Corso di Trasmissione ed Elaborazione Numerica dei Segnali

SUN - Seconda Università degli Studi di Napoli

Laurea Magistrale in
Ingegneria Informatica

aa. 2013-14

Prof.: F. A. N. Palmieri

Appunti sulla codifica aritmetica

Autore: Giovanni Di Gennaro
email: giovannidigennaro@virgilio.it

CODIFICA ARITMETICA

1.1. Introduzione

Lo sviluppo di una codifica senza perdita d'informazione (*lossless*) che risulti la più efficiente possibile rappresenta senza dubbio un traguardo difficile da raggiungere. Ovviamente, è banale affermare che la codifica effettuata potrà risultare tanto migliore quante più informazioni si posseggano sulla sorgente, o ancor meglio sul particolare flusso d'informazioni generato. Tuttavia, poiché una sorgente che trasmetta informazioni è per sua natura aleatoria[1], è facile comprendere che la conoscenza piena dell'ingresso è spesso impossibile; a meno di non essere nell'eventualità di poter eseguire una scansione dell'intero input prima di effettuarne la codifica. Anche nel caso la scansione fosse possibile sorgono ugualmente problemi di efficienza, dovuti al fatto che analizzare un intero flusso d'informazioni potrebbe essere estremamente complesso, e per esempio impiegare troppa memoria o troppo tempo. Conoscendo però il tipo d'informazione atteso, è plausibile che per esso si abbia almeno una stima delle probabilità con cui i simboli sono generati dalla sorgente[2]. Queste probabilità *a priori* rappresentano sicuramente un'informazione in più, che può quindi essere utilizzata nella codifica per sperare di ottenere una "compressione" più proficua.

1.2. Nozioni base

Nel 1948 Claude Shannon ha dimostrato che, dato un insieme di simboli mutuamente esclusivi, derivati da una sorgente discreta senza memoria con alfabeto $\mathcal{A} = \{a_1, a_2, \dots, a_n\}$, ed una distribuzione di probabilità $\Pi = \{p_1, p_2, \dots, p_n\}$ definita su di essi, l'informazione mediamente trasportata da ogni simbolo è rappresentata dalla cosiddetta *entropia di sorgente*:

$$H(\Pi) = \sum_{i=1}^{n} p_i \log_2 \frac{1}{p_i} = -\sum_{i=1}^{n} p_i \log_2 p_i$$

Il contributo più rilevante che Shannon apportò attraverso l'entropia, fu tuttavia quello di riuscire a provare che tale quantità (definita in questa forma[3]) rappresenta una misura media dei valori binari necessari a codificare i simboli di sorgente.

[1] Una sorgente che trasmetta informazioni dovrà, infatti, non essere prevedibile; ogni "nuova informazione" deve appunto essere sconosciuta al ricevitore (altrimenti il senso stesso della trasmissione andrebbe perso).

[2] Ad esempio, un sensore che indichi la presenza di "brutto tempo" o di "bel tempo", se posizionato nella città di Napoli, produrrà in un anno sicuramente valori relativi al "bel tempo" in misura maggiore (o quantomeno ci si aspetta ragionevolmente che sia così).

[3] La base del logaritmo è di per se poco importante, tuttavia l'utilizzo in base 2 permette di misurare l'entropia in *bit/simbolo*.

Egli mostrò, quindi, che il meglio che si possa fare, se si vuole realizzare uno schema di codifica senza perdita, è quello di codificare i simboli dell'alfabeto sorgente con un numero medio di bit pari proprio all'entropia. In altre parole, l'entropia rappresenta quel valore minimo che ci si prefigge di raggiungere per poter ottenere la migliore codifica possibile. Nello stesso articolo[4] Shannon propose anche una soluzione sub-ottima al problema della codifica (attribuita a Fano, che ne pubblicò una versione nel 1949, ed oggi nota come *codifica di Shannon-Fano*) che, pur garantendo per ogni parola codice una lunghezza pari all'approssimazione superiore del limite teorico dell'*autoinformazione* $(-\log_2 p_i)$, non riusciva a raggiungere il più basso valore possibile (ossia quello promesso dall'entropia). Si dovrà aspettare la *codifica di Huffman* (allievo di Fano al MIT) per ottenere il più efficiente sistema di compressione di questo tipo: è stato infatti dimostrato che nessun'altra mappatura dei simboli dell'alfabeto in stringhe binarie possa produrre un risultato migliore[5], poiché la lunghezza media del codice di Huffman è maggiore al più di un bit rispetto all'entropia della sorgente[6].

