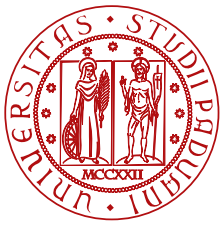


UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA DELL'INFORMAZIONE

Analisi della codifica aritmetica e di una sua implementazione

Relatore
Prof. Giancarlo Calvagno

Laureando
Filippo Bosello

ANNO ACCADEMICO 2023-2024

Data di laurea 17/07/2024

Indice

Capitolo 1

Introduzione

La compressione ha un ruolo fondamentale nella comunicazione e nello scambio di dati. Senza di essa non sarebbe possibile l'ampia fruizione di contenuti multimediali che osserviamo oggi, dovuta, ad esempio, ai social network, ma anche allo streaming di musica e video.

L'obiettivo di questa tesi è analizzare uno dei metodi di compressione senza perdite, la codifica aritmetica, e una sua specializzazione, CABAC. Quest'ultimo è utilizzato, in aggiunta ad altri metodi, per migliorare le prestazioni complessive della compressione di segnali video.

1.1 Compressione senza perdite o con perdite

I metodi di compressione sono suddivisi in due macro categorie: senza perdite e con perdite. La prima consiste nel ridurre la quantità di dati necessari per la memorizzazione o trasmissione senza, però, modificare il messaggio originale, ovvero senza perdita di informazione. Questa tecnica è necessaria e si applica a qualunque situazione in cui una piccola variazione del messaggio originale non sia accettabile, ad esempio: con file di testo, in cui l'inversione di un singolo bit comporta la decodifica incorretta del carattere; o con file contenenti codice macchina, in cui la differenza può cambiare il codice e trasformare un'istruzione in un'altra. La codifica con perdite ammette, invece, una parziale degradazione del messaggio in cambio di un sostanziale miglioramento delle prestazioni di compressione. Essa è usata, ad esempio, per file multimediali, come audio, immagini e video, in cui una piccola differenza nei valori di pressione sonora o di intensità luminosa risultano quasi impercettibili dall'utente finale.

1.2 Cos'è l'informazione

Nel secolo scorso è stata sviluppata la teoria dell'informazione con lo scopo di poter quantificare e trattare matematicamente l'informazione. Uno dei primi a lavorarci e ad accrescere l'interesse verso questo campo fu Claude Shannon che propose di definire l'informazione associata ad un evento aleatorio come funzione della sua probabilità.

La funzione informazione i, per essere tale, deve avere le seguenti proprietà:

- deve essere funzione strettamente decrescente della probabilità;
- deve risultare nulla per l'evento certo;
- l'informazione dell'unione di due eventi indipendenti deve essere uguale alla somma dell'informazione dei due eventi singoli.

Le prime due proprietà possono essere spiegate dal fatto che l'avvenimento dell'evento certo non aggiunge nulla alla nostra conoscenza del sistema, mentre eventi improbabili sono molto più informativi di eventi altamente probabili.

La funzione matematica che rispetta questi requisiti è il logaritmo

$$i(A) = \log_b \frac{1}{P(A)}$$

infatti

$$P(A) > P(B) \rightarrow i(A) = \log_b \frac{1}{P(A)} < \log_b \frac{1}{P(B)} = i(B)$$

$$P(A) = 1 \rightarrow i(A) = \log_b \frac{1}{P(A)} = 0.$$

Se A e B sono indipendenti

$$i(A \cap B) = \log_b \frac{1}{P(A \cap B)} = \log_b \frac{1}{P(A)} \frac{1}{P(B)} = \log_b \frac{1}{P(A)} + \log_b \frac{1}{P(B)} = i(A) + i(B).$$

La scelta della base non è significativa, purché sia maggiore di 1 in modo da garantire la decrescenza del logaritmo. Lavorando con sistemi digitali binari la scelta diventa naturalmente $b = 2$.

Se ora prendiamo una variabile aleatoria X, di alfabeto $\mathcal{A}_X$ e probabilità $p_X(x)$, possiamo definire l'entropia ad essa associata come il valore medio della funzione informazione nel suo alfabeto

$$H(X) = \mathbb{E}\left[i(X)\right] = \sum_{x \in \mathcal{A}_X} p_X(x) \log_2 \frac{1}{p_X(x)}.$$

L'entropia fornisce la misura di quanto è mediamente informativa una variabile aleatoria. Questa nozione può essere estesa anche a vettori aleatori.

Definendo il vettore $\boldsymbol{X} = (X_1, X_2, ..., X_n)$, di alfabeto $\mathcal{A}_{\boldsymbol{X}}$ e probabilità $p_{\boldsymbol{X}}(\boldsymbol{x})$, la sua entropia può essere calcolata come segue

$$H(\boldsymbol{X}) = \sum_{\boldsymbol{x} \in \mathcal{A}_{\boldsymbol{X}}} p_{\boldsymbol{X}}(\boldsymbol{x}) \log_2 \frac{1}{p_{\boldsymbol{X}}(\boldsymbol{x})}.$$

Per i vettori si può anche definire l'entropia media per simbolo come

$$H_n(\boldsymbol{X}) = \frac{1}{n} \sum_{\boldsymbol{x} \in \mathcal{A}_{\boldsymbol{X}}} p_{\boldsymbol{X}}(\boldsymbol{x}) \log_2 \frac{1}{p_{\boldsymbol{X}}(\boldsymbol{x})}.$$

Estendendo ulteriormente la notazione ai processi stocastici, sia X_t un processo aleatorio, definiamo il vettore $\boldsymbol{X}_n = (X_{-n}, ..., X_n)$, l'entropia media per simbolo del processo è definita come

$$H(X_t) = \lim_{n \to \infty} \frac{1}{2n+1} H(\boldsymbol{X}_n).$$

Nel caso in cui le variabili del processo siano indipendenti e identicamente distribuite, l'entropia media per simbolo è uguale all'entropia di una singola variabile; infatti

$$H(\boldsymbol{X}_n) = \mathbb{E}\left[i(X_{-n}, ..., X_n)\right] = \mathbb{E}\left[\sum_{k=-n}^{n} i(X_k)\right] = \sum_{k=-n}^{n} \mathbb{E}\left[i(X_k)\right]$$

e poiché $H(X_t) = H(X_0)$

$$H(\boldsymbol{X}_n) = (2n+1) H(X_0).$$

Nella pratica questa assunzione non è quasi mai verificata, ma può fornire una prima approssimazione nel caso la statistica del processo non sia nota.

Definiamo una sorgente di informazione come un qualunque processo stocastico a tempo discreto e con un alfabeto numerabile.

1.3 Codici e decodificabilità

Un codice può essere definito come una mappa che assegna una sequenza binaria ad ogni simbolo dell'alfabeto di una sorgente d'informazione. Ciascuna sequenza binaria appartenente al codice è chiamata parola e le parole di codice possono essere tutte della stessa lunghezza o avere lunghezze diverse. Poter usare lunghezze diverse per simboli diversi permette di ridurre la lunghezza media del messaggio; infatti, se scegliamo un codice che assegna parole brevi a sim-

boli molto frequenti e lascia quelle lunghe ai simboli meno probabili, possiamo intuire che la lunghezza complessiva del messaggio si riduce.

Ad esempio, consideriamo la seguente sequenza:

111311121211

assegnando a ciascun simbolo lo stesso numero di bit, 2 bit, avremmo una lunghezza codificata di 24 bit, mentre decidendo di assegnare un solo bit al simbolo 1, si può ridurre il messaggio a 15 bit totali.

