali AMPL™ [38], SNOPT™ [39], CPLEX™ [40] e MATLAB™ [41], usufruibili solamente all'interno della rete LAN del Dipartimento di Elettronica e Informazione (DEI) del Politecnico di Milano, ci ha obbligato a sviluppare il tool come un servizio online, separando inoltre il web server, accessibile dagli utenti attraverso la rete Internet, dalla workstation di calcolo dove risiedono i suddetti software, raggiungibile solamente attraverso la rete locale del dipartimento.

Al fine di rendere l'applicativo il più simile possibile ad una vera e propria applicazione eseguita in locale e non ad un sito dinamico, abbiamo deciso di ridurre il più possibile il numero degli aggiornamenti di pagina, preferendo impiegare tecniche Asynchronous JavaScript and XML (Ajax) in background, per la modifica di determinati elementi del Document Object Model (DOM), in seguito a specifici eventi di interazione con l'interfaccia utente. In questo modo possiamo caricare i dati richiesti ogni qual volta si renda necessario, evitando di appesantire enormemente le pagine precaricando tutte le informazioni.

Specifiche funzionali

Lo strumento software deve rispettare alcune caratteristiche necessarie alle sue finalità, le elenchiamo qui di seguito per identificarle in maniera più chiara:

- essendo sviluppato come una web-application, deve essere orientato al Web 2.0, permettendo la creazione di multipli account utente, ognuno dei quali potrà interagire autonomamente con il sistema per la creazione e risoluzione dei patrolling setting, formando la propria base di dati;

- deve permettere la creazione di una mappa a griglia che possa descrivere ambienti il più possibile simili alla realtà, definendone, oltre che la sola topologia, anche gli obiettivi, i punti di ingresso (come vediamo nella Sezione 5.1.2) e tutte le variabili necessarie alla definizione del patrolling setting, quali i tempi di penetrazione, le utilità per gli agenti e la portata del sensore del pattugliatore;

- una volta completata la creazione dell'ambiente e impostate tutte le variabili deve essere fornito un servizio di risoluzione assolutamente trasparente per l'utente, che preveda l'utilizzo di notifiche email[1] una volta trovata la strategia di pattugliamento;

- deve essere sviluppato uno strumento per valutare le strategie, rappresentandole in maniera grafica e intuitiva, mostrando il percorso strategico seguito dal pattugliatore con le relative probabilità di movimento; inoltre deve essere fornito un sistema per il confronto delle strategie risultanti sulla medesima mappa a fronte di variazioni dei parametri di ingresso;

- i risultati di ciascuna esecuzione devono poter essere salvati e gestiti, prevedendo inoltre meccanismi di condivisione tra gli utenti, sia dei patrolling setting che dei risultati strategici, in modo che ognuno possa contribuire alla creazione di un repository per il pattugliamento strategico, che col tempo possa essere identificato dalla comunità di ricerca scientifica, come un punto di riferimento, come è accaduto con Radish [42] per l'esplorazione degli ambienti e creazione delle mappe nella robotica mobile.

5.1.2 Estensioni al modello

Uno dei maggiori limiti dell'approccio BGA presentato nel Capitolo 3 è quello di considerare dei modelli assolutamente ideali per il pattugliatore e l'intruso, che, per determinati aspetti, mal si adattano a descrivere situazioni

[1] Le latenze legate alla risoluzione hanno reso necessario implementare un meccanismo di questo tipo, anche se sono state predisposte anche delle notifiche in tempo reale all'interno dell'applicativo.

reali. A tal fine abbiamo ridefinito ed esteso alcune formulazioni, in questa sezione discutiamo le principali proposte, descrivendo brevemente in che modo siano state introdotte.

La prima modifica riguarda il movimento dell'intruso: abbiamo visto come nel modello BGA è stato assunto che l'agente possa apparire direttamente in un obiettivo per cercare di violarlo. Gli stessi autori però propongono in [43] un modello più realistico, in cui l'intruso può accedere all'ambiente solamente attraverso specifici vertici denominati *punti di ingresso* per poi muoversi lungo dei percorsi per raggiungere un determinato target; pertanto abbiamo deciso di predisporre il nostro tool per prendere in considerazione anche questo tipo di celle durante la fase di creazione dell'ambiente.

É stato assunto che l'intruso possa muoversi infinitamente veloce lungo i percorsi, impiegando un solo turno per raggiungere un obiettivo, indipendentemente dalla lunghezza della traiettoria, il che risulta essere il caso peggiore per il pattugliatore; durante questo singolo turno l'intruso può essere catturato in qualsiasi vertice del percorso seguito. La struttura del gioco non cambia di molto da quella vista in precedenza e i risultati sperimentali in [43] hanno dimostrato che con questa estensione del modello la complessità computazionale per la risoluzione del gioco è minore, poiché è a sua volta ridotto il numero di azioni non dominate da parte dell'intruso.

Un'altra estensione rilevante che abbiamo deciso di prendere in considerazione è esposta in [44], in cui viene considerato un ritardo temporale tra il momento in cui l'intruso decide di attaccare e il momento in cui raggiunge un determinato obiettivo. Combinando i risultati in [43] e [44] è possibile formulare un modello in cui l'intruso si muove lungo i percorsi in un tempo finito.

In [44] è stata presa in considerazione la possibilità di aumentare le capacità sensoriali del pattugliatore: rilassando i Vincoli (3.8) e (3.9) è possibile descrivere una situazione in cui il pattugliatore è in grado di rilevare l'intruso anche in celle differenti da quella corrente; del resto abbiamo visto come la formulazione matematica della funzione che descrive il sensore del pattugliatore, già integra la possibilità di specificare le probabilità di sensing per le celle limitrofe oltre che per il nodo corrente. A tal fine abbiamo preparato il tool a soddisfare anche questa espansione.

L'estensione multi-agente è stata inizialmente considerata in [5], studiando il problema nel caso di un sistema con robot sincronizzati tra loro, ed ulteriormente estesa in [45], dove gli autori hanno approfondito il problema di agenti asincorni, focalizzandosi sulla determinazione del numero minimo di pattugliatori necessari a sorvegliare un ambiente di topologia arbitraria, facendo in modo che nessun obiettivo rimanga esposto a un possibile attacco.

Abbiamo dunque deciso di predisporre l'interfaccia a gestire soluzioni anche per il caso multi-agente, inserendo la possibilità di selezionare la strategia di ogni singolo robot per una visione più chiara.

5.1.3 Interfaccia utente

L'interfaccia di Talos è stata creata in modo che possa essere di semplice utilizzo, permettendo un rapido accesso a tutti i servizi in ogni momento. Con lo scopo di creare il meno confusione possibile, abbiamo distinto delle aree con specifiche funzionalità; possiamo quindi individuare all'interno di ogni pagina i seguenti elementi:

- una barra situata in cima e che riempie la totalità della larghezza della finestra: disegnata in stile *dock*, permette l'accesso ai servizi[2] dell'applicazione attraverso le relative icone;
- un menu verticale in stile accordion[3] posto a sinistra, che può variare a seconda della funzionalità in corso;
- una mappa situata al di sotto del dock e che occupa tutta l'area rimanente a lato del menu, per la creazione dell'ambiente e successiva visualizzazione delle strategie risultanti.

L'interazione dell'utente avviene attraverso il mouse, oltre che con il semplice click, anche mediante eventi *onmouseover* in determinate aree. In Figura 5.1 possiamo vedere l'interfaccia dell'editor delle mappe, i diversi elementi fin qui esposti sono facilmente identificabili al suo interno.

5.2 Architettura

Abbiamo già esposto le necessità che ci hanno spinto a scegliere di sviluppare l'applicativo come un servizio online, una web-application fruibile da rete Internet, senza alcuna necessità di particolari requisiti software o hardware e soprattutto multi piattaforma per definizione. Tuttavia sono state diverse le difficoltà che abbiamo incontrato e risolto allo scopo di trovare una soluzione che risultasse essere funzionale e al tempo stesso elegante.