1.3. I problemi della codifica Huffman

Quando l'alfabeto della sorgente è abbastanza grande, ma soprattutto la probabilità del simbolo più frequente non è eccessivamente elevata, si otterrà quindi che, attraverso la codifica di Huffman, la differenza dall'entropia (spesso espressa in termini percentuali) sarà abbastanza piccola da essere irrilevante. Il codice potrebbe addirittura essere perfetto, ossia avere una lunghezza media pari proprio all'entropia di sorgente.

Un esempio di ciò è il seguente: si consideri un alfabeto del tipo $\mathcal{A} = \{a_1, a_2, a_3, a_4\}$ con una distribuzione di probabilità $\Pi = \left\{\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{8}\right\}$. Valutando l'entropia si ottiene

$$H(\Pi) = \sum_{i=1}^{n} p_i \log_2 \frac{1}{p_i} = \frac{1}{2} \cdot 1 + \frac{1}{4} \cdot 2 + \frac{1}{8} \cdot 3 + \frac{1}{8} \cdot 3 = 1{,}75 \ {}^{bit}/_{simbolo}$$

mentre una possibile codifica di Huffman potrebbe essere la seguente $\{0{,}10{,}110{,}111\}$. Tale codifica possiede lunghezza media pari proprio all'entropia, dimostrando un'efficienza perfetta, infatti

$$\frac{1}{2} \cdot L(0) + \frac{1}{4} \cdot L(10) + \frac{1}{8} \cdot L(110) + \frac{1}{8} \cdot L(111) = \frac{1}{2} \cdot 1 + \frac{1}{4} \cdot 2 + \frac{1}{8} \cdot 3 + \frac{1}{8} \cdot 3 = 1{,}75 \ {}^{bit}/_{simbolo}$$

Tuttavia, nel caso in cui l'alfabeto non sia eccessivamente grande e la probabilità di occorrenza di un dato simbolo sia marcatamente maggiore rispetto a quella degli altri, la codifica di Huffman potrebbe anche risultare decisamente inefficiente, se confrontata con ciò che "promette" l'entropia. Considerando, ad esempio, un alfabeto del tipo $\mathcal{A} = \{a_1, a_2, a_3\}$, con una distribuzione di probabilità $\Pi = \{0{,}97\ ,\ 0{,}01\ ,\ \ 0{,}02\}$, si può facilmente calcolare che l'entropia equivale ad $0{,}222 \ \ bit/simbolo$, mentre una possibile codifica Huffman, del tipo $\{0{,}11{,}10\}$, conduce ad una lunghezza media pari ad $1{,}03 \ \ bit/simbolo$. La differenza tra la lunghezza media del codice ottenuto e l'entropia è quindi di $0{,}808 \ \ bit/simbolo$, che in termini percentuali equivale al 364% dell'entropia!

[4] C.E. Shannon, "*A Mathematical Theory of Communication*", Bell System Technical Journal (Vol.27 - n°3 - Luglio 1948).

[5] Ciò, ovviamente, ammesso che le probabilità utilizzate per i simboli corrispondano effettivamente alla frequenza degli stessi all'interno del flusso in uscita dalla sorgente.

[6] In realtà è stato dimostrato che la lunghezza media è maggiore al più di $p_{max} + 0.086$ rispetto all'entropia, dove p_{max} è la probabilità del simbolo che ha maggiore frequenza relativa: la codifica peggiora se vi sono simboli molto probabili.

È bene precisare che una situazione come quella presentata nell'esempio non è affatto inverosimile, all'interno di una pagina scannerizzata in bianco e nero, infatti, circa il 95% dei pixel risulta essere bianco. Se si suppone di possedere una pagina di 100 pixel la codifica di Huffman utilizzerà 100 bit, assegnando per esempio un bit "0" al colore bianco ed un bit "1" al colore nero, mentre l'entropia ci dice che tale pagina potrebbe essere codificata con soli 29 bit. Un modo per ovviare a tali tipi di problemi potrebbe essere quello di raggruppare in blocco più simboli, ottenendo cioè un alfabeto esteso e generando quindi una corrispondente codifica estesa di Huffman.

