Università degli Studi di Bologna

FACOLTÀ DI SCIENZE MATEMATICHE FISICHE E NATURALI
Corso di Laurea Magistrale in Informatica

Tesi di laurea magistrale

Planning in giochi ad informazione parziale

Candidato:
Andrea Gasparro
Matricola 0000540558

Relatore:
Paolo Ciancarini

Anno Accademico 2010-2011

Ringraziamenti
vari

Abstract

I giochi ad informazione parziale, specialmente quelli in cui si abbia un elevato branching factor, rappresentano un interessante dominio applicativo per il testing di approcci basati sul planning: l'idea di sviluppare una strategia legata all'approfondimento di un sottoinsieme circoscritto dei percorsi possibili, in qualche modo identificato come promettente, appare ragionevole in un ambito in cui l'esplorazione esaustiva dell'albero di gioco, tradizionalmente associata alla forza bruta, risulta estremamente difficoltosa. In questo elaborato è proposta una tecnica di planning legata per la costruzione di piani su differenti livelli di priorità, al fine di sviluppare una strategia che, se da un lato deve includere piani tradizionali per gestire il breve periodo, dall'altro necessita di un sistema per sfruttare piani che siano linee guida nel lungo periodo, sopperendo così alla cecità dell'agente. Dopo una prima fase di studio del dominio, tale tecnica viene prima presentata in astratto e quindi sperimentata sul programma campione del mondo di Kriegspiel, Darkboard, con risultati incoraggianti.

Indice

Capitolo 1

Introduzione

Un problema di scacchi non è altro che un esercizio di matematica pura.
G.H.Hardy *A Mathematician's Apology*

Un problema estremamente complesso nella proposta o nella verifica di algoritmi, non necessariamente connessi al solo mondo dell'informatica, è quello di individuare domini pratici in cui testare quanto è oggetto di studio, mantenendo un sufficiente livello di generalità ed astrazione da permettere una verifica ed un testing, che non siano vincolati ad ipotesi specifiche connesse al contesto.

Sin dalla nascita, o meglio ancora prima che il termine Intelligenza Artificiale fosse coniato, i giochi sono stati un ambito d' elezione per tale scopo: si pensi all'algoritmo di Turing per giocare a scacchi, al contesto dal quale è nato il teorema di Zermelo, o all'intera teoria dei giochi che tanto impatto ha avuto su informatica ed economia, fra le altre.

Di fatto è vero anche l'inverso: le origini dei giochi strategici vengono fatte risalire alla simbolizzazione di modelli concreti e sono riproposizioni in astratto di problematiche reali, ritualizzate attraverso i secoli, le cui strategie sono però ancora riconducibili all'ambito originario: un esempio su tutti è rappresentato dal gioco degli scacchi.

L' argomento centrale di questa tesi è lo studio di una tecnica di planning, denominata *su livelli di priorità* da applicare nella ricerca con avversari ad in-

formazione imperfetta, della quale è stata sviluppata un'implementazione al fine di poter effettuare delle valutazioni sperimentali sull'efficienza dell'algoritmo, implementazione che è stata integrata nel giocatore artificiale campione del mondo di Kriegspiel, Darkboard.

La scelta di questo contesto è dovuta ad un insieme di fattori: il gioco del Kriegspiel è una concretizzazione eccellente dell'incertezza di un dominio ad informazione parziale, in cui un approccio non sufficientemente solido porta ad una rapida debacle, mentre il fattore di branching ed un insieme degli stati estremamente elevato che lo caratterizzano da un punto di vista computazionale, condizionando in negativo tentativi di analisi con la forza bruta.

Avere poi l'opportunità di contribuire allo sviluppo di uno strumento eccezionale quale è Darkboard, è stato un ulteriore motivo di interesse.

1.1 Struttura dell'elaborato

Questa tesi è suddivisa in sette capitoli, strutturati come segue:

Capitolo 2: Stato dell'arte Sono presentate alcune teniche di planning promettenti nell'ambito dei giochi ad informazione imperfetta, per poi concludere trattando un insieme ristretto di tecniche appartenenti al ramo del machine learning.

Capitolo 3: Descrizione del Dominio Viene brevemente descritto Darkboard, concentrandoci sugli aspetti più rilevanti dal nostro punto di vista, dopo una breve introduzione alle caratteristiche del Kriegspiel.

Capitolo 4: Planning su livelli di priorità È presentata una prima trattazione teorica dell'algoritmo proposto in questa tesi, analizzandone le caratteristiche e studiando i contesti in cui è applicabile.

Capitolo 5: Planning su livelli di priorità in Darkboard Viene descritta la realizzazione di un planner su livelli di priorità sviluppato al fine di essere integrato in Darkboard e sono presentati i risultati sperimentali ottenuti in questo contesto.

Capitolo 6: Goals in giochi strategici Vengono ripresi alcuni concetti introdotti nel secondo capitolo, al fine di discutere alcune tecniche interessanti per la costruzione di un goal set sfruttabile dalla tipologia di planner proposta

Capitolo 7: Conclusioni e sviluppi futuri Sono riassunti i punti di interesse identificati nel corso dell'elaborato, discutendo quanto dimostrato e presentando plausibili ulteriori sviluppi della testi sostenuta

Capitolo 2

Stato dell'arte

In questo capitolo saranno presentate brevemente alcune tecniche di planning promettenti nell'ambito dei giochi ad informazione imperfetta, soffermandoci in particolare su quelle che hanno ottenuto risultati interessanti in contesti simili a quello studiato.

Sarà quindi esaminato un insieme ristretto di tecniche appartenenti all'area del machine learning, ipotizzando possibili connessioni con sviluppi relativi alle tipologie di planner appena discusse.

2.1 Planning

Il Planning è giustamente considerato l'approccio più simile a quello adoperato dagli esseri umani, fra quelli a disposizione di un giocatore artificiale; essenzialmente vi è una rinuncia ad analizzare completamente (o nel modo più esaustivo possibile) l'albero di gioco, a favore della ricerca di una strategia che valorizzi elementi specifici della configurazione corrente, la quale cosa viene spesso effettuata ricongiungendosi direttamente o indirettamente a configurazioni conosciute in quanto analizzate precedentemente.

Fu Shannon negli anni '50 in Shannon [1950], prima ancora che fosse mai stato realizzato un giocatore artificiale di scacchi [1], a dividere i possibili approcci per la ricerca in giochi con avversari in *piano A* e *piano B*: A corrispondeva al planning, B alla ricerca sullo spazio degli stati con la forza bruta.

[1] Non considerando tale l'algoritmo eseguito *by hand* da Turing nel 1948

Storicamente, a partire dagli anni '80 il planning, nell'ambito di giochi ad informazione perfetta, è passato in secondo piano rispetto a soluzioni basate sul *piano B*, cedendo il passo agli algoritmi incentrati sull'uso della forza bruta: la ragione di ciò è da ricercarsi essenzialmente nel fatto che con la crescita della potenza di calcolo a disposizione è cresciuto il numero di stati visitabili dai giocatori artificiali e di conseguenza, unendo una strategia di gioco ottima [2] ad una visita dell'albero molto approfondita, un giocatore artificiale non fa altro che scegliere la mossa *oggettivamente* migliore[3], fra quelle visibili: un esempio significativo per comprendere quanto detto è pensare che fino al 1998 programmi basati su Minimax non erano in grado di battere i migliori Grandi Maestri di Scacchi, mentre al contrario da quella data in poi "le macchine hanno inevitabilmente superato gli umani"[4], continuando ad utilizzare lo stesso paradigma.

Il passo più naturale nell'approccio ai giochi ad informazione imperfetta, è stato tentare di riproporre un approccio basato su Minimax, a dispetto di una significativa differenza: in questi contesti il *teorema di Zermelo non vale* e di conseguenza non esiste una mossa oggettivamente ottima, solo mosse probabilmente buone[5], ed in particolare nel Kriegspiel, che è come vedremo in seguito uno degli esempi più interessanti fra queste tipologie di problemi, la conoscenza acquisita sulla posizione al momento corrente, invecchia molto in fretta.

In questo contesto, considerare la possibilità di rinunciare ad una visita approfondita dell'albero (che sarà fuori dalle possibilità dell'hardware per molte decadi, in casi complessi come Go e Kriegspiel) a favore della ricerca di *configurazioni* su cui abbiamo indicazioni positive, acquisisce un nuovo significato: guardando da questo punto di vista anche algoritmi basati sulla forza bruta, è in effetti evidente che la ricerca con avversari in giochi ad informazione imperfetta si sta evolvendo in questa direzione: un esempio significativo è il successo con cui è stato applicato l'algoritmo monte carlo al Go ed al Kriegspiel.

---

[2] si pensi a Minimax

[3] l'esistenza di tale mossa è assicurata dal teorema di Zermelo

[4] Garry Kasparov

[5] esclusi casi in cui il branching dell'albero sia ristretto abbastanza da permettere un calcolo esaustivo di tutte le possibilità

Alla luce di quanto detto, presentiamo due approcci impiegati nella costruzione di strategie che non si basino su una valutazione esaustiva dell'albero di gioco: Planning e Machine Learning.

2.2 Planning in contesti ad informazione imperfetta

Definiamo il dominio in cui opera un planner, come una tripla:

```
D = ( S, Act, G)
```

Dove S sono gli stati validi, Act l'insieme delle azioni consentite, G i goal cioè gli obiettivi da raggiungere.

Il Planning è considerato NP-Hard[6] e sono note una serie di condizioni restrittive poste sul dominio, nella teoria classica:

- Il sistema in cui si opera deve essere Finito, cioè composto da un numero finito di azioni, stati ed eventi.
- Il Controller conosce sempre lo stato della configurazione corrente S.
- Deterministico: ogni azione ha sempre un solo riusultato.
- Statico: non si verificano cambiamenti a meno di azioni da pare del controller.
- Deve esistere un insieme di Goals, che sono stati in S.
- Un piano è una sequenza ordinata di azioni.

Altro fatto noto è che l'interruzione della visita degli stati ad una profondità limite fissata, che nella ricerca di un goal è assolutamente inevitabile in contesti complessi, può portare un planner a non fornire nessuna risposta e dunque a fallire nel compito assegnatogli.

Tradizionalmente i planner vengono divisi in domain specific, domain indipendent e configurabili: le prime due definizioni rispecchiano il fatto che

[6] Nau e Smith in Smith e Nau [1993]

essi dipendano o meno da una conoscenza specifica del dominio applicativo ed abbiano subito un conseguente adattamento(o restrizione) allo stesso.

Intuitivamente, i domain specific planner non funzionano in domini differenti da quelli per cui sono stati programmati e questo li rende meno versatili e inadatti a problemi che richiedano una certa flessibilità, ciò non di meno restano la tipologia maggiormente utilizzata in applicazioni reali.

I domain independent planner al contrario, sono pensati per funzionare bene in qualunque dominio[7] e presentano vantaggi e svantaggi: in un contesto differente rispetto a quello per cui sono stati progettati non richiedono una modifica al loro approccio ma al contempo, tendono ad avere prestazioni inferiori in compiti in cui alla flessibilità sia preferibile l'elaborazione di piani specifici, rispetto a pianificatori pensati appositamente per quel determinato compito.

Alla luce di queste analisi preliminari, è facile intuire che molte tipologie di planner sono inadatte per il problema in analisi: ci concentreremo dunque su quelle più promettenti.

2.2.1 Planner Markoviani

Questa tipologia di planner ben si presta a rappresentare una realtà in cui frequentemente non si conosce lo stato corrente nella sua interezza, ma solo la probabilità del verificarsi di determinate condizioni.

La realtà in esame viene rappresentata come una tripla T= (S, A, P, G), dove:

```
S := insieme degli stati S1,...,Sn
A := insieme delle azioni possibili
P := Probabilita'di transitare da Sn ad Sm
G := Goals in S
```

Il piano elaborato non è dunque un piano classico, ovvero una sequenza di azioni, perchè non abbiamo la garanzia assoluta che ci troveremo in uno stato da cui potremo applicare una determinata azione,si tratta invece di una sequenza di coppie che mappa degli stati appartenenti ad S in determi-

[7] conforme alle condizioni restrittive specificate in precedenze

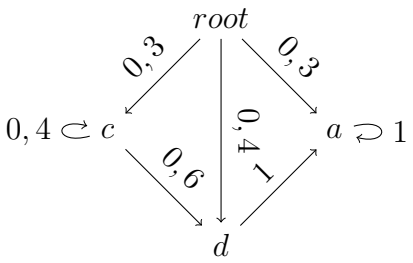


Figura 2.1: *Grafo rappresentante il flusso di un planner Markoviano in cui l'insieme degli stati è composto da (root,a,d,c) mentre le etichette in corrispondenza degli archi descrivono le probabilità di transitare da uno stato all'altro.*

nate azioni appartenenti ad A, da compiere nel caso in cui tali stati vengano effettivamente raggiunti.

Purtroppo questo tipo di approccio è chiaramente inapplicabile in ambiti complessi come il Go od il Kriegspiel, che presentano alberi di gioco con un branching factor nell'ordine di 250 per il go e di più di 30! per il Kriegspiel[8].

Anche restringendo il campo di ricerca ad un limitato set di mosse potenzialmente interessanti come può avvenire nel finale di tali giochi, questa tipologia di planner appare inadeguata.

2.2.2 Planner gerarchici

Un'altra categoria interessante da considerare sono i planner gerarchici in considerazione dei risultati ottenuti dalla loro applicazione in un gioco ad informazione imperfetta, il Bridge.

Il dominio viene rappresentato per mezzo di Task, che possono essere di tre tipologie:

```
Primitivi := Corrispondono ad azioni atomiche
Compositi := Aggregati di Task Primitivi
Goals     := Anch'essi sequenze di azioni primitive,
ma sono rappresentati come un insieme di condizioni
che devono essere verificate.
```

[8] questo aspetto viene trattato in Ciancarini [2004]

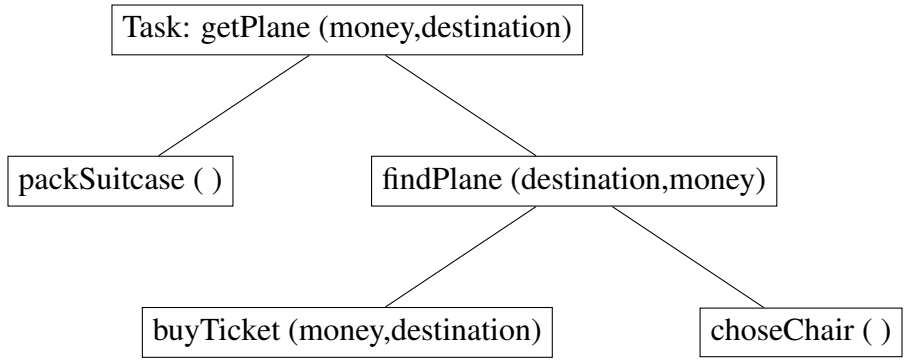


Figura 2.2: *Esempio di Hierarchical Task Network, acquisto di un biglietto aereo*

Vincoli fra Tasks sono rappresentati tramite reti, dette Reti di Task (da qui la definizione HTN, Hierarchical Task Networks): ciascuna rete può a sua volta essere utilizzata come la precondizione di un'altra rete o di un goal, la quale cosa permette di esprimere gerarchie fra vincoli e quindi modellare realtà in cui bisogna pianificare di verificare preliminarmente determinate precondizioni per il raggiungimento di un Goal.

In questo contesto si segnalano i risultati ottenuti da Bridge Baron descritti in Smith et al. [1998] applicando una versione modificata di planning HTN al bridge. Osservando che nel bridge anzichè considerare l'insieme degli stati l'insieme delle possibili configurazioni di carte, la scelta è stata di considerare uno stato come una strategia applicata da un giocatore: se in ogni nodo di un albero minimax-like si inserisce uno dei possibili piani, si ha una sorta di ibrido fra planning e forza bruta; il concetto fra l'altro non è molto dissimile da quanto descritto da Favini in G.P.Favini [2010] dove Darkboard essenzialmente associa all'algoritmo monte carlo una valutazione in termini di possibili messaggi di risposta da parte dell'arbitro.

Un altro approccio interessante in questo senso si ha in Chung [2005], dove per definire il procedimento decisionale in uno gioco strategico-real time è utilizzata una soluzione simile a quella appena descritta, ma al posto di minimax viene utilizzato monte carlo.

Riassumendo, il planning HTN è stato sperimentato con successo domini (quali ad esempio il Bridge) in cui si possa identificare un insieme di piani ristretto e in qualche modo prevedibile; in definitiva si riconduce l'incertezza

fra un grande numero di stati (che descrivono combinazioni di carte) ad un ristretto numero di piani, assumendo che sia possibile prevedere quali tipologie di piano adotterà l'avversario. In ambiti in cui la nozione di piano appare più indefinita, o potrebbe addirittura essere assente a favore di un algoritmo che calcola con la forza bruta, come nel Kriegspiel e nel Go ad esempio, non si ha notizia di tentativi in questo senso.

2.2.3 Priority planning

L' argomento non è ancora molto studiato e non si dispone di una descrizione di un paradigma standardizzato, come scarseggiano i dati sperimentali.

Questo approccio appare concettualmente non molto dissimile rispetto a quello proposto da Nau e Smith: l'idea è essenzialmente quella di definire una coda di priorità fra le azioni possibili, la cui importanza varierà dinamicamente in risposta alla percezione degli agenti coinvolti.

In questo contesto, sono da segnalare gli esperimenti di Bennewitza, Burgarda e Thrun in Bennewitza [2002] nell'ambito della coordinazione di team di automi.

2.3 Pattern recognition in giochi ad informazione imperfetta

Rispetto al Bridge, altri giochi computazionalmente più complessi e privi di fattori di casualità nel determinarsi delle configurazioni correnti, come il Kriegspiel e gli Scacchi, presentano una maggiore ambiguità nell'identificazione di cosa sia un piano: se la definizione data da Dana Nau di un insieme di azioni che portano al raggiungimento di un Goal resta valida, è in effetti la definizione di Goal che non è chiara; se come Goal considerassimo lo *scacco matto* (o equivalentemente in altri contesti, la vittoria della partita) risulterebbe impossibile definire un insieme di azioni tale da raggiungerlo nella maggior parte dei casi.

Dobbiamo dunque definire delle configurazioni auspicabili da utilizzare come Goals quando non vi siano possibilità immediate di vittoria.