Iniziamo presentando nella Sezione 5.2.1 le principali tecnologie software, quali i linguaggi, le librerie e le tecniche impiegate per lo sviluppo dell'applicazione. In seguito nella Sezione 5.2.2 analizziamo in dettaglio la struttura

[2]Si tenga presente che i servizi offerti e visualizzati in quest'area possono variare a seconda che l'utente sia autenticato o meno.

[3]L'accordion è una lista verticale di elementi impilati, dove ogni componente può essere espanso rilevando il contenuto ad esso associato.

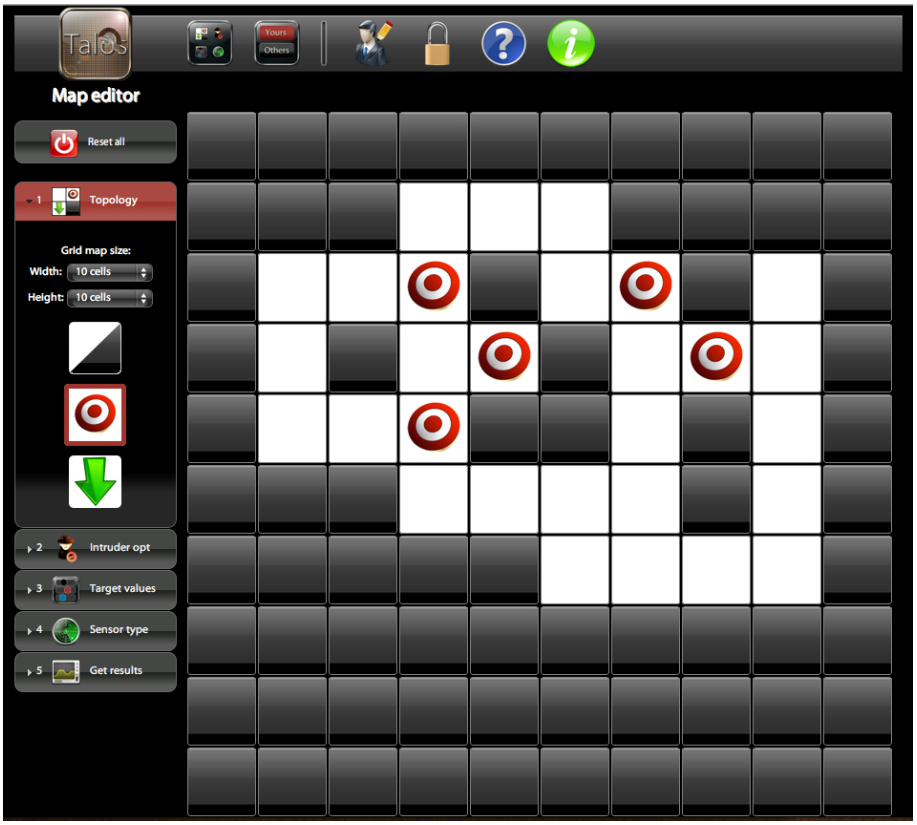


Figura 5.1: L'interfaccia utente per l'editor delle mappe, l'ambiente qui disegnato riporta l'esempio di Figura 3.1.

della web-application presentando i paradigmi adottati e le diverse parti che la compongono. Per ultimo nella Sezione 5.2.3 esaminiamo la struttura dell'API evidenziando alcune delle sue funzionalità.

5.2.1 Tecnologie impiegate

La nostra web-application si presenta come un sito dinamico, sviluppato mediante l'uso di PHP come linguaggio di scripting lato server, che fa uso di un base di dati MySql™ per il mantenimento di tutti i dati. Il framework PEAR (PHP Extension and Application Repository) è stato utilizzato per effettuare le connessioni e le interrogazioni del database, oltre che per la gestione del servizio di notifica via email che presentiamo più avanti nella Sezione 5.3.

Come linguaggio di scripting lato client abbiamo scelto di impiegare *jQuery*, una libreria cross-browser basata su *JavaScript*, che, con un linguaggio davvero user friendly, ci ha permesso di creare tutte le animazioni, sele-

zionare elementi all'interno del DOM per la loro modifica, gestire gli eventi di interazione con l'interfaccia e sviluppare tecniche *Ajax* per comunicare con l'API, che presentiamo nella Sezione 5.2.3.

La risoluzione dei problemi lineari interi di ottimizzazione è affidata ad alcuni software di programmazione matematica su larga scala, la loro specifica funzione viene analizzata in dettaglio nella Sezione 5.4. In realtà questa parte del software era già presente, il nostro lavoro ha riutilizzato ed esteso i modelli e gli algoritmi con le tecniche di riduzione esposte nel Capitolo 4.

Per ciò che concerne lo stile abbiamo scelto di utilizzare *CSS*, uno standard ormai di ampia diffusione su tutti i browser e che permette di avere una maggior flessibilità e controllo nelle specifiche di presentazione, raggruppando in pochi fogli di stile tutto ciò che occorre per definire il design dell'applicazione.

5.2.2 Struttura della web-application

Oltre al classico paradigma client-server tra utenti e web server, la necessità di dover eseguire il calcolo delle strategie su di un computer separato, denominato *Jobe* e situato in area protetta, ci ha indotto a sviluppare anche un *front-end* e un *back-end*.

Il primo, raggiungibile da rete Internet, ha lo scopo di gestire le connessioni verso gli utenti e raccogliere e conservare nella base di dati le variabili di ingresso (i patrolling setting) e uscita (le strategie); il secondo invece, situato nella rete locale del DEI, comunica con il front-end su protocollo http mediante l'API ed elabora il calcolo vero e proprio delle strategie di pattugliamento. Entrambi i computer sono dotati di una base di dati MySql che viene costantemente sincronizzata, attraverso l'API, per ciò che concerne i patrolling setting e i risultati di esecuzione. In Figura 5.2.2 viene mostrato il diagramma della struttura di rete, mostrando i diversi elementi che la compongono.

5.2.3 Application Program Interface

La necessità di far comunicare tra loro front-end e back-end mantenendo la consistenza e sincronizzazione delle due basi di dati, ci ha spinto a sviluppare una REST-API (Representational State Transfer) che permetta di trasferire le informazioni tra i due server in maniera automatica e del tutto trasparente, mediante un sistema di richiesta e risposta del tutto simile a quello impiegato nel protocollo http; inoltre sono stati previsti meccanismi di recupero in caso di un fallimento iniziale della connessione. Possiamo quindi distinguere due parti che compongono l'API: la prima, situata sul

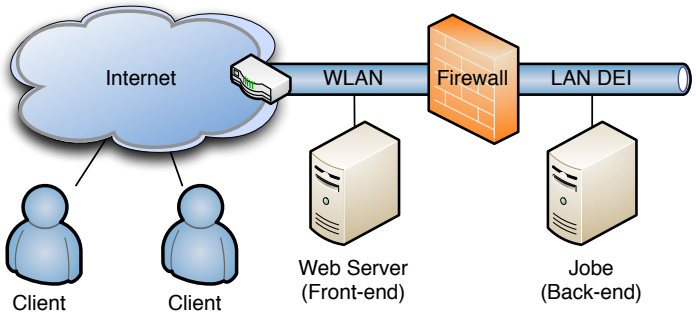


Figura 5.2: Diagramma di rete che rappresenta la struttura della web-application.

web server, ha lo scopo di interagire sia con l'utente per, ad esempio, l'invio in tempo reale dei dati di sessione che compongono il patrolling setting corrente durante la creazione di una mappa; sia con il server di calcolo Jobe per l'invio dei dati e la successiva ricezione dei risultati strategici; la seconda invece, situata sulla workstation in area protetta, ha lo scopo di ricevere i patrolling setting dal web server, gestire l'esecuzione vera e propria dello script per la risoluzione delle strategie e inviare il risultato all'API del server web, per il loro mantenimento nella base di dati. Lo scambio di informazioni tra le due parti avviene su protocollo http mediante l'utilizzo di JSON (JavaScript Object Notation) per la serializzazione delle strutture dati da inviare.

