UNIVERSITÀ DEGLI STUDI DI BOLOGNA

FACOLTÀ DI SCIENZE MATEMATICHE FISICHE NATURALI

Corso di Laurea in Scienze dell'Informazione

IMPLEMENTAZIONE DI UN ALGORITMO PER LA RAPPRESENTAZIONE COMPRESSA DI UNA PARTITA DI SCACCHI

Tesi di Laurea di: **MARCO MASCELLANI**

Relatore: Chiar. mo Prof. **LUCIANO MARGARA**

Sessione I

Anno Accademico 2000 / 2001

Parole chiave:

Teoria dell’informazione

Compressione

Scacchi

Codici ottimali

Entropia

Indice

Capitolo 5

Introduzione

Il recente impulso nella diffusione degli elaboratori in vari campi della vita quotidiana e gli imprevedibili progressi che l'informatica sta compiendo in questi anni hanno portato novità e benefici anche in campi ritenuti estranei o lontani dalle nuove tecnologie. L'inaspettato impatto informatico nel mondo di un gioco antico come gli scacchi è stato tale da rivoluzionare, nel suo piccolo, questo settore, modificando abitudini consolidate e aprendo nuovi orizzonti agli appassionati di questo gioco.

Ha dato un ulteriore impulso a questo veloce sviluppo anche l'eco che la stampa, specializzata e non, ha dato alle recenti sfide *uomo contro macchina*, capaci di evocare nell'immaginario collettivo il confronto tra l'intelligenza umana e quella artificiale, in questo campo realmente bilanciate, visti i notevoli passi avanti fatti dagli sviluppatori di giocatori artificiali, che sono riusciti a produrre macchine in grado di competere ormai con i più forti giocatori al mondo.

Il risultato di questa piccola rivoluzione informatica nel mondo scacchistico è che oggi praticamente ogni giocatore di scacchi professionista si serve di un elaboratore munito di programmi ad hoc, da una parte per analizzare posizioni, utilizzando un programma di gioco, dall'altra per trovare partite degli avversari o prepararsi nelle aperture [1], utilizzando un database di partite.

[1] Il termine apertura indica le fasi iniziali di una partita di scacchi

Lasciando per un momento da parte i programmi di gioco, per quanto riguarda invece i database di partite, dal punto di vista informatico, si è assistito, nel corso degli anni, allo sviluppo di numerose architetture e formati diversi, finalizzati a una efficace memorizzazione di questo tipo di archivi.

Esistono dunque attualmente vari metodi per memorizzare partite, più o meno efficienti per velocità e compattezza dei risultati. La domanda a cui cerca di rispondere questa tesi è: fino a che punto è possibile comprimere tali memorizzazioni? Il che, utilizzando il linguaggio della teoria dell'informazione, equivale a chiedersi: quanta *informazione/entropia* produce la *sorgente* partita di scacchi?

L'idea che sta alla base del progetto di questa tesi, per fornire una risposta nuova a tale domanda, è quella di sfruttare i forti legami logici che una partita di scacchi possiede al suo interno, per ottenere una maggiore compressione. Ciò sarà realizzato utilizzando strumenti scacchistici per cercare di ipotizzare delle previsioni sul possibile svolgimento della partita che si sta considerando, in modo da assegnare una distribuzione di probabilità a tutte le possibilità e di utilizzare queste informazioni statistiche aggiuntive tramite i metodi forniti dalla teoria dell'informazione, in particolare il codice di Huffman.

Nell'ambito di questo lavoro di tesi si è sviluppato quindi un algoritmo di compressione, basato sulla teoria dell'informazione, che, in generale, prende in input dati legati tra loro da relazioni logiche forti di cui sia possibile, algoritmicamente, avere una descrizione probabilistica e restituisce poi tali dati, codificati e compressi, in formato binario. In particolare si è applicato questo procedimento alla compressione di partite di scacchi e si sono confrontati i risultati del metodo proposto con quelli degli algoritmi già esistenti, mirati a questo particolare tipo di

compressione, realizzati negli ultimi anni con metodi via via più raffinati ed efficaci. Quello che si tenterà di dimostrare è che con l'introduzione di questo metodo si otterrà una migliore compressione, sfruttando soluzioni e risultati di quel campo teorico di grande interesse, per la trasmissione e memorizzazione di dati che è la teoria dell'informazione.

La tesi è organizzata come segue:

Nel capitolo 1 sarà introdotta la teoria dell'informazione, che costituisce la base teorica per i metodi utilizzati nel progetto.

Nel capitolo 2 sarà presentata un'introduzione al progetto e nel capitolo 3 una più ampia e dettagliata spiegazione di esso.

Nel capitolo 4 saranno forniti i risultati sperimentali del metodo trattato in questa tesi, anche in confronto con altri metodi già esistenti.

Infine, il capitolo 5 parlerà delle conclusioni e dei possibili sviluppi futuri.

Capitolo 1

Introduzione alla teoria dell'informazione

Nell'ambito di questa tesi verranno utilizzati alcuni concetti chiave della teoria dell'Informazione, di cui questo capitolo propone una breve introduzione.

1.1 Il modello della trasmissione

La teoria dell'informazione studia il sistema di comunicazione che, nella sua forma più semplice, può essere schematizzato come in figura 1.1.

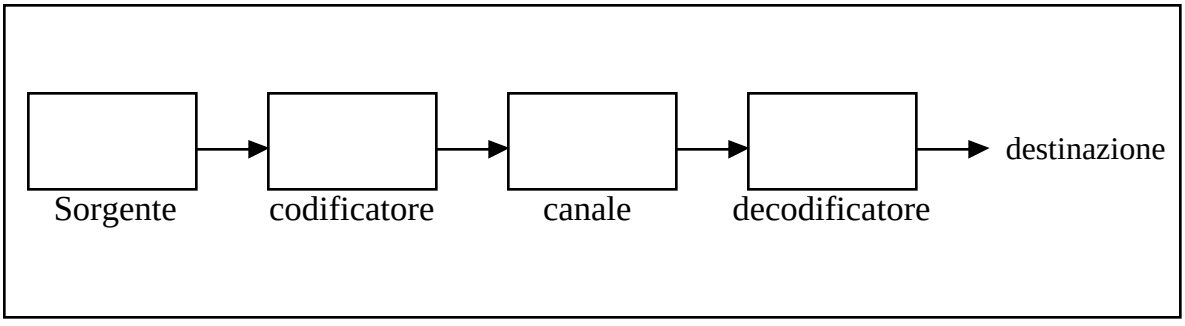

Figura 1.1: il modello del sistema di comunicazione

La sorgente genera il messaggio da trasmettere e si dice discreta se il messaggio consiste in una sequenza di simboli, continua se il messaggio è rappresentato da una funzione continua nel tempo. In campo informatico, si considerano di norma sorgenti discrete perché i sistemi di comunicazione tra elaboratori utilizzano una rappresentazione discreta dell’informazione. In generale, il primo passo consiste nel fare un modello della sorgente. Questi modelli non potranno essere deterministici, dato che non si può sapere in anticipo quale messaggio sarà trasmesso, ma saranno probabilistici, in quanto sono note prima, in minor o maggior misura, le leggi probabilistiche a cui il messaggio deve obbedire.

Il codificatore trasforma il messaggi in una sequenza di simboli di ingresso del canale. Anche quando i simboli della sorgente coincidono con quelli di ingresso del canale, è utile introdurre un codificatore. Questo perché, come si illustrerà nel corso di questo paragrafo, il codificatore svolge anche un altro compito.

Il canale rappresenta il mezzo utilizzato per la trasmissione. I canali si classificano a seconda che il loro ingresso e la loro uscita siano discreti o continui. Come nel caso della sorgente, considereremo solo canali discreti sia in ingresso sia in uscita. Vi è poi un’altra classificazione importante: canali senza rumore o canali rumorosi. Nel primo caso l’alfabeto di ingresso e di uscita del canale coincidono e il canale non fa mai errori; nel secondo caso il canale fa degli errori imprevedibili, per cui avremo bisogno di una descrizione probabilistica.

L’obiettivo del codificatore è diverso nei due casi. Con un canale senza rumore l’unico obiettivo sarà quello di rendere la trasmissione nel canale il più veloce possibile, minimizzando il numero medio di simboli del canale per simboli della sorgente. Nel secondo caso, oltre a rendere la trasmissione più veloce, ci sarà anche da combattere gli errori. I due

compiti possono essere separati, affidando il primo al codificatore della sorgente e il secondo al codificatore del canale, come mostrato in figura 1.2.

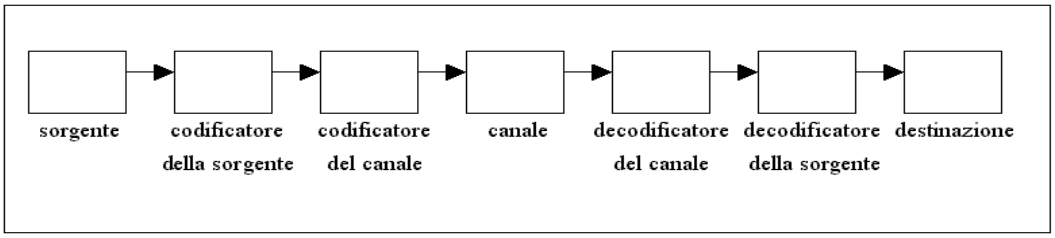

Figura 1.2: il modello completo del sistema di comunicazione

Il decodificatore nel primo caso ha il compito di ritrasformare la sequenza di uscita del canale nella sequenza trasmessa dalla sorgente; nel secondo caso cerca di indovinare meglio che può i simboli trasmessi, che vengono poi ritrasformati dal decodificatore della sorgente in una sequenza di simboli della sorgente stessa. La teoria dell'informazione dice come costruire modelli della sorgente ed introduce una misura della quantità di informazione associata ad ogni simbolo della sorgente, l'entropia. Mostra inoltre che questa misura, nel caso di un canale senza rumore, è legata al limite inferiore del numero medio di simboli del canale necessari per ogni simbolo della sorgente. Infine mostra come ottenere codici che usano il minor numero possibile di simboli del canale.

Nell'ambito di questa tesi verrà presa in considerazione una sorgente discreta, ed un canale di comunicazione discreto e senza rumore. Verrà considerato inoltre un processo di memorizzazione piuttosto che di trasmissione, ma i concetti illustrati in questo paragrafo sono validi sia per l'uno sia per l'altro, essendo da questo punto di vista perfettamente equivalenti. Se consideriamo infatti il dispositivo di memorizzazione come un canale ad alfabeto binario e come codificatore e decodificatore,

rispettivamente, le funzioni di scrittura e di lettura dei dati, otteniamo un modello del tutto equivalente a quello considerato in questo paragrafo.

1.2 L'entropia

Passiamo ora a introdurre la grandezza fondamentale per la teoria dell'informazione, l'entropia.

1.2.1 Definizione di entropia

Sia X un esperimento che abbia un numero finito di possibili risultati e_1, e_2, …, e_q, ciascuno con probabilità p_1, p_2, …, p_q. Vogliamo definire una misura dell'incertezza del risultato di X. Indichiamola con H.

$$H(p_1, p_2, ..., p_q) = -\sum_{i=1}^{q} p_i \log_2 p_i$$

Formula 1.1

Questa incertezza si può indicare anche brevemente con H(X), e si dice *entropia dell'esperimento*. Il nome entropia deriva dal fatto che H(X) ha la stessa espressione dell'entropia in certe formule della termodinamica statistica. L'entropia si misura in bit. Il bit è definito come l'entropia di un esperimento con due eventi equiprobabili, come ad esempio il lancio di una moneta.

L’utilità della misura di questa grandezza sarà evidenziata in seguito, quando sarà assegnata ad una sorgente di informazione e utilizzata per dare un limite alla lunghezza media del codice di codifica utilizzato.

1.2.2 Esempi di entropia

Nel paragrafo precedente è stata fornita la definizione formale di entropia in funzione delle probabilità dei possibili risultati di un esperimento. Quello che l’entropia così calcolata rappresenta intuitivamente è la quantità di informazione che porta con sé la conoscenza del risultato dell’esperimento o, da un altro punto di vista, la quantità di incertezza eliminata una volta conosciuto il risultato.

Ad esempio, se l’esperimento in questione ha un solo risultato possibile con il 100% di probabilità, conoscerlo non porterà alcuna informazione e non toglierà alcuna incertezza, poiché a priori è possibile prevedere perfettamente l’esito dell’esperimento. In questo caso, infatti, l’Entropia ottenuta, come risulta anche applicando la formula, sarà di zero bit. Un'altra prova può essere quella del lancio di una moneta con due risultati, testa e croce, equiprobabili. In questo caso l’entropia sarà diversa da zero, poiché a priori non è possibile sapere quale sarà il risultato dell’esperimento. In particolare, per convenzione, l’entropia di un esperimento come questo, con due risultati equiprobabili, è fissata ad 1 bit. Non è una coincidenza se, in questo caso, questa quantità corrisponde anche alla lunghezza di un codice binario utilizzabile per codificarne il risultato (0 per testa e 1 per croce è una semplice codifica di lunghezza media uguale a 1); infatti entropia e lunghezza media di un codice, come si vedrà in seguito, sono strettamente legate.

Come altro esempio si consideri un esperimento sempre con due risultati possibili, ma con probabilità molto sbilanciate, ad esempio 90% e 10%. In questo caso l'entropia è inferiore a un bit, in quanto è facile prevedere che il risultato ottenuto più di frequente sarà quello al 90%. Infine, un esperimento con 8 possibili eventi equiprobabili (con il 12.5% ognuno di probabilità) ha un risultato intuitivamente più incerto dei precedenti e infatti porta ad un'entropia pari a 3 bit.