Con pattern tecognition in questo ambito si intende il riconoscimento di schemi ricorrenti all'interno di insiemi di dati grezzi: se come dati grezzi consideriamo un insieme di partite giocate e come schemi i piani adottati all'interno di queste partite, appare evidente come questa disciplina sia strettamente connessa alla possibilità di ricavare piani da partite giocate.

2.3.1 Chunks

È la tecnica a cui probabilmente è stato dedicato più interesse nell'ambito del pattern recognition negli scacchi ed è stato in genere molto studiato negli anni '90 nel machine learning come in numerosi altri ambiti (dalla psicologia cognitiva alla medicina) eppure una sua definizione univoca di *cosa sia un chunk* non esiste. Ci limiteremo a dire che nell'ambito dei giochi può essere considerata come *la fotografia* di una determinata configurazione: negli scacchi ad esempio, strutturalmente può essere assimilata alla configurazione di un insieme di pezzi in una posizione.

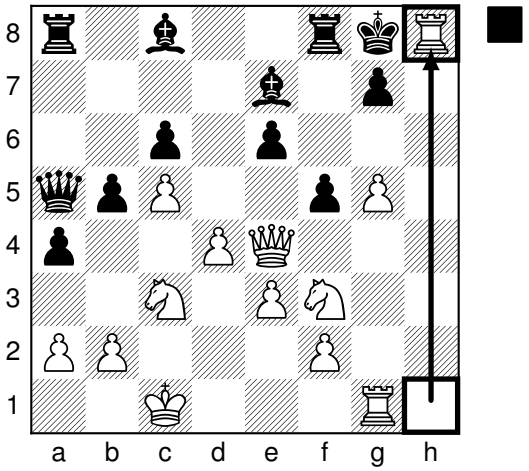


Figura 2.3: *Esempio di chunk contenuto in una posizione scacchistica*

Le caratteristiche che rendono un chunk rilevante sono:

- Occorrere con una frequenza maggiore di una soglia fissata $f \in \mathrm{R}$, all'interno del database in analisi.

- Essere significativo per la funzione di valutazione: non necessariamente questo significa discostarsi dalla media degli stati valutati, piuttosto trovarsi entro un certo arco di mosse precedenti ad una discontinuità maggiore di una certa soglia V per la funzione di valutazione: intuitivamente questo vuol dire che stiamo cercando l'origine fattori che hanno causato questa discontinuità.

- Gli elementi che compongono un chunk hanno associazioni particolarmente significative fra loro: in particolare negli scacchi, ed in generale in molte classi di giochi strategici, le relazioni fra i pezzi dipendono dalla dinamica degli stessi.

- In contesti in cui lo spazio è rilevante (ed è ragionevole affermare che lo è in quasi tutti i giochi strategici, alla luce del fatto che la quasi totalità fu originata da simulazioni di conquista o di protezione di un territorio) un chunk è una posizione assoluta[9].

Un tentativo di sviluppare un giocatore artificiale basato su chunk fu Clamp (Ericsson and Harris's, 1990) che portò alla creazione di un giocatore di livello discreto, ma sopratutto a interessanti risultati dal punto di vista didattico.

Il motivo per il quale questo approccio non è stato approfondito ulteriormente è da ricercarsi principalmente in due fattori: la quantità di configurazioni diverse che possono verificarsi in una partita di scacchi[10], ed il fatto che piccole differenze in una posizione possono mutare radicalmente il significato della stessa, rendendo dunque impossibile comprendere chunk anche solo lievemente differenti in un'unica analisi.

In particolare riguardo al Kriegspiel, c'è da segnalare che la visione parziale della scacchiera rende l'approccio ancora più difficoltoso: non conoscendo lo stato dei pezzi dell'avversario, non è nemmeno possibile essere sicuri che due chunk analoghi dal nostro punto di vista siano effettivamente lo stesso chunk in senso assoluto!

A discapito di quanto detto, il concetto di chunk rimane un punto focale nello studio di goal in giochi strategici, argomento particolarmente interessan-

[9]Per posizionamento assoluto si intende che la configurazione non è soggetta a traslazioni o altre trasformazioni: è una configurazione che può essere ritrovata esattamente come viene proposta.

[10]è stato calcolato che tale numero sia dell'ordine di 10^{42}

te ai fini di questa tesi ed in generale nella ricerca con avversari in giochi ad informazione imperfetta, dunque nel sesto capitolo di questa tesi torneremo su alcuni utilizzi possibili di collection di chunks.

2.3.2 Machine learnings e reti neurali

Implementazioni basate su reti neurali sono divenute una delle principali alternative a minimax nell'ambito dei giochi ad informazione imperfetta dopo il successo ottenuto da Tesauro in G.Tesauro [1994] che le ha impiegate per l'addestramento di un giocatore artificiale nel backgammon: TD-Gammon.

TD-Gammon addestrandosi contro se stesso è stato in grado di sviluppare strategie di gioco che lo hanno portato ai livelli dei migliori giocatori umani, sfruttando una strategia che richiama da lontano il simulated annealing: dopo aver tarato la propria funzione di valutazione attraverso autoaddestramento (altro fatto molto raro) TD-Gammon sceglie frequentemente la mossa dalla quale si aspetta i risultati migliori, ma talvolta testa anche alcune strade considerate promettenti ed aggiorna la sua valutazione in base all'esito.

È stato evidenziato come alcuni fattori determinanti negli incredibili risultati di TD-Gammon, risiedano nella particolarità del backgammon di sciogliere la configurazione corrente da quelle passate, in cui le uniche informazioni importanti sono quelle sulla situazione corrente, quelle passate sono quasi prive di interesse: è un gioco *non Markoviano* che si sposa alla perfezione con l'uso di reti neurali, anzichè con procedimenti che rendono necessaria la costruzione di un albero di gioco; indipendentemente da ciò, le reti neurali restano l'approccio principale per il machine learning di strategie complesse.

2.3.3 Opponent modelling

Questa tecnica dopo essere stata accantonata ha trovato recentemente nuove applicazioni nell'ambito dei giochi ad informazione imperfetta, proprio in Darkboard, come descritto da Favini in G.P.Favini [2010]; tradizionalmente questo approccio è stato trascurato negli scacchi ed in altri ambiti di ricerca con avversari ad informazione perfetta; nonostante ciò, alcuni tentativi sono stati fatti più o meno recentemente.

A. Jansen, D. Dowe, E.Farr in Jansen [2000] associano a partire da un DataBase di partite di Grandi Maestri di scacchi una funzione di distribuzione di probabilità che in una posizione l'avversario compia una certa mossa, per tarare attraverso di essa la funzione di valutazione in modo che, nella scelta delle mosse assegni priorità simili a quelle che stimerebbe un Grande Maestro nella stessa situazione.

Nell'ambito dei giochi ad informazione imperfetta Del Giudice, P. Gmytrasiewicz, J. Bryan in Giudice [2009] sperimentano l'opponent modelling basato su probabilità Bayesiane in una variante semplificata del Kriegspiel giocata su una scacchiera 4x4.

Come anticipato ad ogni modo, in analogia rispetto a quanto avvenuto per il planning, questo approccio per anni è stato ritenuto poco promettente per il semplice fatto che mal si sposa con l'utilizzo della forza bruta, che come detto domina in questa categoria di problemi.

Recentemente, risultati interessanti in questa direzione sono stati ottenuti da Darkboard, il quale ha proposto un approccio in un certo senso simile a quello di Jansen e Dowe: a partire da un DataBase di 12000 partite di Kriegspiel giocate da giocatori umani o da Darkboard contro giocatori umani, viene calcolata la probabilità che alla mossa m, con m $\in$ N, su una matrice 8x8 rappresentante la scacchiera, un giocatore specifico (del quale quindi si possiedono file .pgn di partite giocate in precedenza) o un opponente generico effettuino una determinata mossa.

L'idea alla base di questa strategia è l'ipotesi che in contesti in cui si possiedono scarse informazioni, un giocatore tenda a riproporre mosse che gioca abitualmente, la quale cosa renderebbe prevedibile il suo comportamento, fornendo quindi una sorta di indicatore statistico sullo stato dei pezzi dell'avversario.

In conclusione si può dire che in ambiti in cui si disponga di scarse informazioni, il profiling può risultare una risorsa preziosa per ricostruire il comportamento del proprio avversario: essenzialmente è un'ottima (vitale, citando Favini) arma difensiva.

Per quanto riguarda la pianificazione in senso tradizionale come l'elaborazione di un tradizionale piano offensivo però , l'approccio appare restrittivo: il meccanismo si sposa perfettamente con l'idea di descrivere la pericolosi-

tà di occupare alcune case della scacchiera in un certo momento di gioco ad esempio, ma utilizzarlo per sviluppare un piano stimando che obiettivi strategici si trovino in un settore specifico appare ancora abbastanza azzardato, se non in condizioni particolari (ad esempio, in apertura).

2.4 Riepilogo

Gli approcci legati al Planning in giochi ad informazione parziale ed in particolare quelli legati al Kriegspiel presentano delle difficoltà aggiuntive rispetto al Planning classico, già un problema complesso di per se: la definizione di cosa sia un piano è in se ambigua in questi ambiti, mentre le azioni possibili dipendono non più da valutazioni certe ma da stime di probabilità sull'effettiva situazione dello stato corrente, che può in alcuni casi essere completamente erronea.

In questo contesto è utile tenere presente alcune buone performances ottenute da algoritmi basati su reti gerarchiche e reti neurali, come anche il successo di alcuni approcci basati sull'estrazione di indicazioni statistiche sul comportamento di giocatori umani, a partire da database di partite giocate.

Capitolo 3

Descrizione del dominio

In questo capitolo viene descritto Darkboard, lo strumento di sviluppo scelto e presentato nell'introduzione di questo elaborato, dopo aver proceduto ad una relativamente breve introduzione al dominio applicativo per il quale è stato progettato: il Kriegspiel.

3.1 Kriegspiel

Il Kriegspiel o *Scacchi Ciechi* è una variante degli scacchi (per una trattazione più estesa di quella offerta in questo capitolo, rimandiamo ad Wiki [2012]) ed è un gioco con avversari, ad informazione imperfetta a somma zero.

Storicamente la nascita del Kriegspiel viene fatta risalire al 1899, per opera di Henry Michael Temple in Inghilterra, più precisamente a Londra (allora considerata la Capitale Mondiale degli Scacchi)[1] nel Club *The Gambit*: le prime regole di cui si abbia notizia sono dette "all'inglese, in memoria del loro luogo di origine.

Il Kriegspiel prende le mosse dal war game detto Kriegsspiele, probabilmente inventato da Georg von Rassewitz nel 1812 e già allora utilizzato dall'esercito tedesco, come simulazione di situazioni di guerra su cui addestrare i propri ufficiali.

[1] G.Kasparov, I miei più Grandi Predecessori, Vol 1

3.1.1 Regole di gioco

In ciascuna partita di Scacchi Ciechi sono coinvolti tre attori, due giocatori ed un arbitro, ciascuno dei quali ha a disposizione una scacchiera, ed un totale di due set di scacchi: un colore a ciascun giocatore ed un set completo per l'arbitro.

L'obiettivo del gioco è come negli scacchi catturare il re avversario e le regole per il movimento dei pezzi restano invariate, con la differenza che ciascun giocatore vede solo i propri pezzi e dunque può inconsapevolmente proporre mosse illegali che saranno rifiutate dall'arbitro.

Il compito dell'arbitro, unico attore a conoscere lo stato complessivo del gioco, è quello di monitorare l'andamento del gioco controllando che i giocatori effettuino solo mosse valide, notificando in caso contrario uno dei seguenti messaggi:

- illegale: se la mossa è illegale a causa di impedimenti non visibili dal giocatore.
- nonsense: se la mossa viola le regole sulla mobilità dei pezzi proprie degli scacchi

L'arbitro comunica inoltre il verificarsi di determinate situazioni di gioco regolamentari con i seguenti messaggi:

- pezzo catturato: se l'ultima mossa porta alla cattura un pezzo, l'arbitro informa entrambi i giocatori.
- scacco: se il re del giocatore che ha il tratto si trova sotto scacco (cosa che può avvenire anche a seguito di una mossa tentata dal giocatore stesso, che sarà quindi rifiutata) l'arbitro lo comunica con uno dei messaggi che vedremo in seguito.
- scacco matto: se un giocatore è sotto scacco e non puo muovere il proprio re, nè proteggerlo con un altro pezzo, l'arbitro informa i due giocatori che la partita è terminata.

Ogni messaggio dell'arbitro è pubblico: le comunicazioni sono dirette ad entrambi i giocatori, mentre ogni messaggio inviato da un giocatore all'arbitro viene notificato esclusivamente a quest'ultimo.

È opportuno sottolineare che i messaggi dell'arbitro giocano un ruolo fondamentale nelle deduzioni (potenzialmente anche erronee) effettuate dai giocatori sullo stato della scacchiera e dunque, tentare di massimizzare il numero di comunicazioni che si ricevono per ogni mossa è una strategia sensata, per poter sviluppare delle ipotesi più accurate sullo stato complessivo del gioco da parte di ciascun giocatore, ma non solo; questo pone l'accento su un aspetto trasversale nei giochi ad informazione imperfetta: le informazioni in questo contesto sono preziose.

Come già detto in precedenza, il Kriegspiel non è popolare quanto gli scacchi e la mancanza di una standardizzazione nelle regole di gioco ha portato alla nascita di differenti dialetti; quelle scelte per implementare Darkboard e qui presentate sono le più largamente condivise, giocate dalla maggioranza dei giocatori ed adottate da ICC [2] ed alle Olimpiadi di Informatica tenutesi nel 2006.

In questa variante, a seguito di ogni mossa le comunicazioni dell'arbitro possono essere:

- White move: la mossa è al bianco.
- Black move: la mossa è al nero.
- Pawn at <square> captured: l'ultima mossa effettuata ha portato alla cattura di un pedone nella casella corrispondente
- Piece at <square> captured: l'ultima mossa effettuata ha portato alla cattura di un pedone nella casella corrispondente
- Rank check: Scacco sulla Traversa: notare che in tale caso non viene comunicato se lo scacco arrivi dal lato corto o lungo.
- File check. Scacco sulla Colonna: notare che in tale caso non viene comunicato se lo scacco arrivi dal lato corto o lungo.
- Long-diagonal check: Scacco sulla diagonale lunga (rispetto al re oggetto dello scacco)

[2] Internet Chess Club Server, la più grande comunità di Kriegspiel attiva

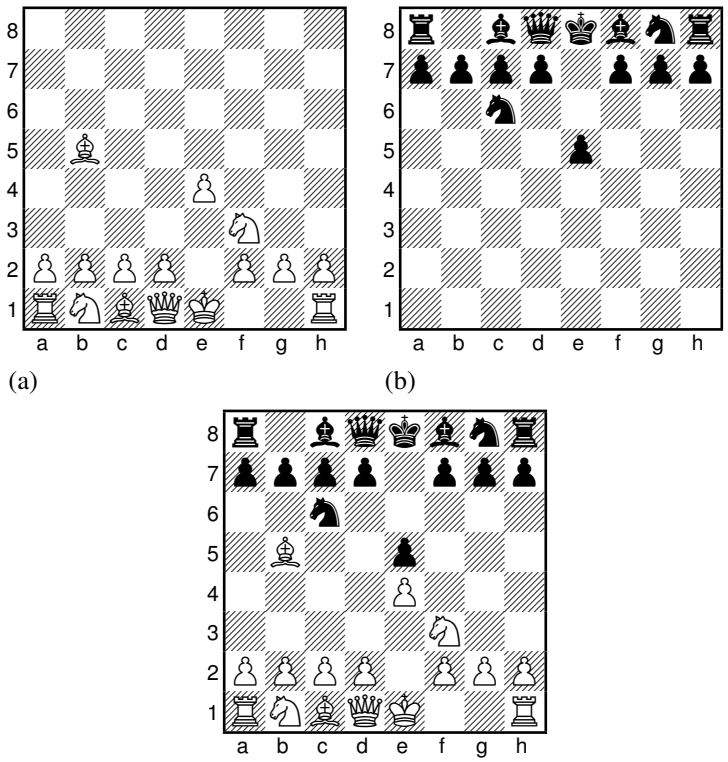


Figura 3.1: *Esempio di configurazione in una partita di Kriegspiel, raffigurante la visuale del giocatore bianco(a), del giocatore nero (b) e quella dell'arbitro*

- Short-diagonal check: Scacco sulla diagonale corta (rispetto al re oggetto dello scacco)
- Knight check: Scacco di Cavallo.
- «INT> pawn tries: Il giocatore che ha il tratto ha a disposizione INT catture di pedone.

3.1.2 Il Kriegspiel da un punto di vista computazionale

Il Kriegspiel, sebbene abbia regole simili al gioco degli scacchi, appartiene alla classe di giochi che come Go, Phantom Go etc, risultano estremamente complessi per un giocatore artificiale a causa del loro branching factor: è significativo considerare che se negli scacchi il branching è nell'intorno delle 35 mosse,in Kriegspiel questo potrebbe essere approssimativamente intorno a 35!

Per avere un'idea di come sia possibile calcolare questo dato, fra le diverse scuole di pensiero a riguardo ci appoggeremo al metodo utilizzato in Ciancarini [2004]: al contrario di quanto avviene negli scacchi, in Kriegspiel bisogna considerare anche i possibili tentativi rifiutati e dunque le possibili *sequenze di tentativi*: questo è rilevante in quanto, una diversa sequenza di tentativi porta ad avere diverse informazioni da parte dell'arbitro, quindi una diversa visione dello stato corrente.

Quello che contiamo in questo caso è dunque il numero di *semimosse possibili* in una posizione: se le mosse possibili sono N, le sequenze possibili sono N*(N-1)*(N-2)...*2.

L'insieme degli stati invece è esattamente lo stesso degli scacchi, quindi come detto pari a 10^{42}

3.2 Darkboard

Darkboard è il programma attualmente campione del mondo, detentore della medaglia d'oro assegnatagli alle Olimpiadi di Informatica del 2006 e del 2009 come miglior giocatore artificiale di Kriegspiel, sviluppato da G.Favini sotto la supervisione del Prof. P.Ciancarini, presso il Dipartimento di Scienze dell'Informazione dell'Alma Mater Studiorum di Bologna.

Per una trattazione estesa sul funzionamento di Darkboard si rimanda a G.P.Favini [2010]; l'obiettivo di questo paragrafo è descriverne la struttura, in modo da contestualizzare l'ambito nel quale testeremo la tecnica di planning proposta in questo elaborato di tesi.

La gestione di una partita da parte di Darkboard è suddivisa in tre momenti distinti, affrontati con approcci distinti e che dunque saranno presentati separatamente.