Riprendendo l'esempio precedente si potrebbero raggruppare i simboli in blocchi di due, ottenendo l'alfabeto esteso $\mathcal{A} = \{a_1a_1, a_1a_2, a_1a_3, a_2a_1, a_2a_2, a_2a_3, a_3a_1, a_3a_2, a_3a_3\}$ con distribuzione di probabilità $\Pi = \{0{,}9409\ , 0{,}0097\ , 0{,}0194\ , 0{,}0097\ , 0{,}0001\ , 0.0002\ , 0.0194\ , 0{,}0002, 0{,}0004\}$. La codifica Huffman estesa diviene quindi $\{0{,}111{,}100{,}1101{,}110011{,}110001{,}101{,}110010, 110000\}$, che possiede una lunghezza media pari ad $1{,}13\ bit/nuovo\ simbolo$, equivalente a $0{,}57\ bit/simbolo$ (se paragonato con l'alfabeto originale). Come si può facilmente intuire, la differenza tra la lunghezza media del codice ottenuto e l'entropia si è quindi ridotta, arrivando ed essere pari al 154% dell'entropia.

Tale approccio, sebbene si dimostri inequivocabilmente convergente, risulta sfortunatamente impraticabile (se si cerca di applicarlo attraverso una codifica di Huffman). Nell'esempio precedentemente visto, infatti, per poter ottenere valori accettabili (ossia pari circa all'entropia di sorgente) si dovrebbero raggruppare assieme almeno dieci simboli, ottenendo cioè un alfabeto esteso formato da $3^{10} = 59049$ simboli. Codificare attraverso Huffman un alfabeto così grande significa utilizzare una quantità di memoria e di tempo decisamente improponibile; senza contare che l'alfabeto dovrebbe poi essere aggiunto alla codifica stessa prima che essa sia inviata sul canale.

1.4. La codifica aritmetica

Il problema relativo all'impossibilità di codificare attraverso Huffman una sequenza di simboli di una certa lunghezza, deriva principalmente dal fatto che tale codifica impone l'obbligo di mappare tutte le possibili sequenze della stessa lunghezza. La *codifica aritmetica* risolve invece questo problema, generando un identificativo unico, per ogni data sequenza, senza la necessità di codificare anche tutte le altre. L'idea principale della codifica aritmetica sembra si debba a Peter Elias (un altro allievo di Fano al MIT, peraltro della stessa classe di Huffman). Tuttavia egli non pubblicò mai nulla, e pertanto tutto ciò che sappiamo della sua idea iniziale ci è pervenuto attraverso una nota presente all'interno di un libro di Abramson[7]. Negli anni la codifica aritmetica fu tuttavia ulteriormente sviluppata, fino ad arrivare alla versione che rappresenta oggi certamente la più comune, ossia quella descritta in [3]. Il principio alla base di tale codifica è quello di cercare di sfruttare una sorgente estesa, senza doversi preoccupare di codificare tutti i vari elementi dell'alfabeto. Per far sì che l'identificativo d'assegnare sia univoco in ogni caso, ossia che risulti univoco per una qualsiasi sequenza arbitrariamente grande di simboli, è però necessario che esso stesso appartenga ad un insieme infinitamente grande[8]. Se quindi si pensa ad un possibile set di simboli, che siano contemporaneamente univoci ed infiniti, non si può far a meno di considerare l'intervallo tra due numeri reali[9]. Ovviamente, dovendo scegliere un intervallo numerico, risulta naturale (in ottica probabilistica) adoperare come base l'intervallo $[0{,}1)$.

[7] N. Abramson, "*Information Theory and Coding*", McGraw-Hill, 1963.

[8] La codifica dovrà infatti assegnare un identificativo univoco per la sequenza di simboli in ingresso, a prescindere dalla quantità di simboli che compongono la sequenza e dalla quantità di simboli presenti nell'alfabeto.

[9] Tra due numeri reali esistono infiniti numeri, ed ogni numero è banalmente distinguibile dagli altri.

1.5. Il processo di codifica

Il processo di codifica consiste pertanto di pochi semplici passi:

- Si realizza un "intervallo corrente", che in principio è pari all'intervallo $[0,1)$;
- Per ogni simbolo s presente nella sequenza:
  - si divide l'intervallo corrente in sottointervalli, la cui dimensione sia proporzionale alle probabilità dei singoli simboli dell'alfabeto della sorgente;
  - si seleziona il sottointervallo relativo ad s e lo si fa diventare il nuovo "intervallo corrente".