Avendo introdotto il concetto base di entropia, vediamo come questo può essere esteso ad una sorgente di informazione discreta.

1.2.3 Definizione di sorgente discreta

Una sorgente d'informazione discreta S è una successione di esperimenti $X_0,X_1,\ldots,X_n,\ldots$ dove ogni esperimento ha lo stesso insieme di possibili risultati, costituito dall'alfabeto A della sorgente.

Questa definizione della sorgente come successione di esperimenti permette l'estensione del concetto di entropia alla sorgente stessa, in modo intuitivamente analogo all'entropia del singolo esperimento.

1.2.4 Entropia di una sorgente

Data una sorgente S, si definisce entropia della sorgente, indicata con H(S), la quantità:

$$H(S) = \lim_{n \longrightarrow \infty} H(X_n \mid X_0, X_1, \ldots, X_{n-1})$$

Formula 1.2

Cioè l'entropia di S è uguale al limite per n che tende ad infinito dell'entropia di X_n condizionata dalla conoscenza di $X_1,\ldots, X_{n-1}$. Indicato con q il numero di simboli dell'alfabeto A della sorgente, se H(S) esiste, si ha che

$$0 \leq H(S) \leq \log q$$

Formula 1.3

Da questa definizione deriva il fatto che a seconda dell'esistenza del limite della definizione in formula 1.2, alla sorgente sia possibile o meno associare un'entropia. In casi particolari, come ad esempio una sorgente stazionaria o una sorgente senza memoria, si può dimostrare che è sempre possibile calcolarne l'entropia.

1.3 Codici unicamente decifrabili ed istantanei

Esaurita la panoramica sulle caratteristiche delle sorgenti di informazione e dell'entropia ad esse riferita, nei prossimi paragrafi si tratterà delle proprietà e delle particolarità dei codici utilizzati per rappresentare i simboli della sorgente e si illustrerà come è possibile sapere se un dato codice, di cui si conosca la lunghezza delle parole, può essere utilizzato o meno per la codifica di una certa sorgente.

1.3.1 Definizione di codice a blocchi

Siano $A=\{A_1, A_2, \ldots, A_q\}$ e $B=\{B_1, B_2, \ldots, B_r\}$ due insiemi finiti di simboli. Un codice a blocchi è una funzione che associa ad ogni simbolo di A una sequenza di simboli di B; queste sequenze di simboli di B si chiamano parole di codice, A si dice alfabeto della sorgente, B alfabeto del codice.

Per codificare con i simboli di B non singoli elementi, ma sequenze di simboli di A è sufficiente esaminare, al posto della sorgente S, la sua estensione k-esima S_k, considerando i simboli di S a gruppi di k come singoli elementi di S_k.

1.3.2 Definizione di codice unicamente decifrabile

Un codice è unicamente decifrabile se ad ogni sequenza di simboli di codice corrisponde al più una sequenza di simboli della sorgente. È opportuno utilizzare una codifica di questo tipo se si desidera una trasmissione fedele, perché solo con questa premessa è possibile decodificare con successo, e in modo univoco, il messaggio originale. Un ulteriore sottoinsieme dei codici a blocchi e dei codici unicamente decifrabili è rappresentato dai codici istantanei.

1.3.3 Definizione di codice istantaneo

Un codice unicamente decifrabile è istantaneo se è possibile decifrare ogni parola del codice in una qualsiasi possibile sequenza senza far riferimento a simboli di codice successivi alla parola stessa.

Un esempio di codice unicamente decifrabile e istantaneo è il seguente:

$A_1 \longrightarrow 0$
$A_2 \longrightarrow 10$
$A_3 \longrightarrow 110$
$A_4 \longrightarrow 1110$

Infatti, una volta che compare lo 0 significa che la parola di codice che si sta leggendo è finita e, a seconda del numero degli uno, la si può decodificare correttamente, comunque prima di conoscere i successivi valori della sorgente.

1.3.4 La disuguaglianza di Kraft

Vogliamo costruire un codice istantaneo con q parole $X_1, X_2, \ldots, X_q$ composte rispettivamente di $n_1, n_2, \ldots, n_q$ simboli di codice. E' possibile? La risposta è la disuguaglianza di Kraft:

Condizione necessaria e sufficiente per l'esistenza di un codice istantaneo con la lunghezza delle parole $n_1, n_2, \ldots, n_q$, è che

$$\sum_{i=1}^{q} r^{-n_i} \leq 1$$

Formula 1.4

Dove r è il numero dei simboli del codice.

1.3.5 La disuguaglianza di Mac-Millan

I codici istantanei sono un sottoinsieme di quelli unicamente decifrabili; ciò nonostante, possiamo limitarci a considerare codici istantanei. Il motivo è che, se esiste un codice unicamente decifrabile con parole lunghe n_1, n_2, …, n_q, allora esiste anche un codice istantaneo con le stesse lunghezze delle parole. Conviene allora prendere quello istantaneo, che è equivalente e più semplice dato che ogni parola si può decifrare da sola.

1.4 Codifica in assenza di rumore

1.4.1 Lunghezza media di un codice

Per trasmettere i simboli emessi da una sorgente mediante un canale di trasmissione, dobbiamo trasformare le sequenza di simboli emessi dalla sorgente in sequenze composte da simboli del canale; dobbiamo quindi realizzare un codice con un dato alfabeto della sorgente $\{A_1, A_2, \ldots, A_q\}$ e un dato alfabeto del codice $\{B_1, B_2, \ldots., B_r\}$ (i due alfabeti possono anche essere gli stessi). Indichiamo con X_i la parola di codice associata al simbolo A_i della sorgente, e con n_i la lunghezza della parola x_i, e con p_i la probabilità del simbolo A_i; la lunghezza media del codice è

$$\bar{n} = \sum_{i=1}^{q} p_i n_i$$

Formula 1.5

Questo numero rappresenta, per il codice dato, il numero di simboli che occorre trasmettere in media per ogni simbolo emesso dalla sorgente. Se il canale di trasmissione è privo di rumore, riproduce all'uscita il simbolo trasmesso senza fare mai errori; in questo caso, l'unico obiettivo per un sistema di comunicazione efficiente è in generale quello di costruire codici che abbiano la minima lunghezza media (codici compatti o ottimali). Vedremo nel prossimo paragrafo che c'è un limite insuperabile.

1.4.2 Limite superiore della lunghezza media di un codice

Per una data sorgente stazionaria S, la lunghezza media n di un codice unicamente decifrabile è

$$\frac{H(S)}{\log r} \leq \bar{n}$$

Formula 1.6

dove l'eguaglianza vale se, e solo se, S è priva di memoria e $p_i = r^{-n_i}$, i=1, 2,…, q.

1.4.3 Teorema della codifica in assenza di rumore

Consideriamo sorgenti stazionarie. Se le probabilità di una sorgente priva di memoria non sono tali che $-\log p_i/\log r$ è, per ogni i, un intero, la lunghezza media di un qualsiasi codice unicamente decifrabile, e quindi anche quella del codice ottimale, sarà maggiore di H(S)/log r. Se però, invece di considerare la sorgente, si considera una sua estensione, è possibile rendere il numero medio di simboli del codice, necessari per codificare un simbolo della sorgente, vicino a H(S)/log r di tanto quanto si vuole. Più precisamente:

Esistono codici per l'estensione k-esima di S tali che, indicata con n_k la loro lunghezza media, si ha:

$$\lim_{k \longrightarrow \infty} \frac{\overline{n_k}}{k} = \frac{H(S)}{\log r}$$

Formula 1.7

1.5 Costruzione di codici ottimali

Nei prossimi paragrafi si illustrerà un metodo algoritmico per ottenere un codice ottimale a partire dalla sola distribuzione di probabilità dei simboli della sorgente.

1.5.1 Il metodo di Huffman

Fissata una sorgente S con simboli A_1, A_2, …, A_k e una certa distribuzione di probabilità sui simboli emessi $p1_1$, p_2, …, p_k, si vuole trovare un metodo che consenta di trovare un codice ottimale per questa sorgente. Per semplicità si consideri un codice di codifica binario, quindi con simboli 0 e 1.

Ordinati i simboli A_1, A_2, …, A_k in modo che $p_1 \geq p_2 \geq \ldots \geq p_q$, si considerano gli ultimi due simboli di S A_{q-1} e A_q come un unico simbolo, il quale avrà probabilità $p_{q-1}+p_q$, che indichiamo con $A_{q-1,q}$: si ottiene in questo modo una sorgente S_1 con q-1 simboli. I simboli di S_1 sono poi di nuovo ordinati, e gli ultimi due simboli considerati come un unico simbolo; si ha così la sorgente S_2 con q-2 simboli. Si continua finché si arriva a una sorgente con solo 2 simboli. La costruzione di questa sequenza di sorgenti, illustrata nell'esempio sotto, è il primo passo per la costruzione dei codici istantanei binari ottimali per S.

Il secondo passo è semplicemente quello di dare alla sorgente con due soli simboli il codice composto dalle due parole 0 e 1; questo codice è ovviamente ottimale per qualunque sorgente con due simboli. Il terzo passo è quello di ripercorrere all'indietro la sequenza di sorgenti ottenuta al primo passo; ogni volta che da una sorgente S_i si passa alla sorgente S_{i-1}, dal codice per S_i si ottiene un codice per S_{i-1} nel modo seguente: ai simboli di S_{i-1} che corrispondono a singoli simboli di S_i si lascia la stessa parola di codice, mentre ai due simboli A_x e A_y di S_{i-1} che corrispondono allo stesso simbolo $A_{x,y}$ di Si, si danno rispettivamente le due parole di codice X0 e X1, dove X è la parola associata a $A_{x,y}$. In questo modo, risalendo fino a S_0=S, si ha un codice per S.

1.5.2 Un esempio di codice di Huffman

Consideriamo una sorgente con quattro simboli x, y, w e z, rispettivamente con probabilità pari a 15%, 20%, 25% e 40% e utilizziamo l'alfabeto binario (zero e uno) per la codifica.

Ordinando S in modo decrescente, risulta:

S: z (0.40) w (0.25) y (0.2) x (0.15)

Unendo i due simboli meno probabili, si ottiene :

S_1: z (0.40) w (0.25) x, y (0.35)

Bisogna ordinare di nuovo e si ottiene:

S_1: z (0.40) x, y (0.35) w (0.25)

E unendo gli ultimi 2 simboli, si ottiene :

S_2: z(0.40) x,y,w(0.60)

Completata questa fase, si passa alla codifica vera e propria delle parole, risalendo da S_2 fino a S finché non si trova il gruppo costituito dalla sola parola che si sta cercando, aggiungendo ad ogni passo 1 se si considera il ramo sinistro e 0 se si considera il ramo destro.
Alla fine di questo procedimento, risultano le seguenti parole di codice:

z	1
w	00
y	011
x	010

La lunghezza media del codice così trovato, calcolata come la sommatoria della lunghezza delle singole parole moltiplicate per la rispettiva probabilità, vale 1*0.40+2*0.25+3*0.20+3*0.15=1.95 bit. Questo valore è ottimale, vale a dire che qualsiasi altra codifica basata su questa distribuzione di probabilità avrebbe una lunghezza media del codice maggiore o uguale rispetto ad esso. La conseguenza, in generale, è che se si conosce la distribuzione di probabilità, per codificare la sorgente è sufficiente applicare l’algoritmo di Huffman con la certezza che fornirà il codice più compatto.

La codifica di Huffman è l’ultimo passo della base teorica su cui si fonda il metodo di compressione trattato in questa tesi.

Capitolo 2

Introduzione al progetto

In questo capitolo verranno introdotti i concetti base scacchistici necessari per una completa comprensione del progetto. Verrà inoltre illustrato il metodo di codifica ideato in questa tesi, presentato come prodotto di raffinamenti incrementali alle codifiche già esistenti.

2.1 La rappresentazione di una partita di scacchi

Nei prossimi paragrafi sarà offerta una panoramica dei metodi, manuali o digitali, per la rappresentazione di una partita di scacchi.

2.1.1 La notazione algebrica

Il metodo comunemente usato dai giocatori di scacchi per trascrivere una partita in modo da potere successivamente riprodurre il suo svolgimento prende il nome di notazione algebrica, e fu introdotto dall'italiano Filippo Stamma nel 1737 [Cap73]. Tale metodo consiste nell'annotare, per ogni mossa, l'iniziale del pezzo che si muove in lettera

maiuscola (in Italia, A per alfiere, C per cavallo, etc.) e, in lettera minuscola e numero, la casella di arrivo dello stesso, denotando con le lettere da ‘a’ a ‘h’ le colonne e con i numeri da ‘1’ a ‘8’ le righe (traverse). Questo sistema è usato anche all’estero, con il solo mutamento delle iniziali dei pezzi.

2.1.2 Il formato PGN

Dal punto di vista informatico, data la crescente proliferazione da una parte di programmi di gioco e dall’altra di grandi database di partite, è nata la necessità di creare dei formati per la memorizzazione di partite singole e di database di partite.

Lo standard che si è affermato in questi anni prende il nome di formato PGN (Portable Game Notation) ed è in pratica nient’altro che una trascrizione testuale della ‘notazione algebrica’, tanto facile da leggere per l’uomo quanto logicamente inefficiente riguardo all’occupazione di spazio.

Per questa ragione quasi ogni sviluppatore di database scacchistici ha sviluppato un formato proprietario mirando all’efficienza e alla compressione dei dati.