3.2.1 Apertura

I giocatori artificiali di scacchi ormai convergono sul gestire questa prima fase attraverso la creazione di Libri di Aperture, cioè sequenze di mosse ormai consolidatesi nel tempo e che la teoria scacchistica ha nei secoli individuato come le varianti migliori di determinate mosse iniziali: essenzialmente si

tratta di sequenze, abitualmente comprese fra le 5 e 10 mosse, che permettono al giocatore artificiale di risparmiare tempo e potenza di calcolo, nel caso in cui le aperture si svolgano secondo schemi consueti, oltre ad avere la fondamentale funzione di evitare le numerose trappole studiate in avvio di partita.

La difficoltà più significativa incontrata nel trasporre questo approccio al Kriegspiel è che in questo ambito non è presente una teoria ben studiata come per gli scacchi, ma questa via resta percorribile: Darkboard ha infatti creato libri di aperture appoggiandosi alle sequenze di mosse iniziali messe in pratica dai migliori giocatori umani su un database di 12.000 partite raccolte su ICC[3].

Al momento della ricezione del primo messaggio dell'arbitro che riporti informazioni sul gioco avversario, la fase di apertura è considerata conclusa e si passa al Mediogioco.

3.2.2 Mediogioco

Il mediogioco è il momento più creativo e ricco di incertezza dell'intera partita sia negli Scacchi che nel Kriegspiel, il che equivale a dire che è la fase in cui il branching factor dell'albero di gioco è più ampio: in questa fase non si hanno pattern noti da seguire o semplificazioni a disposizione: è il momento in cui pesa maggiormente la capacità di pianificare e calcolare sequenze di mosse dinamicamente, cioè su situazioni potenzialmente nuove, ed altresì la fase in cui maggiormente emerge l'aspetto computazionale in astratto rispetto a quello legato alle specifiche regole del gioco.

In questo momento di gioco si inquadrano molte delle innovazioni apportate da Darkboard allo studio del Kriegspiel ed in generale ai giochi ad informazione imperfetta:

Monte carlo tree search

Un'evoluzione del metodo monte carlo standard, proposto da E.Fermi, J. von Neumann e S.M. Ulam negli anni '40 che appartiene alla famiglia dei metodi statistici non parametrici, in cui essenzialmente il computer simula un ampio numero di partite che potrebbero seguire alla scelta delle mosse disponibili

[3]Internet Chess Club

nella configurazione corrente, per ricavarne indicazioni statistiche sulla bontà di tali mosse.

Nella versione *tree search*, l'algoritmo si basa sulla ripetizione, fino all'esaurimento del tempo disponibile, dei seguenti quattro passi:

Scelta viene scelta una foglia dell'albero basandosi sul numero di visite e dei i risultati medi ottenuti.

Espansione in questa fase é possibile aggiungere nuovi nodi all'albero di gioco.

Simulazione il punto di maggior rilievo: secondo differenti metodologie che dipendendono dal contesto in cui il metodo è impiegato, viene simulato il proseguimento della partita sin dallo stato corrente e la media dei risultati prodotti è restituita in output ed associata alla foglia.

Backpropagation Il valore associato a ciascun nodo è propagato verso l'alto a ciascun padre fino alla radice ed i valori di ciascun nodo sono ricalcolati.

Al termine di questo ciclo l'algoritmo sceglie il nodo che ha ricevuto il maggior numero di visite e *non quello con il valore statistico più alto, che è sperimentalmente risultato essere un indicatore inaffidabile*, là dove il nodo da visitare viene scelto in base alla formula:

$$Ui = V_i + c * \sqrt{\frac{lnN}{n_i}} \tag{3.1}$$

Dove V_i è il valore del nodo i, N è il numero di volte in cui il nodo padre di i è stato visitato, n_i è il numero di volte in cui i stato visitato e *c* è una costante che può essere modificata verso il basso nel caso in cui si voglia favorire l'exploitation, verso l'alto nel caso in cui si voglia favorire l'esplorazione.

Monte carlo tree search in Darkboard

Il problema principale nell'applicare monte carlo al Kriegspiel è che simulare per intero una partita dallo stato corrente non fornisce indicatori statisticamente rilevanti: il branching nel mediogioco è talmente ampio che simulare

per un tempo ridotto i possibili sviluppi risultava in un comportamento non distinguibile dal giocatore casuale[4].

La scelta implementativa è stata quella di utilizzare una versione del MC-TS in cui non viene simulata la scacchiera dell'avversario, una volta constatata l'inefficenza che ne derivava, bensì i messaggi che è possibile ricevere da parte dell'arbitro in seguito ad ogni azione.

In aggiunta, ad ogni istante una matrice 8x8 rappresentante la scacchiera descrive la probabilità che un pezzo appartente ai tre gruppi Pedoni, Pezzi, Re si trovi sulla casa *i,j* e tali indicatori sono utilizzati per meglio modellare la scelta dei messaggi che verranno restituiti dall'arbitro durante la simulazione.

La probabilità che un pezzo occupi una casa sulla matrice è calcolata in base alle regole sulla mobilità dei pezzi, in base ai messaggi ricevuti dall'arbitro negli ultimi n turni(messaggi ricevuti prima di un certo lasso di tempo sono considerati inattendibili in quanto come detto, nel Kriegspiel la conoscenza attuale invecchia e diviene inservibile in modo molto rapido, a causa della grande mobilità dei pezzi e della scarsità di informazione) ed ultimo ma forse più importante, basandosi sul meccanismo di profiling anticipato nel secondo capitolo, che vedremo in dettaglio nel prossimo paragrafo.

Un ulteriore discostarsi di Darkboard dall'algoritmo monte carlo standard si ha per quanto riguarda la lunghezza delle simulazioni effettuate; sperimentalmente è emerso che più sono profonde le simulazioni, minore è la precisione nell'approssimare lo stato della scacchiera e di conseguenza l'affidabilità del risultato: i tentativi che hanno dato risultati migliori sono le simulazioni ad un passo ed a 4 passi, al termine delle quali l'indicatore sulla bontà della posizione ottenuta è ricavato in base al calcolo del materiale residuo per Darkboard e per l'avversario.

Attualmente, la versione più performante di Darkboard è quella che effettua simulazioni ad un passo, esemplificata in figura:

[4]G.P.Favini,in G.P.Favini [2010]

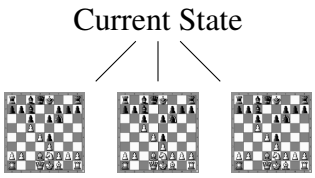


Figura 3.2: *Esempio di visita monte carlo ad un passo: questa è in realtà una semplificazione in quanto ciascuno stato è una coppia di metaposizioni in Darkboard, ma ai fini della nostra trattazione abbiamo preferito astrarre su questo meccanismo, che non giocherà un ruolo determinante nella gestione del planner*

Un'ultima considerazione, interessante anche ai fini dell'algoritmo che presenteremo, è che l'utilizzo del metodo monte carlo riduce significativamente la conoscenza specifica del dominio necessaria al programma, rendendo l'approccio al mediogioco di Darkboard interessante non solo dal punto di vista del Kriegspiel.

Per una trattazione più approfondita sull'uso del metodo monte carlo nel caso specifico del Kriegspiel si rimanda a Ciancarini e Favini [2009a].

Profiling

L'opponent modelling in ambito di giochi strategicamente complessi come gli scacchi è stato per molto tempo e con alcune valide ragioni [5] considerato un approccio scarsamente attrattivo, ma parlando di giochi ad informazione parziale è significativo soffermarsi su alcuni aspetti.

Durante il mediogioco il *goal* ricercato sia da giocatori artificiali che da giocatori umani è contingente alle situazioni che si verificano sulla scacchiera; salvo eccezioni non è più direttamente correlato allo scopo finale del gioco(lo scacco matto in questo caso), bensì ad obiettivi che si possono considerare *locali*, cioè a quello che scacchisticamente viene definito come miglioramento della posizione.

Una previsione che si basi su abitudini di gioco in questo contesto rischia per tanto di trascurare parametri oggettivi rilevanti, che dovrebbero incidere nella scelta delle strategie da impiegare.

[5] di cui abbiamo discusso nel capitolo precedente, nel paragrafo relativo all'Opponent Modelling

L' intuizione che ha portato all'approccio usato in Darkboard è che in un certo senso, *i parametri oggettivi nel Kriegspiel sono effettivamente poco oggettivi*: si ha una scarsa conoscenza della situazione reale di gioco ed anche eventi, che in un contesto ad informazione perfetta devierebbero l'attenzione di un giocatore dai suoi consueti schemi di gioco verso considerazioni relative alla situazione contigente, in questo caso spesso non forniscono informazioni abbastanza significative per elaborare altri schemi di gioco. L'importanza, anche in termini psicologici dei pattern ricorrentemente messi in atto da un giocatore, è drasticamente maggiore a fronte della scarsità di informazioni disponibili.

Essenzialmente l'ipotesi è che in contesti in cui si possiedono scarse informazioni, un giocatore tenda a riproporre mosse e schemi che gioca abitualmente e dunque se tali schemi sono opportunamente identificati è possibile ricavare altri preziosi indicatori sullo stato del gioco.

Il modello sviluppato da Darkboard è originato da un Database di 12000 partite in cui semplicemente si effettua, in corrispondenza dell'istante di gioco corrente, una ricerca statistica sulla probabilità che l'avversario occupi determinate case della scacchiera.

Questo semplice meccanismo viene raffinato con alcune ottimizzazioni contingenti alle situazioni specifiche della partita: in base al colore scelto dal giocatore il modello cambia, viene applicata una sorta di neighboring col procedere del gioco, in considerazione della già menzionata tendenza insita nello stesso ad una volatilità delle informazioni che erano valide anche solo poche mosse addietro.

Si ha inoltre la suddivisione dei modelli in due differenti tipologie: modelli specifici per uno specifico avversario noto ed un modello generico per un avversario sconosciuto.

3.2.3 Finale

I risultati sul finale ottenuti sono forse l'apporto più significativo fornito da Darkboard al Kriegspiel, dal punto di vista del gioco in se.

Per mezzo di un approccio che richiama quello delle tavole per i finali attualmente utilizzate nei programmi di scacchi professionali (essenzialmente costruite attraverso un'analisi brute force retrograda a partire dalle foglie di

un albero in cui vengono inizialmente memorizzate delle metaposizioni rappresentanti finali vinti) il programma è in grado di offrire gioco perfetto nel caso pessimo, in situazioni cui si sia effettivamente in presenza di un finale vinto, riconducendo la trattazione dello stesso ad un caso particolare di gioco ad informazione perfetta.

Segnalazione necessaria é che questi risultati non hanno precedenti nell'ambito del Kriegspiel e sono dunque di assoluto interesse, ma vista la complessità dell'argomento ed il fatto che esula dall'interesse di questo elaborato, per una trattazione esaustiva si rimanda a G.P.Favini [2010] ed a Ciancarini e Favini [2010a].

3.3 Riepilogo

Riepilogando quanto visto, il Kriegspiel è un gioco ad informazione parziale con avversari ed a somma zero, molto complesso per un giocatore artificiale: le informazioni sono scarse, invecchiano in fretta e l'incertezza è non monotona, il branching factor è nell'intorno del fattoriale di 30 e lo spazio degli stati è pari a 10^{42}.

In questo contesto Darkboard, il programma campione del mondo di Kriegspiel, è in grado di giocare perfettamente finali vinti ed ha a disposizione quello che è probabilmente il miglior repertorio di aperture che sia stato sviluppato in questo ambito.

Da un punto di vista difensivo, specie nelle fasi iniziali del gioco, il meccanismo del profiling e la creazioni di matrici associate a pezzi con differente mobilità, forniscono un eccellente strumento di previsione riguardo alla pericolosità di ciascuna casa sulla scacchiera, il che viene ulteriormente rafforzato dall'innovativo sfruttamento di un algoritmo *monte carlo-like* che è il cuore del programma nel medio gioco, da un punto di vista computazionale.

Anche alla luce di quanto visto, si deve però rilevare che Darkboard, pur essendo il *programma* campione del mondo di Kriegspiel, non è ancora al livello dei migliori giocartori umani; un parametro oggettivo per misurare la distanza che separa giocatori artificiali da quelli umani è l'Elo[6] che questi

[6]Con Elo si intende il metodo standard per calcolare la forza relativa di un giocatore

hanno su ICC: quello ottenuto da Darkboard varia intorno ai 1700 punti, non avvicinandosi agli Elo oltre i 2000 punti dei migliori umani.

La differenza oggettiva che emerge da un analisi dell'approccio di gioco è che, sebbene nelle fasi iniziali Darkboard sia persino superiore nel prevedere lo stato della scacchiera a parità di informazioni e mediamente anche nella soluzione del finale, l'approccio al mediogioco (che è il cuore della partita)sia troppo passivo e limitato a strategie a corto raggio, il che lo porta a rispondere correttamente alle problematiche che sorgono nel breve periodo, senza però imprimere al gioco nessun andamento: essenzialmente quello che manca è una direzione strategica.

A seguito di quanto detto riguardo ai risultati ottenuti con metodo monte carlo nel Kriegspiel, questa non deve essere una sorpresa d' altronde : sperimentalmente Darkboard ha confermato che l'incertezza insita nel Kriegspiel porta metodi basati sulla forza bruta ad offrire i risultati migliori nel brevissimo periodo, la qual cosa corrisponde essenzialmente a non realizzare piani strategici bensì mosse sensate allo stato presente. In un ambito in cui la cecità del difensore sembra favorire un attacco strutturato rispetto ad una difesa attendista, per quanto efficace, l'impressione è che vi sia ancora spazio per ulteriori miglioramenti, per quanto riguarda la visione nel lungo periodo del programma.

Nel seguente capitolo viene proposta una tecnica di planning che abbiamo chiamato *su livelli di priorità*, orientata alla gestione di alcuni degli aspetti che qui abbiamo individuato come critici.

Capitolo 4

Planning su livelli di priorità

È importante giocare secondo un piano ragionevole,
piuttosto che cercare di fare il colpo migliore.
Eugenio Snosko-Borovsky

In questo capitolo è presentato il *planning su livelli di priorità*, un approccio pensato per operare in contesti caratterizzati da informazione parziale e da un branching factor e numero degli stati elevati.

Nel capitolo corrente l'algoritmo viene trattato da un punto di vista teorico, in quello successivo descriveremo i risultati ottenuti da un'implementazione di tale tecnica, integrata in Darkboard.

4.1 Modellare il dominio applicativo

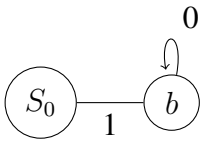
Riassumendo quanto visto nel secondo capitolo, tradizionalmente il dominio applicativo di un planner è descritto come una quadrupla *D=(*s_0*,* S_n *A,* G_n*)*

Dove :

- s_0 è detto stato iniziale.
- S: è l'insieme di stati attraversabili
- A: le azioni legali nel Dominio Applicativo, che portano il sistema a trasitare da uno stato S_i a uno stato S_{ii}.

- G_n: un insieme di Goal, con n >= 1.

Il problema di realizzare un Piano in un dominio D corrisponde a generare una sequenza di azioni ordinate che porti da s_0 a G.



Esempio: nel dominio giocattolo esemplificato dal transition system in figura, se Val(B) ∈ G, in termini pratici il piano consiste nell'individuazione della transizione 1

Nel dominio applicativo in esame, assumeremo sia disponibile una funzione di valutazione totale sulla bontà dello stato corrente in S, che verrà definita come:

$$Val : s \in S \Rightarrow n \in N \tag{4.1}$$

4.1.1 Contesti ad informazione imperfetta e riflessi sulla tipologia di piani

Richiamandoci ancora a quanto anticipato nel corso del secondo capitolo, alcuni dei problemi classici nella pianificazione sono:

- L'impossibilità di garantire l'esistenza o la calcolabilità di un percorso da s_0 a G, comporta la possibilità di un fallimento.
- Ricercare solo sequenze di azioni che portano al raggiungimento dei Goal in G, in un contesto complesso può portare a scartare strade promettenti o peggio, in contesti con avversari questo può significare non considerare fattori di rischio non preventivati nel modello su cui viene generato l'insieme G.

In contesti ad informazione imperfetta in cui si abbia un avversario, sono necessarie ulteriori considerazioni.

Il teorema di Zermelo afferma che in giochi ad informazione perfetta con avversari ed a somma zero, l'esito di un confronto si può prevedere già a partire dalla configurazione iniziale; per quanto gli effetti del suddetto teorema siano in pratica limitati dalla complessità computazionale di calcolare tale esito, esso resta uno dei capi saldi nell'ambito della ricerca con avversari, in quanto assicura l'esistenza di una *scelta corretta*; esteso al planning questo vuol dire che esiste una serie di transizioni, cioè un piano, preferibile alle altre.

Come anticipato nel secondo capitolo, il Teorema di Zermelo nel caso dei giochi con avversari ed a informazione imperfetta non vale e questo si riflette sulla concezione di *piano*; la componente dell'informazione imperfetta di per se non impedisce ad un planner probabilistico, o Markoviano, di generare soluzioni plausibili, mentre la presenza congiunta di un avversario e della mancanza di informazioni, rende approcci pratici di questo tipo insoddisfacenti in domini con uno spazio degli stati di ampie dimensioni.

Un ulteriore aspetto da prendere in esame è l'affidabilità richiesta al planner: un approccio tradizionale, basato sul generare la sequenza di mosse che più probabilmente porterà all'esito sperato, in questo campo comporta, nel caso pessimo, la possibilità che il percorso preso sia fallimentare e dunque il piano stesso lo sia: questo può essere accettabile in alcuni contesti ma in altri, in cui viene richiesto un approccio più conservativo o cauto se vogliamo, è una delle considerazioni che porta spesso a scegliere l'utilizzo di metodi basati sulla forza bruta, anzichè sulla pianificazione.

La tecnica proposta in questo capitolo parte da queste considerazioni per presentare una tipologia di piano orientata verso il concetto di miglioramento dello stato attraverso l'individuazione di fattori strutturali dello stesso.

4.2 Planning su livelli di priorità

Siano G_1, .., G_n i goals appartenenti al dominio D=(s_0, S_n, A, G_n) definito in precedenza, il dominio viene arricchito con un nuovo insieme G_l che associa a ciascun g $\in G_n$ un valore i $\in$ N: questi valori descrivono il livello di appartenenza del goal.