Un codice per essere utile, oltre ad essere corto, deve anche essere univocamente decodificabile, altrimenti il messaggio originale non può essere ricostruito con certezza. Questa proprietà è necessaria, in quanto le parole binarie vengono trasmesse in modo contiguo come sequenze di bit e non c'è modo di distinguere dove una inizia e l'altra termina. La creazione di un codice univocamente decodificabile è tutt'altro che banale; infatti, per dire con certezza che un codice rispetta questa proprietà bisogna controllare che ogni possibile sequenza di bit possa essere ricondotta al messaggio originale. Inoltre, per alcuni codici, decodificare un simbolo può richiedere di dover proseguire la lettura della sequenza anche oltre i bit che codificano quello specifico simbolo prima di poterlo dedurre. Ai fini pratici queste osservazioni si traducono in una maggiore complessità di implementazione e di analisi non trascurabili.

1.4 Codici a prefisso

Per ovviare ai problemi appena discussi è stata ampiamente studiata una specifica classe di codici chiamati a prefisso. La peculiarità di questi codici è che sono creati in modo tale per cui nessuna parola sia prefisso di un'altra. Questi codici sono semplici da decodificare e garantiscono che la decodifica di un simbolo sia istantanea; ovvero, appena letti i bit che formano la parola di codice, il ricevitore può immediatamente prendere la decisione corretta su che simbolo è stato inviato. Ciò è intuibile in quanto, non appena viene trovata una successione di bit che corrisponde a una parola di codice, sappiamo che essa non può essere il prefisso di nessun'altra parola, di conseguenza la parola deve sicuramente terminare in quel punto e può, quindi, essere decodificata con certezza. Anche la costruzione del codice è semplificata, dato che esso può essere rappresentato da un albero binario in cui ad ogni foglia è associata una parola di codice i cui bit si trovano nei rami che vengono percorsi dalla radice verso la foglia. Infatti percorrendo l'albero dalla radice si compongono delle sequenze di bit tali che, fermandosi ad ogni nodo interno, esse formano un prefisso di qualche parola di codice, in quanto a quella sequenza verranno aggiunti altri bit proseguendo verso le foglie, mentre arrivati alle foglie la sequenza non può proseguire oltre rendendola adatta a formare una parola di codice valida.

Riproponendo l'esempio visto in precedenza, un possibile codice a prefisso con il relativo albero è riportato in Figura 1.1.

Codice	
Simbolo	Parola
1	1
2	01
3	00

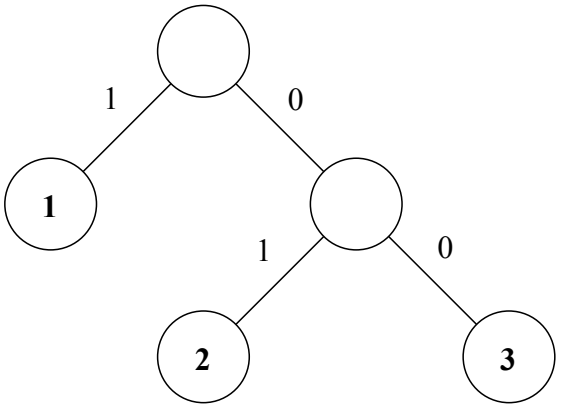


Figura 1.1: Esempio di codice a prefisso.

Uno potrebbe ora chiedersi se, però, restringendo l'attenzione a questa specifica classe di codici si rischia di ignorare codici che forniscano una maggiore efficienza. Può essere dimostrato che per ogni codice non a prefisso esiste un codice a prefisso con le stesse lunghezze di parola; perciò, dati i notevoli vantaggi di questo tipo di codifica, non ha senso investigare oltre altre classi di codici.

Capitolo 2

Codifica aritmetica

La codifica aritmetica, a differenza di metodi precedentemente proposti, non va a codificare una sequenza simbolo per simbolo, ma si pone come obiettivo quello di codificare la sequenza nel suo insieme. Metodi come la codifica di Huffman, che comprimono ogni simbolo in modo indipendente, hanno l'implicito svantaggio di non poter usare meno di un bit per definire una parola di codice, di conseguenza la lunghezza media dei messaggi non può scendere sotto tale soglia; alcuni metodi sono stati proposti per aggirare il problema, ma questi diventano poco pratici quando la cardinalità dell'alfabeto è grande e le probabilità sono molto sbilanciate.

Per codificare ogni possibile sequenza di simboli, data una certa distribuzione di probabilità, dobbiamo poter assegnare loro un identificatore univoco ed, essendo un numero infinito di sequenze, abbiamo bisogno di un numero infinito di identificatori. La soluzione adottata è stata quella di utilizzare l'intervallo $[0,1)$ dei numeri reali. Ora dobbiamo, però, mappare il messaggio all'intervallo unitario, a tale scopo utilizzeremo la funzione di distribuzione associata alla sorgente del messaggio.

2.1 Generazione dell'identificatore

Il metodo di generazione dell'identificatore consiste nel ridurre l'intervallo in cui esso è compreso all'arrivo di più e più simboli da codificare.

Consideriamo il seguente alfabeto della variabile aleatoria X

$$\mathcal{A}_X = \{a_1, a_2, ..., a_m\}, \quad |\mathcal{A}_X| = m$$

e la sua funzione di distribuzione, FdD,

$$F_X(i) = \sum_{k=1}^{i} p_X(a_k).$$

Possiamo, ora, suddividere l'intervallo unitario in sotto-intervalli del tipo $[F_X(i-1), F_X(i))$, $i = 1, ..., m$. Essendo, la FdD, crescente e avendo come minimo zero e massimo uno, gli intervalli considerati formano una partizione dell'intervallo unitario. Associamo ciascun intervallo al simbolo i-esimo dell'alfabeto. La presenza del simbolo i come primo elemento della sequenza costringe la posizione dell'identificatore ad essere all'interno dell'intervallo i-esimo. Si può, quindi, partizionare ulteriormente questo intervallo usando nuovamente la funzione di distribuzione, con un fattore di scala e traslazione, nel seguente modo:

$$[F_X(i-1) + F_X(j-1) \cdot p_X(a_i), F_X(i-1) + F_X(j) \cdot p_X(a_i)), \quad j = 1, ..., m.$$

Il secondo simbolo ridurrà ancora lo spazio in cui si trova l'identificatore, costringendolo all'interno del suo intervallo.

Si può procedere ricorsivamente a partizionare gli intervalli per ogni nuovo simbolo in ingresso fino ad arrivare al termine del messaggio. Possiamo notare che gli intervalli associati a tutte le possibili sequenze di una data lunghezza formano a loro volta una partizione dell'intervallo unitario e sono, di conseguenza, disgiunti; quindi ogni valore compreso nell'intervallo definito dal messaggio può essere utilizzato come identificatore. I valori comunemente usati sono il limite inferiore o il punto medio.

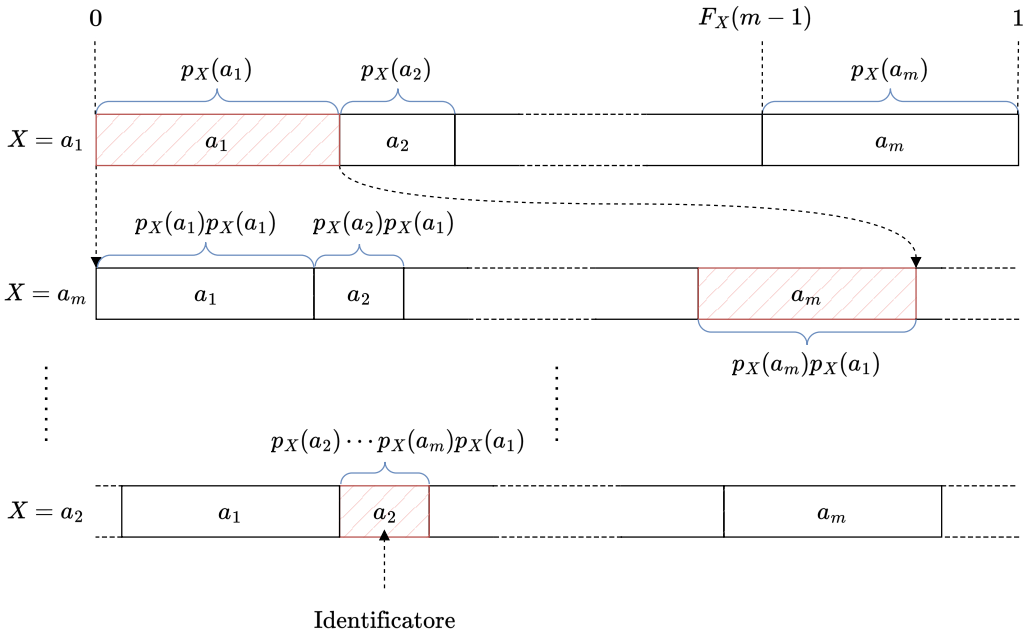


Figura 2.1: Dimostrazione grafica del procedimento di riduzione dell'intervallo

Per sequenze di lunghezza unitaria e utilizzando il valore medio, ad esempio, possiamo