2.1.3 I formati digitali compressi

Esistono dunque numerosi formati digitali differenti, identificati dall’estensione caratteristica del file che li contiene. Sostanzialmente quello che varia da formato a formato è il metodo di compressione che sfrutta la ridondanza dell’archivio quando questo inizia a contenere un notevole numero di partite. Il procedimento che sta alla base dell’algoritmo di compressione della singola partita, invece, nella quasi totalità dei casi è il

medesimo, ed è quello che nei prossimi paragrafi sarà illustrato come ‘codifica dipendente dalla posizione’ o ‘algoritmo delle mosse possibili’. È un algoritmo che dà un rapporto tra compressione e overhead computazionale molto buono, ma, come vedremo nei prossimi paragrafi, almeno dal punto di vista della compattezza, può essere ulteriormente migliorato.

2.2 Raffinamento dei metodi esistenti

Osserviamo più da vicino i metodi fino ad ora usati per ridurre al minimo la codifica di una partita: nei prossimi tre paragrafi si illustreranno le idee che stanno alla base dei metodi più comuni e se ne studierà l’efficacia, apportando successivi miglioramenti che condurranno al metodo utilizzato nel progetto di questa tesi.

2.2.1 Notazione compatta per corrispondenza

È una codifica concettualmente semplice e computazionalmente leggera e veloce. Considera per ogni mossa la casella di partenza del pezzo mosso e quella di arrivo, considerando quindi in tutto 4 coordinate (2 per la casella di partenza, 2 per la casella d’arrivo) con numeri da 1 a 8, codificabili indipendentemente in ‘parole’ di tre bit, per un totale di dodici bit per mossa [2].

[2] Solitamente, in ambito scacchistico, si indica con ‘mossa’ l’insieme della mossa del bianco e di quella del nero, a cui ci si riferisce singolarmente col nome ‘semimossa’. All’interno di questa tesi, invece, per semplicità, per mossa si intenderà la mossa effettuata da un solo giocatore.

In questo caso, dunque, in cui la lunghezza in bit per mossa è costante e uguale a dodici, la lunghezza media in bit di una partita corrisponde a 960 bit, cioè il prodotto della lunghezza media in bit di una mossa per il numero medio di mosse per partita, che vale circa 80.

2.2.2 Codifica dipendente dalla posizione

Un'idea per migliorare il metodo sopra esposto si basa sull'utilizzo delle informazioni che si hanno sullo stato della scacchiera in un dato momento. Queste informazioni si sfruttano utilizzando un programma che, data la posizione ottenuta fino a quel momento della partita, calcola tutte le mosse possibili della posizione e le numera. A questo punto la codifica della mossa si riduce alla codifica di questo numero, che tipicamente non va oltre i 64 (risultando quindi quasi sempre codificabile al massimo in 6 bit). Questo metodo, quindi, garantisce mediamente una occupazione di spazio inferiore al 50% rispetto a quello precedente, e ha come contropartita solo quella di dover calcolare tutte le mosse possibili in ogni posizione incontrata.

Si noti che la numerazione che si fa delle mosse possibili deve essere deterministica e univoca, (ad esempio, con la regola che siano etichettate con numeri più bassi le mosse di pezzi con casella di partenza più in basso a sinistra e con casella di arrivo anch'essa più in basso e a sinistra); si potrà così essere sicuri che quando il decodificatore riceverà il numero di mossa che è stato eseguito effettuerà la stessa numerazione fatta al momento della codifica, in modo da ricostruire correttamente la mossa. Formalmente, il criterio sopra descritto si può riassumere nella funzione di ordinamento tra mosse giudicando una mossa m1 minore di una mossa m2 se:

- la colonna della casella di partenza di m1 è minore di quella di m2,
- a parità, se la riga della casella di partenza di m1 è minore di quella di m2
- a parità, se la colonna della casella di arrivo di m1 è minore di quella di m2
- a parità, se la riga della casella di arrivo di m1 è minore di quella di m2

Ad esempio, partendo dalla posizione iniziale e applicando tale procedimento, si ottiene la seguente tabella indicante la numerazione e i codici relativi a tutte le mosse possibili.

Mossa	a3	a4	Ca3	Cc3	b3	b4	c3	c4
Numero	0	1	2	3	4	5	6	7
Codice	00000	00001	00010	00011	00100	00101	00110	00111
Mossa	d3	d4	e3	e4	f3	f4	Cf3	Ch3
Numero	8	9	10	11	12	13	14	15
Codice	01000	01001	01010	01011	01100	01101	01110	01111
Mossa	g3	g4	h3	h4				
Numero	16	17	18	19				
Codice	10000	10001	10010	10011				

Tabella 2.1: esempio di codifica dipendente dalla posizione

In questo caso, ad esempio, le mosse possibili sono 20, per cui la codifica risulta di 5 bit contro i 12 della codifica della notazione compatta per corrispondenza.

2.2.3 Codifica ottimale a mossa singola

Il passo ulteriore, rispetto al metodo precedentemente descritto, che si compie con questa tesi, concettualmente è quello di non considerare tutte le mosse possibili come equiprobabili, ma di assegnare ad ognuna una probabilità di essere giocata e di utilizzare le tecniche e gli strumenti della teoria dell'informazione per trovare il metodo di codifica ottimale, data la distribuzione di probabilità.

Questa probabilità è assegnata ad ogni mossa tramite due passi successivi: in un primo tempo si attribuisce un valore numerico ad ogni mossa in base alla sua validità dal punto di vista scacchistico. Poi, a seconda di questo giudizio, viene assegnata ad essa una probabilità di essere giocata, basandosi sull'ipotesi che tanto una mossa è migliore (o più precisamente sia considerata tale dal programma di gioco), più probabilità avrà di essere giocata. Tornando all'esempio della posizione iniziale e affrontandolo con questo metodo, otteniamo la tabella 2.2.

Come si può vedere, maggiore è il valore assegnato a una certa mossa, maggiore sarà la probabilità ad essa corrispondente. Analogamente a mosse più probabili corrisponderanno minori lunghezze delle rispettive parole di codice. In questo modo, se la previsione è esatta, la codifica sarà più compatta.

Il procedimento appena illustrato porta a una importante implicazione: il metodo così descritto si propone come codifica ottimale di una partita a scacchi *ben giocata*.

Mossa	a3	a4	Ca3	Cc3	b3	b4	c3	c4
Numero	0	1	2	3	4	5	6	7
Valore	-0.20	-0.19	-0.10	+0.01	-0.23	-0.37	-0.14	-0.11
Probabilità	0.046	0.046	0.050	0.057	0.045	0.036	0.048	0.050
Codice	00100	00001	1100	0110	00110	00111	1101	1010
Lunghezza	5	5	4	4	5	5	4	4
Mossa	d3	d4	e3	e4	f3	f4	Cf3	Ch3
Numero	8	9	10	11	12	13	14	15
Valore	0.00	0.00	0.00	+0.09	-0.09	-0.08	0.00	-0.10
Probabilità	0.056	0.057	0.058	0.063	0.048	0.048	0.056	0.050
Codice	1001	0111	0101	0100	1110	1111	1000	1011
Lunghezza	4	4	4	4	4	4	4	4
Mossa	g3	g4	h3	h4				
Numero	16	17	18	19				
Valore	-0.21	-0.22	-0.20	-0.19				
Probabilità	0.046	0.045	0.046	0.047				
Codice	00010	00101	00011	00000				
Lunghezza	5	5	5	5				

Tabella 2.2: esempio di codifica predittiva

Una partita tra amatori rientra già in questa definizione di ben giocata e verrà compressa con successo dall'algoritmo; al contrario, se si tenta di codificare una partita fatta ad esempio di mosse generate in modo casuale, la compressione in generale non sarà efficace.

Già Althöfer nel 1990 [Alt90] propone un metodo empirico basato sulla previsione delle tre mosse migliori in ogni posizione, ottenendo un buon risultato in termini di percentuale di compressione. Lui stesso in conclusione dell'articolo, ipotizza la possibilità di un approccio più estensivo basato sulla teoria dell'informazione.

L'approccio scientifico basato sulla teoria dell'informazione, oltre a portare una maggiore compressione, dà la prova che la soluzione così trovata, a patto di scegliere una buona distribuzione di probabilità, è ottimale, vale a dire che, avvicinandosi molto al limite teorico, difficilmente può essere migliorata.

Questo risultato, interessante dal punto di vista accademico, ha l'importante risvolto pratico di fornire la definizione di un formato compresso molto vicino al limite teorico. L'utilizzo pratico di questo algoritmo in database commerciali, tuttavia, è improbabile, poiché produce un overhead computazionale non trascurabile e questo, almeno ad oggi, rappresenta un problema per i maggiori programmi di gestione di database scacchistici.

2.2.4 Codifica ottimale a mossa composta

Un ulteriore passo per avvicinarsi ancora di più al limite teorico è quello di codificare, al posto delle singole mosse, gruppetti di due o tre mosse (in generale di n mosse), perché così facendo, per il teorema della codifica in assenza di rumore (cfr. 1.5), è possibile avvicinarsi quanto si vuole (nel caso teorico in cui il numero di mosse di cui sono composti tali gruppi si può spingere avanti a piacere) al limite dell'entropia. Ciò avviene poiché a ogni incremento di cardinalità nel gruppetto di mosse codificate, corrisponde, in generale, una diminuzione della quantità di informazione

trasmessa non utilizzata per la codifica. Le possibilità di applicazione di questo metodo saranno comunque illustrate nei prossimi capitoli.

2.3 Il programma di gioco: Crafty

Per la valutazione di posizioni e mosse all'interno del progetto è necessario l'utilizzo di un programma di gioco. E' stata scelta un'applicazione già esistente: Crafty, un programma freeware e opensource, ideato da Robert M. Hyatt, professore associato della University of Alabama a Birmingham, sviluppato e perfezionato in collaborazione da vari programmatori di tutto il mondo. Pur non avendo la forza di gioco di alcuni programmi commerciali, è adatto allo scopo di questo progetto, essendo relativamente leggero computazionalmente e sufficientemente accurato nel giudizio delle posizioni.

Un importante accorgimento da tenere presente nell'interazione con il programma di gioco, è che il risultato numerico da esso fornito come valutazione di una mossa deve essere deterministico e univoco. Cioè tale risultato deve essere uguale ogni volta che si richiede sulla stessa posizione con gli stessi parametri, dovendo essere usato in tempi e magari su macchine differenti prima dal codificatore poi dal decodificatore.

Un errore da evitare, ad esempio, sarebbe quello di richiedere il risultato di Crafty mettendo come parametro il tempo di riflessione, ad esempio di un secondo. In tal modo, infatti, il risultato potrebbe variare a seconda della velocità di calcolo della macchina su cui si esegue il programma.

2.3.1 Le valutazioni numeriche di Crafty

I risultati forniti da Crafty nel giudicare una posizione sono delle quantità numeriche decimali, in cui un numero positivo evidenzia un vantaggio per il bianco, mentre un numero negativo un vantaggio per il nero. Una unità di tale valutazione numerica corrisponde al vantaggio di un pedone (minimo vantaggio materiale possibile), mentre nella maggior parte dei casi un vantaggio non materiale, ma strategico o tattico, incide solo nella parte decimale di tale valutazione. Ad esempio, una valutazione di +1.5 significa che il Bianco ha un pedone in più e una posizione preferibile, oppure un vantaggio equivalente.

2.4 Metodi utilizzati per la previsione delle probabilità

Un parametro importante per il funzionamento efficiente del programma è l'assegnazione della distribuzione di probabilità sulla base dei risultati del programma di gioco. Infatti più questa distribuzione è aderente alla realtà, migliori saranno i risultati di compressione. Si parte dal presupposto che le mosse ritenute più forti da Crafty siano quelle giocate con più frequenza e cioè quelle a cui attribuire probabilità più alte. Ma qual è la probabilità che chi sta giocando faccia una delle mosse migliori? Per rispondere correttamente a questo interrogativo, tenendo presente che anche la valutazione di partenza di Crafty può essere inesatta, è necessaria un'accurata fase di testing, ma anche altre valutazioni possono spingere verso certe scelte.

Se ad esempio si ritiene di utilizzare l'algoritmo con partite di giocatori di alto livello, per le quali si ritiene probabile un'alta conformità

con le previsioni fatte da Crafty, la codifica può essere impostata in modo da codificare in modo particolarmente efficiente queste ultime, a discapito del fattore di compressione che si osserverebbe su una partita piena di errori. Cioè, se ci si aspetta una notevole conformità tra le mosse previste dal computer e quelle effettivamente giocate, si possono codificare le mosse migliori con parole di codice molto corte, dovendo poi però codificare le altre, al contrario, con parole lunghe; dal punto di vista dei risultati si ottiene un vantaggio se succede con frequenza alta che la mossa giocata sia tra quelle più probabili, uno svantaggio se si scende sotto una certa soglia di attendibilità.

Come si evince dai ragionamenti di cui sopra, le scelte sul passaggio dai valori di giudizio da parte del motore di gioco ai valori in probabilità utilizzati per la codifica, sono delicate. Nel prossimo paragrafo si illustreranno i criteri e i metodi adottati in pratica.

2.4.1 Scelte effettuate per l'impostazione delle probabilità

Per effettuare una scelta ottimale per ogni tipo di partita, i parametri di distribuzione probabilistica sono stati 'registrati' tramite prove effettuate su database già esistenti e sono stati fissati sui valori più efficaci. Per quanto riguarda le prove successive sull'efficienza dell'algoritmo, queste sono state effettuate, per correttezza, anche su database non utilizzati per la 'registrazione' e hanno fornito risultati perfettamente omogenei.