Durante la costruzione di un piano, l'analisi dello stato corrente determina la scelta di quale livello conterrà i goal destinazione per il piano corrente: ciascun livello contiene differenti categorie di Goal, da un punto di vista semantico: goal di livello 0 rispondo a determinate esigenze, goal di livello n rispondono a determinate altre esigenze.

Una volta scelto il livello su cui ricercare i goal per la generazione del piano nello stato corrente, vengono rimossi fino al termine del piano o al verificarsi di un evento che altera la nostra valutazione sullo stato corrente, i goal con l diverso da i.

Nel presente elaborato è stata individuata la necessità di definire tre livelli:

Livello 0 Goal di importanza cruciale o *prioritari*, generati in risposta al rilevamento del verificarsi di una condizione critica: richiedono l'attuazione di tipologie di piani specifiche dall'esito certo; in altre parole si tratta di raggiungere un vantaggio nel breve periodo, un obiettivo verificabile in termini di *funzione di valutazione*.

Livello 1 Goal testati e ritenuti *affidabili*: il raggiungimento di un goal appartenente a questo insieme è stimato come un vantaggio nel lungo termine; possono essere considerati come relazioni fra elementi verificabili su uno stato S_x, dal cui raggiungimenti ci aspettiamo effetti positivi.

La definizione di questa tipologia di goal sarà affrontata in modo più esteso in seguito, ma per fornire una prima intuizione, essi consistono in un insieme di relazioni strutturali verificabili sul singolo stato.

Se un'analisi a posteriori, cioè non eseguita a run time durante l'elaborazione di piani ma in un secondo momento, rileva che dopo n utilizzi, con n $\in$ N fissato in base a osservazioni sperimentali sul dominio applicativo dell'algoritmo, il valore di: $\frac{Val(g1)-Val(S0)}{n}$ è inferiore ad una valutazione V fissata, il piano viene retrocesso a piano di livello 2

Livello 2 sui Goals in questo insieme non si hanno indicazioni sufficienti a stimare che porteranno un vantaggio sicuro; vengono ricavati da analisi su possibili piani vantaggiosi nel dominio applicativo (argomento discusso nel sesto capitolo) e non guidano la ricerca della prossima transizione; se ne memorizza il soddisfacimento in uno storico.

Quando un'analisi a posteriori, cioè non eseguita durante l'elaborazione di piani, rileva che un goal appartenente al livello 2 è stato frequentemente associato a stati favorevoli, o in altre parole viene rilevato che : $\frac{Val(g2)-Val(s0)}{k}$ con k $\in$ N che rappresenta il numero di utilizzi (fissato in base a osservazioni sperimentali sul dominio applicativo dell'algoritmo), il piano viene promosso a piano di livello 1.

L'assunzione che i goal siano appartententi ad un numero di livelli pari a tre può essere considerata un *lower bound*; questi tre livelli sono necessari ma niente vieta di specializzare ulteriormente l'algoritmo, ad esempio scomponendo i goal di livello 2 in differenti tipologie.

Un altro aspetto importante è l'individuazione di un meccanismo per specificare connessioni o priorità fra goal di livello 1: il concetto di gerarchia, candidato naturale a questo compito, appare non abbastanza flessibile da gestire efficacemente i cambi di scenario, cui sarà prevedibilmente soggetto un agente in un gioco con avversari ad informazione parziale. La soluzione scelta è quella di specificare un punteggio per ciascun goal appartenente a questa fascia, così da rappresentarne la priorità quando si possa scegliere fra diversi obiettivi possibili: in questo modo viene superato il concetto di *memoria* che limita la flessibilità dell'agente ed è possibile definire un meccanismo di scelta come nell'esempio in figura:

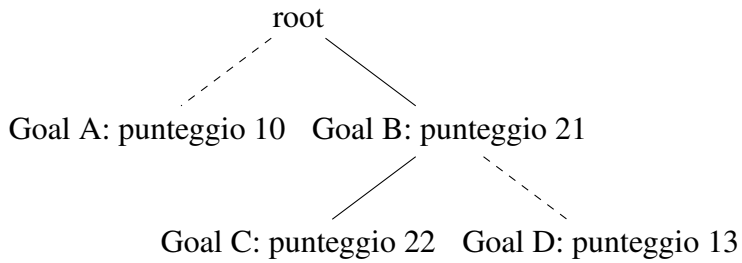


Figura 4.1: *Si supponga che fra il goal B ed il goal V vi sia una sequenzialità, in altre parole sia noto che l'ottenimento di A renderà raggiungibile in una transizione B: scegliendo il valore più alto fra i figli, in caso a run time siano raggiungibili altri obiettivi più importanti l'agente avrà la flessibilità di adeguarsi cambiando percorso, ma in caso contrario, la scelta del valore più alto porterà sarà sicuramente a selezionare il goal con punteggio 21 fra quelli disponibili.*

Riguardo alla nozione di *piano prioritario*, è opportuno aggiungere che essa è stata oggetto di varie evoluzioni durante la stesura di questo elaborato: se suppore che vi siano condizioni di importanza critica, il cui rilevamento è un passo necessario sia nella definizione dell'insieme dei Goal che per il raggiungimento dell'obiettivo dell'agente, appare una assunzione accettabile (significa in altre parole che l'agente non brancola totalmente nel buio, ma riconosce al verificarsi un certo insieme di caratteristiche particolari), successivamente a questa categoria sono state aggiunte tutte quelle azioni che hanno esiti certi, la qual cosa riconduce la generazione di un piano ad un contesto ad informazione perfetta: l'idea di fondo è sfruttare il verificarsi di queste situazioni come un modo per aggiungere informazioni che *guidino* l'agente, in un contesto in cui proprio le informazioni sono cruciali.

4.2.1 Algoritmo

L'algoritmo del planning su livello si divide concettualmente in quattro fasi, che vengono iterate per un numero n $\in$ N di passi:

VALUTAZIONE Viene valutato lo stato corrente S_0, in conseguenza di tale valutazione, che include il rilevare condizioni critiche, viene identificato il livello l corrente (in modo che i goals bersaglio siano scelti fra quelli con G_l = l).

INDIVIDUAZIONE Si individuano i Goals in G tali che G_l <= di liv se liv > 0, se liv = 0 si individuano i Goals in G tali che G_l=0 (piani prioritari).

COSTRUZIONE Viene generato l'insieme di Stati s $\in$ S, Path(s_1, .., s_n) che sono raggiungibili in un passo da S_0 e su di essi viene individuato l'insieme O definito in precedenza.

Chiaramente essere in grado di effettuare una potatura degli stati interessanti da $S_0 \Rightarrow S_1$ identificando le azioni più promettenti e generando solo i percorsi che comprendano tali azioni come passi iniziali, aumenta drasticamente l'efficacia e diminuisce il costo dell'algoritmo; si può quindi opzionalmente prevedere il seguente passo, precedente a quello di valutazione:

OTTIMIZZAZIONE Ove sia disponibile una funzione di valutazione:

ValAct(a) : $S \Rightarrow S$

che fornisce indicazioni in termine di valutazione dello stato raggiunto, riguardo alla bontà della transizione a, questa può essere utilizzata per rimuovere dall'insieme Act nello stato presente azioni con una valutazione inferiore ad una soglia fissata.

I goal ritenuti *inaffidabili* non sono considerati in questo procedimento, ma riscontrare che vengono realizzati in uno stato viene comunque memorizzata in un vettore per delle analisi a posteriori che permettano di evolverli in piani *affidabili* al verificarsi delle condizioni descritte in precedenza.

Lo pseudocodice dell'algoritmo è il seguente:

```
Function priorityLevelPlanning(state,goals,deepth){

  Int level = levelEvaluation(state);
  Vector goalSet = goalIndividuation(goals,level);

  Function construct(state,deepth){

    state.rate += evaluate(state,goals);
    state = evolve();
    Vector transitions = legalTransitions(state);

    prune(transitions);     /* optimization step

    if(deepth > 0){
      deepth--;
      for(k=0; k< transitions.Size(); k++){
       transitions(k)= construct(state,deepth);
      }

    }
  }
  return maxRate(state);
}
```

La visita dell'albero delle transizioni possibili avviene secondo il paradigma iterative deepening search, con l'unica differenza che prima di visitare gli stati vengono generate tutte le transizioni per ogni livello visitato come evidenziato in figura.

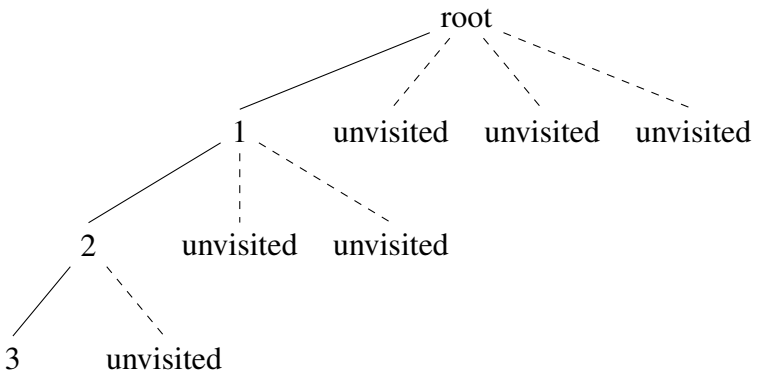


Figura 4.2: *Visita dell'albero delle transizioni da parte dell'algoritmo*

Una descrizione visivamente più intuitiva dell'intero procedimento si ha nell'activity diagram di seguito.

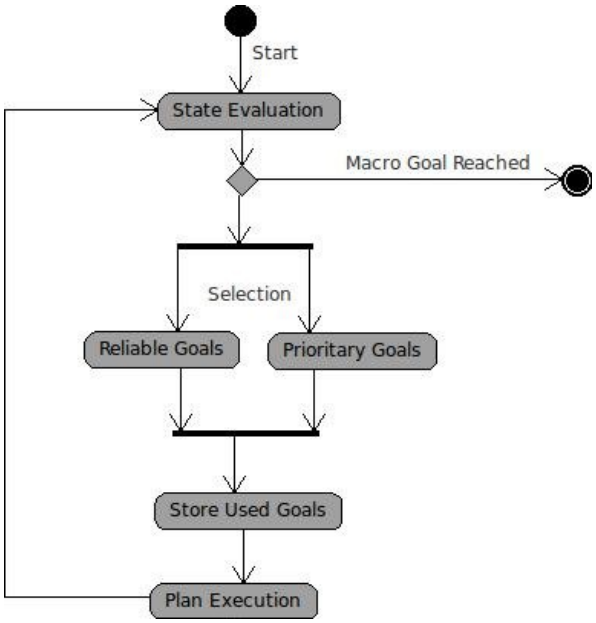


Figura 4.3: *Activity Diagram dell'algoritmo di planning su livelli di priorità*

Sulla natura di un piano

Dal punto di vista pratico, l'algoritmo consiste nell'analizzare i livelli contigui dell'albero delle transizioni alla ricerca di goal appartenenti al Goal Set: il

piano che ne risulta è in realtà un insieme di *micro-piani* orientati ad obiettivi multipli, che sono i principi strategici di cui sopra.

Il planner punta a migliorare lo stato corrente, iterando questo procedimento fino a quando non riesce a trovare un aggancio che lo porti a passare da una fase più ricca di incertezza gestita attraverso *piani affidabili*, ad una fase finale in cui il branching diminuisce, gestita attraverso piani *prioritari*, che saranno tipicamente piani nel senso classico del termine, cioè insiemi di transizioni ordinate con un unico goal definito.

4.2.2 Progresso

Considerando che il fine di questa tipologia di planning consiste nel miglioramento costante della posizione, il conseguimento del progresso non può che dipendere dagli effetti delle singole tipologie di azioni selezionabili dal planner: queste possono appartenere a piani prioritari o a piani affidabili.

Piano Prioritario

Un piano prioritario viene definito per essere utilizzato in situazioni specifiche: la stessa creazione di un piano prioritario implica definire una serie di transizioni *automatiche* in risposta ad una situazione specifica, con la certezza di trarre un vantaggio (o limitare una perdita, in contesti con avversari) da tale sequenza.

L' esecuzione di un *piano prioritario* è dunque, astraendo, assimilabile ad un modo certo di migliorare la posizione.

Piani Affidabili

In questo caso, la ricerca del progresso è sottile quanto costante.
Configurazioni ordinarie, cioè non incluse fra quelle previste dai goal prioritari, vengono gestite ricercando un miglioramento strutturale delle stesse, in un modo che ricorda il concetto di *formazione*: inizialmente l'obiettivo è la costruzione di una configurazione strutturalmente indicata come favorevole dal planner dai *goal affidabili*, successivamente esplorare gli stati considerati promettenti reiterando il procedimento, cioè trasponendo da una formazione ad

un'altra attraverso transizioni viste ancora una volta come il raggiungimento di goal strutturali: questo assicura che l'esplorazione degli stati avvenga incrementando il valore strutturale della configurazione.

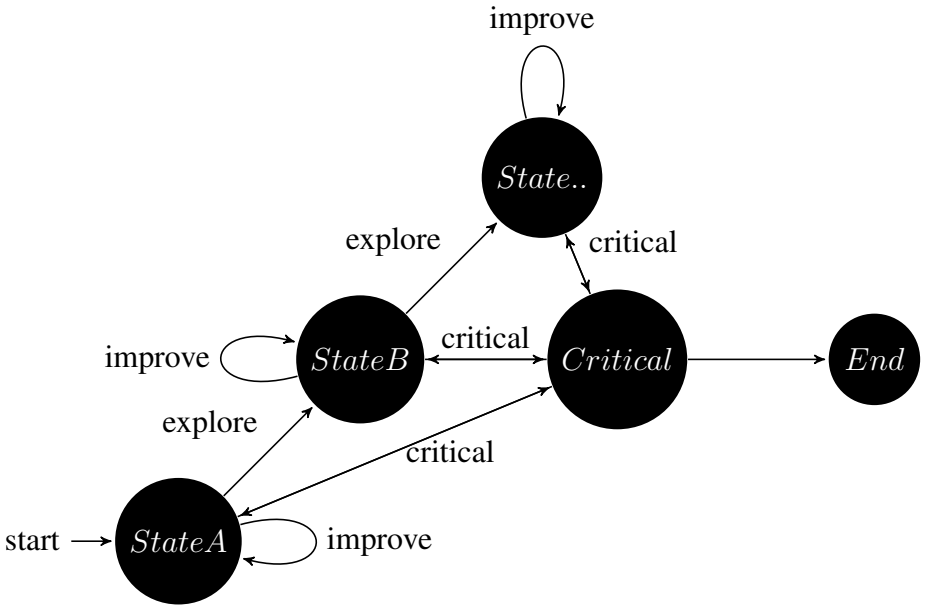


Figura 4.4: *Grafo che rappresenta il progresso di un planner su livelli di priorità*

Perchè il meccanismo descritto abbia effetto, un aspetto chiaramente fondamentale è la definizione del goal set, in modo che obiettivi concettualmente collegati si concatenino spontaneamente; in altre parole se il raggiungimento di una configurazione A porta alla possibilità di raggiungere una configurazione B che ne è il naturale proseguimento, il sistema di punteggi da attribuire ai goal (descritto in questo capitolo) permette di definire le priorità fra le relazioni sfruttando il loro punteggio, evitando il concetto di gerarchia proprio del planning HTN, che non si è dimostrato adeguato in questo contesto.

Riassumendo, il conseguimento del progresso nell'applicazione di piani affidabili, è garantito dall'esistenza delle stesse relazioni che sono alla base della definizione di questa tipologia di goal.
Quanto appena affermato è determinante per comprendere in quali ambiti la tecnica proposta sia promettente: contesti in cui sia presente una significativa nozione di *struttura* nell'insieme degli stati (struttura che è anche definibile

in termini di *patterns*) come Kriegspiel, Phantom Go , o eventuali giochi di strategia *a la Risiko*, presentano per loro natura caratteristiche identificabili per il raggiungimento del progresso. Al contrario contesti in cui tali elementi manchino o scarseggino, porterebbero un planner su livelli di priorità ad un approccio passivo in quanto privo di un goal set significativo.

4.2.3 Classi di problemi interessanti

Riprendendo il concetto del paragrafo precedente, una chiave per sfruttare efficacemente questa tecnica è comprendere che essa performa al meglio in ambiti in cui vi siano principi *strategici* individuabili e scomponibili in relazioni strutturate in modo *piatto* o in altre parole, che tali relazioni siano definite in modo che verificarne l'esistenza su uno stato richieda di esplorare esclusivamente lo stato in questione.

Confronto per contesti

A titolo di esempio, presentiamo alcune ipotesi su come si comporterebbe un planner su livelli in alcuni domini con caratteristiche differenti:

Gioco	Informazione	State space	Casualità	Prospettive
Scacchi	perfetta	molto ampio	assente	*scarse*
Risiko	imperfetta	ampio	presente	*interessanti*
Kriegspiel	imperfetta	molto ampio	assente	*buone*
Bridge	imperfetta	ampio	presente	*scarse*

Figura 4.5: *Esempi di giochi più o meno interessati dal planning su livelli*

Un planner su livelli abbinato ad esempio ad un'analisi monte carlo o Minimax secondo il paradigma spiegato nei prossimi paragrafi, potrebbe essere una combinazione molto interessante in contesti quali eventuali giochi strategici *a la risiko*, i quali seppure offrano talvolta un minore grado di incertezza, presentano una struttura ideale per valorizzare un sistema di costruzione del gioco *su due fronti*: quello prettamente tattico adeguato ad un'analisi brute force e quello strategico a lungo termine, più affine al concetto dei *piani affidabili* descritti in precedenza.

Al contrario, a nostro avviso un approccio nel campo di giochi in cui il caso eserciti una componente molto forte quali il bridge per esempio, che pure tanto bene ha risposto a planner gerarchici, sarebbe poco promettente in quanto viene a mancare il concetto di strategia nel lungo termine, catturato da questa tipologia di planning.

Le valutazioni espresse sono puramente indicative in quanto non oggetto di verifiche sperimentali, se non nel caso del Kriegspiel: l'obiettivo non è quello di consigliare un ambito specifico, piuttosto quello di inquadrare le caratteristiche salienti nell'approccio proposto.

4.2.4 Complessità Computazionale

Scomponiamo l'analisi del costo complessivo dell'algoritmo nella somma delle tre fasi in cui si divide:

VALUTAZIONE Costo della funzione Val da applicare ad S_0; dipende dal contesto, solitamente sarà trascurabile.

Notare che in questo passo si tratta di valutare il verificarsi di condizioni critiche o meno che determinerebbero la scelta di goal di livello prioritario, non di valutare lo stato.