5.3 Servizi

In questa sezione presentiamo tutti i servizi che la nostra web-application fornisce, presentandoli in una sorta di ordine cronologico derivato dall'utilizzo stesso dell'applicazione. Iniziamo presentando nella Sezione 5.3.1 il funzionamento del servizio automatico di registrazione e attivazione degli account utente. Successivamente nella Sezione 5.3.2 analizziamo il funzionamento dell'editor delle mappe per l'impostazione del problema: la creazione dell'ambiente e il setting delle variabili ad esso annesse, così da poter definire un patrolling setting che sia coerente col modello. Nella Sezione 5.3.3 spieghiamo in che modo avviene il processo di risoluzione delle mappe, elencando brevemente i software impiegati, approfondiamo questo aspetto nella Sezione 5.4. Nella Sezione 5.3.4 presentiamo il map-manager il cui scopo è quello di permettere ad ogni utente di gestire il proprio archivio di mappe e strategie annesse, oltre che poter prendere visione delle strategie e delle mappe eseguite da altri utenti. Nella Sezione 5.3.5 descriviamo il servizio

di valutazione delle strategie messe in atto dal pattugliatore, delineando gli aspetti grafici impiegati per loro esposizione. Inoltre viene qui mostrato il meccanismo di comparazione delle strategie risultanti su di una stessa mappa.

5.3.1 Registrazione e gestione utenti

Come abbiamo già detto nelle specifiche uno dei requisiti che l'applicazione deve avere è quello di poter gestire account multipli, così che ogni utente possa essere autenticato e possano essere identificate le mappe a lui associate. Devono pertanto essere definiti dei meccanismi, oltre che di autenticazione ad ogni nuova sessione per gli utenti già esistenti, anche di registrazione e attivazione per i nuovi utilizzatori. Il sistema prevede dunque una semplice pagina di registrazione dove poter definire il proprio nome utente, password e indirizzo email: la verifica della validità di tale indirizzo viene effettuata mediante l'invio di un link personalizzato per l'attivazione del proprio account, altrimenti inutilizzabile. L'indirizzo verrà impiegato in seguito dal sistema di notifica una volta calcolata una strategia di pattugliamento richiesta dall'utente. É stata inoltre definita una pagina per la modifica dei dati personali, quali password e indirizzo di posta elettronica per gli utenti già esistenti.

5.3.2 Editor

L'editor delle mappe è stato il punto di partenza di Talos: una pagina web che permetta all'utente di impostare il pattrolling setting: disegnare la griglia che descrive l'ambiente e configurare tutte le variabili necessarie alla formulazione del problema.

Al fine di rendere il procedimento il più semplice possibile, abbiamo identificato cinque passi principali per l'impostazione del gioco, che devono essere eseguiti cronologicamente, soddisfacendo determinati vincoli di successione controllati in modo automatico ed eventualmente segnalati all'utente. Riportiamo qui di seguito le operazioni effettuabili in ciascun passaggio con i relativi vincoli da soddisfare per procedere al successivo:

- nel Passo 1 è possibile definire la topologia dell'ambiente, le dimensioni della mappa (nei termini di numero di celle), le zone percorribili dagli agenti (in bianco) e non percorribili (in nero), la posizione degli obiettivi all'interno delle zone libere, e l'ubicazione di eventuali

punti di ingresso[4], come previsto dall'estensione esposta in precedenza; i vincoli da soddisfare per il Passo 1 sono che la topologia della mappa sia completamente connessa e esistano almeno due obiettivi da pattugliare;

- nel Passo 2 è richiesto di impostare le preferenze relative all'intruso, la strategia adottata, se strettamente competitiva o meno e il rinforzo negativo ricevuto in caso di cattura; non ci sono vincoli per questo passo in quanto vengono assunti di default una strategia strettamente competitiva e un valore di cattura per l'intruso pari a -1;

- nel Passo 3 è necessario definire tutti i valori per ciascun obiettivo, i tempi di penetrazione, il valore per il pattugliatore ed eventualmente quello per l'intruso, a seconda che la strategia selezionata al Passo 2 sia strettamente competitiva o meno; il vincolo per questo passo è che siano stati impostati tutti i valori necessari per ciascun target, non è altrimenti possibile procedere oltre;

- nel Passo 4 è possibile selezionare la portata del sensore per il pattugliatore, se limitata alla singola cella in cui si trova o aumentata ai nodi limitrofi secondo quattro livelli selezionabili; non ci sono vincoli in questo passo in quanto anche qui viene assunta di default una portata limitata alla cella corrente;

- nel Passo 5, l'ultimo, viene selezionata la modalità di riduzione desiderata per l'esecuzione, se senza riduzione, con rimozione delle strategie dominate, con astrazioni senza perdita o con astrazioni con perdita di informazione; viene inoltre richiesto di scegliere un nome alfanumerico da associare all'esecuzione corrente, tale nome viene impiegato all'interno del map manager per l'identificazione della mappa.

Abbiamo visto un esempio dell'editor in Figura 5.1 quando abbiamo parlato più genericamente dell'interfaccia utente, si noti il menu laterale a sinistra che identifica i cinque passi necessari alla definizione del problema: ogni elemento di tale menu accordion si espande in seguito al click del mouse, mostrandone il contenuto. Nell'esempio in Figura 5.1 vengono mostrate le opzioni del Passo 1 per la definizione della topologia della mappa.

Grazie all'ausilio di tecniche Ajax e dell'API da noi implementata, tutte le interazioni dell'utente con l'interfaccia dell'editor, quali l'impostazione delle aree libere piuttosto che il setting di una variabile, vengono salvate in

[4]Si tenga presente che in assenza dei punti di ingresso si assume che l'intruso possa entrare in qualsiasi punto dell'ambiente, obiettivi compresi.

tempo reale nei dati di sessione, evitando in questo modo che un cambio della pagina, ad esempio per esplorare il manager, possa causare un'improvvisa perdita di informazione.

5.3.3 Risoluzione dei setting

La risoluzione del problema è, come abbiamo già detto, affidata alla workstation di calcolo Jobe, connessa in rete locale al web server su cui risiede il front-end tramite protocollo http. Attraverso l'API dunque i due server comunicano fra loro, inizialmente per lo scambio del patrolling setting e, una volta calcolata la strategia, per la permuta dei risultati, il tutto in maniera assolutamente trasparente da parte dell'utente.

Oltre alla strategia di pattugliamento viene anche fornito un log contenente tutte le informazioni di esecuzione, comprensive di eventuali errori che potrebbero essere stati generati durante l'esecuzione, tale file è disponibile per il download in formato testuale.

Come già abbiamo accennato in precedenza il calcolo delle strategie avviene mediante software licenziatari per la risoluzione dei problemi di programmazione matematica, lo specifico uso di questi programmi viene discusso più avanti nella Sezione 5.4, presentando il flusso di esecuzione effettuato.

Il sevizio di risoluzione dei setting agisce calcolando una soluzione alla volta, attraverso l'uso di una pila nella base di dati contenente tutte le esecuzioni con relativo stato: *ricevuto*, *in esecuzione*, *eseguito*, *inviato*, *errore*. Oltre al meccanismo atto a garantire una singola esecuzione alla volta, attraverso l'uso dello stato di ogni mappa è possibile implementare anche il sistema di recupero in caso di fallimento della connessione, con lo scopo di mantenere coerenti e sincronizzate le due basi di dati su ciascun server.

Una volta calcolata la strategia è previsto un meccanismo automatico di notifica email, gestito su un server esterno mediante protocollo Simple Mail Transfer Protocol (SMTP) con messaggi in formato HTML e testuale. Tuttavia è stato anche implementato un servizio di notifiche in tempo reale all'interno dell'applicazione nel caso in cui l'utente sia ancora connesso al sistema al momento della risoluzione della mappa.