Per limitare i gradi di libertà delle svariate possibilità di mappatura dall'insieme dei valori a quello delle probabilità, è stata effettuata l'ipotesi che a uguali differenze di valori tra due mosse corrisponde un ugual rapporto tra le rispettive probabilità. In questo modo le svariate possibilità di scelta del passaggio alle probabilità sono state limitate alla sola scelta di

questi rapporti. In termini più formali, si è stabilito che ad una differenza di una unità tra due risultati numerici di valutazione, corrisponde un certo rapporto k tra le corrispondenti probabilità. Quindi se la differenza sarà uguale a 2, il rapporto sarà k^2, e in generale, se indichiamo con Δv la differenza tra i due valori, il rapporto varrà $k^{\Delta v}$. Per illustrare questo concetto, ipotizziamo una posizione con sole 4 mosse possibili.

Mossa	Giudizio	Rapporto	es. k=3	Probabilità
Mossa 1	+0.5	k^2	9	0.45
Mossa 2	-0.5	k	3	0.15
Mossa 3	-1.5	1	1	0.05
Mossa 4	+0.2	$k^{1.7}$	7.1	0.35

Tabella 2.3: esempio di calcolo delle probabilità

Come si può vedere, si è fissato per semplicità ad 1 il valore della mossa con giudizio più basso, per calcolare poi il rapporto tra questo e gli altri giudizi. Per la coppia di mosse uno e due e per quella delle mosse due e tre, che presentano entrambe una differenza di valore pari a uno, risulta il medesimo rapporto k. Per la mossa numero quattro, invece, che non ha una distanza unitaria dalla mossa col giudizio minimo, si è comunque calcolato il rapporto con un esponente frazionario, conservando le proporzioni scelte. Il passo successivo è il passaggio a probabilità vere e proprie dove, mantenendo le proporzioni dei rapporti appena calcolati, si è ottenuta una lista di valori la cui somma fosse uguale a 1.

In conclusione, con questo metodo si cerca di tradurre in pratica la volontà di far corrispondere innanzitutto a valore più alto probabilità maggiore e in particolare alla stessa differenza di valutazione un costante rapporto tra probabilità. In tal modo, il fattore che determina il settaggio della probabilità si riduce semplicemente ad una scelta del valore k sopra descritto.

Capitolo 3

Il progetto

Il progetto è stato sviluppato in 3 fasi successive: una prima di implementazione dell'algoritmo con alcune variabili parametrizzate, una seconda di testing e settaggio di parametri tramite prove effettuate su database reali e una terza di rifinitura e di aggiunta di alcune funzionalità. Il risultato finale è un algoritmo che fornisce una buona compressione, prestazioni accettabili e, dal punto di vista pratico, semplicità di utilizzo dovuta alla compatibilità col formato standard PGN (vedi paragrafo 2.1.2).

3.1 Scelte implementative

Nei paragrafi che seguono sarà illustrato il processo di sviluppo che ha portato alla realizzazione dell'algoritmo proposto, a partire dagli strumenti utilizzati fino ad arrivare agli schemi riguardanti il suo funzionamento.

3.1.1 Ambiente di sviluppo

Il progetto è stato implementato in C++, ed è stato utilizzato l'ambiente di sviluppo Visual C++ sotto il sistema operativo Windows. I

sorgenti del programma di gioco “Crafty”, sviluppato esternamente a questa tesi, sono invece scritti in parte in C, in parte in C++.

3.1.2 Progettazione

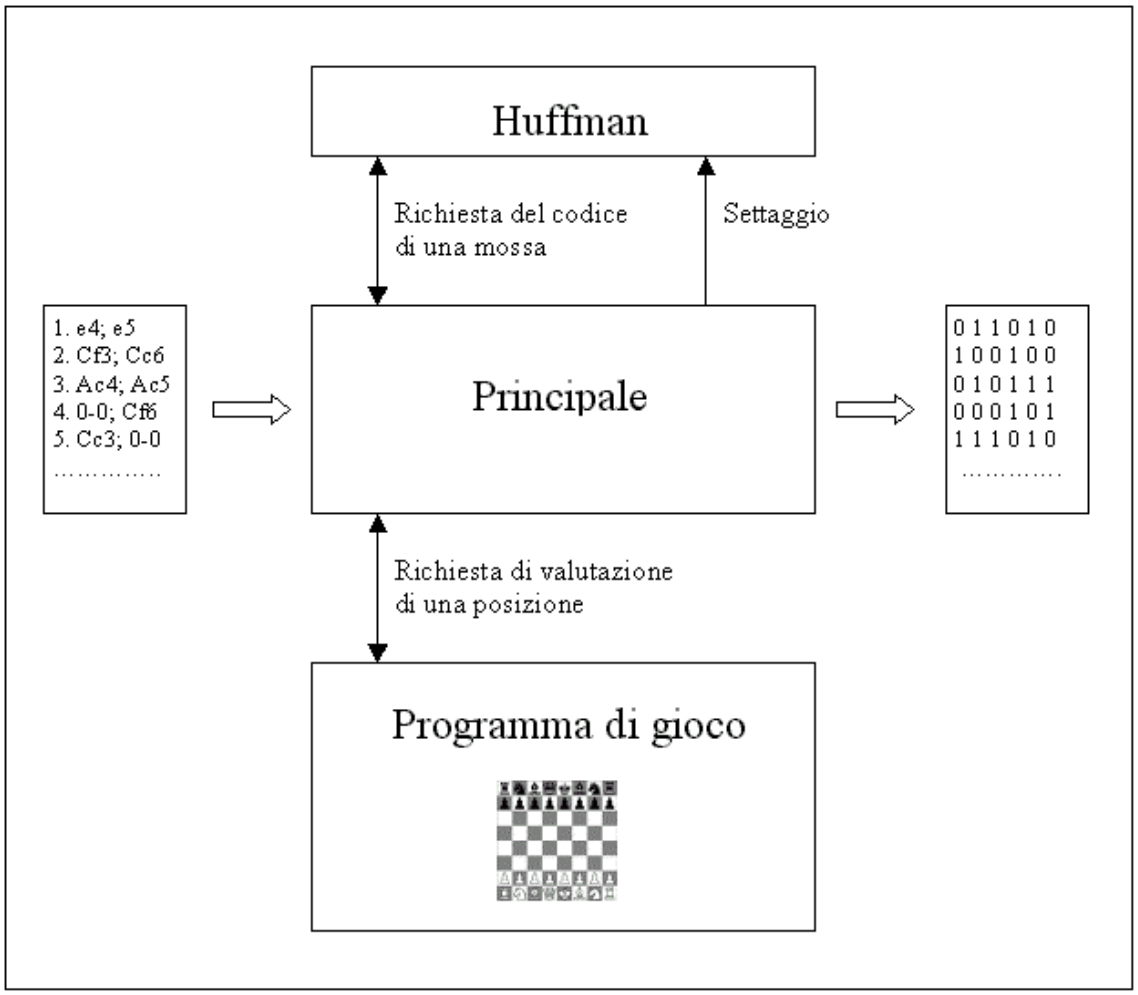

Figura 3.1: schema generale di funzionamento

Nella figura 3.1 è stato raffigurato il funzionamento del programma a partire da una partita di scacchi in input fino a produrre un file binario come risultato della compressione di tale partita. Come si può vedere, il progetto si può suddividere intuitivamente in tre parti distinte: una principale di elaborazione, una di gestione del codice di Huffman e infine una terza costituita dal programma di gioco.

Le interazioni del corpo principale con il modulo di Huffman sono di due tipi: il settaggio, per aggiornare la struttura dell'albero gestito da questo modulo, e di richiesta del codice per una certa mossa. Tali interazioni verranno approfondite nella parte riguardante l'implementazione del codice di Huffman (vedi paragrafo 3.4). Le comunicazioni con il programma di gioco, invece, sono limitate a richieste di valutazioni numeriche di mosse o posizioni. Le problematiche di interfacciamento sono approfondite nel paragrafo 3.3.2. Dal punto di vista esterno, infine, il programma trasforma semplicemente, nella sua funzione di codifica, la partita di scacchi che riceve in input in codice binario che emette in output.

3.1.3 Schema di funzionamento del codificatore

Entrando maggiormente nel dettaglio dell'implementazione, il funzionamento dell'algoritmo di codifica può essere schematizzato come in figura 3.2. Seguendo lo schema, il primo passo è aggiornare la posizione con l'ultima mossa effettuata. Se si sta considerando la prima mossa della partita, la posizione sarà quella iniziale. Successivamente il programma trova tutte le mosse possibili a partire dalla posizione corrente. Questo insieme di mosse passa poi attraverso varie fasi di analisi: in un primo tempo, tramite l'utilizzo di un programma di gioco, viene assegnata ad ogni mossa una valutazione numerica.

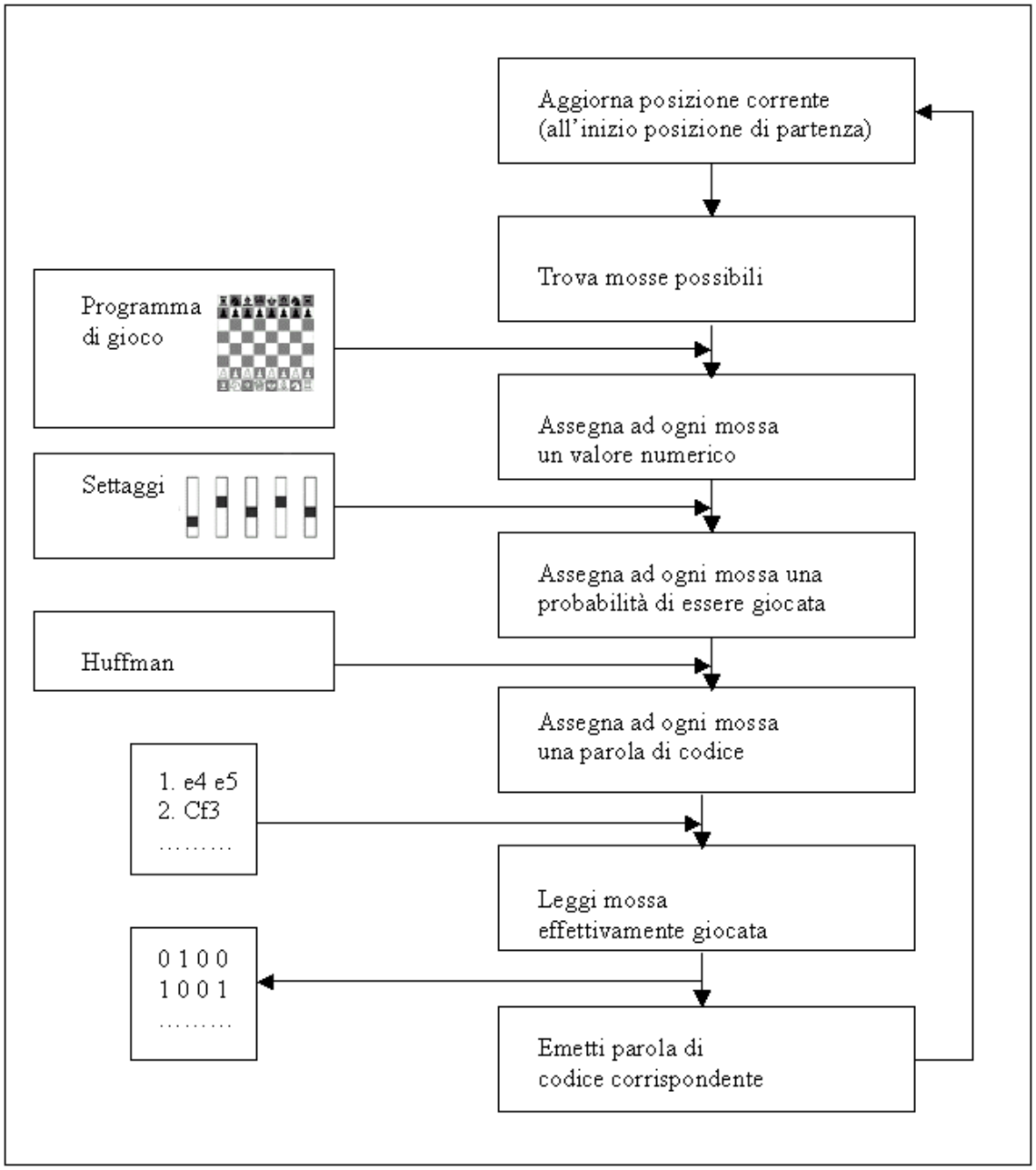

Figura 3.2: schema di funzionamento del codificatore

Il secondo passo è ottenere dalla valutazione numerica una probabilità di essere giocata, che sarà tanto più alta quanto più il giudizio

numerico effettuato dal programma di gioco sarà positivo. Una volta ottenuta la distribuzione di probabilità su tutte le mosse, tramite il metodo di Huffman, si assoceranno ad ogni mossa delle parole di codice, in modo da minimizzare (se le percentuali sono esatte) il numero medio di parole di codice utilizzate.

3.1.4 Schema di funzionamento del decodificatore

Una logica funzionalità del progetto è quella che permette di tornare dalla partita codificata a quella originale. All'interno della teoria dell'informazione il ruolo del decodificatore infatti è quello di tradurre il linguaggio usato dal canale di comunicazione, il cui ruolo è giocato in questo caso dal processo di memorizzazione binaria, nel linguaggio originale che è stato processato a monte dal codificatore, che in questo caso è la rappresentazione, che può avvenire nel linguaggio standard PGN o in altro modo, di una partita.