INDIVIDUAZIONE Una costante, determinata dal numero di goal appartenenti al livello scelto in fase di valutazione.

COSTRUZIONE (Act/Ottimizzazione)*(Act/Ottimizzazione)*Piani

Il tutto moltiplicato per la Profondità a cui interrompiamo la visita. L'aspetto dominante è chiaramente la fase di Costruzione ed il costo totale è:

Deepth ((Act/Ottimizzazione)*(Act/Ottimizzazione)* Piani)*

Come si nota un ruolo centrale è affidato alla funzione di ottimizzazione, che impedisce all'algoritmo di crescere in modo esponenziale sul numero di azioni (che tende a valori alti, abitualmente) per ogni livello di profondità nell'albero delle transizioni: su di essa ci soffermeremo nel prossimo paragrafo e nel prossimo capitolo.

È altresì opportuno evidenziare che per come abbiamo presentato il concetto di *miglioramento dello stato corrente*, un planner su livelli di priorità necessita di compiere visite particolarmente brevi dell'albero delle transizioni (al contrario di quanto avviene per algoritmi basati sulla forza bruta) dunque il fattore di rilievo per giudicare la complessità dell'algoritmo sarà dato dal numero di Goal appartententi a Piani, che tenderà ad essere alto, per il numero di azioni.

Si noti in conclusione che essendo basato su una visita Iterative Deepening Search l'algoritmo è completo, cioè riconosce ogni obiettivo all'interno dell'orizzonte.

4.2.5 Planning su livelli di priorità e forza bruta

L'applicazione pratica di questa tipologia di planning, descritta nel capitolo successivo, ha messo in luce un ulteriore fattore di interesse: l'approccio descritto si presta in modo quasi naturale ad un'integrazione con algoritmi di forza bruta.

Consideriamo un algoritmo di forza bruta come la funzione di ottimizzazione descritta in precedenza, quale una via per individuare un insieme di transizioni interessanti, evitando di analizzare tutti i percorsi possibili: i due approcci traggono l'uno beneficio dall'altro in quanto l'uso della forza bruta è limitato ad individuare percorsi in fase iniziale, evitando l'esplosione combinatoria che si ha costruendo alberi di gioco che vanno oltre una certa profondità ed al contempo, l'utilizzo del planning su livelli beneficia di una notevole potatura sulle transizioni candidate il che lo porta ad aumentare drasticamente l'efficienza aggiungendo una nozione di safety, cioè un controllo sul fatto che le transizioni generate non porteranno ad un peggioramento drastico secondo una funzione di valutazione(che tipicamente orienta gli algoritmi di forza bruta).

Questo approccio ibrido non restringe il campo di scelta iniziale anzi, permette un alto grado di astrazione al momento della definizione del set di goals che guidano il planner nel miglioramento dello stato corrente: transizioni con un risultato oggettivamente negativo non vengono neanche prese in esame, di conseguenza qualsiasi risultato selezionato dal planner considera comunque il concetto di cautela, menzionato nell'introduzione di questo capitolo.

Capitolo 5

Planning su livelli di priorità in Darkboard

Il capitolo corrente sarà dedicato alla descrizione dei risultati sperimentali, ottenuti integrando l'implementazione di un algoritmo di planning su livelli di priorità nel giocatore artificiale di Kriegspiel, Darkboard.

5.1 Planning su livelli nel Kriegspiel

Come segnalato nel capitolo precedente, la tecnica proposta risulta adeguata per classi di problemi che presentino determinate caratteristiche.

La scelta del Kriegspiel come dominio di testing per questo tipo di pianificazione è stata dettata dal fatto che un giocatore di Kriegspiel presenta tutte le caratteristiche "ideali" definite in fase di discussione teorica, necessarie per essere valorizzato da questa tecnica:

COMPLESSITÀ COMPUTAZIONALE Il branching dell'albero di gioco è molto alto, come visto nel capitolo 3 può raggiungere valori intorno a 30!-35! [1], il che rende algoritmi basati sulla pura forza bruta inefficienti nella gestione del mediogioco.

INCERTEZZA L'alto grado di incertezza, causato dal progressivo deteriorarsi delle informazioni sulla configurazione corrente della scacchiera,

[1] Per ulteriori approfondimenti su questo aspetto,si rimanda a Ciancarini [2004]

rende inefficienti algoritmi di planning più ”tradizionali”, se non in un insieme molto ristretto di condizioni (ad esempio, una serie di transizioni che porti allo scacco matto non è pianificabile se non quando questo è molto vicino in termini di orizzonte) e porta naturalmente verso una strategia di graduale miglioramento dello stato corrente, in fase di mediogioco.

GOAL Rispetto alle considerazioni sull'importanza dell'esistenza di relazioni, che abbiamo definito come *principi strategici* nel corso capitolo precedente, giochi appartenenti alla famiglia degli scacchi sono un eccellente dominio di testing: l'aspetto strategico-strutturale è stato largamente studiato nella letteratura scacchistica ed è dunque disponibile una grande mole di informazioni, utili ai fini della definizione di goals di testing per il programma.

5.2 Il Mediogioco in Darkboard 2.0

Richiamando brevemente quanto visto nel terzo capitolo, una partita per Darkboard consiste nella gestione delle tre fasi seguenti:

APERTURA Viene gestita attraverso un libro di aperture, consistente essenzialmente in varie sequenze di mosse precalcolate modellate sulle strategie dei migliori giocatori umani attivi sul server di web-gaming ICC; alla ricezione del primo messaggio da parte dell'arbitro questa fase viene considerata conclusa.

MEDIOGIOCO La parte più sostanziosa di una partita in termini di quantità di mosse generate; il nucleo nella generazione delle mosse è basato sul metodo monte carlo coadiuvato da una strategia di profiling.

Il programma considera concluso il mediogioco quando riesce ad individuare con sufficiente precisione il re avversario, il che comporta spesso l'eliminazione della maggior parte dei pezzi dalla scacchiera.

FINALE La scarsità di materiale in questa fase rende possibile trattare il finale come un caso di gioco ad informazione perfetta, appoggiandosi al concetto di metaposizione.

La fase su cui ci siamo concentrati è naturalmente il mediogioco; l'attore principale di questo segmento è come detto il metodo monte carlo, riportiamo dunque lo pseudocodice che descrive il processo di generazione delle mosse:

```
function approach_C(Node root){
  while(availableTime){
    node n=root;
    Move move;
    while(!isLeaf(n)){
      if(programTurn(n)){
n=uctSelection(n);
Message msg = probabilisticMessage(n);
n = getChild(n,Msg);
      }
      else{
  Message msg = probabilisticMessage(n);
  n = getChild(n,msg);
      }
    }
    if(n.explored) n = expand(n);
    double outcomeProbabilities(n,outValues,probab);
    getOutcomeProbabilities(n,outValues,probab);
    for( int a=0; a< outValues.length; a++}
value += outcomes[a]* probab[a];
    n.explored = true;
    backpropagate(outcome,n);
  }
  return mostVisitedChild(root);
}
```

Figura 5.1: *Generazione di mosse in Darkboard 2.0, tratta da Ciancarini e Favini [2007b]*

Semplificando, il programma crea N alberi di gioco di profondità 1, ipotizzando i possibili esiti in termini di messaggi di risposta da parte dell'arbitro,

il che gli fornisce un'indicazione statistica sulla bontà di una mossa nel breve periodo.

Per una trattazione più approfondita del procedimento, si rimanda a G.P.Favini [2010].

5.3 Definizione dei Livelli di Priorità in Darkboard

I livelli proposti nel corso del capitolo precedente sono:

PRIORITARIO Goal associati a vantaggi nel breve termine, considerabili come piani "tradizionali", con un unico obiettivo individuato come centrale e una serie ordinata di transizioni volte a raggiungerlo.

AFFIDABILE Goal che descrivono insiemi di *relazioni* verificabili su uno stato: sono associati a fattori strutturali della posizione e si evolvono nel tempo.

INAFFIDABILE Goals da testare (ad esempio ottenuti attraverso l'utilizzo di tecniche di machine learning etc.) sulla cui affidabilità è richiesta una verifica sperimentale prima di essere considerati affidabili.

La suddivisione delle differenti situazioni riscontrabili da Darkboard nel corso del Mediogioco ha portato a identificarli con le seguenti tipologie di piano:

PRIORITARIO Reazioni "semplici" legate ad un insieme ristretto di situazioni: spostamento del re in caso di scacco, ricattura se attaccati, etc: come ipotizzato nell'algoritmo sono sequenze di azioni ben definite, volte a raggiungere un unico obiettivo che al momento è prioritario.

AFFIDABILE Allo stato attuale ogni *goal affidabile* descrive una relazione fra i pezzi in gioco, tali relazioni sono state definite sulla base della strategia scacchistica; in questa fase era necessario che i goal fossero sicuramente portatori di effetti positivi non essendo l'elaborato incentrato sul machine learning.

La verifica di un goal su uno stato ha un esito booleano (il goal è verificato, il goal non è verificato) ed il raggiungimento di quest'ultimo è visto come un fattore positivo in termini di prospettive della posizione.

INAFFIDABILE La categoria di goal è stata predisposta, in modo da poter in seguito integrare algoritmi di pattern recognition o di data mining; allo stato attuale non viene posto l'accento su questa funzione, in quanto l'obiettivo di questa tesi è sperimentare il funzionamento della tecnica come metodo di pianificazione, senza concentrarsi troppo sugli aspetti connessi al machine learning; lo sfruttamento di questa funzionalità è previsto come uno dei principali sviluppi futuri.

5.3.1 Definizione del goal set dei piani affidabili

La gestione di piani *affidabili* ha richiesto la codifica di una struttura, adatta a memorizzare il concetto di relazioni descritto, così composta:

```
public class Goal {

public int level;   //Affidabile,inaffidabile
public int rate;
public int source;
public int dest;

public boolean isPieceConnection = false;
public boolean isPieceActivity   = false;
}
```

Figura 5.2: *Struttura dei Goals per piani di livello 1 e 2*

I campi *source* e *dest* descrivono dei pezzi (che in Darkboard sono codificati come valori interi), mentre chiaramente il campo level descrive il livello di priorità a cui appartiene il goal.
Il campo *rate* descrive il punteggio dell'obiettivo, in accordo con il sistema trattato nel capitolo precedente.

Ciascun goal codifica una relazione binaria fra due pezzi, individuando quelli che sono i possibili obiettivi: il planner identifica il piano che è più vantaggioso scegliendo quello che ha il valore strategico più elevato, dove

tale valore è ottenuto sommando i punteggi dei goal realizzati in ciascuno degli stati attraversati nella costruzione del piano.

```
Goal RookToRook = new Goal();
  RookToRook.source = Chessboard.ROOK;
  RookToRook.dest = Chessboard.ROOK;
  RookToRook.rate = 4;
  RookToRook.isPieceConnection = true;
goals.add(RookToRook);
```

Figura 5.3: *Esempio di goal in Darkboard, in cui connettere due torri viene individuato come un obiettivo*

L'assegnazione dei punteggi a ciascun goal è un aspetto cruciale nella definizione di quello che è un piano in Darkboard.
In questo elaborato abbiamo parlato di goal come relazioni, poichè questi non rappresentano l'obiettivo finale di un piano bensì le relazioni strutturali che portano all'attuazione dello stesso: è dunque necessario scomporre un piano nella verifica di un insieme di passi il cui ordine è specificato assegnando punteggi progressivi alle relazioni coinvolte.
Così facendo, il planner analizza due stati per volta e costruisce un piano di gioco a partire dalle relazioni, scegliendo sì lo stato che porta ad una configurazione più vantaggiosa nel breve termine, ma al contempo gettando progressivamente le basi per la costruzione di un piano più ampio, grazie al fatto che non c'è mai una rappresentazione esplicita dello stesso.

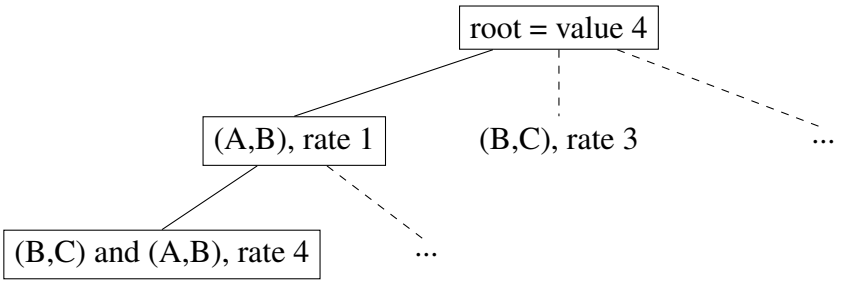


Figura 5.4: Esempio di costruzione del piano *collega B e C, proteggendo B con A lungo il cammino*

Al momento, esistono due tipologie di goal affidabili in Darkboard: PieceConnection goals, che descrivono il processo di costruzione d'una formazione e PieceActivity goals, che descrivono il processo di esplorarazione de l territorio nemico, come nel ciclo *miglioramento-esplorazione* descritto nel capitolo precedente.

5.4 Lo sviluppo di un piano in Darkboard

Non ci soffermeremo sullo sviluppo dei piani prioritari, trattandosi essenzialmente di piani tradizionali che rispondo ad esigenze molto circoscritte: protezione del re quando è sotto scacco, tentativo di promuovere un pedone in prossimità dell'ultima traversa,etc.

Nello sviluppo di *piani affidabili*, chiameremo così i piani che mirano al raggiungimento di goal definiti come *affidabili*, ciascuna transizione può essere considerata come un piano multiobiettivo, inclusa in unico piano di cardinalità maggiore volto al raggiungimento di un macro obiettivo, non necessariamente singolo, che in questo caso corrisponde alla costruzione di presupposti strutturali che permettando di trasporre il mediogioco in un finale vincente.

A livello intuitivo, dallo stato corrente si compie un'analisi dell'albero di gioco fino ad una profondità $d \in N >= 1$ che nell'implementazione più efficace si ferma a profondità due, cercando di scegliere la mossa che verifica il maggior numero possibile di Goals nei percorsi che partono da essa.

Un aspetto da prendere in considerazione è che non essendo basato su una funzione di valutazione o su un'analisi esaustiva dell'albero di gioco, un giocatore artificiale di Kriegspiel che affidi il suo processo decisionale interamente all'algoritmo appena descritto, sarebbe costretto a specificare un set di goal talmente ampio da rendere l'algoritmo stesso una sorta di *altamente inefficiente* analisi con la forza bruta; se invece questo opera sinergicamente ad un'analisi delle mosse preliminari con l'algoritmo monte carlo, il quale si è dimostrato un metodo di analisi con al forza bruta più efficace di minimax in questo contesto, può demandargli il compito di scartare scelte deleterie in ter-

mini di funzione di valutazione, scegliendo fra le mosse giudicate accettabili: questo permette una pianificazione altamente flessibile.

Vediamo adesso come si traduce lo scheletro teorico dell'algoritmo esposto nel capitolo precedente nell'implementazione realizzata per Darkboard. I seguenti passi vengono iterati un numero di volte pari alla profondità massima definita per l'algoritmo che, come detto, in seguito ad esperimenti su quale valore desse le migliori performances, è stata fissata a due:

OTTIMIZZAZIONE Come anticipato questo ruolo è svolto dal metodo monte carlo: viene calcolato il valore dei nodi come da pseudocodice in fig.5.1 e sono scelte le $n \in N >= 1$ transizioni con valori più alti, a condizione che queste non si discostino per più di una soglia fissata dal valore migliore; questa è una ottimizzazione per evitare di prendere in considerazione mosse che perderebbero materiale

VALUTAZIONE E INDIVIDUAZIONE L'aspetto centrale di questo passo è quello di individuare situazioni che richiedano piani critici: è stato sufficiente posizionare la chiamata al planner al termine dell'esecuzione della scelta delle mosse da parte del modulo che implementava la chiamata a monte carlo, identificando le situazioni che richiedono piani prioritari in base ai messaggi dell'arbitro, per svolgere questo compito senza ulteriori calcoli: questa ottimizzazione è stata possibile grazie al fatto che Darkboard prevedeva mosse definite come *automatiche*, le quali sono state assimilate al concetto di piano prioritario.

COSTRUZIONE Il passo più complesso dal punto di vista del planning: vengono prese in input le *transizioni candidate* filtrate attraverso il passo di Ottimizzazione, viene generato il set delle mosse legali a partire da ciascuna di esse incrementandone il valore strategico per ciascun goal verificato, reiterando il procedimento per ciascuna di esse.

È opportuno ricordare che un goal è una relazione, dunque la sua verifica è un risultato booleano; in caso sia true, il punteggio strategico della mossa viene incrementato del punteggio memorizzato nel campo *rate* del goal.

Al termine del ciclo viene scelta la transizione candidata che ha il valore strategico più alto (non più quindi quella migliore secondo monte carlo); vediamo come cambia lo pseudocodice per la generazione delle mosse:

```
function generateMove()
 if(refree_message == critical condition){
    return simple_reaction(refree_message);
    }

 else
 {
  function approach_C(Node root){
  while(availableTime){
    node n=root;
    Move move;
    while(!isLeaf(n)){
      if(programTurn(n)){
n=uctSelection(n);
Message msg = probabilisticMessage(n);
n = getChild(n,Msg);
      }
      else{
  Message msg = probabilisticMessage(n);
  n = getChild(n,msg);
      }
    }
    if(n.explored) n = expand(n);
    double outcomeProbabilities(n,outValues,probab);
    getOutcomeProbabilities(n,outValues,probabs);
    for( int a=0; a< outValues.length; a++}
value += outcomes[a]* probabilities[a];
    n.explored = true;
    backpropagate(outcome,n);
  }
  }

  Vector Candidate;
  for(k=0; k< root.Child.Size(); k++)
  {
    if( root.child[k].score > root.bestRate-treesold)
    {
Candidate.add(root.child[k]);
```

```
    }
  }

  for(k=0; k< CandidateMoves.Size(); k++)
  {
    int Strategic Rate = 0;
    Vector Moves = genMoves(State,candidate(k),Deepth);
    Candidate(k).rate= EvaluateReachedGoals(State,Moves);
  }
  return maxRate(Candidate);

}
```

Figura 5.5: *Pseudocodice del mediogioco in Darkboard*

In figura è presentato l'activity diagram della generazione di una mossa nel mediogioco, in cui sono evidenti le similitudini rispetto all'omologo proposto nel capitolo precedente:

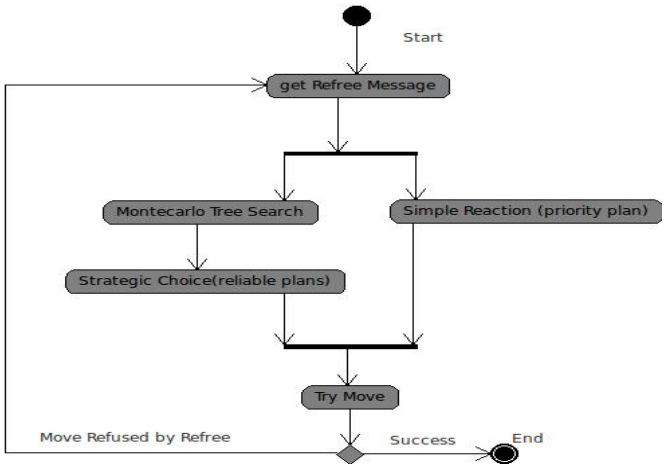


Figura 5.6: *Activity Diagram del Mediogioco in Darkboard*

5.4.1 Progresso

La ricerca del progresso nel mediogioco di Darkboard, vera mancanza individuata in fase di analisi dell'agente, avviene esattamente come descritto nel

capitolo precedente: in ogni situazione in cui non siano disponibili Simple-Reactions, Darkboard riorganizza i pezzi sulla scacchiera in formazioni che rispecchiano le strategie che hanno guidato la definizione dei goal.