In pratica, la codifica aritmetica utilizza la *funzione di densità cumulativa* (cdf) per assegnare ad ogni simbolo dell'alfabeto il corretto valore massimo all'interno dell'intervallo corrente; riducendo di volta in volta l'intervallo ma suddividendolo sempre come se fosse unitario[10]. Un esempio chiarirà certamente ogni dubbio.

Si consideri un alfabeto di sorgente del tipo $\mathcal{A} = \{a, b, c\}$, con distribuzione di probabilità pari ad $\Pi = \{0{,}7\ ,\ 0{,}1\ ,\ 0{,}2\}$, e si supponga di possedere in ingresso la stringa

$$aaaabcaaca$$

All'inizio il nostro intervallo è pari ad $[0,1)$, e pertanto dividendolo in ragione della *cdf* otterremmo:

$$\mathcal{F}(a) = \mathcal{F}(1) = 0{,}7 \quad , \quad \mathcal{F}(b) = \mathcal{F}(2) = 0{,}8 \quad , \quad \mathcal{F}(c) = \mathcal{F}(3) = 1$$

Al primo simbolo verrà quindi assegnato il sottointervallo $[0\ ,0{,}7)$, al secondo simbolo il sottointervallo $[0{,}7\ ,0{,}8)$ ed al terzo simbolo il sottointervallo $[0{,}8\ ,1)$, e pertanto l'intervallo iniziale risulterà frazionato in maniera proporzionale alle probabilità dei singoli simboli. A questo punto, considerando il primo simbolo in ingresso, che nel nostro caso è a, si sceglierà il sottointervallo ad esso relativo, ossia $[0\ ,0{,}7)$, e lo si considererà come il nuovo "intervallo corrente" suddividendolo in maniera tale da mantenere sempre le medesime proporzioni. Per far ciò, che non rappresenta altro che un mero esercizio di calcolo proporzionale, si consideri di rappresentare il nuovo intervallo nella forma $[low, high)$. In tal caso sarà possibile assegnare al simbolo i-esimo un nuovo sottointervallo che sia composto nella maniera seguente:

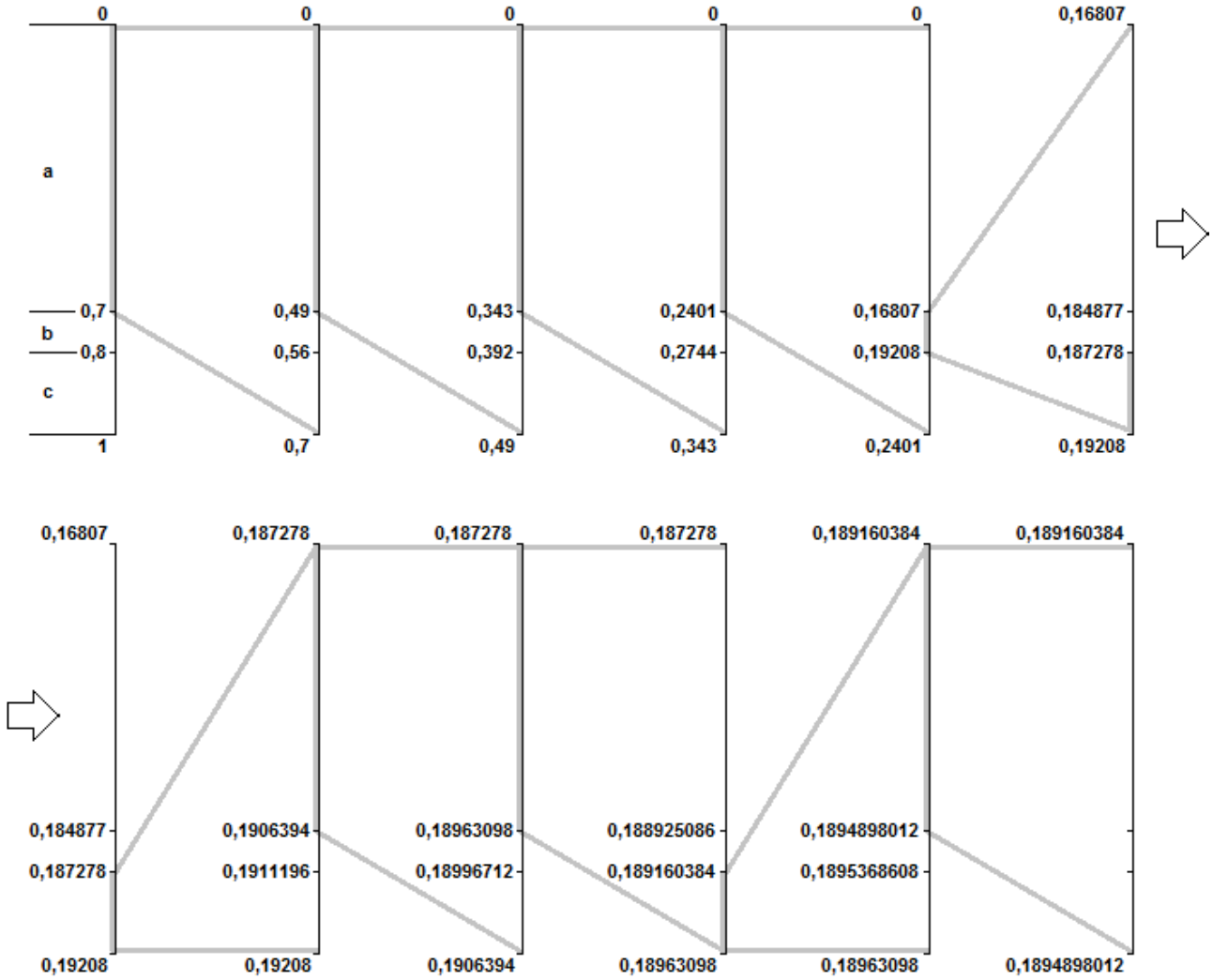
$$\big[low + (high - low) \cdot \mathcal{F}(i-1)\ \ ,\ \ low + (high - low) \cdot \mathcal{F}(i)\big)$$