definire la funzione che restituisce l'identificatore come

$$\begin{aligned}\overline{T}_X(a_i) &= \sum_{k=1}^{i-1} p_X(a_k) + \frac{1}{2}p_X(a_i) \\ &= F_X(i-1) + \frac{1}{2}p_X(a_i).\end{aligned} \tag{2.1}$$

Possiamo estendere la notazione a sequenze di lunghezza arbitraria, imponendo un ordinamento, come può essere quello lessicografico,

$$\overline{T}_{\boldsymbol{X}}^{(m)}(\boldsymbol{x}_i) = \sum_{\boldsymbol{y}<\boldsymbol{x}_i} p_{\boldsymbol{X}}(\boldsymbol{y}) + \frac{1}{2}p_{\boldsymbol{X}}(\boldsymbol{x}_i) \tag{2.2}$$

in cui $\boldsymbol{x}_i, \boldsymbol{y}$ sono sequenze di lunghezza m.

Si può notare dall'equazione 2.2 che per calcolare l'identificatore è necessario conoscere la probabilità di ogni sequenza che viene prima di quella da codificare. Questo approccio, però, risulta irrealizzabile nella pratica a causa della crescita esponenziale del numero di probabilità da calcolare in funzione della lunghezza del messaggio. Possiamo, invece, trovare una definizione ricorsiva che permette di ricavare l'identificatore utilizzando solo le probabilità di un singolo simbolo. La definizione si basa sul metodo precedentemente utilizzato per restringere l'intervallo in cui si trova l'identificatore. Infatti, può essere dimostrato che per una sequenza $\boldsymbol{x} = (x_1 x_2 ... x_n)$, dove x_i è l'indice dell'i-esimo simbolo da inviare, i limiti superiori e inferiori dell'intervallo di appartenenza dell'identificatore sono rispettivamente

$$\begin{aligned}l^{(n)} &= l^{(n-1)} + (u^{(n-1)} - l^{(n-1)})F_X(x_n - 1) \\ u^{(n)} &= l^{(n-1)} + (u^{(n-1)} - l^{(n-1)})F_X(x_n)\end{aligned} \tag{2.3}$$

con $l^{(0)} = 0$, $u^{(0)} = 1$; da cui possiamo ricavare l'identificatore

$$\overline{T}_{\boldsymbol{X}}(\boldsymbol{x}) = \frac{u^{(n)} - l^{(n)}}{2}.$$

Pertanto, per ogni sequenza, possiamo calcolare l'identificatore in modo sequenziale e conoscendo solo la FdD della sorgente.

2.2 Decodifica dell'identificatore

Come precedentemente detto, un codice, per essere utile, deve essere decodificabile e, nel caso della codifica aritmetica, il metodo per recuperare il messaggio originale dall'identificatore è del tutto analogo al procedimento di codifica.

Partendo dall'intervallo unitario, questo viene partizionato nei sotto-intervalli corrispondenti a ciascun simbolo utilizzando la FdD. Uno degli intervalli deve necessariamente contenere l'identificatore, se quest'ultimo si trova nell'intervallo i-esimo allora il primo simbolo decodificato è a_i. Risolto, ora, il primo simbolo, si procede col successivo dividendo l'intervallo corrente con la FdD allo stesso modo visto per la codifica. Fatto ciò, si sceglie ancora una volta l'intervallo, e conseguentemente il simbolo, in cui l'identificatore è compreso e si continua in questo modo per il resto del messaggio. Si può notare, però, che utilizzando questo metodo, la decodifica continuerebbe senza mai terminare, in quanto l'intervallo può essere ristretto indefinitamente; è necessario, quindi, utilizzare un metodo che permetta al ricevente di capire quando ha letto tutti i simboli. Un modo consiste nell'inviare il numero di simboli del messaggio insieme all'identificatore, oppure si può includere nell'alfabeto un simbolo di fine trasmissione che viene aggiunto al termine del messaggio per notificare il ricevente.

2.3 Unicità ed efficienza

Per poter implementare l'algoritmo appena descritto dobbiamo poter rappresentare l'identificatore in codice binario, ma essendo un numero reale, la sua rappresentazione binaria potrebbe avere un numero di cifre infinito, o comunque molto lungo. Dobbiamo, perciò, poter troncare la rappresentazione binaria in modo tale da mantenere l'unicità della codifica e allo stesso tempo usare il minor numero possibile di bit.

Indichiamo con $\lfloor \overline{T}_{\boldsymbol{X}}(\boldsymbol{x}) \rfloor_{l(\boldsymbol{x})}$ il troncamento di $\overline{T}_{\boldsymbol{X}}(\boldsymbol{x})$ a $l(\boldsymbol{x})$ bit dopo la virgola. Vogliamo trovare una funzione $l(\boldsymbol{x})$ tale che garantisca che $\lfloor \overline{T}_{\boldsymbol{X}}(\boldsymbol{x}) \rfloor_{l(\boldsymbol{x})}$ rappresenti ancora in modo univoco il messaggio $\boldsymbol{x}$, il che equivale a dire

$$\lfloor \overline{T}_{\boldsymbol{X}}(\boldsymbol{x}) \rfloor_{l(\boldsymbol{x})} \in [F_{\boldsymbol{X}}(\boldsymbol{x}-1), F_{\boldsymbol{X}}(\boldsymbol{x})).$$

Sappiamo che $\lfloor \overline{T}_{\boldsymbol{X}}(\boldsymbol{x}) \rfloor_{l(\boldsymbol{x})} < F_{\boldsymbol{X}}(\boldsymbol{x})$ perché $\overline{T}_{\boldsymbol{X}}(\boldsymbol{x})$ lo è; resta, quindi, da dimostrare che

$$\lfloor \overline{T}_{\boldsymbol{X}}(\boldsymbol{x}) \rfloor_{l(\boldsymbol{x})} \geq F_{\boldsymbol{X}}(\boldsymbol{x}-1).$$

Sappiamo, inoltre, che $\overline{T}_{\boldsymbol{X}}(\boldsymbol{x}) - F_{\boldsymbol{x}}(\boldsymbol{x}-1) = \frac{p_{\boldsymbol{X}}(\boldsymbol{x})}{2}$, pertanto vogliamo mostrare che

$$\overline{T}_{\boldsymbol{X}}(\boldsymbol{x}) - \lfloor \overline{T}_{\boldsymbol{X}}(\boldsymbol{x}) \rfloor_{l(\boldsymbol{x})} \leq \frac{p_{\boldsymbol{X}}(\boldsymbol{x})}{2}$$

Ora, per come è costruito $\lfloor \overline{T}_{\boldsymbol{X}}(\boldsymbol{x}) \rfloor_{l(\boldsymbol{x})}$, è vero che

$$\overline{T}_{\boldsymbol{X}}(\boldsymbol{x}) - \lfloor \overline{T}_{\boldsymbol{X}}(\boldsymbol{x}) \rfloor_{l(\boldsymbol{x})} < \frac{1}{2^{l(\boldsymbol{x})}}$$