5.3.4 Gestione e condivisione

Una delle peculiarità della nostra applicazione è quella di poter gestire il proprio archivio di mappe e risultati annessi. Attraverso il gestore delle mappe (*map manager*) l'utente può prendere visione di tutte le esecuzioni finora effettuate, sia personalmente che da parte di altri utenti.

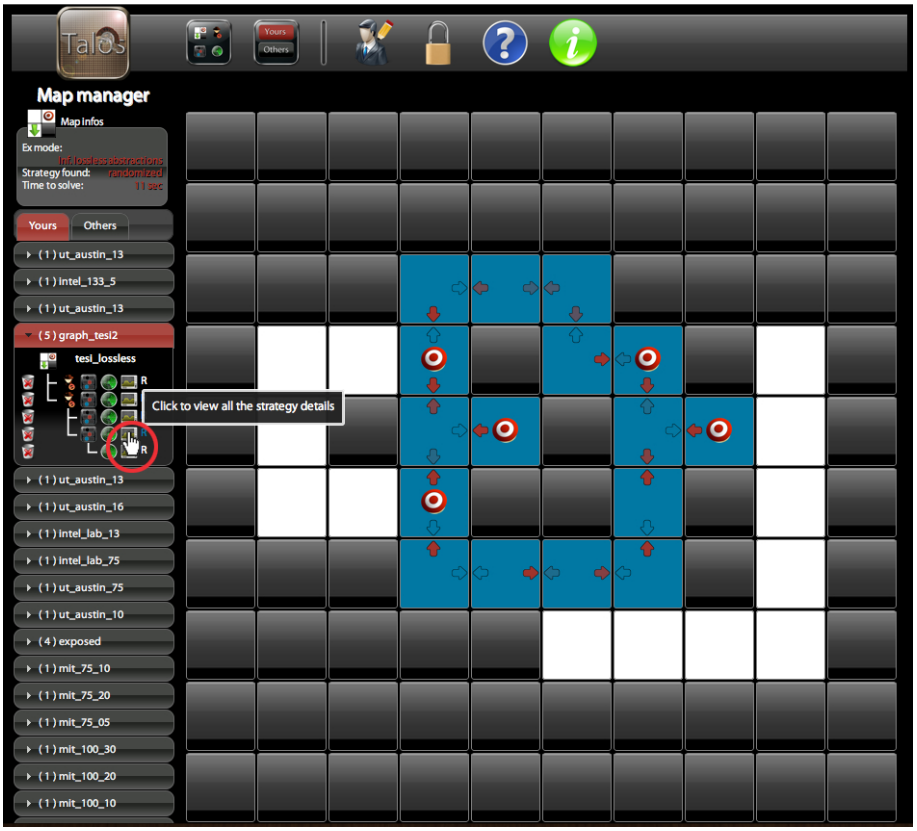


Figura 5.3: L'interfaccia utente di Talos per il gestore delle mappe.

L'interfaccia è anche qui molto semplice, possiamo vederne un esempio in Figura 5.3: a sinistra abbiamo un menu in stile accordion, dove ogni riga rappresenta una raccolta di mappe aventi la stessa topologia, a destra abbiamo la mappa che descrive l'ambiente dell'elemento espanso. All'interno di ciascun componente del menu, una volta selezionato, possiamo osservare le mappe ad esso appartenenti, elencate riga per riga, secondo una gerarchia ad albero, in questo modo è possibile avere una visione più chiara dei setting tra loro simili e che ereditano determinati aspetti da un'altra mappa. A sinistra di ogni riga è presente un cestino per l'eliminazione della relativa mappa e strategie annessa.

Facendo riferimento all'esempio in Figura 5.3 possiamo notare come il gruppo di mappe espanso sia costituito da cinque istanze differenti: tutte, eccetto che la prima, hanno le stesse impostazioni per ciò che concerne i parametri dell'intruso che sono ereditati da ogni riga; sono invece diversi tra loro i parametri riferiti agli obiettivi eccetto che per la quarta e quinta riga che si differenziano invece per la portata del sensore.

Muovendo il mouse sopra a ciascuna icona è possibile prendere visione dei parametri ad essa associati, sempre facendo riferimento all'esempio in Figura 5.3 possiamo notare come in questo momento il puntatore (cerchiato in rosso) sia situato sull'icona dei risultati, come effetto si ottiene sulla mappa un'anteprima della strategia di pattugliamento.

Attraverso il click del mouse possiamo invece copiare il patrolling setting selezionato all'interno dell'editor per una nuova esecuzione, ad esempio cliccando l'icona riferita ai parametri degli obiettivi, vengono copiati la topologia dell'ambiente, i valori riferiti all'intruso e quelli riferiti ai target, vengono invece tralasciati l'ampiezza del sensore e la modalità di esecuzione, che devono quindi essere impostati manualmente.

Oltre che la gestione delle proprie mappe è possibile prendere visione delle esecuzioni effettuate dagli altri utenti, implementando così un meccanismo di condivisione: a partire dal tab *others* è possibile visualizzare gli ultimi gruppi eseguiti di recente, permettendo, oltre che la semplice visualizzazione delle strategie risultanti, anche un meccanismo di copia per aggiungere una determinata mappa e relativa strategia alle proprie, oppure, come abbiamo già visto precedentemente, è possibile impostare l'editor delle mappe ad un determinato "stato" ereditando tutto o parte del patrolling setting di una mappa. In questo modo diversi ricercatori possono collaborare alla creazione di mappe topologicamente uguali ma con setting diversi, per valutare la bontà dei risultati strategici a fronte del cambiamento dei parametri di ingresso.

5.3.5 Valutazione e comparazione

A partire dal gestore delle mappe (*map-manager*) è possibile avere un'anteprima della strategia prodotta per ciascun mappa muovendosi con il mouse sopra la relativa icona, cliccandola potremo invece avere una visione completa di tutti i dettagli: sia dei parametri di ingresso (il patrolling setting) che delle probabilità di movimento associate alla strategia.

Ogni cella appartenente alla strategia messa in atto appare di colore blu: più chiare quelle che sono state rimosse dalle strategie dominate e dalle astrazioni, e che vengono pertanto solamente transitate dal pattugliatore; più scure quelle in cui invece viene effettuata una scelta sulla prossima azione da compiere, in questi nodi vengono mostrate le frecce con le probabilità associate. Ciascuna freccia è di colore rosso, con un'intensità diversa in relazione alla probabilità di movimento, muovendosi con il mouse al di sopra di ognuna è possibile prendere visione del valore numerico esatto.

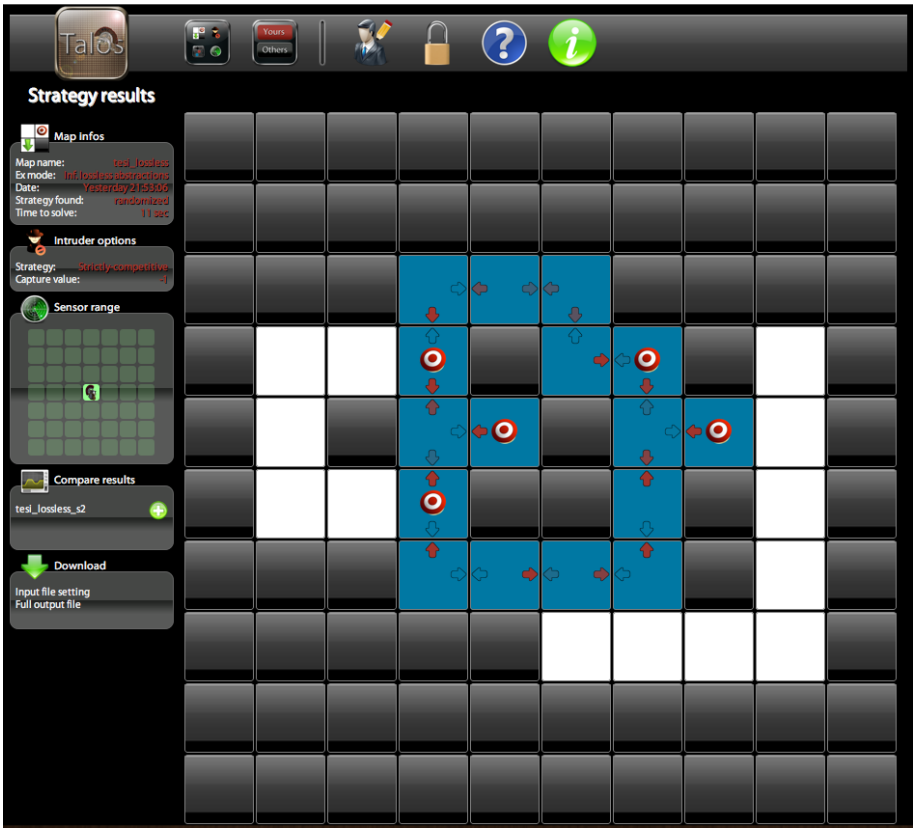


Figura 5.4: L'interfaccia utente per il visualizzatore delle strategie, la soluzione esposta è derivata dal patrolling setting di Figura 5.1.