Come si può osservare in figura 3.3, tecnicamente il processo di decodifica è del tutto analogo a quello di codifica. Da notare che anche per il processo di decodifica è necessario processare le posizioni incontrate con il programma di gioco e se questo comporta un appesantimento dal punto di vista del tempo di elaborazione, non si può tuttavia limitare questo dispendio di tempo alla sola codifica. Un altro passaggio interessante poi è quello della lettura dal file binario delle mosse, che dovranno poi essere confrontate con quelle elaborate tramite il metodo di Huffman in modo da ottenere la mossa effettivamente giocata. Il problema infatti risiede nel fatto che non si conosce a priori la lunghezza in bit della parola di codice che si sta leggendo, per cui non si sa quando fermarsi nella lettura della mossa codificata.

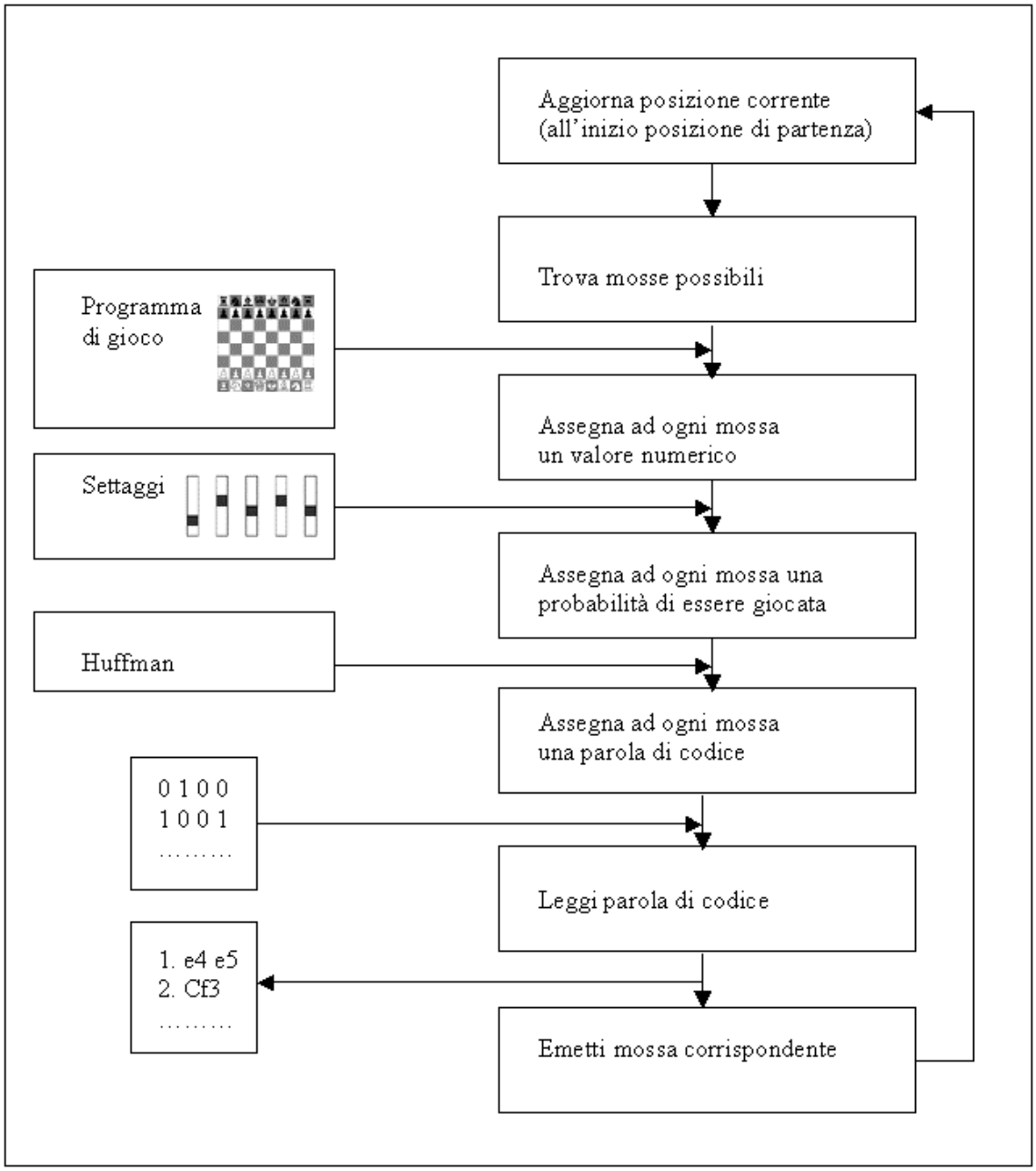

Figura 3.3: schema di funzionamento del decodificatore

Per risolvere questo problema è sufficiente ricordare che il codice prodotto dall'algoritmo di Huffman è istantaneo, cioè al suo interno nessuna

parola di codice è prefisso di un'altra (vedi paragrafo 1.3.3), per cui confrontando a ogni bit letto la parola fino a quel momento ottenuta con quelle possibili, la prima che si trova che corrisponda ad una esistente, è certamente quella definitiva.

3.2 Organizzazione del codice

Le parti logiche in cui si può suddividere il programma sono la gestione di mosse e posizioni scacchistiche, la gestione dell'input, con decodifica del formato standard PGN, quella dell'output binario, l'implementazione del codice di Huffman, l'interfaccia con il programma esterno Crafty e le infrastrutture per collegare le varie parti tra loro. Il codice è stato creato con un esteso utilizzo delle strategie object-oriented, con un occhio di riguardo alle possibilità di riuso, soprattutto per le parti in qualche modo slegate da questa particolare applicazione, come il gruppo di classi relative alla codifica di Huffman e le interfacce per la lettura del formato PGN.

3.2.1 Caratteristiche dell'applicazione

Di seguito un elenco delle principali funzionalità offerte dell'applicazione:

- possibilità di importare archivi di partite nel formato standard PGN
- confronto dinamico del metodo di conteggio veloce con quello predittivo

- possibilità di analisi del fattore di compressione al variare dell'impostazione delle probabilità
- possibilità di impostazione dei parametri mirata all'archiviazione di partite di livello alto piuttosto che di partite di livello medio-basso
- possibilità di osservare il funzionamento dell'algoritmo di compressione mossa per mossa

3.3 Integrazione con il "motore di gioco" scelto

Il "motore di gioco" è sostanzialmente un programma che gioca a scacchi ed è utilizzato all'interno del progetto nella sua funzione di valutazione di una determinata posizione o mossa.

3.3.1 Motivazioni della scelta

La scelta del programma di gioco è caduta su un programma esistente, Crafty, sviluppato da Robert M. Hyatt, professore associato della University of Alabama a Birmingham. È il prodotto di un progetto che prevede la distribuzione dei sorgenti e la possibilità di contribuire al miglioramento del codice estesa a chiunque. Ha visto la sperimentazione di tutti gli algoritmi di valutazione più avanzati ed è arrivato ad un livello di gioco piuttosto elevato, quindi perfettamente in grado di fornire un giudizio di riferimento sulla maggiore o minore validità scacchistica di una certa mossa (vedi paragrafo 2.3).

3.3.2 Interfacciamento dell'applicazione con Crafty

La parte di interfacciamento con Crafty è stata dal punto di vista pratico piuttosto difficoltosa a causa della relativa lentezza della fase di inizializzazione del programma. Per comprendere il problema basti pensare che, per ogni partita codificata, sono necessarie in media più di mille richieste di valutazione da parte del corpo principale del programma. Ciò significa che i tempi di inizializzazione pesano su una singola partita in misura di mille volte tanto. È stato pertanto necessario trovare il modo di sfruttare la stessa inizializzazione per la richiesta di più giudizi posizionali, modificando opportunamente il codice di Crafty. Quello che si è ottenuto è dal punto di vista pratico un buon compromesso tra affidabilità dei giudizi e velocità. È risultata però allo stesso tempo evidente la necessità di utilizzare, se si vogliono migliorare le prestazioni dell'algoritmo in termini di tempo di esecuzione, un programma opportunamente progettato.

3.4 Implementazione del metodo di Huffman

Il modulo contenente il codice delle classi che gestiscono l'algoritmo di Huffman può essere visto come una componente separabile dal resto del progetto e ha validità generale, fornendo semplicemente, dato un insieme in cui ogni oggetto ha una data probabilità, una codifica in cui ad ogni elemento corrisponde una parola di codice, in modo da minimizzare la lunghezza media del codice prodotto.

3.4.1 Realizzazione tecnica dell'algoritmo di Huffman

L'utilizzo di questa implementazione del metodo di Huffman si divide in due fasi: una prima di setup in cui, data la distribuzione di

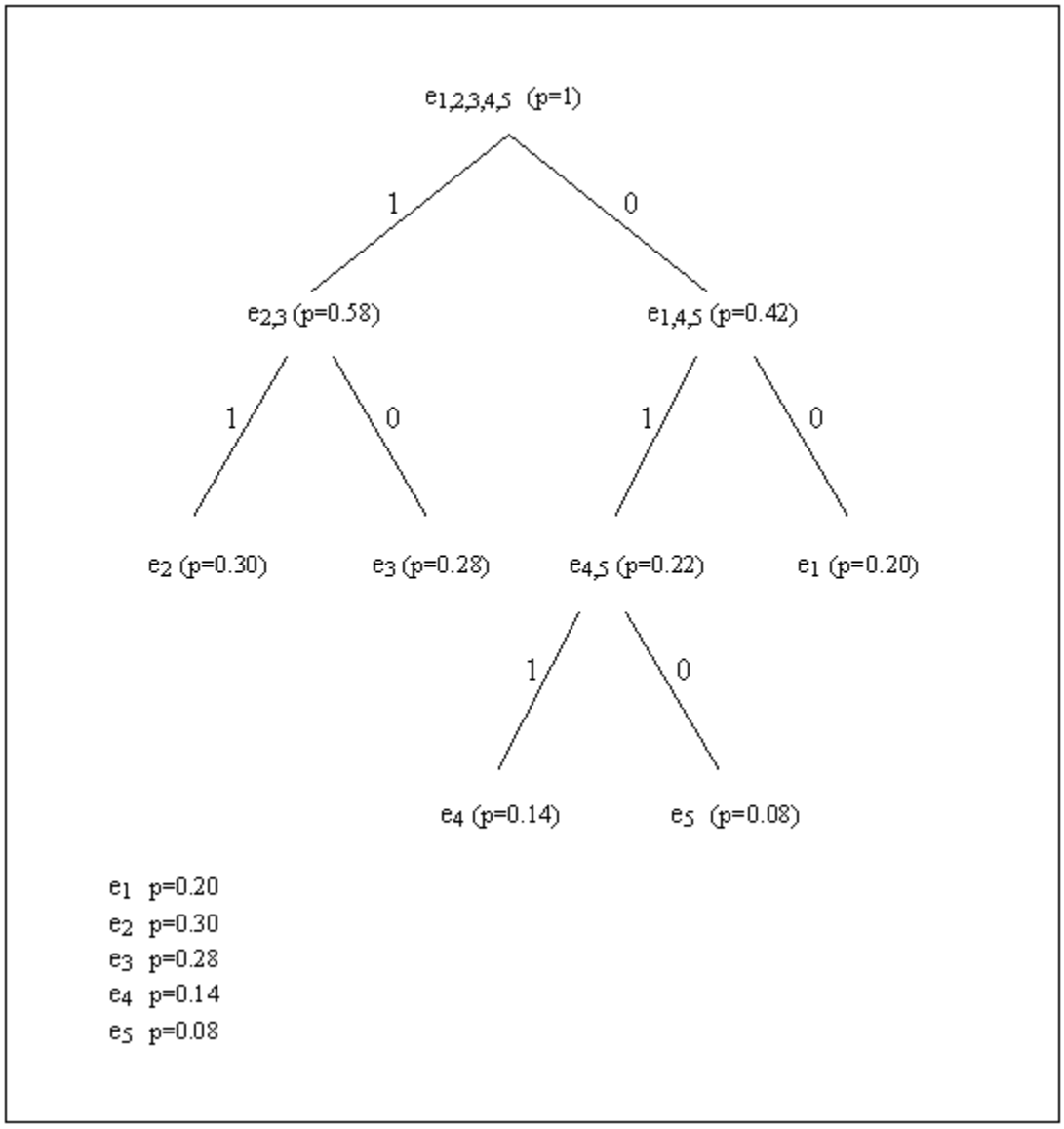

Figura 3.4: applicazione dell'algoritmo di Huffman

probabilità, viene creato un albero in cui ogni foglia corrisponde a un elemento dell'insieme da codificare. La seconda parte è la richiesta del codice di un dato elemento da parte del programma. In questo caso, una funzione, esplorando tale albero, a seconda della posizione in cui trova l'elemento cercato, restituisce in modo univoco il codice corrispondente. In figura 3.4 è schematizzato l'albero così costruito.

Come si può osservare, l'albero è stato costruito man mano (a partire dal basso) con i due insiemi di eventi meno probabili fino a raggiungere l'elemento finale, la radice dell'albero, contenente tutti gli elementi singoli e avente probabilità uguale a 1. In seguito, ogni diramazione dell'albero è stata etichettata con un simbolo del codice di output. Nel nostro caso, utilizzando un codice binario, si è scelto di usare l'etichetta 1 per tutti i rami sinistri e 0 per quelli destri. A questo punto, per la fase di ricerca del codice per una data parola e_i sarà sufficiente scorrere l'albero dalla radice fino alla foglia etichettata con la parola che si sta cercando e leggere le etichette dei 'rami' che si incontrano via via. Ad esempio, per codificare e_4, bisogna spostarsi sull'albero, a partire dalla radice, nel ramo destro, poi in quello sinistro, poi ancora il sinistro, ottenendo come parola di codice 011.