Risolvendo ora per $l(\boldsymbol{x})$ e imponendo quanto segue

$$\frac{1}{2^{l(\boldsymbol{x})}} \leq \frac{p_{\boldsymbol{X}}(\boldsymbol{x})}{2}$$

$$\log_2 \frac{1}{2^{l(\boldsymbol{x})}} \leq \log_2 \frac{p_{\boldsymbol{X}}(\boldsymbol{x})}{2}$$

$$-l(\boldsymbol{x}) \leq \log_2 p_{\boldsymbol{X}}(\boldsymbol{x}) - 1$$

otteniamo

$$l(\boldsymbol{x}) \geq \log_2 \frac{1}{p_{\boldsymbol{X}}(\boldsymbol{x})} + 1$$

Non potendo usare lunghezze reali di bit, utilizziamo il primo numero intero maggiore o uguale al limite appena trovato

$$l(\boldsymbol{x}) = \left\lceil \log_2 \frac{1}{p_{\boldsymbol{X}}(\boldsymbol{x})} \right\rceil + 1$$

Pertanto, $\lfloor \overline{T}_{\boldsymbol{X}}(\boldsymbol{x}) \rfloor_{l(\boldsymbol{x})}$ rappresenta unicamente $\overline{T}_{\boldsymbol{X}}(\boldsymbol{x})$ e non può rappresentare l'identificatore di nessun altra sequenza.

Per dimostrare che il codice è anche univocamente decodificabile è sufficiente dimostrare che è un codice a prefisso. Dato un numero a nell'intervallo $[0,1)$ con una rappresentazione binaria $[b_1 b_2 ... b_n]$, per ogni altro numero b che abbia $[b_1 b_2 ... b_n]$ come prefisso, esso deve essere compreso in $[a, a + \frac{1}{2^n})$.

Se x e y sono due sequenze distinte, $\lfloor \overline{T}_{\boldsymbol{X}}(\boldsymbol{x}) \rfloor_{l(\boldsymbol{x})}$ e $\lfloor \overline{T}_{\boldsymbol{y}}(\boldsymbol{y}) \rfloor_{l(\boldsymbol{y})}$ devono appartenere agli intervalli disgiunti $[F_{\boldsymbol{X}}(\boldsymbol{x}-1), F_{\boldsymbol{X}}(\boldsymbol{x}))$ e $[F_{\boldsymbol{Y}}(\boldsymbol{y}-1), F_{\boldsymbol{Y}}(\boldsymbol{y}))$ rispettivamente; quindi, per dimostrare che la codifica di una sequenza non può essere il prefisso di un'altra, dobbiamo dimostrare che l'intervallo $[\lfloor \overline{T}_{\boldsymbol{X}}(\boldsymbol{x}) \rfloor_{l(\boldsymbol{x})}, \lfloor \overline{T}_{\boldsymbol{X}}(\boldsymbol{x}) \rfloor_{l(\boldsymbol{x})} + \frac{1}{2^{l(\boldsymbol{x})}})$ è interamente contenuto in $[F_{\boldsymbol{X}}(\boldsymbol{x}-1), F_{\boldsymbol{X}}(\boldsymbol{x}))$.

È già stato dimostrato che $\lfloor \overline{T}_{\boldsymbol{X}}(\boldsymbol{x}) \rfloor_{l(\boldsymbol{x})} > F_{\boldsymbol{X}}(\boldsymbol{x}-1)$, quindi resta da dimostrare che

$$F_{\boldsymbol{X}}(\boldsymbol{x}) \geq \lfloor \overline{T}_{\boldsymbol{X}}(\boldsymbol{x}) \rfloor_{l(\boldsymbol{x})} + \frac{1}{2^{l(\boldsymbol{x})}} \quad \rightarrow \quad F_{\boldsymbol{X}}(\boldsymbol{x}) - \lfloor \overline{T}_{\boldsymbol{X}}(\boldsymbol{x}) \rfloor_{l(\boldsymbol{x})} \geq \frac{1}{2^{l(\boldsymbol{x})}}$$

che è verificato da quanto segue

$$F_{\boldsymbol{X}}(\boldsymbol{x}) - \lfloor \overline{T}_{\boldsymbol{X}}(\boldsymbol{x}) \rfloor_{l(\boldsymbol{x})} \geq F_{\boldsymbol{X}}(\boldsymbol{x}) - \overline{T}_{\boldsymbol{X}}(\boldsymbol{x}) = \frac{p_{\boldsymbol{X}}(\boldsymbol{x})}{2} \geq \frac{1}{2^{l(\boldsymbol{x})}}.$$

Il che dimostra che il codice è a prefisso e, di conseguenza, univocamente decodificabile.

Cerchiamo adesso di capire quanto, la codifica aritmetica, riesce ad avvicinarsi all'entropia della sorgente. Ricordando che $l(\boldsymbol{x})$ è il numero di bit necessari per codificare l'intero messaggio

$\boldsymbol{x}$, la lunghezza media del codice per una sequenza di lunghezza m è la seguente

$$\begin{aligned} l_{A^{(m)}} &= \sum p_{\boldsymbol{X}}(\boldsymbol{x}) l(\boldsymbol{x}) \\ &= \sum p_{\boldsymbol{X}}(\boldsymbol{x}) \left[\left\lceil \log_2 \frac{1}{p_{\boldsymbol{X}}(\boldsymbol{x})} \right\rceil + 1 \right] \\ &< \sum p_{\boldsymbol{X}}(\boldsymbol{x}) \left[\log_2 \frac{1}{p_{\boldsymbol{X}}(\boldsymbol{x})} + 1 + 1 \right] \\ &= \sum p_{\boldsymbol{X}}(\boldsymbol{x}) \log_2 \frac{1}{p_{\boldsymbol{X}}(\boldsymbol{x})} + 2 \sum p_{\boldsymbol{X}}(\boldsymbol{x}) \\ &= H(X^m) + 2. \end{aligned}$$

Sapendo che la lunghezza media è sempre maggiore dell'entropia si ha

$$H(X^m) \leq l_{A^{(m)}} < H(X^m) + 2.$$

La lunghezza media per simbolo può essere calcolata dividendo per m

$$\frac{H(X^m)}{m} \leq l_A < \frac{H(X^m)}{m} + \frac{2}{m}$$

e nel caso di variabili indipendenti e identicamente distribuite, dove vale

$$H(X^m) = mH(X),$$

abbiamo

$$H(X) \leq l_A < H(X) + \frac{2}{m}.$$

Pertanto, aumentando la lunghezza della sequenza, possiamo avvicinarci in modo arbitrario all'entropia.

Nel caso la sorgente non generi simboli i.i.d. e avessimo più informazioni sulla sua statistica, ad esempio, la probabilità congiunta dei simboli, potremmo facilmente implementare queste informazioni nell'algoritmo per ottenere una maggiore efficienza di compressione. Di fatto, è sufficiente sostituire $p_X(x)$ con $p_{X|\boldsymbol{X}}(x|\boldsymbol{x})$ e di conseguenza $F_X(x)$ con $F_{X|\boldsymbol{X}}(x|\boldsymbol{x})$ ad ogni iterazione, dove x rappresenta il simbolo da codificare e $\boldsymbol{x}$ rappresenta la sequenza di simboli già codificati.

2.4 Nella pratica

Il metodo per eseguire la codifica aritmetica appena descritto è ottimo per caratterizzare le sue proprietà statistiche e trattarlo in modo matematico, ma nella realtà risulta impossibile da implementare per sequenze arbitrariamente lunghe, in quanto i dispositivi che utilizziamo non hanno precisione infinita e non possono, quindi, rappresentare tutti i possibili numeri nell'intervallo $[0,1)$. I limiti dell'intervallo che contiene l'identificatore, $u^{(n)}$ e $l^{(n)}$, si avvicinano sempre di più ad ogni iterazione e quando la loro differenza è minore della risoluzione della macchina non forniscono più un'informazione accurata dello stato del codificatore. È necessario quindi adottare delle tecniche che permettano di svolgere la codifica della sequenza mantenendo tutti i numeri rappresentati negli intervalli di operazione a disposizione. Un altro requisito importante nella pratica è avere la possibilità di eseguire la codifica in modo incrementale, ovvero, poter inviare parte della sequenza codificata durante il processo di codifica, senza dover prima attendere che esso sia terminato.