Regolando i punteggi di tali goal in modo opportuno, è stato possibile definire una successione fra di essi, che porta il raggiungimento di una configurazione stabile dei pezzi ad evolvere naturalmente in un'esplorazione organizzata, quindi nuovamente ad un consolidamento della posizione, in un ciclo che ha come obiettivo la conquista del territorio avversario attraverso un'avanzata *in formazione*, riprendendo la definizione del capitolo precedente, invece che attraverso l'uso di singole mosse vantaggiose nell'immediato, come avviene quando Darkboard è guidato esclusivamente dall'analisi monte carlo.

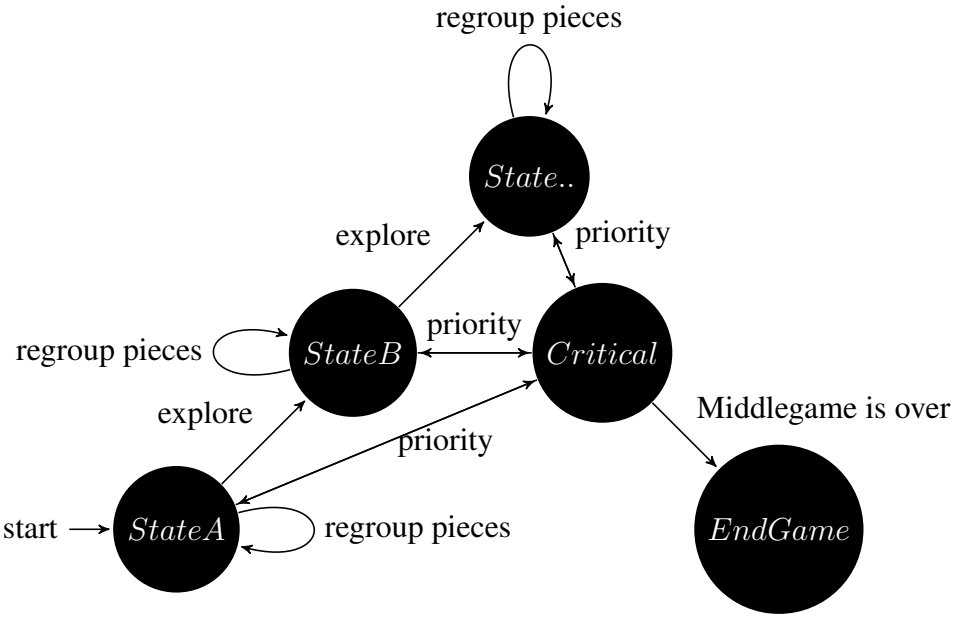


Figura 5.7: *Grafo che rappresenta la ricerca del progresso nel mediogioco in Darkboard: notare che i passi ricalcano quelli dell'omologo presentato nel capitolo precedente*

Concretamente il processo non sfrutta ancora a pieno le potenzialità che derivano da questa tecnica di pianificazione, dal momento che non è stato ritenuto opportuno definire un goal set altamente specializzato sulle caratteristi-

che del dominio, al fine di mantenere un certo grado di generalità nei risultati ottenuti testando l'algoritmo: questo non impedisce comunque di riscontrare la presenza di una logica di insieme nello sviluppo che và al di là delle singole mosse (riassunta dal diagramma in figura), aspetto questo che era quasi assente in precedenza *in questa fase di gioco*, come evidenziato nelle conclusioni del secondo capitolo.

5.5 Costo Computazionale

Il costo computazionale dell'algoritmo limitato a profondità 2, è trascurabile rispetto al costo dell'esecuzione del metodo monte carlo.

Il fattore trainante è chiaramente O(Costruzione * Deepth) che come anticipato in fase di presentazione teorica dell'algoritmo, è nell'ordine di:

Deepth(MosseLegali/Ottimizzazione*MosseLegali*Piani)*

In astratto,l'algoritmo cresce esponenzialmente sul numero di mosse legali, ma considerando che Deepth è pari a due e che invece il numero di Piani sarà auspicabilmente molto più alto, il costo dell'algoritmo è in realtà dominato da:

```
MosseLegali * Piani
```

Una ulteriore considerazione riguarda il fatto che un qualsiasi contesto in cui divenga sensato aumentare la profondità della visita è implicitamente minormente incerto, quindi non appartiene alla classe di problemi a cui si interessa questo algoritmo.

Di conseguenza non solo l'algoritmo non cresce in modo esponenziale in questo caso, ma non lo farà in nessun caso in cui l'algoritmo stesso sia di interesse concreto.

5.6 Risultati sperimentali ed altre considerazioni

Il testing della versione 2.1 di Darkboard, che abbiamo chiamato Strategic Darkboard è stato condotto contro i suoi predecessori:

Monte carlo DB La versione più forte fino ad oggi, la prima in cui è stato introdotto l'uso dell'algoritmo monte carlo e del Profiling

Darkboard 1.0 Il giocatore basato sulle metaposizioni, vincitore della medaglia d'oro alle olimpiadi del 2006

Random Player Una versione rinforzata di giocatore casuale: usa i libri di aperture di Darkboard, ricattura se possibile ed evita mosse "suicide" con il re

I test consistono in tornei da venti partite fra le differenti versioni, con l'assegnazione di un punto per ogni vittoria e di mezzo punto ad entrambi i giocatori per il pareggio.

I risultati ottenuti sono i seguenti:

Players	Strategic DB	Opponent
Strategic DB Vs Monte carlo DB	12,5	7,5
Strategic DB Vs Darkboard 1.0	15,5	4,5
Strategic DB Vs Random Player	17	3

Figura 5.8: *esiti dei tornei fra le differenti versioni di Darkboard*

Come si attendeva, Montecarlo DB è risultato l'avversario più significativo, mentre i punti persi contro l'avversario casuale sono nell'ordine dei valori precedentemente riscontrati nei testing sulle precedenti versioni di Darkboard: si tratta di pareggi per stallo o per la regola delle 50 mosse senza che l'avversario abbia mai vinto una sola partita.

Conseguentemente ai risultati ottenuti, i test fra Montecarlo DB e Strategic DB sono stati ripetuti, variando le cadenza dei tornei:

Durata del torneo	Strategic DB	Montecarlo DB
20 partite	61%	39%
10 partite	50%	50%
5 partite	80%	20%

I test vedono Strategic Darkboard superare costantemente in modo significativo la versione 2.0 su tornei di cadenza lunga, con un risultato del 61% dei

punti totalizzato, mentre su esperimenti ripetuti di lunghezza inferiore i risultati nel complesso si attestano su una percentuale pari al 66,6%, ma oscillano in modo troppo significativo per essere di qualche interesse: essenzialmente in un dominio complesso come il Kriegspiel, il caso gioca un ruolo troppo forte per ottenere dati attendibili su un numero di partite inferiore a 20, mentre invece su un numero di partite superiore a tale soglia i risultati si stabilizzano attorno al 61% di successo.

Una terza serie di test è stata infine eseguita fra Strategic Darkboard e Darkboard Montecarlo, privando i due giocatori di informazioni specifiche riguardo all'avversario: come anticipato nel terzo capitolo, Darkboard può usufruire di due tipologie di profilo, una specifica per un particolare avversario conosciuto, un'altra generica per un avversario ignoto.
Considerando che Darkboard conosce se stesso meglio di chiunque altro, il profilo dell' *avversario Darkboard* ha una mole di informazioni pari a 14 volte quello del secondo avversario più affrontato e 100 volte più estesa di quella dell'avversario medio: abbiamo dunque ritenuto interessante testare le due versioni cancellando le informazioni che ciascuna aveva dell'altra, seppur consapevoli del fatto che questo penalizzerà inevitabilmente il giocatore più difensivo, cioè Darkboard Montecarlo.
In effetti i risultati sono stati in linea con le nostre previsioni, con una percentuale di punti totalizzata da Strategic Darkboard pari al 70%.

Una riflessione significativa sulle performances di Strategic Darkboard riguarda il fatto che l'insieme dei goal che hanno orientato il planner è stato volutamente definito in modo minimale; alla base di questa decisione vi è l'intenzione di evitare di ottenere risultati sperimentali che fossero troppo strettamente legati all'alto grado di teorizzazione del gioco degli scacchi.

Limitare i dettagli inerenti al dominio nei goal ha però sicuramente penalizzato Strategic Darkboard, che dipende da essi per orientare il proprio gioco, ed è un dato di fatto che raffinando e portando il goal set da 8 a 12 unità, il programma sia passato da una percentuale di successo di 54% a 63,75%.

Una discussione più approfondita sul alcune strategie possibili per l'espansione dell'insieme dei goal è stata comunque oggetto di studio, e sarà affrontata nel corso del capitolo successivo.

Capitolo 6

Goals in giochi strategici

Il capitolo corrente si distacca, seppur non totalmente, dall'argomento principale di questa tesi, per soffermarsi come anticipato nel secondo capitolo, su alcune considerazioni relative alla possibilità di individuare goals visti come *fattori strategico-strutturali* a partire da un insieme di chunk, con un evidente riferimento alle strategie future per la definizione del goal set nel caso del planning su livelli di priorità.

6.1 Goals come Strategie

Come abbiamo avuto modo di notare nei capitoli precedenti, nel campo dei giochi con avversari in contesti che presentino un branching factor molto elevato e non abbiano elementi di casualità, le differenze fra l'approccio dei giocatori umani e quelli artificiali basati sulla forza bruta sono evidenti, fin dalla scelta del metodo con cui le due categorie analizzano le configurazioni degli stati.

Un giocatore umano, al contrario di un programma basato sulla forza bruta, tende a scansionare gli stati di gioco alla ricerca di elementi significativi, scartando arbitrariamente l'analisi di determinati percorsi e focalizzandosi su altri: ad oggi questa capacità di selezione di quelli che sono i *punti di interesse* è forse il più grosso fattore in favore del giocatore umano rispetto a quello artificiale: il caso specifico a cui siamo interessati è quello in cui egli compia questa scelta sulla base di considerazioni strategiche o in altre parole, identificando obiettivi che si è *addestrato* a riconoscere e che gli permettono di avere

una vista profonda sullo stato futuro del gioco, che non dipende direttamente da un procedimento di calcolo.

È utile rimarcare come ancora una volta, la discriminante fra il successo dell'approccio del giocatore umano e quello artificiale sia il branching factor o in altri termini, la profondità a cui riesce a giungere una visita con la forza bruta: basta pensare che giochi come la Dama e gli Scacchi, che sono alla portata di un'analisi brute force, sono dominati da giocatori artificiali mentre Go,Kriegspiel,Phantom Go ed altri giochi dal branching factor più elevato sono stati assolutamente dominati dai giocatori umani fino all'introduzione del metodo monte carlo e negli ultimi due casi, ai livelli più alti rimangono tali.

L'obiettivo di questo capitolo è presentare alcuni aspetti critici da prendere in considerazione per l'utilizzo tecniche di analisi che permettano di estrapolare i *fattori strategici* sopra menzionati, in ultima analisi con il fine di sfruttare goals set così ricavati in un planner su livelli di priorità.

6.2 Chunks come Goals

Abbiamo già spiegato nel corso del secondo capitolo che i chunks sono *fotografie* di elementi contenuti in una configurazione, individuati come particolarmente significativi.

Una prima idea, sperimentata e poi accantonata durante le fasi iniziali di questa tesi, nell'ottica di costruire un numero significativo di *piani prioritari*, consisteva nell'individuare posizioni che presentino caratteristiche considerate desiderabili: per esempio, compaiono in partite vinte per un numero $n \in N$ maggiore di una certa soglia fissata, sono presenti con una certa frequenza in prossimità di discontinuità nella funzione di valutazione favorevoli al giocatore,etc. ed impiegare ad ogni mossa una piccola quantità di tempo per cercare percorsi che portassero a raggiungere una di queste posizioni dallo stato corrente.

Sfortunatamente questo approccio non si è rivelato promettente per le tipologie di domini a cui ci stiamo interessando: le considerazioni precedenti sull'utilità di definire un insieme di "goal strategici" valgono in ambiti in cui

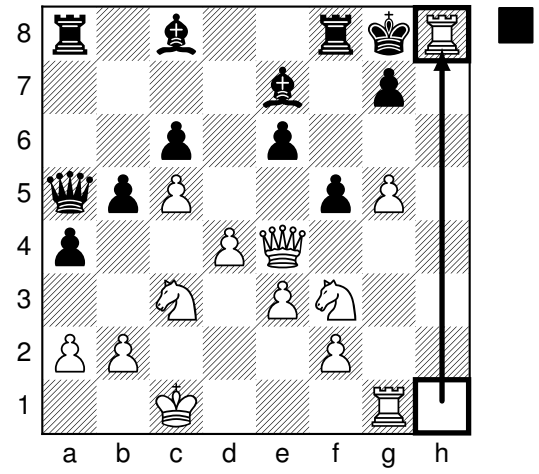


Figura 6.1: *Esempio di chunk contenuto in una posizione scacchistica*

un'analisi con la forza bruta sia inefficiente, cioè in casi in cui il branching factor sia molto alto; questa caratteristica porta però il numero di stati possibili ad essere estremamente elevato [1] il che rende il numero di chunk identificabili irrisorio, rispetto al numero di stati che il giocatore artificiale si troverà a visitare in pratica.

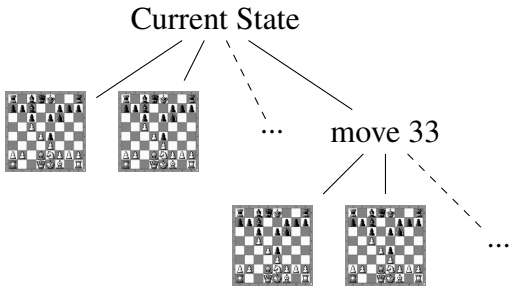


Figura 6.2: *Progressione dei chunks negli scacchi, dove il fattore di branching è pari a 33*

[1] Ad esempio, nel gioco degli scacchi il numero di configurazioni possibili è nell'ordine delle 10^{42}

6.3 Chunks come contenitori di Goals

Giocatori umani non percepiscono una posizione come un'entità statica,
ma come un insieme di azioni potenziali
Finkelstein e Markovitch L. Finkelstein [1998]

I chunks, seppur non utilizzabili direttamente come obiettivi per un giocatore artificiale nel contesto in esame, restano un concetto chiave nella emulazione dei processi cognitivi di giocatori umani; ricerche in questo campo [2] dimostrano come alcune delle differenze più significative fra scacchisti di fascia bassa e scacchisti di fascia alta siano proprio nella gestione del processo di memorizzazione e sfruttamento dei chunks.

Nel 1973, un esperimento in questo senso compiuto da Chase e Simon, consistette nel far memorizzare a giocatori di diversi livelli di abilità una partita di scacchi di 25 mosse, chiedendogli in seguito di ricostruire quanto visto. I risultati furono i seguenti:

Players	corretti	frammenti	posizione
Maestro	99	4	7.7
Esperto	95	2.5	10.5
Novizio	90	1.2	22

Fig 6.3 *Percentuali di successo da parte delle tipologie di giocatori*

Un punto significativo emerso dalle ricerche di Chase e Simon fu che i giocatori migliori tendono a riconoscere nelle posizioni analizzate caratteristiche o *relazioni fra i pezzi*: si può quindi considerare l'intero procedimento di riconoscimento e memorizzazione dei chunks come riconducibile all'individuazione di schemi noti ai quali assimilare la posizione, similmente a quanto avviene per il concetto di *stereotipo* in psicologia.

Un approccio interessante ai fini dell'individuazione di questi fattori può dunque essere quello di compiere una sorta di analisi retrograda, volta ad individuare schemi ricorrenti a partire da una collection di chunks.

[2] W.G.Chase [1973]

I problemi da risolvere in questo ambito sono identificare in modo algoritmico:

- Cos'è un chunk in una partita
- Cos'è una relazione (concetto che coincide con i *goal affidabili e inaffidabili* in un planner su livelli di priorità) in una collezione di chunks
- Quali relazioni sono interessanti all'interno di un goal set: alcune relazioni saranno facilmente verificabili per un giocatore artificiale, altre seppur interessanti per un giocatore umano saranno più difficilmente sfruttabili

6.4 Conclusioni

Il goals set di Strategic Darkboard è stato ottenuto identificato una soluzione parziale ai problemi esposti nel paragrafo precedente, ma è frutto di una selezione arbitraria e volutamente minimale per le ragioni esposte nel quinto capitolo: di conseguenza ai fini di questa tesi non è stato possibile focalizzarsi su aspetti quali machine learning, data mining o pattern recognition per la costruzione di goal set.

A dispetto di ciò, è significativo sottolineare come lo sfruttamento di queste tecniche sia un fattore di rilievo per tipologie di planning che si basino sulla nozione di strategia e nel caso specifico, futuri sviluppi del planning su livelli di priorità non potranno prescindere da approfondimenti in tal senso.

Capitolo 7

Conclusioni e sviluppi futuri

In questo elaborato abbiamo trattato il planning nella ricerca con avversari in giochi ad informazione imperfetta a somma zero, concentrandoci in particolare verso problemi caratterizzati da un elevato branching factor.