La Figura 5.4 mostra l'interfaccia del visualizzatore delle strategie per l'esempio di Figura 5.1: il menu laterale accordion, situato a sinistra di ogni finestra dell'editor e del map manager è qui sostituito con le informazioni di esecuzione e del patrolling setting.

Nel caso in cui la soluzione trovata preveda l'utilizzo di più robot pattugliatori è possibile evidenziare la strategia di pattugliamento per ciascuno di essi attraverso un apposito box a lato: per ogni agente selezionato verranno "attivate" le relative aree all'interno della mappa.

Oltre alla visualizzazione di una singola strategia è possibile confrontare tra loro due risultati ottenuti da esecuzioni differenti. Attraverso il box laterale *compare results* viene mostrato l'elenco delle sole ed eventuali strategie con cui è possibile effettuare un paragone con quella corrente: requisito necessario è che le due mappe abbiano la stessa topologia e posizione degli obiettivi. Una volta aggiunta una mappa per il confronto, le due strategie vengono mostrate una sotto l'altra con il relativo patrolling setting a lato.

Un ulteriore box permette di effettuare il download del file di ingresso completo sotto forma di file testuale, comprendente la matrice di adiacenza che esprime la topologia della mappa e i vettori delle utilità e dei tempi di penetrazione. Inoltre è possibile scaricare il file di log relativo all'output generato dall'esecuzione.

5.4 Flusso del servizio di risoluzione

In questa sezione analizziamo il diagramma di flusso che descrive i passi eseguiti per la risoluzione di un setting, come possiamo vedere in Figura 5.5. Tale servizio è richiamato ogni qual volta è stata ricevuta da Jobe, attraverso l'API, una nuova richiesta. La risoluzione avviene attraverso l'utilizzo di MATLAB^{TM} e l'ausilio di altri software da esso richiamati. Inizialmente viene controllato che il grafo rappresentante l'ambiente sia completamente connesso e non ci siano aree separate l'una dall'altra. In caso di esito negativo la computazione termina, altrimenti si procede con l'esecuzione dell'algoritmo di Floyd-Warshall per il calcolo dei cammini minimi tra tutti gli obiettivi esistenti: se esiste un target tale per cui nessun cammino minimo che lo connette ha un tempo di percorrenza minore del suo tempo di penetrazione, allora può essere considerato *esposto* e pertanto viene rimosso dall'insieme e segnalato all'utente. In seguito procediamo con la ricerca di una strategia deterministica, come discusso in [46]. La formulazione di tale strategia avviene ricercando, nella totalità dei cammini minimi, se esiste un percorso ciclico che permette la copertura di tutti gli obiettivi, in tal caso abbiamo trovato una strategia di pattugliamento deterministica che assicura la messa in sicurezza dell'ambiente, l'esecuzione può quindi considerarsi terminata. Nel caso in cui tale cammino non esista procediamo con la rimozione delle strategie dominate, come evidenziato nella Sezione 4.1, controllando se il problema di programmazione matematica ammette una soluzione. Una volta terminata l'eliminazione delle strategie dominate proseguiamo con il calcolo delle astrazioni senza perdita di informazione per ridurre ulteriormente il problema, come abbiamo visto nella Sezione 4.3, tale risoluzione avviene attraverso l'uso di CPLEX^{TM} e AMPL^{TM}. In seguito alla determinazione del problema ridotto viene cercata una sua soluzione con l'uso di SNOPT^{TM} e AMPL^{TM}, come descritto in [7], se tale procedimento supera i limiti di memoria disponibile, o una determinata soglia temporale, proseguiamo oltre con il calcolo delle astrazioni con perdita di informazione e successiva rimozione delle dominanze iterando nuovamente il processo di risoluzione; in caso si riesca a terminare l'esecuzione senza incorrere nelle suddette limitazioni, effettuiamo un ultimo controllo sulla strategia calcola-

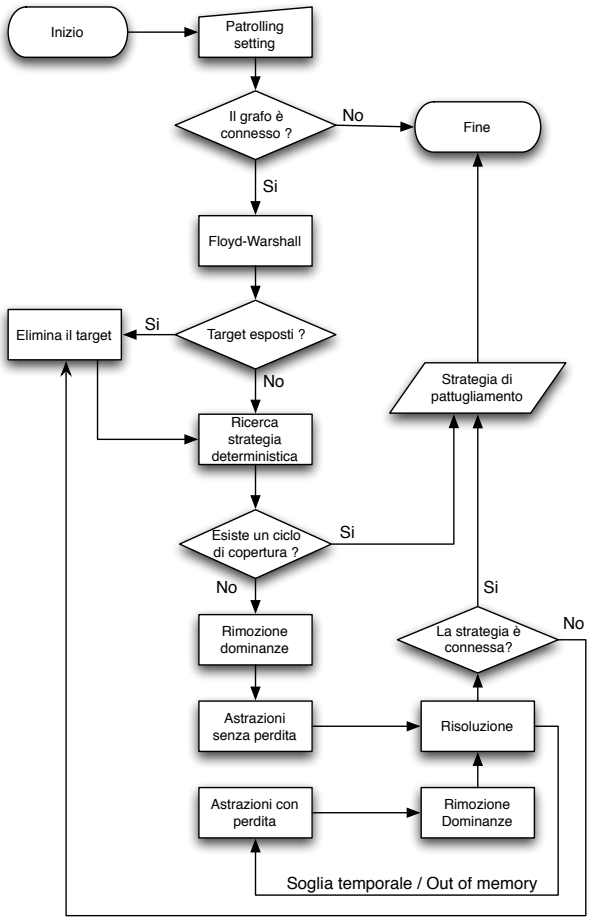


Figura 5.5: Diagramma di flusso che descrive i passi effettuati durante la risoluzione di un patrolling setting.

ta in modo che risulti completamente connessa, in caso affermativo abbiamo trovato una strategia di pattugliamento randomizzata e il processo termina; in caso negativo invece procediamo con l'eliminazione di un target esposto e il flusso riparte dalla ricerca di una strategia deterministica.

Nel prossimo capitolo esponiamo le valutazioni sperimentali delle tecniche di riduzione proposte per valutarne la validità, impiegando lo strumento software fin qui descritto.

Capitolo 6

Valutazioni sperimentali

In questo capitolo mostriamo i test sperimentali effettuati atti a dimostrare la validità delle tecniche da noi formulate e la praticità di utilizzo dello strumento software. Iniziamo nella Sezione 6.1 con alcune considerazioni sull'impostazione del problema: la generazione delle mappe, il posizionamento degli obiettivi al loro interno e le diverse densità di target considerate. Nella Sezione 6.2 riportiamo i test sperimentali che abbiamo effettuato, presentando la topologia delle mappe e i loro risultati. Il capitolo termina con la Sezione 6.3 in cui valutiamo la validità e l'efficacia delle tecniche di riduzione proposte.