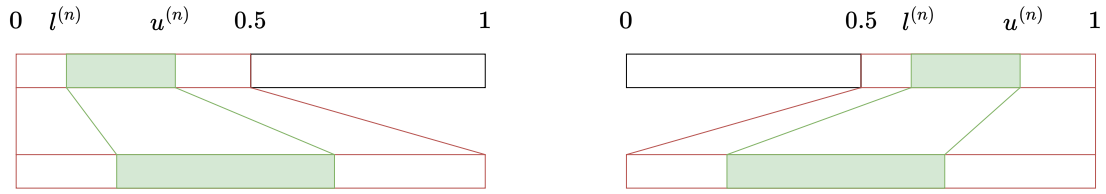
Possiamo osservare che al procedere della codifica, i limiti dell'intervallo possono rientrare in uno dei seguenti casi:

- l'intervallo è interamente contenuto metà inferiore dell'intervallo unitario, $[0,0.5)$;
- l'intervallo è interamente contenuto metà superiore dell'intervallo unitario, $[0.5,1)$;
- il punto medio dell'intervallo unitario, 0.5, è un punto interno all'intervallo.

Nei primi due casi sappiamo che l'identificatore deve essere confinato in uno dei due sottointervalli e che questo non può cambiare con l'arrivo di nuovi simboli da codificare; di conseguenza possiamo inviare il primo bit, che varrà 0 nel primo caso e 1 nel secondo. Questo ci permette successivamente di concentrarci solo su una delle due metà dell'intervallo unitario e scartare l'altra. Per risolvere, quindi, il problema della precisione finita possiamo ora prendere la metà di interesse ed espanderla a riempire l'intervallo unitario; mappando i valori dell'identificatore e degli estremi degli intervalli generati con la FdD nel seguente modo:

$$E_1 : [0,0.5) \to [0,1); \quad E_1(x) = 2x$$
$$E_2 : [0.5,1) \to [0,1); \quad E_2(x) = 2x - 1$$

In questo modo scartiamo l'informazione sul bit più significativo, che, però, non ha importanza dal momento che è già stato inviato (vedi Figura 2.2). Questo metodo, ovviamente, non si applica solo nel caso del primo bit, ma viene utilizzato ogni volta che una delle due condizioni si verifica nell'intervallo aggiornato.



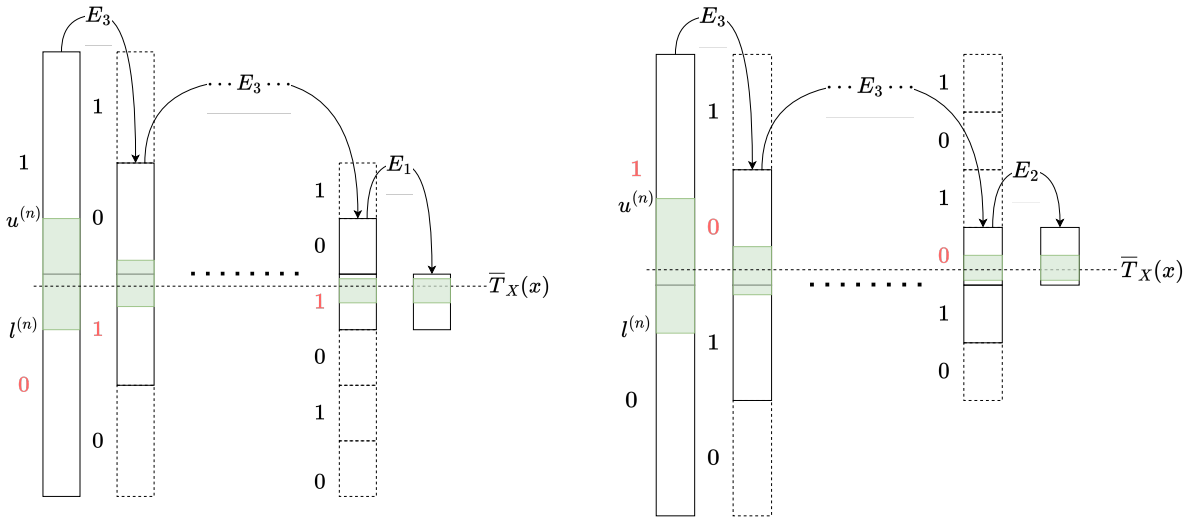
(a) Condizione E_1 (b) Condizione E_2

Figura 2.2: Esempio di mappatura dell'intervallo

Nel caso in cui l'intervallo continui a restringersi, ma rimanga sempre a cavallo del punto medio, le condizioni E_1 e E_2 non sono in grado di risolvere il problema; in questo caso va adottata una soluzione diversa. Quando l'intervallo è contenuto in $[0.25, 0.75)$ possiamo espanderlo in modo analogo alle altre due condizioni

$$E_3 : [0.25,0.75) \to [0,1); \quad E_3(x) = 2x - 0.5$$

in questo caso, però, non viene inviato nessun bit, ma viene memorizzato che è stata effettuata la mappatura. Questa condizione può verificarsi anche più volte consecutivamente, pertanto, viene memorizzato il numero di mappature. Non appena si verifica una delle altre due condizioni, E_1 o E_2, possiamo procedere a inviare il bit 0 per la prima o 1 per la seconda e successivamente inviamo lo stesso bit negato, per un numero di volte pari al numero di volte consecutive in cui si è verificata la condizione E_3, che è stato memorizzato precedentemente. Il procedimento è illustrato in Figura 2.3.



(a) Condizione E_3 seguita da E_1 (b) Condizione E_3 seguita da E_2

Figura 2.3: Esempio di mappatura dell'intervallo in condizione E_3

Per terminare la codifica, una volta che l'ultimo simbolo è stato processato e l'intervallo $[l^{(n)}, u^{(n)})$ è stato aggiornato, dobbiamo procedere a inviare gli ultimi bit dell'identificatore in modo da raggiungere la lunghezza sufficiente per renderlo univocamente decodificabile, $\left\lceil \log_2 \frac{1}{p_{\boldsymbol{X}}(\boldsymbol{x})} \right\rceil + 1$. Potrebbe sembrare di avere perso l'informazione relativa alla probabilità dell'intera sequenza eseguendo la codifica in modo incrementale, ma in realtà è sufficiente conoscere la dimensione dell'insieme contente l'identificatore per sapere quanti bit sono ancora necessari. Sappiamo, infatti, che la probabilità della sequenza corrisponde alla larghezza dell'intervallo che contiene l'identificatore, che a questo punto risulta

$$p_{\boldsymbol{X}}(\boldsymbol{x}) = \frac{(u^{(n)} - l^{(n)})}{2^k}$$

dove indichiamo con k il numero di volte in cui è stata effettuata una mappatura, che effettua un'espansione di un fattore due di ogni intervallo. Abbiamo, quindi, che

$$\begin{aligned} l(\boldsymbol{x}) &= \left\lceil \log_2 \frac{1}{p_{\boldsymbol{X}}(\boldsymbol{x})} \right\rceil + 1 \\ &= \left\lceil \log_2 \frac{2^k}{u^{(n)} - l^{(n)}} \right\rceil + 1 \\ &= \left\lceil \log_2 \frac{1}{u^{(n)} - l^{(n)}} \right\rceil + 1 + k \end{aligned}$$

e, avendo già inviato k bit, rimane il seguente numero di bit da inviare

$$l(\boldsymbol{x}) - k = \left\lceil \log_2 \frac{1}{u^{(n)} - l^{(n)}} \right\rceil + 1$$

presi dai bit più significativi dell'identificatore $\overline{T} = \frac{u^{(n)} - l^{(n)}}{2}$.