In fase di analisi, abbiamo riscontrato come nel contesto in esame vi siano fra le altre, alcune criticità:

- Nei giochi ad informazione imperfetta con avversari il Teorema di Zermelo non vale, di conseguenza non esiste nessuna assunzione possibile sull'esistenza di una mossa esatta, salvo casi particolari in cui l'insieme degli stati sia molto ristretto.

- Un alto branching factor limita un giocatore basato sulla forza bruta, l'approccio storicamente più efficace nella ricerca con avversari, ad un orizzonte estremamente breve.

Ulteriori riflessioni sul dominio hanno evidenziato come, in molti casi in cui giocatori umani ottengono risultati migliori di giocatori artificiali in questo campo (vedesi gli esempi di Go, Phantom Go, Kriegspiel e molti giochi strategici complessi), tali risultati siano connessi alla loro capacità di focalizzare la propria attenzione su determinati rami dell'albero di gioco che riconoscono come promettenti, elaborando strategie basate più su linee guida estremamente flessibili, che vengono abbandonate e riprese in base a valutazioni dinamiche sullo stato corrente, anzichè su piani ben calcolati in termini ad esempio di una funzione di valutazione classica.

A partire da queste osservazioni abbiamo elaborato l'approccio proposto, al quale abbiamo dato il nome di *Planning su livelli di Priorità*, basato principalmente sulla gestione di due tipologie di piani:

Raggiungimento di Goal Prioritari vengono elaborati piani tradizionali in presenza di situazioni di gioco *eccezionali*, che corrispondono ai momenti in cui il branching dello stato corrente si restringe abbastanza da poterlo gestire come un caso ad informazione perfetta (si pensi ad esempio all'obbligo di proteggere il re dopo uno scacco, nel Kriegspiel), o a considerazioni di altra natura che guidano verso un ramo specifico.

Miglioramento dello Stato L'approccio si basa sulla possibilità di identificare delle relazioni sugli stati, in un certo senso analoghe alle considerazioni strategiche che guidano le scelte di giocatori umani; l'obiettivo è per l'appunto quello di introdurre una nozione di strategia slegata da un calcolo su vantaggi espliciti in termini di funzione di valutazione, che offra quella visione a lungo termine che manca in questi ambiti.

Per sperimentare questo approccio abbiamo scelto un gioco particolarmente complesso, il Kriegspiel, potendo contare su uno strumento di testing eccezionale quale il programma campione del mondo, Darkboard.

7.1 Risultati

Avere la possibilità di effettuare dei test in un contesto concreto e in quanto tale valutabile, ha portato ad evolvere l'approccio ben oltre gli obiettivi iniziali (che non comprendevano la terza tipologia dei goal *inaffidabili* poi introdotta, ad esempio) ottenendo il significativo risultato di migliorare sensibilmente la versione campione del mondo del programma.

Un altro aspetto di rilievo è stato riscontrare come l'algoritmo proposto si presti in modo naturale ad un'integrazione con altri metodi basati sulla forza bruta, arrivando ad ottenere un effetto in potenza volto ad emulare la visione di un giocatore umano: l'algoritmo monte carlo integrato in Darkboard interpreta in modo eccellente quella che è la visuale a *corto raggio*, mentre la costruzione del gioco attraverso *linee guida strategiche* scelte fra gli elementi

della posizione corrente, introduce quel concetto di *obiettivo o direzione*, che era la vera ragione alla base della scelta di un algoritmo di planning.

In ultima analisi è opportuno segnalare che il già consistente miglioramento delle prestazioni di Darkboard si è verificato nonostante il goal set sia ancora decisamente poco ottimizzato: in fase di testing abbiamo infatti preferito limitare la conoscenza *specifica del dominio* affinchè i risultati ottenuti non fossero da ritenere interessanti solo nell'ambito del Kriegspiel, ma potessero essere auspicabilmente estesi alla classe di problemi da cui siamo partiti.

7.2 Sviluppi futuri

Per quanto riguarda Darkboard, una volta provato che l'approccio offre prospettive interessanti non dovute ad assunzioni specifiche sul dominio, il passaggio naturale è quello di introdurre un set esteso ed efficace di goals al fine di aumentare ulteriormente l'impatto del planner; questo potrà a nostro avviso garantire ulteriori e più significativi miglioramenti in termini di prestazioni.

La strada più promettente in questo senso è già stata individuata e sono stati predisposti dei meccanismi per integrarla in Darkboard: se i goal sono definiti come relazioni speculari alle considerazioni strategiche che guidano a lungo termine un giocatore umano, appare ragionevole pensare che lo studio di un database di partite, che è già a disposizione di Darkboard, possa permettere di identificare altre relazioni analoghe, auspicabilmente anche più efficaci, essendo tratte dalle partite dei più forti giocatori di Kriegspiel di ICC.

Per quanto riguarda il planning su livelli di priorità, interessanti ambiti applicativi per sviluppi futuri possono essere rappresentati da altri contesti strategici, caratterizzati da un avversario meno imprevedibile ed aggressivo rispetto a quello incontrato nel Kriegspiel, anche se giova ricordare che tale avversario era il campione del mondo, in modo da testare la possibilità di costruire piani più strutturati ed a più lungo termine.
A titolo esemplificativo, altri domini interessanti potrebbero spaziare da giochi di strategia *a la Risiko* a simulazioni economiche, fra gli altri: in ultima analisi, domini caratterizzati dalla presenza di informazione parziale, ambito

che a nostro avviso si segnala come uno dei più promettenti per lo sviluppo di algoritmi legati al planning.

Appendice A

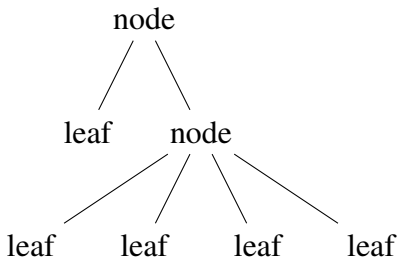
Glossario

1: Agente È detto agente una entità in rado di interagire con il suo ambiente in maniera *efficace*, è detto Agente Razionale un agente che disponga di un criterio oggettivo di valutazione delle proprie prestazioni.

2: Analisi retrograda L' analisi retrograda è un paradigma impiegato per risolvere alcuni problemi di scacchi: lo scopo principale è determinare quale sequenza di mosse ha portato alla posizione corrente.

I problemi su cui viene più frequentemente utilizzata sono noti come retro-problemi e sono tipicamente riconducibili al finale.

3: Branching factor Riferendosi ai nodi di un albero, il suo fattore di branching è pari al numero di figli di ciascun nodo; in caso tale numero non sia uniforme, il branching factor di un albero è dato dalla media sul branching dei figli.



Esempio di albero con Branching Factor pari a 3

4: Effetto orizzonte L'effetto orizzonte è un fenomeno che si verifica quando l'analisi di uno stato (effettuata ad esempio con minimax), dà origine ad un albero di gioco non interamente visitabile e di conseguenza, la visita viene interrotta ad una profondità che non corrisponde a quella in cui si trovano foglie: in tale caso la valutazione è approssimata ed elementi oltre l'orizzonte (profondità massima di analisi) sfuggono alla valutazione.

5: Giochi a somma zero È detto a somma zero, un gioco in cui ad ogni guadagno da parte di un giocatore, corrisponde una perdita di eguale entità da parte di un altro giocatore: la somma dei guadagni e delle perdite è un punteggio costante.

6: Gioco ad informazione perfetta Un gioco in cui ciascun giocatore è informato di tutte le mosse operate dagli altri giocatori.

7: Gioco ad informazione parziale Un gioco in cui i giocatori coinvolti non sono informati su ogni mossa compiuta dagli altri giocatori fino allo stato corrente: possono conoscerne un sottoinsieme, ma non la totalità.

8: Iterative deepening search o IDP, è una strategia di ricerca sullo spazio degli stati che consiste nell'iterare una ricerca depth-limited (cioè fino ad un orizzonte d fissato), sino al raggiungimento del goal ricercato.

La tecnica è completa, ottimale se ogni passo di esplorazione ha costo costante, e richiede tempo O($branching^{deepth}$) e spazio O($branching * deepth$)

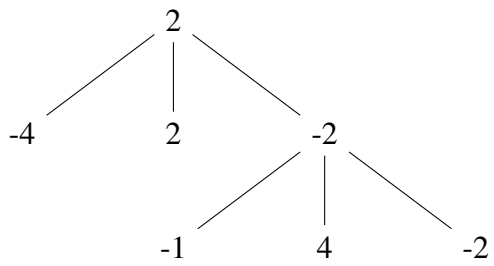
9: Machine learning Un ramo dell'intelligenza artificiale, connesso allo sviluppo di algoritmi volti a permettere ad un agente di evolvere il proprio comportamento, a partire dall'analisi di dati empirici.

10: Metaposizioni Una strategia adottata in giochi ad informazione imperfetta, basata sull'unificare un insieme di stati possibili in un unico *metastato*, che comprenda l'insieme delle caratteristiche considerate rilevanti del set di posizioni originarie, così da poterlo trattare come lo stato di un gioco ad informazione perfetta.

La gestione del finale di Darkboard si basa questo approccio.

11: Minimax È un algoritmo dimostrato nel 1928 da J.von Neumann , basato sul simulare il comportamento di due avversari, in cui uno tenta di massimizzare il valore dello stato attraverso la scelta di una transizione sull'albero di gioco, mentre l'altro cerca di minimiazzarla.

L'algoritmo è completo ed una soluzione esiste sempre per i giochi a somma perfetta a somma zero; computazionalmente ha un costo di O($branching^{deepth}$), ma esiste una strategia di riduzione detta potatura Alfa-Beta che lo riduce a O($branching^{\frac{deepth}{2}}$).



Esempio di costruzione di albero di gioco con Minimax

12: Strategia In contrapposizione alla tattica, una strategia corrisponde ad un piano d'azione di lungo termine usato come linea guida per impostare e *successivamente* coordinare azioni tese a raggiungere un obiettivo.

13: Tattica In contrapposizione alla strategia, un piano tattico corrisponde ad una serie di azioni a breve termine volte al raggiungimento di un obiettivo specifico.

14: Teorema di Zermelo Il teorema afferma che in ogni gioco finito ad informazione perfetta con un numero finito di posizioni, tra giocatori che muovono alternativamente e senza elementi di casualità ad intervenire nella selezione delle mosse, si dimostra che uno dei due giocatori deve avere una strategia vincente oppure esiste una strategia che porta alla patta.

La dimostrazione fu fornita nel caso degli scacchi ed è elementare: se il bianco ha una successione di mosse che lo porta alla vittoria la partita è vinta per il bianco, se così non fosse ed il nero ha una successione di mosse, qualsiasi mossa compia il bianco, che lo porta alla vittoria, la

partita è vinta per il nero: in caso nessuno dei due abbia una sequenza di mosse che lo porti a vincere, evidentemente la partita è patta.

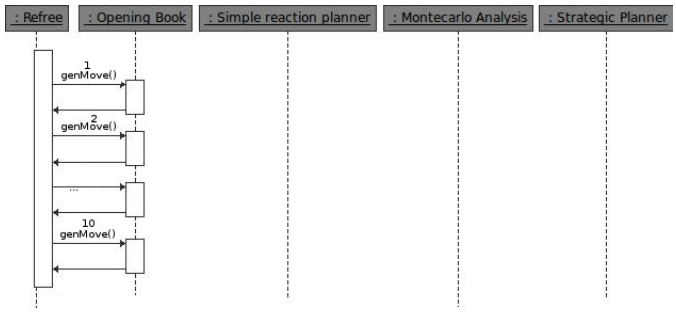
Appendice B

Una partita dal punto di vista dell'agente

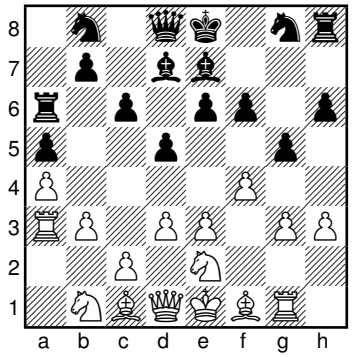
La partita in esame è stata giocata fra la versione di Darkboard in cui abbiamo integrato il planner, che gioca come bianco e Darkboard 2.0, che gioca come nero.

Fase di apertura:

Le prime 10 mosse vengono selezionate dal libro di aperture di Darkboard



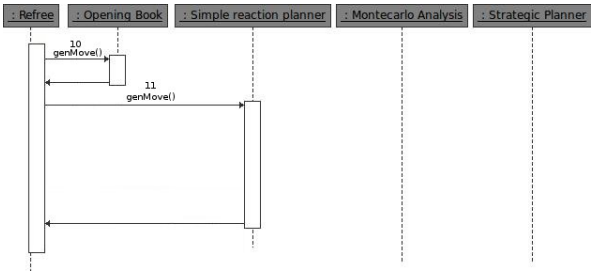
1. h3 a5 2. a4 Ra6 3. Ra3 c6 4. e3 e6 5. d3 h6
6. Ne2 Be7 7. Rg1 d5 8. g3 f6 9. b3 Bd7 10. f4 g5



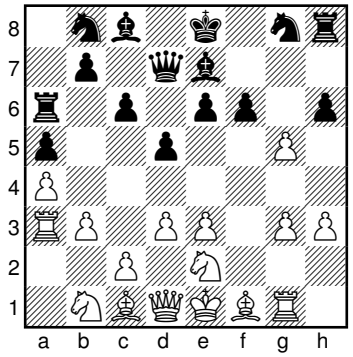
11) Darkboard riceve il primo messaggio e passa al mediogioco:

11. white's turn; 1 Try;

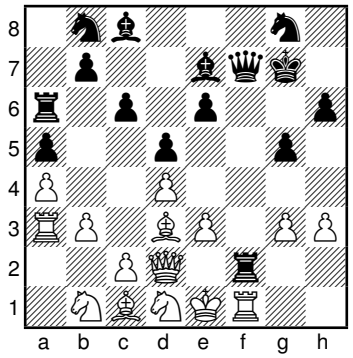
Essendo disponibile una cattura di pedone, viene utilizzata una simple-Reaction e Darkboard cattura automaticamente



11. fxg5



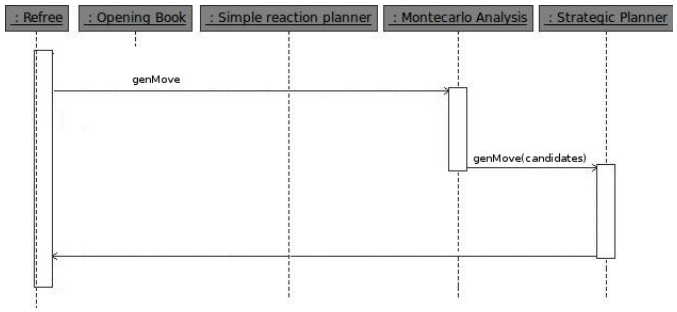
11. fxg5 fxg5 12. d4 Rh7 13. Qd3 Rf7 14. Qd1 Rf6 15. Qd2 Kf7
16. Nec3 Qe8 17. Be2 Kg7 18. Nd1 Qf8 19. Rf1 Qf7 20. Bd3 Qe8
21. Be2 Qf8 22. Bd3 Qf7 23. Be2 Qe8 24. Bd3 Qf8 25. Be2 Rf2
26. Bd3 Qf7



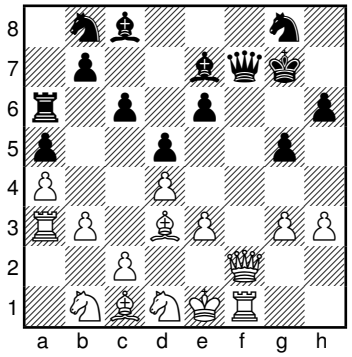
27) Le mosse valutate come migliori dall'analisi monte carlo sono Bd3-e2, c2-c3, Qd2-f2:

monte carlo selezionerebbe la prima, il planner seleziona Qd2-f2 al fine di formare una batteria di torre-donna; notare che in questo caso il giocatore non sapeva della presenza della torre nemica in f2.

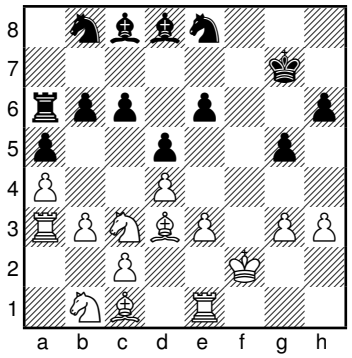
strategic evaluation: [12, 6, 14]
monte carlo chose: Bd3-e2 planner chose: Qd2-f2



27. Qxf2

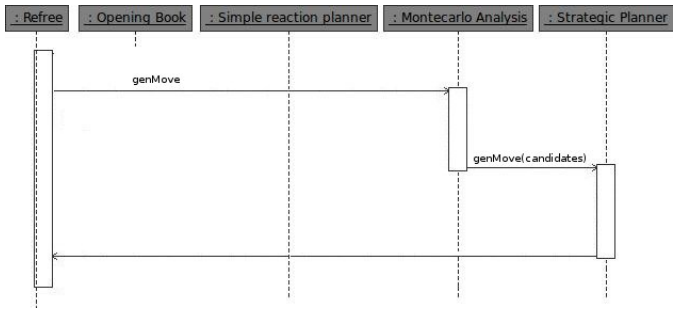


27. Qxf2 Qxf2+ 28. Kxf2 b6 29. Ndc3 Nf6 30. Rg1 Ne8 31. Re1 Bd8

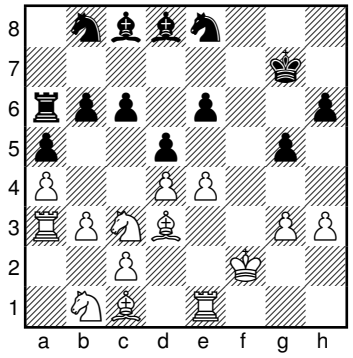


32) Solo una mossa supera il filtro dell'analisi Montecarlo, di conseguenza il planner e Darkboard concordano sulla scelta:

strategic evaluation: [4, 0, 0]
monte carlo chose:e3-e4 planner chose: e3-e4



32. e4



[...]
107. 1-0

Appendice C

Listato

Questa appendice non vuole essere un listato della totalità del codice sviluppato che, oltre ad essere molto più esteso, consiste in svariate aggiunte all'interno di Darkboard ed altre classi di supporto, che stilizzano il procedimento dell'estrazione ed identificazione dei goals al fine di gettare le basi per gli sviluppi futuri menzionati in precedenza: qui viene riportata solo la classe principale per lo sviluppo dei piani affidabili, al fine di descrivere la gestione di questa tipologia di goals e agevolare la comprensione di come viene trattato il concetto di *micro-obiettivi* su cui ci siamo molto soffermati in questo elaborato.