Concludendo, il codice di Huffman fornisce un metodo algoritmico, quindi facilmente informaticizzabile, per assegnare codici ai simboli della sorgente e fornisce anche la garanzia che il codice così trovato sia ottimale.

3.5 Variabili libere

Senza effettuare modifiche nel corpo del programma, costruito come specificato nei paragrafi precedenti, sono tuttavia possibili alcuni cambiamenti (vedi paragrafo 2.4.1) concernenti da una parte il passaggio

dalla valutazione numerica del programma di gioco alla probabilità assegnata in funzione a tale valutazione e dall'altra la decisione della grandezza dei gruppi di mosse da codificare insieme. I fattori che hanno guidato queste scelte sono, rispettivamente, l'efficacia della compressione di partite anche non di alto livello e il compromesso tra compressione e overhead computazionale.

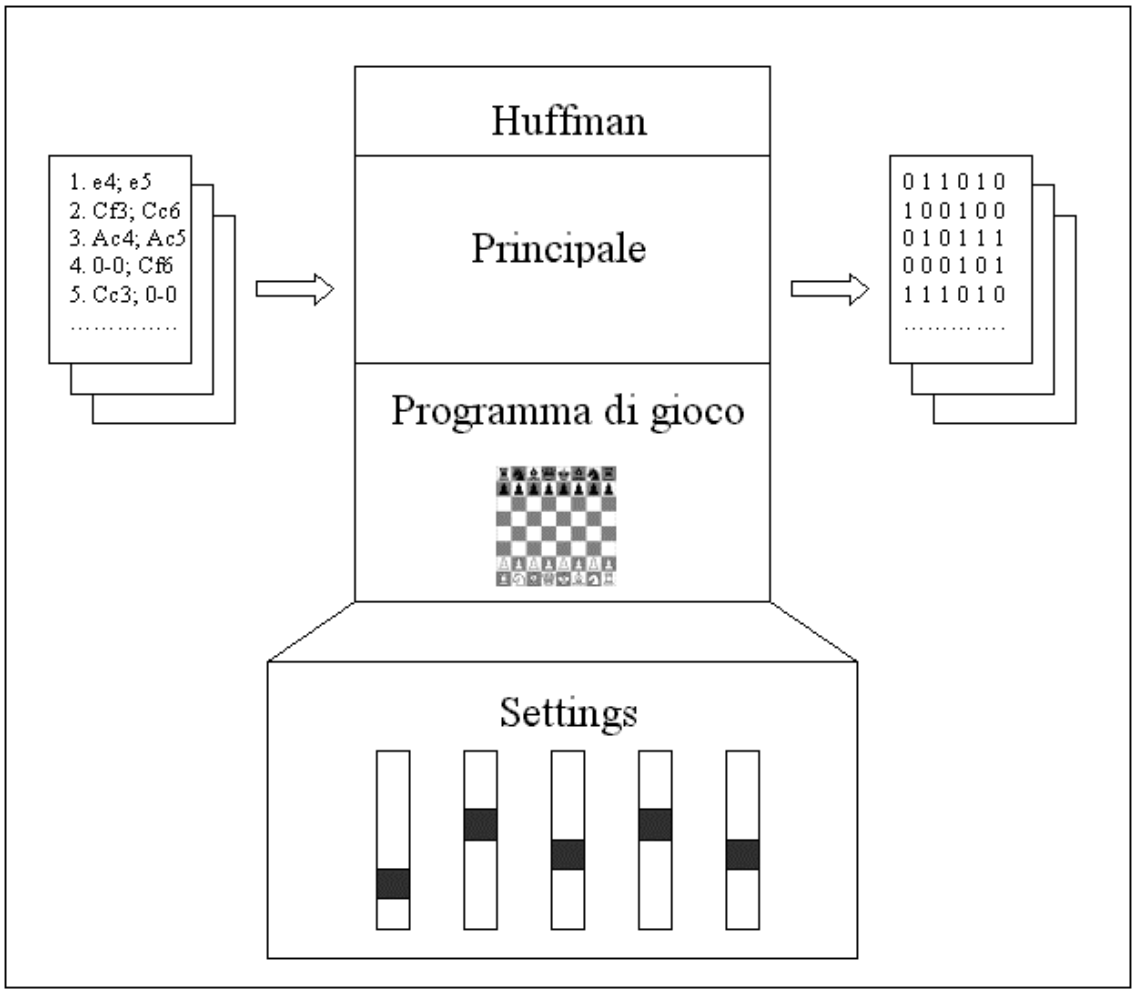

Figura 3.5: schema della fase di testing

Nel prossimo paragrafo si illustra il procedimento che ha portato al settaggio ottimale delle variabili ‘libere’.

3.5.1 Schema di testing

Le variabili libere sono rappresentate in figura 3.5 come manopole visibili dall’esterno, e le scelte effettuate su tali variabili sono rappresentate dallo stato delle manopole. Sono state effettuate varie prove con molte partite variando questi parametri man mano nel modo più opportuno, fino ad arrivare ai risultati che sono sembrati migliori, tenendo presente fattori come efficienza e buon livello di compressione.

3.5.2 Risultati dei settaggi

La scelta effettuata sui parametri per il passaggio alle probabilità è stata fatta sulla base di database di giocatori di livello medio-alto (con un punteggio ELO[3] medio indicativamente uguale a 2350). Il valore moltiplicativo k tra le probabilità di due mosse che distano 1 è stato fissato a 80, mentre per ottenere una velocità di computazione accettabile è stato necessario considerare mosse singole piuttosto che gruppi di mosse.

Il valore di 80 è stato scelto sulla base dei test effettuati con diversi valori sugli stessi database di partite, come sarà illustrato anche nel prossimo capitolo che è incentrato sui risultati sperimentali.

[3] ELO è un sistema di valutazione della forza di un giocatore tramite un numero che va grossomodo dai 1000 punti di un principiante agli oltre 2700 dei più forti giocatori al mondo

Capitolo 4

I risultati ottenuti

In questo capitolo vengono presentati, sottoforma di grafici o di dati numerici, i risultati sperimentali dell'algoritmo proposto in questa tesi. Innanzitutto verrà presentato il risultato del fattore di compressione in rapporto ai metodi di compressione già esistenti. In seguito saranno analizzati alcuni dati sulle sperimentazioni effettuate in fase di testing. Infine saranno mostrati i risultati numerici più significativi.

4.1 Grafici di confronto con gli altri metodi

Il risultato più importante per l'algoritmo sviluppato con questa tesi, a cui ci si riferirà nel seguito col nome '*predittivo probabilistico'*, è la sua effettiva percentuale di compressione, che sarà presentata e commentata in questo paragrafo. Lasciando per un momento da parte i valori di compressione per mossa o per partita assoluti che si ottengono con questo metodo, come primo dato saranno piuttosto mostrati i risultati della compressione in rapporto ai due principali metodi utilizzati nei formati di database esistenti, vale a dire la notazione *compatta per corrispondenza* e l'algoritmo delle *mosse possibili*.

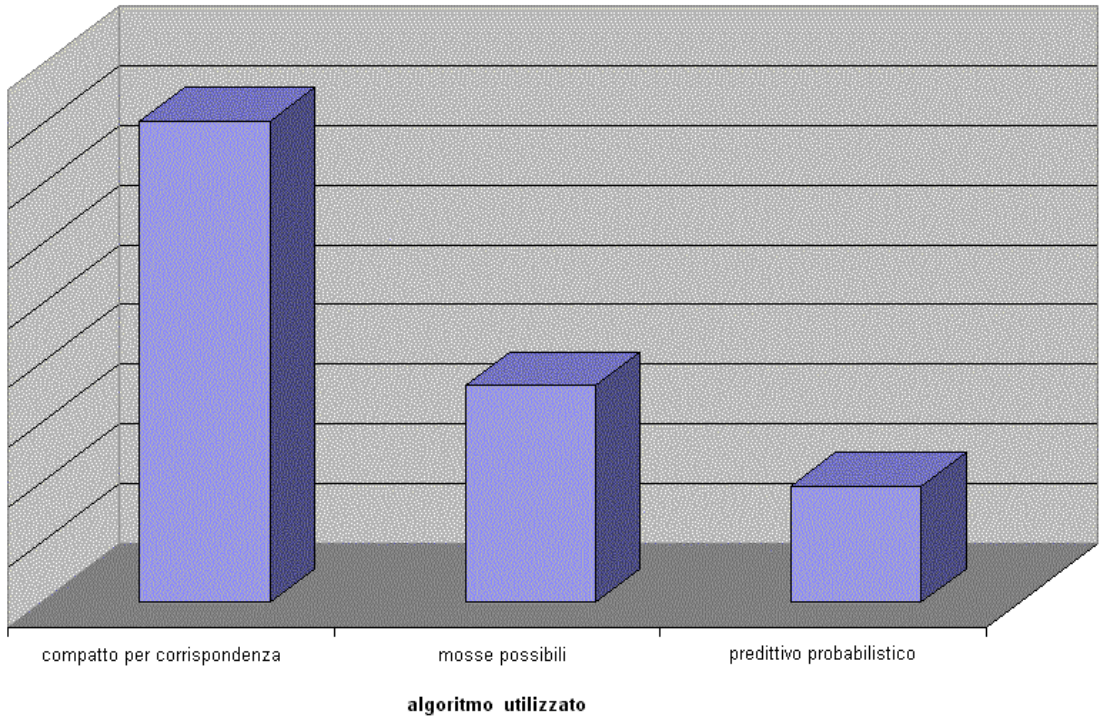

Figura 4.1: confronto tra i tre metodi principali

Come si nota dal grafico l'algoritmo *compatto per corrispondenza* è il meno efficiente dei tre, richiedendo una occupazione di spazio più di due volte superiore già a quello della codifica delle *mosse possibili.* Il successivo confronto tra quest'ultimo e quello proposto in questa tesi dà buoni risultati, dimostrando l'ipotesi che il fattore di compressione fissato dagli altri formati fosse migliorabile e infatti si può vedere un ulteriore dimezzamento (la percentuale precisa è del 55,7%) dello spazio necessario alla memorizzazione ottenuto con il metodo *probabilistico predittivo* introdotto in questa tesi.

4.1.1 Confronto tra il metodo compatto per corrispondenza e quello delle mosse possibili

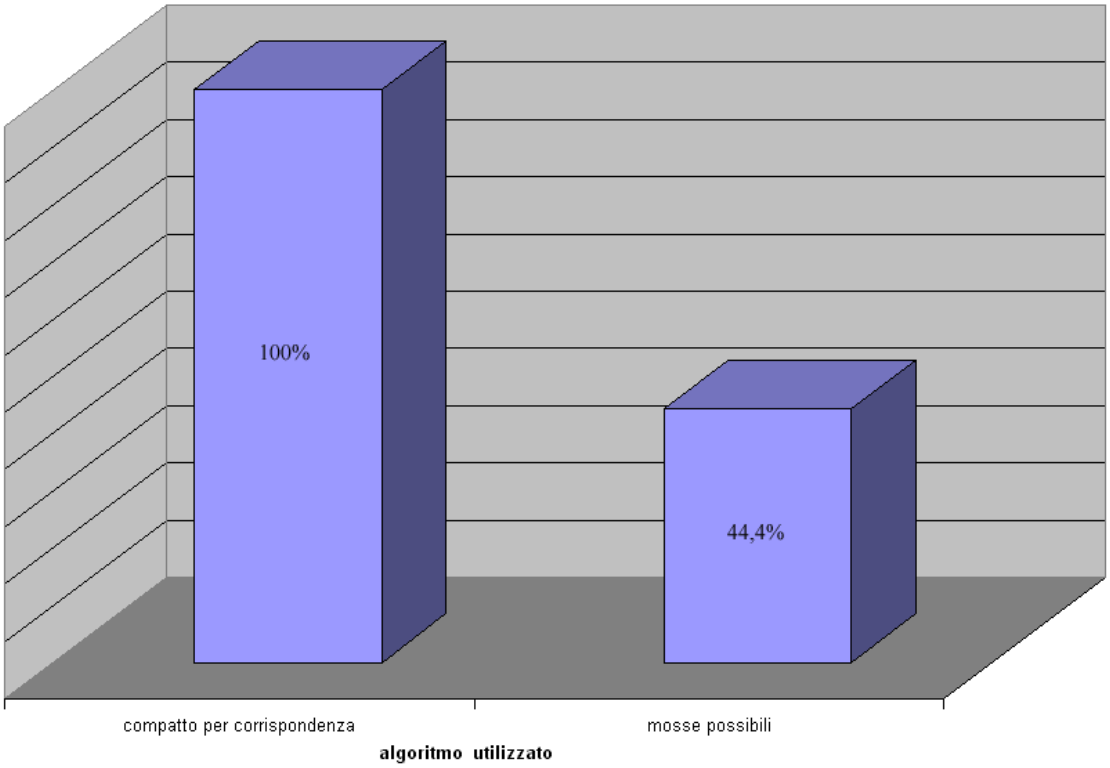

Figura 4.2: confronto tra il metodo compatto p.c. e il metodo delle mosse possibili

Analizzando più approfonditamente la differenza tra i primi due metodi, si nota che il secondo occupa mediamente circa il 45% di spazio rispetto al primo, con un risparmio di spazio quindi pari al 55%. La variabilità dei valori reali rispetto a questo dato medio dipende sostanzialmente da un solo fattore, cioè il numero di mosse possibili in ogni posizione incontrata nel corso di una partita. In particolare, più alto è questo numero peggiore sarà la compressione e viceversa più basso è il numero più la codifica sarà efficace. Il rapporto medio del 45% varierà comunque in

modo minimo, e comunque in nessun caso, neanche su una singola mossa, si otterrà una compressione meno efficace del primo metodo proposto.