Dal lato del decodificatore il procedimento è quasi lo stesso, ma con alcune variazioni dovute al fatto che non possiamo leggere la sequenza per intero. Ad esempio, quando la decodifica viene iniziata, quanti bit della sequenza è necessario conoscere prima di poter definire qual'è il primo simbolo del messaggio? Ragionando al caso peggiore, ovvero quello in cui il primo simbolo sia quello meno probabile, sappiamo che, per codificare univocamente una sequenza di lunghezza unitaria contenente quel simbolo, abbiamo bisogno di $l(\boldsymbol{x})$ bit e, di conseguenza, prendendo in ingresso quel numero di bit ci permette di distinguere con certezza il primo simbolo. Questo ragionamento funziona correttamente per la prima iterazione, ma proseguendo nella decodifica gli intervalli e le probabilità vanno a restringersi rendendo non più valido il ragionamento appena fatto. Per capire, quindi, quanti bit sia effettivamente necessario conoscere, istante per istante, per eseguire la decodifica in modo corretto dobbiamo capire quanto gli intervalli possono restringersi. L'intervallo più piccolo che possiamo osservare, prima di rientrare

in una delle condizioni per cui avvenga una mappatura, è quello con $l^{(n)}$ poco minore di 0.25 e $u^{(n)}$ poco maggiore di 0.5, oppure $l^{(n)}$ poco minore di 0.5 e $u^{(n)}$ poco maggiore di 0.75; in entrambi i casi tutti gli intervalli risultano moltiplicati per un fattore maggiore di $\frac{1}{4}$. Ricordando che moltiplicare in binario per un fattore 2^{-2} corrisponde a traslare i bit a destra di due posizioni, per essere in grado di distinguere il più piccolo intervallo in ogni momento aggiungiamo due bit al valore precedentemente stimato:

$$m = l(\{a_{\text{min}}\}) + 2$$

indicando con $\{a_{\text{min}}\}$ la sequenza di lunghezza unitaria contenente il simbolo meno probabile.

Un'ultima considerazione di interesse pratico riguarda la scelta della rappresentazione dei valori binari utilizzati nell'algoritmo. Utilizzare la rappresentazione in virgola mobile risulta costoso da un punto di vista computazionale rispetto all'uso di numeri interi. Di fatto, non abbiamo bisogno della flessibilità fornita dalla virgola mobile, in quanto lavoriamo nell'intervallo unitario e ci interessano soltanto i primi bit dopo la virgola. Allora possiamo pensare di utilizzare i numeri interi considerando la virgola come posta a sinistra del bit più significativo.

Considerando un'implementazione a m bit, abbiamo che lo 0 è rappresentato come

$$\overbrace{00...0}^{m \text{ volte}}$$

Il valore 0.5 è rappresentato come

$$1 \overbrace{0...0}^{m-1 \text{ volte}}$$

Mentre il valore massimo rappresentabile è $1-2^{-m}$. Ora, il valore $u^{(0)}$ dovrebbe essere inizializzato a 1, che, però, non è rappresentabile nel sistema che abbiamo adottato e, conseguentemente, darebbe luogo ad un overflow. Quindi, decidiamo di non assegnare più a $u^{(n)}$ l'estremo superiore dell'intervallo che contiene l'identificatore, che è escluso dall'intervallo stesso essendo un estremo aperto, $[l^{(n)}, u^{(n)})$, ma andiamo ad assegnare ad esso il più grande valore rappresentabile che sia contenuto nell'intervallo, diventando quest'ultimo $[l^{(n)}, u^{(n)} + \epsilon)$, con $\epsilon = 2^{-m}$; ad esempio nella prima iterazione verrà inizializzato con tutti i bit uguali a 1.

La funzione di distribuzione viene ridefinita nel seguente modo

$$F_X(i) = \frac{\sum_{k=1}^{i} n_k}{N}$$

dove n_k è il numero di occorrenze del simbolo a_k e N è il numero totale di simboli presenti nel messaggio da codificare, quindi $\frac{n_k}{N}$ rappresenta una stima di $p_X(a_k)$.

Definiamo anche

$$C(i) = F_X(i) \cdot N = \sum_{k=1}^{i} n_k$$

Le equazioni per aggiornare l'intervallo diventano

$$\begin{aligned} l^{(n)} &= l^{(n-1)} + \left\lfloor \frac{(u^{(n-1)} - l^{(n-1)} + 1) \cdot C(x_n - 1)}{N} \right\rfloor \\ u^{(n)} &= l^{(n-1)} + \left\lfloor \frac{(u^{(n-1)} - l^{(n-1)} + 1) \cdot C(x_n)}{N} \right\rfloor - 1 \end{aligned} \tag{2.4}$$

dove l'addizione o sottrazione di uno è dovuta al fatto che abbiamo cambiato il significato di $u^{(n)}$.

Il controllo della presenza delle condizioni E_1 o E_2 si riduce a verificare se il bit più significativo di $l^{(n)}$ e $u^{(n)}$ coincide, in tal caso, si procede a trasmettere quel bit e a traslare gli altri a sinistra di una posizione, inserendo a destra uno 0 per $l^{(n)}$ e un 1 per $u^{(n)}$. L'Inserimento di 1 è nuovamente dovuto al fatto che $u^{(n)}$ rappresenta il più grande numero contenuto nell'intervallo ed è come se fossero presenti un numero infinito di 1 dopo i bit che possiamo rappresentare. Per controllare la presenza di E_3 andiamo a guardare il secondo bit più significativo, se per $l^{(n)}$ risulta 1 e per $u^{(n)}$ è uguale a 0, allora la condizione è verificata e procediamo traslando i bit a sinistra, complementiamo il bit più significativo, che corrisponde a sottrarre 0.5, e inserendo quello meno significativo come per le altre due condizioni.

2.5 Metodo adattivo

Abbiamo visto come utilizzare la codifica aritmetica per comprimere i dati generati da una sorgente d'informazione nel caso in cui la distribuzione di probabilità sia nota. Consideriamo, ora, il caso in cui non abbiamo alcun dettaglio riguardante la statistica della sorgente. L'algoritmo può essere semplicemente modificato per far sì che la distribuzione possa essere appresa durante il corso della codifica e aggiornata ogni volta che viene elaborato un nuovo simbolo.

Quando la statistica non è nota l'approssimazione più sensata è quella di considerare ogni simbolo equiprobabile. Partendo dalla distribuzione uniforme, ogni volta che riceviamo un simbolo possiamo aggiornare il modello in modo che la probabilità dell'ultimo simbolo sia aumentata. Sapendo che in media ci aspettiamo di osservare un dato simbolo in una sequenza un numero di volte proporzionale alla sua probabilità, è ragionevole stimare la probabilità come

$$p_X(a_i) = \frac{n_i}{N}$$

dove n_i è il numero di volte in cui abbiamo ricevuto il simbolo a_i e N è il totale di simboli codificati fino a quel momento.

Nell'algoritmo con rappresentazione binaria a interi possiamo inizializzare tutti gli $n_i = 1$ e $N = |\mathcal{A}|$ e, a ogni iterazione, incrementare n_i e N di uno, per i corrispondente al simbolo codificato, a_i. Va sottolineato, infine, che l'aggiornamento del modello va effettuato solo dopo la codifica del simbolo, altrimenti il ricevente non sarebbe in grado di decodificare correttamente la sequenza, avendo un modello non aggiornato delle probabilità.