N.B Essendo Darkboard commentato in lingua inglese, ho ritenuto opportuno allinearmi commentando a mia volta in inglese, nonostante questo elaborato sia stato scritto in lingua italiana.

```
package ai.strategicplanner;

import java.lang.reflect.Array;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Vector;

import ai.player.Player;
import ai.strategicplanner.Goal;

import core.Chessboard;
import core.Metaposition;
import core.Move;

/** @author Andrea Gasparro
 * the aim of this class is to add a strategic view to
 * the offensive play  of Darkboard:
 * we take the candidate moves that Darkboard chose with MC
 * at deepth one,we watch what kind of configurations we
 * can reach (the known desiderable configuration are chosen
 * in the class planextractor) and we select the best one */

public class PlanExecutor {

public Player owner;     //the player we play for i.e Darkboard

private Move best;
  // the number of candidates moves we take from monte carlo
private int candidateNumber = 3;
  // deepth to explore  in game tree
private static final int fixeddeepth = 2;


/****static constants to move on the board representation ****/
```

```
private static final int left = -1;
private static final int leftUp = -9;
private static final int leftDown = 7;
private static final int right = 1;
private static final int rightUp = -7;
private static final int rightDown = 9;
private static final int up = -8;
private static final int down = 8;

private static final int upLeftCorner = 0;
private static final int upRightCorner = 7;
private static final int downLeftCorner = 56;
private static final int downRightCorner = 63;

//every square of the board is represented inside vector "cover"
SquareDescriptor cover[] = new SquareDescriptor[64];


/***********auxiliary functions ***************************/

/*check if "piece" is active on the given "square":
 * that means that it attack a contiguous square and "square" is
 * occupied by a friendly piece: note that if "piece" actually
 * occupy it,it is NOT active on it */

public boolean isPieceActive (int square,int piece)
{
    /*holds values of pieces standing on (if existing)
    * north,south,east,west squares*/
Vector<Integer> sides = new Vector();

    //...same as above,but with diagonals
Vector<Integer> diagonals = new Vector();
```

```
/*funzione che calcola i valori delle case laterali*/
if(exists(left,square))  sides.add(square+left);
if(exists(right,square))  sides.add(square+right);
if(exists(up,square))  sides.add(square+up);
if(exists(down,square))  sides.add(square+down);
if(exists(leftUp,square))  diagonals.add(square+leftUp);
if(exists(leftDown,square)) diagonals.add(square+leftDown);
if(exists(rightUp,square))  diagonals.add(square+rightUp);
if(exists(rightDown,square))diagonals.add(square+rightDown);

/* check if the (surely existing) squares that are close to
 * "square" are occupied by a piece who can move from "close
 * square" to "square"*/

switch(piece)
{
   case Chessboard.BISHOP:
     for(int k=0; k < diagonals.size(); k++) {
       if(cover[diagonals.get(k)].dest.contains(piece)) return true;
     }

    case Chessboard.ROOK:
      for(int k=0; k < sides.size(); k++) {
if(cover[sides.get(k)].dest.contains(piece)) return true;
      }
    case Chessboard.QUEEN:
for(int k=0; k < diagonals.size(); k++) {
  if(cover[diagonals.get(k)].contains(piece)) return true;
}
for(int k=0; k < sides.size(); k++) {
  if(cover[sides.get(k)].contains(piece)) return true;
}
}

/*...else,return false,the area is not protected by any piece*/
```

```
return false;
}



/*check if a square on the "side" of the square "square" exists */

public boolean exists(int side,int square)
{

switch(square)
{

/* here we check corners: if square is on a corner & side is
*  out of the board return false*/

 case upLeftCorner: {
 if(side==left||side==up||side==leftUp||side==leftDown||side==rightUp)
      return false;
 }

 case downLeftCorner: {
 if(side==left||side==leftUp||side==leftDown||side==rightDown||side==down)
      return false;
 }

 case upRightCorner:{
 if(side==leftUp||side==up||side==right||side==rightUp||side==rightDown)
  return false;
 }

 case downRightCorner: {
 if(side==right||side==rightUp||side==rightDown||side==down||side==leftDown)
  return false;
 }
```

```
}

/* here we check side: if square is on a side of the board and side
*  is out of the board,return false*/

for(int j=0; j <= 56;j=j+8)
{
    if( j == square && (side==Left||side==leftUp||side==leftDown) )
  return false;
    if( j-1 == square && (side==right||side==rightUp||side==rightDown) )
  return false;
}

/* last but not least,we check if the square is on the bottom or the top*/

if(square < 8 && (side==up||side==leftUp||side==rightUp) )
      return false;
if(square > 55  && (side==down||side==leftDown||side==rightDown) )
      return false;


return true;
}


/* initialize the cover matrix with 64 vectors,one for each square*/

private void initCover()
{
for(int k=0; k < 64;k++)
   {
   cover[k]= new SquareDescriptor();
   }
}
```

```
/* we can define a function to prune moves we can identify as
 * not promising, to reduce the cost of analyzing deeper levels*/

private boolean prune(Move roughcandidate)
{
    /*TODO PUT HERE TO EASE FURTHERE DEVELOPMENTE
    * We may decide to use other functions to reduce the moves
    * analized other then monte carlo; for intance we may avoid
    * moves that do not affect the side of the board where we
    * want to play*/

    return true;
}

/**********************end of auxiliary function***************/


/* set the owner of the class,i.e the player whom we are going
        *  to inherit the current Metaposition */

public void setOwner(Player own)
{
owner = own;
}

/* takes n = candidateNumber moves and select the best one from
*  a strategic point of view */

public Vector<Integer> bestStrategicMove(Vector<Move> candidates)
{

    Vector candidateRating = new Vector();
```

```
    candidateNumber= candidates.size();

      // we get a rating for each candidate move
    for (int k=0; k < candidateNumber; k++)
      {
/*we rate every darkboard's received move*/
candidateRating.add(strategicalValue( candidates.get(k),
    fixeddeepth,owner.simplifiedBoard));
      }

/*we just give back the candidate move ratings*/
return candidateRating;
}


/* the heart of the class: we use this method to evaluate the
 * strategical outcome of a move;
 * generate all legal moves in the actual position,
 * compute the goals reached at the current state,
 * prune the less interesting ones,
 * iterate the previous step till the fixed depth,
 * in the end compute and return the integer rate of the move*/

int strategicalValue(Move refinedCandidate,int deepthToExplore,
     Metaposition simplBoard)
{
  //this field contains the strategic value computed for the move
  int evaluation = 0;
  //we erase previously computed info on the board state
  initCover();

  /*we evolve the current Metaposition to the one we'd get
  * executing the input move */
  Metaposition futurePosition =
simplBoard.evolveAfterMove(simplBoard,refinedCandidate,
```

```
   Chessboard.NO_CAPTURE, -1, -1,
   Chessboard.NO_CHECK,
   Chessboard.NO_CHECK, 0);

  /*we generate all the possible legal moves from the present state*/
  Vector<Move> nextLevelMoves =
  futurePosition.generateMoves(true,owner);

  /* search for goals realized in the current state: it's all about
  *  piece mobility*/
  evaluation = getActiveGoals(nextLevelMoves);


  /* recursion: we call strategicalValue on new generated moves,
  *  if we're within depth levels*/

  if(deepthToExplore > 0)
  {
/* put here for further development using the prune function;
* here we prune the new generated moves,deleting the ones
*  that have no strategic interest to us*/

/*for (int k=0; k < nextLevelMoves.size(); k++) {
    if(!prune(nextLevelMoves.get(k))) nextLevelMoves.remove(k);
}*/

    for (int k=0; k < nextLevelMoves.size(); k++) {
      //one level deeper in gametree exploration
      evaluation += strategicalValue(nextLevelMoves.get(k),
     deepthToExplore-1,futurePosition);
}
}
    return evaluation;
}
```

```
/* search for active goals in the current position: the result
*  is the sum of those goal's rate */

private int getActiveGoals(Vector<Move> newMov)
{
  int rate = 0; //sum of the rates of every active goal

  /********description of piece activity on the board********/

  //the squares occupied by our pieces
  Vector<Integer> inhabitedSquares = new Vector();
  //last square were we detected a friendly piece
  int lastSquareFound = -1;

  /* add to every square the pieces attacking it*/
  for (int j=0; j < newMov.size(); j++)
  {

    //point to the currently analyzed squares
    int squarePointer=
    (7-newMov.get(j).toX )+(newMov.get(j).toY)*8;
    int sourcePointer=
    (7-(newMov.get(j).fromX)+(newMov.get(j).fromY)*8);

    cover[squarePointer].dest.add((int)newMov.get(j).piece);

    /*to add the inhabited squares: squares FROM WHERE a
    * moves start*/

    if( lastSquareFound != sourcePointer )
    {
      //we found a new inhabited square
      inhabitedSquares.add(sourcePointer);
      cover[sourcePointer].piece = (newMov.get(j).piece);
```

```
      lastSquareFound= sourcePointer;
    }

  }

    /* if the squares inhabited by a piece are contiguous to
    * the ones attacked by another piece we add the inhabited
    * square to the covered ones */
    for (int j=0; j < inhabitedSquares.size(); j++)
    {
      if(isPieceActive(inhabitedSquares.get(j),Chessboard.QUEEN))
      {
      cover[inhabitedSquares.get(j)].dest.add((int) Chessboard.QUEEN);
      }

      if(isPieceActive(inhabitedSquares.get(j),Chessboard.BISHOP))
      {
      cover[inhabitedSquares.get(j)].dest.add((int) Chessboard.BISHOP);
      }

      if(isPieceActive(inhabitedSquares.get(j),Chessboard.ROOK))
      {
      cover[inhabitedSquares.get(j)].dest.add((int) Chessboard.ROOK);
      }

      if(isPieceActive(inhabitedSquares.get(j),Chessboard.KNIGHT))
      {
      cover[inhabitedSquares.get(j)].dest.add((int) Chessboard.KNIGHT);
      }

    }

    /***************piece activity description is over**************/

//check what goals are reached within the current position
```

```
PlanExtractor goalCollection= new PlanExtractor();
Vector<Goal> pieceActivityGoals=
goalCollection.getPieceActivityGoals();
Vector<Goal> pieceConnectionGoals=
      goalCollection.getPieceConnectionGoals();

/*********piece activity goals**********/
/* for every piece activity goal
 *  check any move to see if
 *    the piece moving is of the type goal.source
 *   and it attack a square reachable by piece goal.source
 */
for(int k=0; k < pieceActivityGoals.size();k++){

  for (int j=0; j < newMov.size(); j++) {

  //point to the currently analyzed squares
  int squarePointer = (7-newMov.get(j).toX )+(newMov.get(j).toY)*8;

  if(pieceActivityGoals.get(k).dest == newMov.get(j).piece)
  if(cover[squarePointer].dest.contains(pieceActivityGoals.get(k).source))
{
 rate += pieceActivityGoals.get(k).rate; } //goal is reached
}
}


for(int j=0; j < pieceConnectionGoals.size();j++)
{
  for(int occupata=0; occupata <inhabitedSquares.size();occupata++)
  {
    //square containing the piece "attacked"
    int source= pieceConnectionGoals.get(j).source;
```

```
    //square containing the piece that connect to the other
    int attacker= pieceConnectionGoals.get(j).dest;

    if(cover[inhabitedSquares.get(occupata)].piece == source)
    {
      if(cover[occupata].dest.contains(attacker))
if(pieceConnectionGoals.get(j).level < Goal.UntrustedLevelGoals)
    rate += pieceConnectionGoals.get(j).rate;

    /*// PUT HERE TO EASE FURTHER DEVELOPMENT:
      // use this field to store goals in history,
      // for goal evolution
    * else:
    * goalHistory.add(pieceConnectionGoals.get(j));*/
    }
  }
}

return rate;
}
}
```

Bibliografia

A. Bolognesi and P. Ciancarini. Computer programming of Kriegspiel endings: the case of KR vs K. In J. van den Herik, H. Iida, and E. Heinz, editors, *Advances in Computer Games 10*, Graz, Austria, pg 325-342, 2003. Kluwer.

A. Bolognesi and P. Ciancarini. Searching over metapositions in Kriegspiel. In J. van den Herik, Y. Björnsson, and N. Netanyahu, editors, *Computer and Games 04*, volume 3846 of *Lecture Notes in Computer Science, Ramat-Gan, Israel, Springer*, pg 246-261, 2004.

E. Zermelo. On an Application of Set Theory to the Theory of the Game of Chess. In *Proceedings of the Fifth International Congress of Mathematicians*, volume 2, pg 501-504, Cambridge, UK, 1913.

P. Ciancarini, F. DallaLibera, and F. Maran. Decision making under uncertainty: a rational approach to Kriegspiel. In J. van den Herik and J. Uiterwijk, editors, *Advances in Computer Chess 8*, pg 277-298, University of Rulimburg, 1997.

J. Blair, D. Mutchler, and C. Liu. Games with imperfect information. In *Proceedings of the AAAI Fall Symposium on Games: Planning and Learning, AAAI Press Technical Report FS93-02, Menlo Park CA*, pg 59-67, 1993.

P.Ciancarini La scacchiera invisibile: Appunti su un gioco di guerra. *URL: http://www.cs.unibo.it/cianca/wwwpages/chesssite/kriegspiel/scacchierainvisibile.pdf.*

P.Ciancarini, G.Favini. Representing Kriegspiel states with metapositions. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07)*, pg 2450-2455, Hyderabad, India 2007, a.

P. Ciancarini, G. Favini. A program to play Kriegspiel. *ICGA Journal*, vol. 30:3-24, 2007.

P. Ciancarini, G. Favini. Monte carlo tree search techniques in the game of Kriegspiel. In *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI-09)*, pg 474-479, Pasadena (CA), USA, 2009a.

A. Del Giudice, P. Gmytrasiewicz, and J. Bryan. Towards strategic kriegspiel play with opponent modeling. In *AAMAS '09: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems*, pg 1265-1266, Richland, SC, 2009. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 978-0-9817381-7-8.

A. Parker, D. Nau, and VS. Subrahmanian. Game-tree search with combinatorially large belief states. In *Int. Joint Conf. on Artificial Intelligence (IJCAI05)*, pg 254-259, Edinburgh, Scotland, 2005.

S.Smith e D.Nau. Strategic planning for imperfect-information games. University of Maryland, Department of computer science, 1993. *URL: citeseer.ist.psu.edu/smith93strategic.html*.

S. Smith, D. Nau, and T. Throop. Computer bridge - a big win for AI planning. *AI Magazine*,vol 19, pg 93-106, 1998.

A.Cook. Computational chunking in chess. *PhD thesis, University of Birmingham, Department of Philosophy, England*, 2011.

W.Burgarda, S.Thrun, M.Bennewitza. Finding and optimizing solvable priority schemes for decoupled path planning techniques for teams of mobile robots. *Department of Computer Science, University of Freiburg, Freiburg, Germany,School of Computer Science, Carnegie Mellon University, Pittsburgh, USA*, pg 3-9, 2002.

B.Goethals. Survey on frequent pattern mining. *University of Helsinki, Department of Computer Science*, pg 6-22, 2003.

J. Blair, D. Mutchler, and C. Liu. Games with imperfect information. In *Proceedings of the AAAI Fall Symposium on Games: Planning and Learning, AAAI Press Technical Report FS93-02, Menlo Park CA*, pages 59–67, 1993.

M. Chung, M. Buro, and J. Schaeffer. Monte Carlo planning in RTS games. *Proceedings of the ieee symposium on computational intelligence and games, IEEE Conference on Computational Intelligence and Games,Essex University, Colchester, Essex, UK*, 4-6 Aprile 2005.

P. Gmytrasiewicz J. Bryan Del Giudice. Particle filtering approximation of kriegspiel play with opponent modeling. *Proc. of 4th Workshop in the MSDM Series, Budapest, Hungary*, pg 11-15, 2009.

C.E. Shannon. Programming a computer for playing chess. *Philosophical Magazine (Series 7)*, pg 256-275, 1950.

G.Kasparov. *My Greatest Predecessors, Vol 4*. Everyman Chess, 2003.

G.P.Favini. The dark side of the board: advances in chess kriegspiel. *PhD. thesis, Department of Computer Science, University of Bologna*, 2010.

G.Tesauro. Td-gammon, a self-teaching backgammon program, achieves master-level play. neural computation. *Journal Neural Computation archive Volume 6 Issue 2*, 1994.

D. Dowe E.Farr A. Jansen. inductive inference of chess player strategy. *Proceedings of the 6th Pacific Rim international conference on Artificial intelligence,Springer-Verlag Berlin,Germany, Heidelberg*, 2000.

S. Markovitch L. Finkelstein. A selective macro-learning algorithm and its application to the nxn sliding-tile puzzle. *Journal of Artificial Intelligence Research, Vol 8*, pg 223-263, 1998.

M.Guid. Search and knowledge for human and machine problem solving. *PhD. thesis, University of Ljubljana,Faculty of Computer and Information Science*, 2010.

R.Norvig. *Intelligenza artificiale, vol I.* Prentice Hall, 2005.

P.Ciancarini. Imparare dai programmi di scacchi. *Conferenza sull'intelligenza artificiale,Scaccomatto, Torino*, 5 Dicembre 2008.

R.Kumar. An analysis of chess games on the free internet chess server. *Term Project, Winter 2011, University of California, Los Angeles*, 18 Marzo 2011.

A.Sabharwal B.Selman R.Ramanujan. On adversarial search spaces and sampling-based planning. *29th International Conference on Automated Planning and Scheduling, pg 242-245, Toronto, Canada*, Giugno 2010.

H.A.Simon W.G.Chase. *The mind's eye in chess*. *Visual Information Processing, New York, Academic Press*,pg 215-281, 1973.

P. Ciancarini e G. Favini. Solving Kriegspiel endings with brute force: the case of KR vs. K. In J. van den Herik and P. Spronck, editors, *Proc. 12th Int. Conf. on Advances in Computer Games (ACG)*, volume 6048 of *Lecture Notes in Computer Science*, pages 136–145, Pamplona, Spain, 2010a. Springer.

A. Parker, D. Nau, and VS. Subrahmanian. The role of imperfect information. In A. Kott and W. McEneaney, editors, *Adversarial Reasoning. Computational approaches into reading the Opponent's Mind*, pages 209–230. Chapman and Hall, Edinburgh, Scotland, 2006.

Wikipedia. *WikiBook: scacchi*, *URL: http://it.wikibooks.org/wiki/Scacchi*.