6.1 Impostazione del problema

Nella Sezione 6.1.1 riportiamo le decisioni prese in merito alla creazione degli ambienti su cui abbiamo effettuato i test sperimentali. Nella Sezione 6.1.2 enunciamo le scelte effettuate per il posizionamento e il numero di obiettivi presenti all'interno delle mappe.

6.1.1 Generazione delle mappe

Come abbiamo già ampiamente detto, non esiste un dataset che possa essere impiegato per la risoluzione dei problemi di pattugliamento; pertanto, al fine di valutare la bontà delle nostre tecniche di semplificazione, quando applicate a problemi di grandi dimensioni, abbiamo creato un insieme di patrolling setting mediante una selezione di mappe dal già citato repository Radish [42], di modo che fossero le più rappresentative dal punto di vista della struttura. Gli ambienti sono stati generati a diversi gradi di risoluzione, considerando più o meno celle per descriverlo; in pratica se consideriamo n

il numero totale di nodi che compongono la mappa, maggiore è tale numero e maggiore è la risoluzione della mappa, poiché minore è l'area sottesa ad un vertice; viceversa, minore è il numero di nodi e minore è la risoluzione, poiché maggiore è l'area a cui ciascuna cella si riferisce.

6.1.2 Considerazioni sugli obiettivi

Sono diversi gli aspetti che abbiamo dovuto prendere in considerazione per ciò che concerne gli obiettivi:

- per il posizionamento degli obiettivi all'interno degli ambienti, dato che non erano presenti nelle mappe originali, abbiamo scelto di considerare anzitutto, per quanto possibile, tutti i nodi "foglia" che si trovano agli estremi dell'ambiente, ovvero tutti quei nodi con una sola cella adiacente, in questo modo evitiamo che alcune porzioni della mappa vengano tagliate dalle riduzioni perché non vi è alcun interesse in esse; in aree più ampie invece, quali ad esempio stanze, abbiamo scelto di posizionare i target al centro;

- abbiamo considerato diverse densità (δ) di obiettivi presenti, stabilendo quattro livelli percentuali sulla totalità dei nodi che costituiscono l'ambiente: 5%, 10%, 20% e 30%;

- i tempi di penetrazione $d(t)$ degli obiettivi sono stati impostati casualmente in modo che siano compresi nell'intervallo $I = \{\overline{D_t}, \overline{D_t} + 1, \ldots, 2\overline{D_t} - 1\}$, dove $\overline{D_t}$ è la massima distanza di t da tutti i vertici che compongono il grafo, questo per evitare da un lato che siano presenti obiettivi esposti e dall'altro che possano essere generate delle strategie deterministiche, non oggetto di questa valutazione;

- i payoff degli agenti per ciascun obiettivo sono stati scelti in modo che siano dei valori compresi tra 0 e 1 in una distribuzione di probabilità uniforme.

6.2 Test

In questa sezione riportiamo la struttura degli ambienti selezionati e i risultati sperimentali ottenuti in seguito alla loro risoluzione. Tutti gli esperimenti sono stati riprodotti si di una macchina con sistema operativo Linux (kernel 2.6.24) equipaggiata con due processori Intel Xeon Quad-core 2.33 GHz, 8GB di RAM e 4MB di cache, tuttavia la memoria RAM indirizzabile è limitata a 4GB a causa della struttura a 32-bit di AMPL$^{\text{TM}}$. Le computazioni

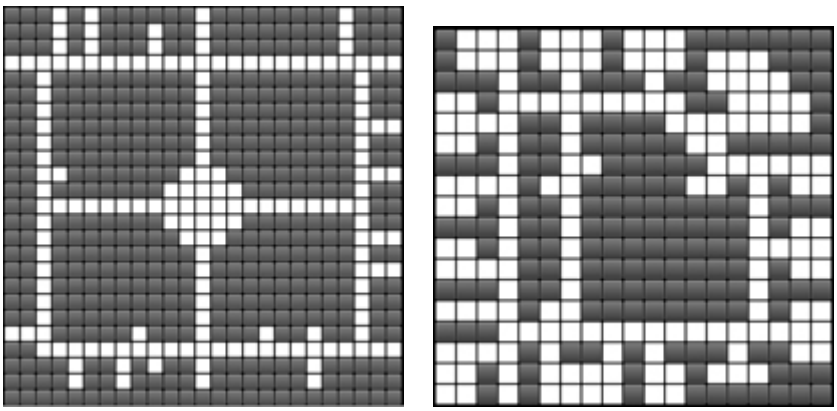
sono state effettuate senza alcuna soglia temporale, l'unico limite è dunque l'esaurimento della memoria disponibile. Abbiamo selezionato otto mappe dal repository Radish e le abbiamo create alle seguenti risoluzioni: con 50, 75, 100, 133 e 166 nodi per descrivere le aree libere.

Citiamo la nomenclatura delle mappe così come vengono riportate nel Radish: *ut_austin_aces3*, *intel_lab*, *fr079*, *ubremen-cartesium*, *albert-b-laser*, *sdr_site_b*, *kwing_wld*, *DLR-Spatial_Cognition*. In Figura 6.1 riportiamo, alla risoluzione più alta, le mappe a griglia degli otto ambienti considerati per effettuare i test.

La Tabella 6.1 mostra i risultati ottenuti sulla totalità degli ambienti generati, considerando il numero di celle n e la densità di obiettivi presenti (δ). Riga per riga vengono riportati, sia per il caso con perdita di informazione che per quello senza perdita: il tempo computazionale medio necessario alla risoluzione (in secondi), la sua deviazione standard (riportata fra parentesi e sempre espressa in secondi) e il numero di azioni non dominate per l'intruso (posto al di sotto dei precedenti). Non è stata presa in considerazione la semplificazione tramite le sole strategie dominate, in quanto producono sempre soluzioni peggiori rispetto all'utilizzo delle astrazioni senza perdita, che abbiamo visto essere in grado di elaborare soluzioni ottimali; così come la soluzione *base* senza alcun tipo di riduzione, in quanto in tutti i casi necessitava di più di 4GB di RAM.

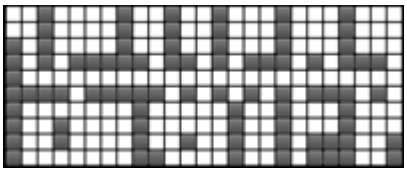
6.3 Valutazione dei risultati

I risultati esposti dimostrano un notevole guadagno rispetto alla risoluzione dei patrolling setting senza alcuna semplificazione: è stato possibile risolvere problemi che prima non erano nemmeno trattabili per la quantità di memoria richiesta. Le astrazioni senza perdita hanno dimostrato di essere una tecnica assolutamente valida, permettendoci di risolvere in maniera ottimale problemi fino a 75 vertici con il 10% di obiettivi, per problemi più grandi si esauriscono i 4GB di memoria indirizzabile. Le astrazioni con perdita hanno dimostrato di essere ancora più potenti, permettendoci di risolvere problemi fino a 166 vertici con il 10% di obiettivi, con un peggioramento dell'utilità per il pattugliatore che si aggira intorno al 5%, anche se talvolta è stato necessario diverso tempo. Tuttavia dobbiamo sottolineare che sono stati riscontrati valori molto alti nella deviazione standard di alcuni casi di risoluzione, evidenziando come la topologia di un ambiente possa essere davvero molto influente. A titolo di esempio possiamo dire che per risolvere *Austin_aces_3* con 166 nodi e il 10% di obiettivi sono stati necessari 20874 s, mentre per *Intel_lab* alla stessa risoluzione e densità di target solo 529 s.