Capitolo 3

CABAC

Context-Based Adaptive Binary Arithmetic Coding (CABAC) [1] è un metodo di compressione dati che si basa sulla codifica aritmetica, originariamente sviluppato come parte dello standard H.264/AVC e utilizzato anche per il suo successore, H.265/HEVC. I due standard sono utilizzati per la compressione video con perdite e sono la scelta principale in diversi ambiti, come il broadcast televisivo, la distribuzione di video e film su supporti fisici o in streaming, fino al comune utilizzo degli utenti finali per creare e memorizzare video privati.

3.1 Struttura

Prima di entrare nel dettaglio è utile capire in generale come CABAC è suddiviso e organizzato, in quanto non si limita solo ad eseguire la codifica aritmetica, ma cerca anche di stimare dei modelli di probabilità da utilizzare per codificare parti diverse della sorgente, chiamati contesti.

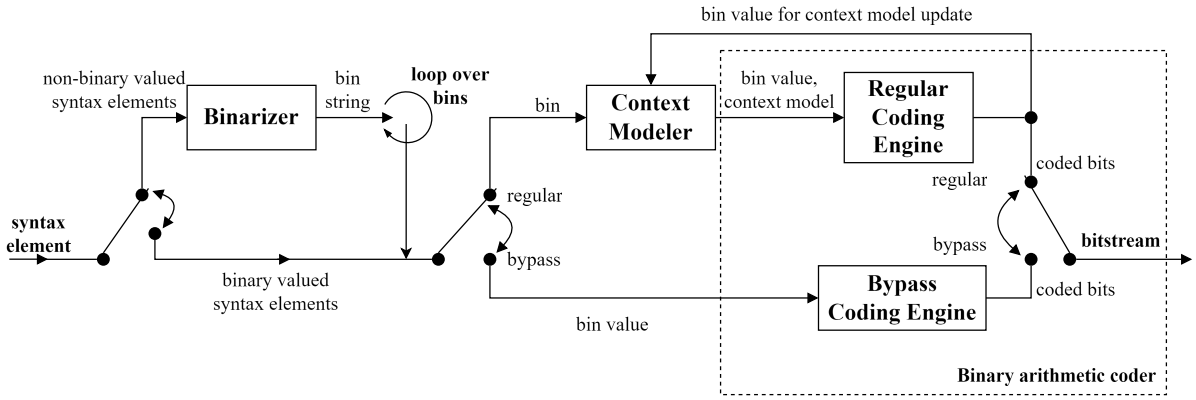


Figura 3.1: Schema a blocchi del codificatore CABAC

Binarizzatore

Il primo passaggio che viene eseguito è quello di prendere i valori in ingresso, che possono rappresentare elementi diversi, e assegnare loro un'adeguata rappresentazione binaria. Questo permette di ottenere una prima riduzione della lunghezza media. Le rappresentazioni binarie sono state definite utilizzando delle distribuzioni statistiche che nella pratica si adattano bene al tipo di sorgente che lo standard vuole andare a comprimere: ad esempio, per assegnare un codice binario a differenze tra vettori (che rappresentano il movimento da un fotogramma all'altro) viene usata una codifica che assegna parole brevi a numeri piccoli in modulo, ciò risulta sensato considerando che differenze piccole sono generalmente più frequenti. Questo primo passaggio risulta importante a garantire l'efficienza finale della codifica.

Context-Based

Lo standard H.264 definisce una serie di operazioni da effettuate prima di eseguire la compressione vera e propria utilizzando CABAC, necessarie a trovare e rimuovere le correlazioni statistiche che sono presenti nella sorgente. Così facendo, i dati risultanti non sono più omogenei tra loro e non lo è nemmeno la loro descrizione probabilistica. I dati generati possono essere byte di intestazione, che definiscono le proprietà dell'immagine o del blocco che si sta elaborando, vettori di movimento, che vanno a stimare lo spostamento dei pixel da un fotogramma all'altro, o i coefficienti risultanti dalla trasformata coseno applicata al singolo blocco. Risulta, quindi, necessario avere più modelli di probabilità per codificare in modo efficiente ogni tipo di dato.

Un contesto non è altro che il modello probabilistico della sorgente per un certo tipo di dato che deve essere codificato. Per ogni bit, sapendo cosa stiamo ricevendo, abilitiamo il modello adatto per eseguire la codifica.

Essendo il modello di probabilità di una variabile aleatoria binaria rappresentabile in modo efficiente da un solo numero, ovvero la probabilità di uno dei due simboli, lo standard definisce ben 399 contesti, uno o più di uno per ogni tipo di dato da codificare. Vedremo che, per motivi di efficienza di calcolo, non viene memorizzato il valore di probabilità, ma un indice che concettualmente "punta" ad esso.

Adaptive

CABAC modella le probabilità apprendendole dalla sorgente con l'arrivo dei dati. L'idea di base è simile a quella illustrata nel capitolo precedente, ovvero, ogni volta che un bit viene codificato la sua probabilità nel modello aumenta. Questo ci libera dal dover trasmettere tutti i contesti generati e allo stesso tempo non riduce l'efficienza di compressione.

Binary Arithmetic Coding

L'ultimo stadio in CABAC è la codifica aritmetica binaria vera e propria. Questa viene effettuata utilizzando l'algoritmo M coder (o modulo coder), una versione ottimizzata della codifica aritmetica e pensata per ridurre l'impatto computazionale delle operazioni matematiche riducendo al minimo i calcoli. Sono state definite, inoltre, due modalità di codifica: modalità regolare, in cui la probabilità è definita dal contesto in uso; e modalità bypass, che viene usata quando si assume che i simboli in ingresso abbiano distribuzione uniforme, essa va a disabilitare il contesto e prosegue la codifica considerando la probabilità di simbolo uguale a 0.5. Illustriamo ora il principio di funzionamento di M coder.

3.2 M coder

Nella codifica binaria l'alfabeto è composto di soli due simboli, tipicamente rappresentati come simbolo più probabile, MPS, e simbolo meno probabile, LPS, invece di 0 e 1. L'intervallo che rappresenta la sequenza codificata $[l, u)$, è ora definito dal suo limite inferiore l e dalla sua ampiezza $A = u - l$. Quando un simbolo deve essere codificato l'intervallo viene suddiviso in due parti, di ampiezza $A \cdot p_{LPS}$ e $A(1 - p_{LPS})$; in questo caso la parte inferiore rappresenta il simbolo più probabile. L'aggiornamento dell'intervallo viene effettuato come segue:

- per la codifica di MPS:

$$\begin{aligned} l^{(n)} &= l^{(n-1)} \\ A^{(n)} &= A^{(n-1)} - A^{(n-1)} p_{LPS} \end{aligned}$$

- per la codifica di LPS:

$$\begin{aligned} l^{(n)} &= l^{(n-1)} + A^{(n-1)} - A^{(n-1)} p_{LPS} \\ A^{(n)} &= A^{(n-1)} p_{LPS} \end{aligned}$$

La moltiplicazione presente nella procedura di aggiornamento dell'intervallo risulta molto costosa da calcolare per le architetture dei dispositivi utilizzati per la codifica. La soluzione fornita da M coder è quella di limitare i possibili valori di A e p_{LPS} ad un insieme discreto di numeri e di eseguire la moltiplicazione mediante la lettura di una tabella contenente i risultati precalcolati per ogni possibile coppia (A, p_{LPS}).