4.1.2 Confronto tra il metodo delle mosse possibili e quello predittivo probabilistico

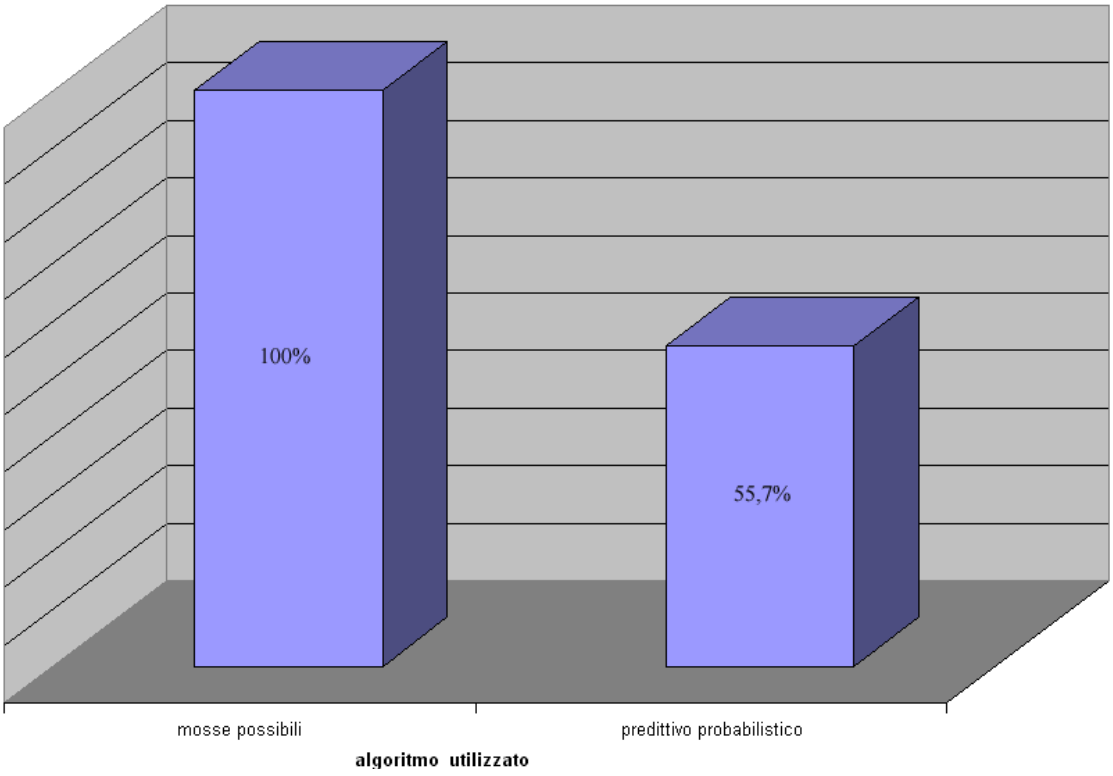

Figura 4.3: confronto tra il metodo delle mosse possibili e il metodo predittivo

Leggermente più complesso è il discorso che riguarda il confronto tra il secondo metodo e il terzo, quello proposto in questa tesi. Il rapporto di compressione del metodo più efficace è comunque valido, attestandosi su un valore di poco superiore alla metà del metodo precedente, precisamente il 55,7%, ma la variabilità di questo risultato da partita a partita merita

un’analisi più approfondita. Diversamente dal caso precedentemente analizzato, al variare del numero di mosse possibili totali di tutte le posizioni della partita il rapporto in esame non varia, partendo entrambi i metodi qui considerati dall’esclusione a monte delle mosse impossibili.

Il punto principale per analizzare la variabilità rispetto al rapporto medio di compressione parte dall’ipotesi che le mosse che compongono la partita siano di buona qualità dal punto di vista scacchistico. Ciò può non avvenire in tutti i casi e, in generale, capita che alcune partite possano essere compresse notevolmente meglio o peggio di altre, a seconda della loro fedeltà alle previsioni effettuate dal programma di gioco. Il risultato, in definitiva, è che si può notare una possibilità di variazioni piuttosto elevate da una partita all’altra rispetto alla percentuale media del 55,7%. Inoltre non vi è una garanzia certa di compressione maggiore da parte dell’algoritmo predittivo rispetto a quello delle mosse possibili, cioè può accadere che singole mosse o, più raramente, intere partite, risultino meno compresse con il primo metodo di quanto sarebbero compresse con il secondo. Se l’osservazione considera un certo numero di partite, però, risulta evidente che in media il metodo *predittivo* è decisamente più compatto, come è sottolineato anche dal valore medio.

4.2 Analisi delle variabili libere

Un punto importante dell’algoritmo di compressione è la ‘registrazione’ dei parametri che regolano la mappatura dei valori attribuiti dal programma di gioco a ogni mossa sulla distribuzione di probabilità ad esse attribuita. Si è illustrato (vedi paragrafo 2.4.1) come i gradi di libertà di

questa mappatura siano stati semplificati e limitati al variare di una singola variabile k, che definisce il rapporto in termini di probabilità tra due mosse il cui giudizio numerico differisce di una unità. A questo punto è stato necessario fissare per k un valore definitivo (per k variabile vedi il paragrafo degli sviluppi futuri) e per farlo si è effettuata una prova sullo stesso campione di partite variando k. Il risultato di questo procedimento è illustrato nella figura 4.5.

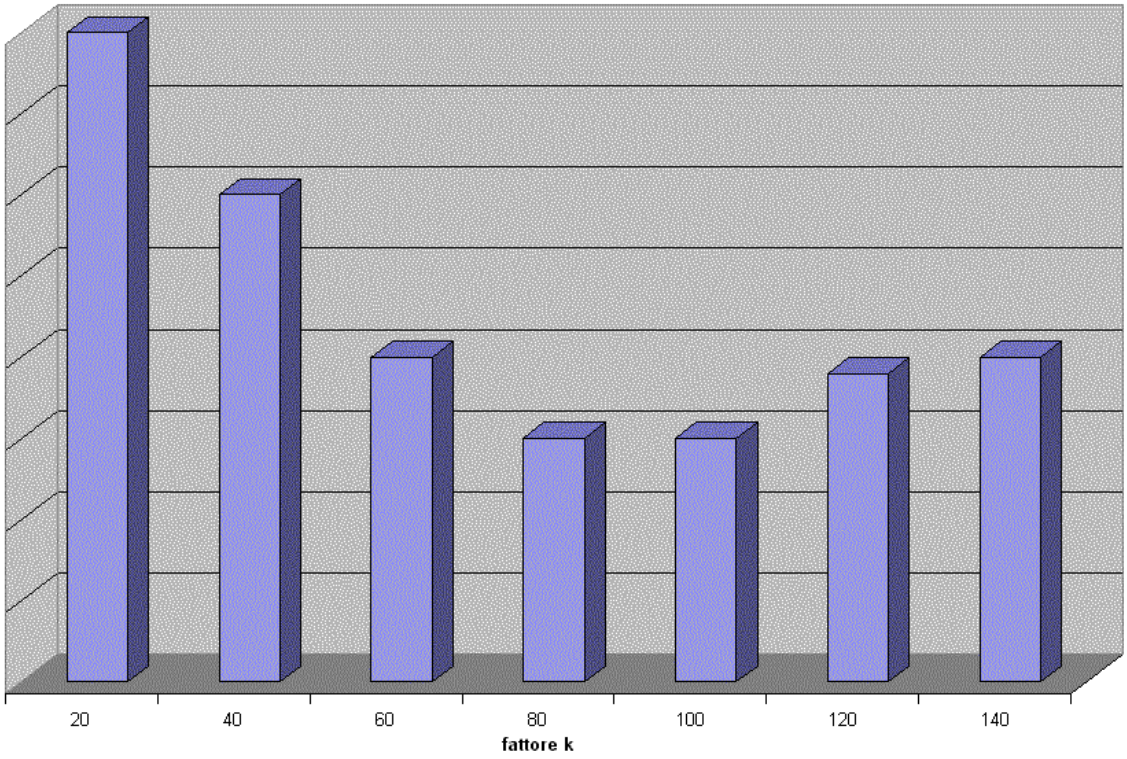

Figura 4.4: studio della compressione al variare del fattore k

Come si può osservare dal grafico, i valori più convenienti per il parametro k si trovano nella parte centrale del grafico. È bene tener presente che l'andamento del grafico è esasperato nelle differenze tra le varie

colonne; in realtà non c'è una differenza così marcata tra questi k possibili, basti pensare che tra le colonne più distanti c'è una differenza che non supera il 3%. Rimane comunque il fatto che i valori più efficaci si ottengono per k uguale a 80 e k uguale a 100. È stato scelto il valore 80 ed è questo il settaggio definitivo che è stato fissato per questa variabile. Il campione di partite considerate per la realizzazione di questo grafico ha rivestito un ruolo importante nella scelta del k, ed esso stesso è stato scelto con cura, alternando partite di alto livello a partite di medio livello. Concludendo, il valore scelto, relativamente basso, mantiene un buon rapporto di compressione per un ampio spettro, in termini di qualità scacchistica, di partite.

4.3 Tutti i risultati numerici

In questo paragrafo saranno illustrati i risultati numerici prodotti da una parte nella fase di testing e di settaggio dei parametri e dall'altra nelle prove di compressione effettuate una volta regolato l'algoritmo.

4.3.1 La sperimentazione

La sperimentazione è stata effettuata su un campione di 84 partite scelto con cura, innanzitutto utilizzando fonti diverse, come archivi tematici per apertura, database commerciali, partite prodotte da giocatori artificiali, oppure materiale preso dalla *rete*. Allo stesso tempo, si è controllato che la forza dei giocatori (dedotta dal rispettivo punteggio ELO), particolare

rilevante per l'algoritmo predittivo, fosse varia, andando da dilettanti a giocatori ai vertici mondiali.

I dati più salienti, relativi al fattore di compressione dell'algoritmo, sono illustrati nella tabella 4.1. Si è indicato con metodi 1, 2 e 3 rispettivamente il metodo compatto per corrispondenza, il metodo delle mosse possibili e infine quello predittivo probabilistico. Una prima osservazione sul campione di partite su cui è stata fatta la sperimentazione è che il numero medio di mosse per partita (si ricordi che per mossa, come già spiegato, si intende la mossa di un solo giocatore) è leggermente inferiore alla media generale Questo leggero sfasamento comunque non incide in alcun modo sui risultati finali di compressione, riuscendo l'algoritmo a essere in egual modo efficiente nelle fasi iniziali di partita come in quelle finali, per cui il fatto di avere tante partite di poche mosse o poche partite lunghe non provoca differenze rilevanti.

Partite considerate	84
numero mosse totali	4932
numero medio mosse per partita	58,7
numero bit necessari con metodo 1	59184
numero bit necessari con metodo 2	26283
numero bit necessari con metodo 3	14643
Rapporto di compressione tra i metodi 3 e 2	55,7%

tabella 4.1: dati numerici sui confronti di compressione tra i vari metodi

Il risultato finale tra il miglior metodo utilizzato ad oggi e quello proposto con questa tesi è una compressione pari al 55,7% a favore di quest'ultimo. Tale risultato comprova l'ipotesi da cui si era partiti, cioè che

i legami logici esistenti nella serie di mosse che formano una partita di scacchi siano abbastanza forti da permettere, se sfruttati con un metodo adeguato, una notevole riduzione della quantità di informazione necessaria a trasmettere o memorizzare la partita, ottenendo una buona compressione.

4.3.2 Analisi di alcuni dati statistici

Altri dati interessanti da analizzare sono i valori medi in bit della codifica di una partita ovvero di una singola mossa. Le proporzioni tra i tre metodi rimangano ovviamente le stesse illustrate nei precedenti paragrafi, ma tali valori numerici riescono a fornire un'idea più precisa anche su fino a che punto si è arrivati a compattare l'informazione trasmessa.

	Num medio bit per mossa
Metodo 1	12
Metodo 2	5,33
Metodo 3	2,97

tabella 4.2: numero medio di bit richiesti per la codifica di una mossa

Nella tabella 4.2 sono indicati i valori in bit necessari in media per memorizzare una singola mossa. Come si vede, sono valori piuttosto bassi, e in particolare il terzo metodo porta ad una codifica media inferiore ai 3 bit per mossa. Si deduce quindi che l'Entropia che la conoscenza della mossa porta con se, cioè l'informazione trasmessa in media dalla comunicazione della mossa eseguita, sia sotto i 3 bit, e, certamente, ancora inferiore al valore ottenuto con il terzo metodo.

I prossimi valori illustrano invece di quanto variano le percentuali di compressione da una partita all'altra, tramite l'analisi dello scarto quadratico medio.