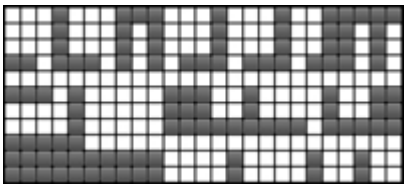


(a) austin_aces3

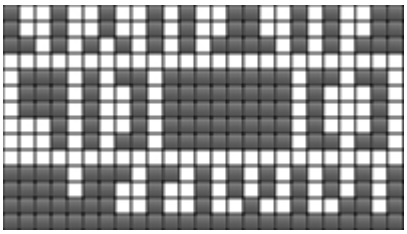
(b) intel_lab



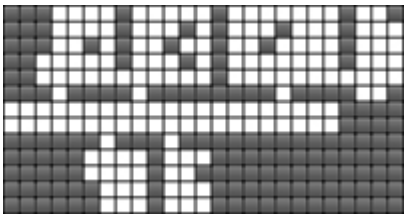
(c) fr079



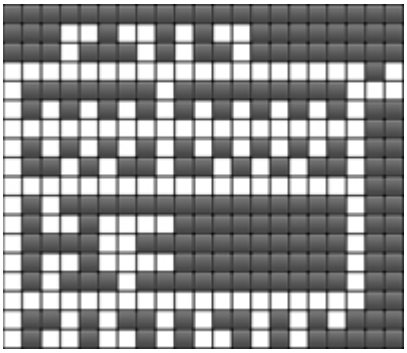
(d) ubremen-cartesium



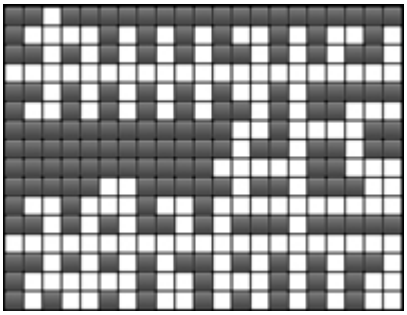
(e) kwing_wld



(f) albert-b-laser



(g) sdr_site_b



(h) DLR-Spatial_Cognition

Figura 6.1: La topologia delle otto mappe a griglia impiegate per le prove sperimentali alla massima risoluzione (166 nodi).

			percentuale di obiettivi / vertici (δ)			
			5%	10%	20%	30%
numero di vertici (n)	50	senza perdita	1.54 (2.25) 16	138.63 (209.35) 71	1676.68 (1342.76) 254	6061.99 (4947.69) 379
		con perdita	0.08 (0.01) 6	1.26 (1.48) 29	104.13 (170.51) 124	1023.46 (1571.75) 245
	75	senza perdita	394.71 (398.81) 82	6413.94 (9377.90) 238	–	–
		con perdita	0.48 (0.35) 20	76.32 (69.61) 101	1598.66 (954.95) 273	8819.38 (4177.12) 568
	100	senza perdita	–	–	–	–
		con perdita	4.62 (4.11) 28	1405.78 (1279.85) 133	7570.56 (3821.34) 434	–
	133	senza perdita	–	–	–	–
		con perdita	150.61 (186.58) 52	4857.47 (6352.03) 191	–	–
	166	senza perdita	–	–	–	–
		con perdita	817.34 (699.84) 102	16684.98 (16726.63) 335	–	–

Tabella 6.1: Tabella riassuntiva dei risultati sperimentali: per ogni riga vengono riportati il tempo necessario alla risoluzione in secondi, la sua deviazione standard (posta fra parentesi) e il numero di azioni non dominate per l'intruso.

Possiamo quindi concludere, che un'analisi sperimentale esauriente e precisa sia alquanto difficile da svolgere quando abbiamo a che fare con ambienti di topologia arbitraria, resta tuttavia il fatto che il confronto diretto tra i risultati delle tecniche proposte e il caso base su ciascuna mappa ha mostrato elevatissimi margini di crescita, sia come prestazioni (nei termini di tempo necessario alla risoluzione), che come complessità (nei termini di ampiezza del problema).

Talos ha dimostrato di essere estremamente utile durante il processo di creazione dei patrolling setting, permettendoci di creare le diverse variazioni del problema in pochissimo tempo. Anche il sistema di risoluzione a pila è stato estremamente efficace, consentendoci di accumulare decine di setting da risolvere creandoli tutti insieme e attendendo la loro risoluzione senza sovraccaricare il server.

Capitolo 7

Conclusioni

Lo studio di tecniche atte a formulare delle strategie di pattugliamento per robot autonomi è un'area dell'Intelligenza Artificiale oggetto di crescente interesse. Caratterizzati da un ambiente da sorvegliare e due agenti, il pattugliatore che deve difendere degli obiettivi, e l'intruso che deve tentare di violarli, il problema di pattugliamento è stato per lo più formulato come un gioco in cui i due agenti agiscono contemporaneamente. Il modello BGA introduce invece la possibilità che l'intruso osservi il comportamento del pattugliatore prima di definire la propria strategia, definendo il concetto di *equilibrio leader-follower*. Tuttavia la scarsa applicabilità ai problemi di grandi dimensioni, che come tali possano essere considerati verosimili, ci ha spinto a sviluppare alcune tecniche per la riduzione del problema, attraverso il concetto di *strategia dominata* e *astrazione*, il risultato è stato un netto miglioramento nei termini di ampiezza dei problemi risolvibili e tempo per effettuare la produzione delle strategie.

Grazie a questi nuovi metodi siamo riusciti a risolvere problemi più complessi e in minor tempo, ma la crescita esponenziale delle matrici di adiacenza che descrivono l'ambiente e le strategie ha reso necessario lo sviluppo di uno strumento software che permettesse una visione immediata e intuitiva delle strategie prodotte attraverso un'interfaccia grafica.

Talos ha dimostrato di essere uno strumento estremamente utile, sia nel processo di creazione dei patrolling setting, che per l'interpretazione delle strategie prodotte attraverso l'interfaccia grafica.

Sono state diverse le modifiche che abbiamo introdotto al modello al fine di renderlo ancora più verosimile: la definizione dei *punti di ingresso*, una funzione di densità di probabilità sull'ampiezza del sensore e il pattugliamento mediante più robot. Tuttavia tali considerazioni sono state elaborate

solamente per ciò che concerne l'interfaccia, sia per la creazione che per la visualizzazione delle strategie. Sarebbe pertanto opportuno che vengano estese anche ai modelli e agli algoritmi di risoluzione, così da essere effettivamente considerate nella produzione delle strategie.

Un altro aspetto di discreta rilevanza che deve essere considerato è l'utilizzo di soli software a 64-bit, permettendoci così di superare il limite altrimenti invalicabile dei 4GB di memoria RAM indirizzabile.

Il caso multi-robot rimane sicuramente lo sviluppo più interessante in quanto permetterebbe di trovare soluzioni ottimali in tutti casi, fornendo il numero minimo di robot necessari al pattugliamento dell'ambiente; tale formulazione inoltre assicurerebbe anche una riduzione della complessità necessaria a produrre le strategie di pattugliamento, dividendo il problema in più sotto problemi, attraverso il concetto di *divide et imperat*.

Bibliografia

[1] L. Martins-Filho, E. Macau, Patrol mobile robots and chaotic trajectories, in: Mathematical Problems in Engineering, Hindawi, 2007.

[2] Y. Elmaliach, A. Shiloni, G. Kaminka, A realistic model of frequency-based multi-robot polyline patrolling, in: Proceedings of the International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), 2008, pp. 63–70.

[3] N. Agmon, S. Kraus, G. Kaminka, Multi-robot perimeter patrol in adversarial settings, in: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Pasadena, USA, 2008, pp. 2339–2345.

[4] P. Paruchuri, J. Pearce, J. Marecki, M. Tambe, F. Ordonez, S. Kraus, Playing games for security: An efficient exact algorithm for solving Bayesian Stackelberg games, in: Proceedings of the International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS), Estoril, Portugal, 2008, pp. 895–902.

[5] F. Amigoni, N. Basilico, N. Gatti, Finding the optimal strategies in robotic patrolling with adversaries in topologically-represented environments, in: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Kobe, Japan, 2009, pp. 819–824.