Per quanto riguarda CABAC, l'intervallo unitario è rappresentato con 10 bit, di conseguenza, $l, A \in [0{,}2^{10})$. In realtà il valore di A viene mantenuto all'interno di $[2^8, 2^9)$: se $A < 2^8$ allora l'intervallo ha ampiezza minore di 0.25, di conseguenza una delle condizioni E_1, E_2 o E_3

deve essere verificata e la procedura di normalizzazione riporta il valore nell'intervallo valido; se, invece, una delle tre condizioni è verificata quando $A \geq 2^8$ la normalizzazione non viene eseguita. Si sono scelti, quindi, quattro valori equidistanti per rappresentare A:

$$A_k = A_{min} + (k+1)\Delta A, \quad k = 0,1,2,3$$

con

$$\Delta A = \frac{A_{max} - A_{min}}{4}.$$

Per la probabilità del LPS, sono stati definiti i seguenti 64 valori possibili:

$$p_m = \alpha \cdot p_{m-1}, \quad m = 1, ...,63$$

con

$$\alpha = \left(\frac{0.01875}{0.5}\right)^{\frac{1}{63}} \approx 0.95, \quad p_0 = 0.5.$$

Questa scelta pratica risulta essere un buon compromesso tra la memoria necessaria a salvare la tabella e l'efficienza di codifica che si può ottenere.

Inoltre, A non deve necessariamente assumere uno dei quattro valori indicati, ma, a livello logico, viene assunto che il suo valore sia uguale al il più vicino dei quattro valori ammissibili di A per ottenere il risultato della moltiplicazione. Nella pratica vengono usati i due bit corrispondenti a 2^6 e 2^7 di A come indice nella tabella delle moltiplicazioni, mentre la probabilità è utilizza solo implicitamente e non viene mai memorizzato il suo valore, in quanto è sufficiente l'indice m per eseguire l'algoritmo, esso viene usato per accedere alla tabella delle moltiplicazioni.

3.3 Aggiornamento del contesto

Lo standard definisce delle regole per l'inizializzazione dei modelli di probabilità in base al tipo di dato e alcuni altri parametri, ma l'aggiornamento del modello avviene allo stesso modo per ciascuno di essi. Il valore $\alpha \approx 0.95$ e la cardinalità $N = 64$ dell'insieme delle probabilità rappresenta il compromesso tra l'esigenza di adattamento veloce del modello ($\alpha \to 0, N$ piccolo) e sufficiente stabilità e precisione di esso ($\alpha \to 1, N$ grande).

L'aggiornamento del contesto avviene dopo la codifica del simbolo, come spiegato nel capitolo precedente, e la modifica del modello dipende solo dall'indice m, corrispondente alla probabilità p_m, e dal simbolo codificato, che sia MPS o LPS. Quando viene codificato un MPS l'indice m viene incrementato di uno, decrementando la p_m di conseguenza, fino ad un valore massimo di 62, che rappresenta il minimo valore che la probabilità p_{LPS} può raggiungere; l'indice 63 è riservato ed utilizzato per la terminazione della sequenza. Dopo la codifica di un

LPS, invece, se $m = 0$, ovvero MPS e LPS equiprobabili, l'indice rimane uguale, ma i simboli più probabile e meno probabile vengono scambiati; nel caso di $m \neq 0$ la transizione dell'indice risulta meno immediata rispetto al caso precedente e può essere derivata dalla seguente formula:

$$p^{(n)} = \alpha \cdot p^{(n-1)} + (1 - \alpha)$$

scegliendo l'indice che fornisce il valore della probabilità più vicino a $p^{(n)}$. In Figura 3.2 viene mostrato come varia la probabilità del LPS dopo la codifica di un simbolo, in accordo con quanto appena detto.

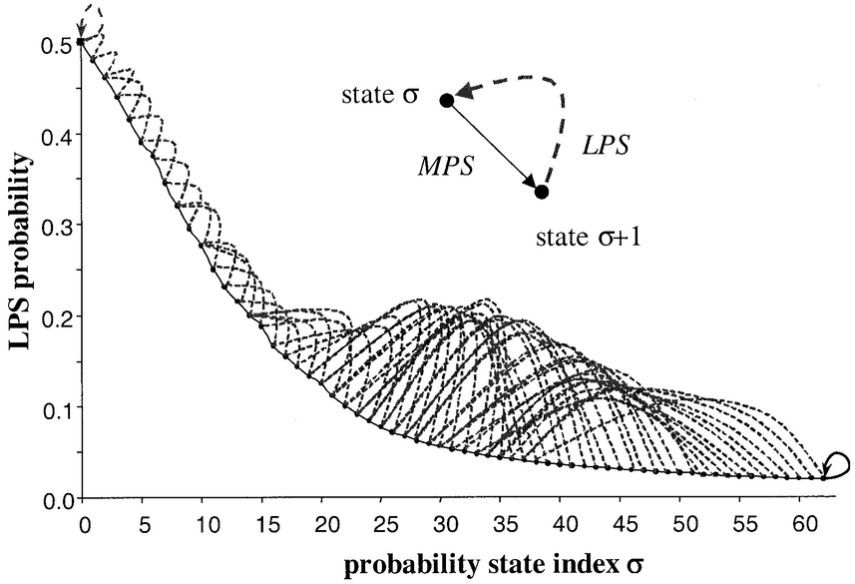


Figura 3.2: Transizioni della probabilità dopo la codifica di un simbolo. Immagine originale [1], Copyright © 2003, IEEE

Nella definizione dello standard alcuni valori calcolati per le tabelle si discostano leggermente dalla definizione matematica, ma la trattazione fatta rimane comunque valida.

Capitolo 4

Conclusione

In questa tesi è stata affrontata la codifica aritmetica come metodo di compressione dati senza perdite, che ha riscontrato un importante utilizzo pratico grazie alla sua flessibilità derivante al fatto che permette di separare la modellazione probabilistica della sorgente dalla codifica vera e propria. Questo ha permesso ad implementazioni, come CABAC, di codificare in un unica sequenza tutti i dati in ingresso, ciascuno con il proprio modello di probabilità. Risulta, inoltre, una scelta obbligata quando si opera con simboli binari, in quanto la codifica a lunghezza variabile non fornisce nessuna riduzione nella lunghezza media della sequenza a meno di concatenare simboli adiacenti, ma complicando così la descrizione probabilistica. È stato successivamente mostrato come CABAC affronta i problemi di complessità computazionale della codifica aritmetica per diventare una valida alternativa al suo predecessore, CAVLC (Context-Adaptive Variable-Length Coding), che è basato sulla codifica a lunghezza variabile. Nella pubblicazione originale [1] è stato mostrato che, in media e su sorgenti che rappresentano il caso tipico di utilizzo del codec H.264 nella pratica, CABAC offre una riduzione del bit-rate che va dal 9% al 14% rispetto all'utilizzo della codifica a lunghezza variabile. Questo incremento delle prestazioni a discapito di un piccolo incremento della complessità lo ha reso il metodo principale di codifica entropica nello standard H.264 e l'unico utilizzato in H.265, dimostrando l'ampia rilevanza pratica della codifica aritmetica.

Bibliografia

[1] D. Marpe, H. Schwarz e T. Wiegand, «Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard,» *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, n. 7, pp. 620–636, 2003. doi: 10.1109/TCSVT.2003.815173.

[2] K. Sayood, *Introduction to Data Compression (Fourth Edition)*, Fourth Edition, K. Sayood, cur. Morgan Kaufmann, 2012, isbn: 978-0-12-415796-5.

[3] T. M. Cover e J. A. Thomas, *Elements of Information Theory*. John Wiley e Sons, Ltd, 2005, isbn: 9780471748823.

[4] C. E. Shannon, «A mathematical theory of communication,» *The Bell System Technical Journal*, vol. 27, n. 3, pp. 379–423, 1948. doi: 10.1002/j.1538-7305.1948.tb01338.x.

[5] A. Moffat, R. M. Neal e I. H. Witten, «Arithmetic coding revisited,» *ACM Trans. Inf. Syst.*, vol. 16, n. 3, pp. 256–294, 1998, issn: 1046-8188. doi: 10.1145/290159.290162. indirizzo: https://doi.org/10.1145/290159.290162.

[6] T. Wiegand, G. Sullivan, G. Bjontegaard e A. Luthra, «Overview of the H.264/AVC video coding standard,» *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, n. 7, pp. 560–576, 2003, issn: 1558-2205. doi: 10.1109/TCSVT.2003.815165.