	Num medio bit per partita	Scarto quad. medio
Metodo 1	704,6	± 0,0 %
Metodo 2	312,9	± 3,5 %
Metodo 3	174,3	± 9,1 %

tabella 4.3: dati numerici sulle medie di compressione di singole partite

Lo scarto quadratico medio, qui espresso come percentuale del valore medio, considerato in proporzione al numero di mosse della partita in esame, dà un'idea di quanto si possa discostare dal numero medio di bit per partita un singolo campione di partita osservato. In particolare, nel primo caso si ha una variazione nulla, in quanto il valore per mossa, e quindi per partita con un numero medio di mosse, è costante e uguale a 12 bit, come già illustrato quando si è definita la codifica compatta per corrispondenza (vedi 2.2.1). Nel secondo caso la variazione è normale, mentre nel terzo caso si nota un'ampiezza dell'intervallo piuttosto elevata, in quanto, come è già stato detto, le prestazioni dell'algoritmo possono essere incostanti.

4.3.3 Analisi delle previsioni

Analizziamo ora in che misura le previsioni effettuate dall'algoritmo, come diretta conseguenza della valutazione del programma di gioco, siano attendibili. Nella tabella 4.4 è mostrata la frequenza con cui le previsioni

fatte da Crafty ed elaborate poi dal nostro algoritmo fanno centro. La prima cosa che si nota osservando la figura è il salto di percentuali tra la prima mossa ritenuta più probabile e le altre.

Riscontro delle previsioni	Frequenza
Prima mossa ritenuta più probabile	55 %
Seconda mossa ritenuta più probabile	14 %
Terza mossa ritenuta più probabile	10 %
Fuori dalle prime 3	21 %

tabella 4.4: dati numerici sull'attendibilità delle previsioni effettuate

Ciò è dovuto al fatto che nel corso della partita capitano con una certa frequenza posizioni in cui una sola è la mossa scacchisticamente 'possibile', come ad esempio fasi intermedie di scambi di materiale (cioè quando si cattura un pezzo avversario, con un pezzo di valore corrispondente, sapendo di essere subito ricatturati). In questi casi sarà facile che l'algoritmo e il giocatore arrivino alla stessa conclusione scegliendo la mossa 'ovvia'. La frequenza relativamente alta in cui casi come questo si verificano, è sicuramente uno dei fattori che porta alla buona compressione ottenuta dall'algoritmo, poichè molto probabilmente la percentuale assegnata alla mossa ritenuta 'unica' da parte del programma sarà superiore al 50%, con la conseguenza che l'applicazione dell'algoritmo di Huffman produrrà una codifica lunga un solo bit per questa mossa, in modo da ottenere la trasmissione con la minima unità di codice.

Capitolo 5

Conclusioni

In questo capitolo si riepilogherà quanto si è detto in quelli precedenti, riassumendo il processo che ha portato alla creazione dell'algoritmo di compressione dalle premesse teoriche che ne stanno alla base fino alle scelte implementative e ai risultati sperimentali che si sono ottenuti, illustrandone pregi, difetti e possibilità di miglioramento.

5.1 Limiti dell'applicazione e sviluppi futuri

Gli sviluppi futuri del lavoro svolto in questa tesi possono andare in due direzioni: da una parte l'ulteriore ottimizzazione dell'algoritmo esistente, la quale si può ottenere sia cercando di adottare una migliore distribuzione di probabilità (vedi 5.1.1), sia provando a ridurre al minimo la pesantezza computazionale (vedi 5.1.2), sia con l'utilizzo della codifica a mossa composta descritta nel capitolo di introduzione al progetto (vedi 2.2.4).

D'altra parte, si può sfruttare il procedimento utilizzato nell'applicazione in altri campi (vedi 5.1.3). Il parallelo è immediato per quanto riguarda la memorizzazione legata ad altri giochi da tavolo, anche se

gli scacchi ad oggi sono l'unico a conoscere un così diffuso utilizzo di database di partite. In modo del tutto analogo, poi, si può riutilizzare questo metodo in campi di memorizzazione discreta di dati in cui sia possibile impostare un algoritmo capace di effettuare previsioni probabilistiche di una certa attendibilità.

5.1.1 Difficoltà di una distribuzione di probabilità verosimile

Un punto importante per il progetto di questa tesi e per i risultati dell'algoritmo di compressione è il passaggio tra l'insieme dei risultati numerici di un gruppo di mosse e la corrispondente distribuzione di probabilità. Si è cercato di semplificare i gradi di libertà di questa funzione che prende valori e restituisce probabilità con l'utilizzo di una singola variabile k, definita come il rapporto di probabilità tra due mosse che differiscono, nel valore, di una unità. Si è cercato il valore più efficace per questa variabile e si è utilizzato nel processo di codifica e decodifica. Questo non significa che la distribuzione di probabilità non possa essere migliorata, magari con una mappatura differente rispetto ai valori numerici ottenuti, ma sostanzialmente si ritiene che le ipotesi che hanno portato alla creazione del metodo utilizzato in questa tesi siano corrette e gli eventuali miglioramenti apportabili non determinanti dal punto di vista dei risultati finali.

5.1.2 Complessità elevata dell'algoritmo

In termini di prestazioni, sono diversi i fattori che portano pesantezza computazionale all'algoritmo presentato con questa tesi, alcuni eliminabili, altri impliciti nel processo di compressione. La parte più complessa e lenta del procedimento è costituita dal programma di gioco, cioè, dal punto di vista funzionale, da quella parte del metodo che prende in esame una posizione e restituisce una valutazione numerica. Ciò è dovuto principalmente al fatto che il programma di gioco utilizzato è esterno e non facilmente adattabile ai propri fini. Ad esempio sarebbe utile limitare la fase di inizializzazione, con la quale si predispone il programma allo svolgimento di intere partite, piuttosto che alla richiesta di valutazione di una singola posizione, al minimo indispensabile, risparmiando cos' un notevole tempo di esecuzione. Questo si può risolvere non utilizzando un programma di gioco esterno come per semplicità si è scelto qui, ma riprogettando questa unità in modo da inserirla all'interno all'applicazione. Una volta ridotta la pesantezza computazionale e quindi il tempo di esecuzione dell'algoritmo, si può procedere anche ad una codifica a mossa composta; si possono cioè codificare, al posto delle singole mosse, gruppetti di due o tre mosse. Questo minimizza gli sprechi di spazio e dovrebbe portare ad una ulteriore riduzione del fattore di compressione. Il prezzo che si paga computazionalmente per questo passaggio, tuttavia, è pesante. Per quanto riguarda il numero di posizioni da analizzare da parte del programma di gioco, infatti, questo varia esponenzialmente con la cardinalità dei gruppetti di mosse considerati. Per dare un'idea, ipotizzando, per posizione, un numero di mosse possibili medio di circa 40, rispetto a

considerare mosse singole, gruppi di 2 mosse appesantirebbero di 40 volte il tempo di esecuzione, gruppi di 3 mosse di 1600 volte. Sostanzialmente è questo il motivo che ha portato alla scelta forzata di considerare mosse singole in questo progetto.

5.1.3 Riusabilità del metodo alla base del progetto

Lo stesso procedimento utilizzato in questo caso per la compressione di partite di scacchi, può analogamente essere utilizzato per la compressione di qualsiasi gioco di cui sia possibile progettare un giocatore artificiale, e, in generale, si può riutilizzare in ogni campo in cui, basandosi su proprietà logiche della sorgente, un programma deterministico possa fornire una previsione della successiva mossa, stato, o azione relativamente di frequente [Alt90].

5.1.4 Codifica di gruppi di partite

L'utilizzo pratico della trascrizione di partite di scacchi, in ambito informatico, è legato da una parte allo studio del gioco da parte dell'utente, con particolare attenzione per le fasi iniziali di una partita, dall'altra alla possibilità di perfezionare algoritmi di gioco, basati su reti neurali o comunque su metodi che prevedano una fase di apprendimento; in entrambi i casi è utile avere, piuttosto che singole partite, archivi contenenti un certo numero di partite. Naturalmente, anche per codificare un archivio, è possibile semplicemente applicare l'algoritmo presentato in questa tesi per ogni partita; tuttavia, in questa circostanza, sono possibili ulteriori

miglioramenti. Innanzitutto in modo praticamente indipendente dall'algoritmo predittivo è possibile sfruttare la proprietà di ridondanza di grossi insiemi di partite, in particolare nelle fasi iniziali. È noto infatti che le prime mosse di ogni partita (in numero che varia grossomodo da 10 fino a 30 o 40) ricalcano schemi noti e abituali, consentendo quindi una notevole riduzione dell'*informazione* da trasmettere comunicando semplicemente fino a che punto la partita abbia ricalcato uno degli schemi preventivamente memorizzati e quale esso sia. In secondo luogo, con riferimento questa volta diretto all'algoritmo predittivo, è possibile calcolare, sull'intero archivio o su un campione di esso, quale k (fattore di confidenza con le previsioni effettuate dal programma di gioco) sia più efficace per la compressione e venga incluso in un header all'inizio dell'archivio il valore di k così stabilito.

5.2 Conclusioni

5.2.1 Panoramica

Il recente sviluppo informatico in campo scacchistico ha portato un diffuso utilizzo di database di partite in questo settore. Si è affermato un formato standard non compresso, il PGN (vedi paragrafo 2.1.2), e numerosi formati compressi ideati dagli sviluppatori di database scacchistici. In questa tesi si è provato a costruire un nuovo algoritmo di compressione basato sullo sfruttamento dei forti legami logici presenti all'interno di una

partita di scacchi tramite i metodi forniti dalla teoria dell'informazione e lo si è confrontato con gli algoritmi già esistenti.

5.2.2 Ipotesi di partenza

Analizzando i metodi adottati in precedenza per la rappresentazione compatta di una partita di scacchi, si è notata la mancanza di uno sfruttamento adeguato di alcune proprietà logiche, non deterministiche ma probabilistiche in essa presenti, utilizzabili al fine di ottenere una ulteriore compressione. L'ipotesi da cui si è partiti, in sostanza, è che una mossa, migliore è dal punto di vista scacchistico, maggiore probabilità avrà di essere giocata. Questa informazione aggiuntiva, tenendo conto delle possibilità di errore sia dei giocatori, sia della valutazione effettuata in sede di codifica, sarà sfruttata probabilisticamente, come probabilisticamente saranno valutati i risultati della compressione a cui essa porta.

5.2.3 L'algoritmo di compressione

L'algoritmo di compressione sviluppato all'interno di questa tesi (vedi paragrafo 3.1.3) si può schematizzare come segue: prima di conoscere l'effettiva mossa eseguita in una data posizione (la prima mossa sarà attesa partendo dalla posizione iniziale), il programma calcola tutte le mosse possibili, le numera univocamente, quindi richiede al "motore di gioco", un programma esterno che gioca a scacchi, una valutazione numerica per ognuna di queste mosse. Dalla valutazione numerica deduce quindi un valore probabilistico che assegna ad ogni mossa come 'la probabilità di essere giocata'. A questo punto, tramite il metodo di Huffman, assegna,

sempre univocamente, un codice binario a ogni mossa e trasmette quello corrispondente alla mossa effettivamente giocata. In questo modo, ad ogni mossa in input, emette una sequenza di bit in output che un programma di decodifica, strutturato in modo del tutto analogo a quello di codifica, potrà decifrare come la mossa originale.

5.2.4 Compendio dei risultati

Il risultato del confronto tra l'algoritmo *predittivo probabilistico* sviluppato in questa tesi e l'algoritmo delle *mosse possibili*, il più competitivo tra quelli già esistenti, porta ad una percentuale di occupazione di spazio inferiore, in misura del 55.7%, del primo rispetto al secondo.

5.2.5 Conclusione

I risultati delle prove sperimentali hanno dimostrato la maggiore efficienza del metodo sviluppato in questa tesi rispetto a quelli già esistenti. Inoltre il percorso di sviluppo, basato sulla teoria dell'informazione, che ha portato alla costruzione dell'algoritmo, ci fa credere che il risultato che si è ottenuto sia tanto vicino al limite teorico dell'entropia (vedi paragrafo 1.4.2), da poter essere considerato ottimale. Si può concludere quindi che un'ulteriore compressione, se non con una ottimizzazione del metodo proposto, sia difficilmente ottenibile.

Bibliografia

[Ges91] Giuseppe Gestri, *Teoria dell'Informazione*, Pisa, ETS, 1991.

[Alt90] Ingo Althöfer *Compressing chess games with the help of a fast deterministic chess program*, Icca Journal, December 1990.

[Cap73] Adolivio Capece, *Storia degli scacchi*, Milano, De Vecchi, 1973

[LelHir98] Debra A. Lelewer and Daniel S. Hirschberg, *Data Compression*, 1998

[Sei97] Jerzy A. Seidler, *Information systems and data compression*, Dordrecht, Kluwer Academic Publishers, 1997

[HanHarJoh98] Darrel R. Hankersonh, Greg A. Harris, Peter D. jun. Johnson, *Introduction to information theory and data compression*, CRC Press Series on Discrete Mathemathics and its applications, 1998.

[Lau94] Leonard Laub, *Data compression*, Heidelerg, 1994.

[OrnWei93] Donald Samuel Ornstein, Benjamin Weiss, *Entropy and data compression schemes*, IEEE Trans. Inf. Theory No.1, 1993

[MarSch90] T. Anthony Marsland, Jonathan Schaeffer, *Computers, chess and cognition*, New York, Springer-Verlag. XII, 1990

[NewMon] Newborn, Monroe, *Computer chess*, ACM Monograph series, 1975.

[Fre83] Peter W. Frey, *Chess Skill in man and machine*, New York, Springer-Verlag. XII, 1983

[BelJac79] A. G. Bell, N. Jacobi, *How to Read, make and store chess moves*, Computer J. 22, 1979