[6] M. Simaan, J. B. Cruz, On the Stackelberg strategy in nonzero-sum games, Springer Netherlands, 2005.

[7] N. Basilico, N. Gatti, F. Amigoni, Leader-follower strategies for robotic patrolling in environments with arbitrary topologies, in: Proceedings of the International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS), Budapest, Hungary, 2009, pp. 57–64.

[8] A. Gilpin, T. Sandholm, Lossless abstraction of imperfect information games, Journal od the ACM 54 (5).

[9] A. Gilpin, T. Sandholm, T. Sørensen, A heads-up no-limit texas hold'em poker player: discretized betting models and automatically generated equilibrium-finding programs, in: Proceedings of the International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS), Estoril, Portugal, 2008, pp. 911–918.

[10] O. M. John von Neumann, Theory of Games and Economic Behavior, terza ed. 1953 Edition, Princeton University Press, 1944.

[11] O. Häggström, Finite markov chains and algorithmic applications, Cambridge University press.

[12] Y. Guo, L. Parker, R. Madhavan, Collaborative robots for infrastructure security applications, in: N. Nedjah, L. dos Santos Coelho, L. de Macedo Mourelle (Eds.), Mobile Robots: The Evolutionary Approach, Book Series on Intelligent Systems Engineering, Springer-Verlag, 2006, pp. 185–200.

[13] D. Carroll, C. Nguyen, H. Everett, B. Frederick, Development and testing for physical security robots, in: Proceedings of the International Society for Optical Engineering (SPIE) Unmanned Ground Vehicle Technology VII, 2005, pp. 550–559.

[14] A. Girard, A. Howell, J. K. Hedrick, Border patrol and surveillance missions using multiple unmanned air vehicles, in: Proceedings of the IEEE Conference on Decision and Control (CDC), 2004, pp. 620–625.

[15] V. Yanovski, I. Wagner, A. Bruckstein, A distributed ant algorithm for efficiently patrolling a network, Algorithmica 37 (2003) 165–186.

[16] A. Almeida, G. Ramalho, H. Santana, P. Tedesco, T. Menezes, V. Corruble, Y. Chevaleyre, Recent advances on multi-agent patrolling, in: Proceedings of the Brazilian Symposium on Artificial Intelligence (SBIA), Vol. LNCS 3171, 2004, pp. 126–138.

[17] Y. Chevaleyre, Theoretical analysis of the multi-agent patrolling problem, in: Proceedings of the IEEE/WIC/ACM International Conference on Agent Intelligent Technology (IAT), Beijing, China, 2004, pp. 302–308.

[18] A. Machado, G. Ramalho, J.-D. Zucker, A. Drogoul, Multi-agent patrolling: An empirical analysis of alternative architectures, in: Proceedings of the Third International Workshop on Multi-Agent-Based Simulation (MABS2002), Vol. LNAI 2581, 2003, pp. 155–170.

[19] H. Santana, G. Ramalho, V. Corruble, B. Ratitch, Multi-agent patrolling with reinforcement learning, in: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS), 2004, pp. 1120–1127.

[20] A. Glad, O. Simonin, O. Buffet, F. Charpillet, Theoretical study of ant-based algorithms for multi-agent patrolling, in: Proceedings of European Conference on Artificial Intelligence (ECAI), Patras, Greece, 2008, pp. 626–630.

[21] Y. Elmaliach, N. Agmon, G. Kaminka, Multi-robot area patrol under frequency constraints, in: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 2007, pp. 385–390.

[22] S. Ruan, C. Meirina, F. Yu, K. Pattipati, R. Popp, Patrolling in a stochastic environment, in: Proc. CCRTS, 2005.

[23] J. Marier, C. Besse, B. Chaib-draa, Solving the continuous time multiagent patrol problem, in: Proceedings of IEEE International Conference on Robotics and Automation (ICRA), Anchorage, USA, 2010.

[24] N. Agmon, V. Sadov, G. Kaminka, S. Kraus, The impact of adversarial knowledge on adversarial planning in perimeter patrol, in: Proceedings of the International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS), Estoril, Portugal, 2008, pp. 55–62.

[25] T. Sak, J. Wainer, S. Goldenstein, Probabilistic multiagent patrolling, in: Proceedings of the Brazilian Symposium on Artificial Intelligence (SBIA), 2008, pp. 124–133.

[26] N. Gatti, Game theoretical insights in strategic patrolling: Model and algorithm in normal-form, in: Proceedings of the European Conference on Artificial Intelligence (ECAI), Patras, Greece, 2008, pp. 403–407.

[27] F. Amigoni, N. Gatti, A. Ippedico, A game-theoretic approach to determining efficient patrolling strategies for mobile robots, in: Proceedings of the IEEE/WIC/ACM International Conference on Agent Intelligent Technology (IAT), Sydney, Australia, 2008, pp. 500–503.

[28] N. Agmon, S. Kraus, G. Kaminka, Uncertainties in adversarial patrol, in: Proceedings of the International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS), Budapest, Hungary, 2009, pp. 1267–1268.

[29] N. Agmon, On events in multi-robot patrol in adversarial environments, in: Proceedings of International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), Toronto, Canada, 2010, pp. 591–598.

[30] N. Agmon, S. Kraus, G. Kaminka, V. Sadov, Adversarial uncertainty in multi-robot patrol, in: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 2009, pp. 1811–1817.

[31] J. Tsitsiklis, Special cases of traveling salesman and repairman problems with time windows, Networks 22 (3) (1992) 263–282.

[32] A. Kolen, A. Kan, H. Trienekens, Vehicle routing with time windows, Operations Research 35 (2) (1987) 266–273.

[33] N. Christofides, J. Beasley, The period routing problem, Networks 14 (2) (1984) 237–256.

[34] P. Francis, K. Smilowitz, M. Tzur, The period vehicle routing problem with service choice, Transportation Science 40 (4) (2006) 439–454.

[35] B. Raa, E. Aghezzaf, A practical solution approach for the cyclic inventory routing problem, European Journal of Operational Research 192 (2) (2009) 429–441.

[36] E. Halvorson, V. Conitzer, R. Parr, Multi-step multi-sensor hider-seeker games, in: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), Pasadena, USA, 2009, pp. 159–166.

[37] J. Pita, M. Jain, J. Marecki, F. Ordonez, C. Portway, M. Tambe, C. Western, P. Paruchuri, S. Kraus, Deployed ARMOR protection: the application of a game theoretic model for security at the Los Angeles International Airport, in: Proceedings of the International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS), Estoril, Portugal, 2008, pp. 125–132.

[38] R. Fourer, D. Gay, B. Kernighan, A modeling language for mathematical programming, Management Science 36 (5) (1990) 519–554.

[39] Stanford Business Software Inc. (2010). [link].
URL `http://www.sbsi-sol-optimize.com/`

[40] Ibm ilog cplex optimizer.
URL `http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/`

[41] A. F. S. Quarteroni, Scientific Computing with MATLAB and Octave., Springer., 2006.

[42] A. Howard, N. Roy, The robotics data set repository (radish) (2003).
URL `http://radish.sourceforge.net/`

[43] N. Basilico, N. Gatti, T. Rossi, S. Ceppi, F. Amigoni, Extending algorithms for mobile robot patrolling in the presence of adversaries to more realistic settings, in: Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT), Milan, Italy, 2009, pp. 557–564.

[44] N. Basilico, N. Gatti, T. Rossi, Capturing augmented sensing capabilities and intrusion delay in patrolling-intrusion games, in: Proceedings of the IEEE Symposium on Computational Intelligence in Games (CIG), Milan, Italy, 2009, pp. 186–193.

[45] N. Basilico, N. Gatti, F. Villa, Asynchronous multi-robot patrolling against intrusion in arbitrary topologies, in: Proc. AAAI, 2010.

[46] N. Basilico, N. Gatti, F. Amigoni, Developing a deterministic patrolling strategy for security agents, in: Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT), Milan, Italy, 2009, pp. 565–572.