Nell'esempio che si sta considerando, dopo aver scelto il primo simbolo, ed imposto così il nuovo intervallo $[0\ ,0{,}7)$, i tre simboli dell'alfabeto potranno essere rappresentati attraverso i sottointervalli

$$a = \big[0 + (0{,}7 - 0) \cdot \mathcal{F}(0)\ \ ,\ \ 0 + (0{,}7 - 0) \cdot \mathcal{F}(1)\big) = [0\ ,0{,}49)$$
$$b = \big[0 + (0{,}7 - 0) \cdot \mathcal{F}(1)\ \ ,\ \ 0 + (0{,}7 - 0) \cdot \mathcal{F}(2)\big) = [0{,}49\ ,0{,}56)$$
$$c = \big[0 + (0{,}7 - 0) \cdot \mathcal{F}(2)\ \ ,\ \ 0 + (0{,}7 - 0) \cdot \mathcal{F}(3)\big) = [0{,}56\ ,0{,}7)$$

[10] Si rammenta che per una sorgente con alfabeto $\mathcal{A} = \{a_1, a_2, \dots, a_n\}$ e distribuzione di probabilità $\Pi = \{p_1, p_2, \dots, p_n\}$ la funzione di densità cumulativa può essere scritta come: $\mathcal{F}(i) = \sum_{k=1}^{i} p_k$. Poiché tale funzione può essere al più unitaria, all'interno di un intervallo unitario (come quello iniziale) ad ogni simbolo a_i dell'alfabeto può essere assegnato il sottointervallo $\big[\mathcal{F}(i-1), \mathcal{F}(i)\big)$ con $i = 1, \dots, n$.

Nella figura sottostante è rappresentato l'intero processo di codifica della stringa in ingresso, unitamente ai valori dei relativi intervalli.



Come si può facilmente comprendere, ciò che la codifica fornisce alla fine sarà ovviamente un intervallo; nell'esempio in oggetto l'intervallo ottenuto è pari a $[0{,}189160384\ ,\ \ 0{,}1894898012)$. Naturalmente, per gli scopi di codifica, è necessario ricavare, da tale intervallo, un unico valore che rappresenti la codifica (poiché l'intervallo in sé è difficilmente gestibile). Il chiarimento appena fatto potrebbe sembrare una banale inezia, poiché sembrerebbe possa bastare anche solo affermare che un qualsiasi valore all'interno dell'intervallo individuato rappresenta una codifica valida per la stringa in ingresso, tuttavia tale precisazione ci pone dinnanzi ad un piccolo particolare che non si può non considerare. Se scegliessimo, ad esempio, di assegnare alla nostra stringa il valore $0{,}189453125$, che è sicuramente interno all'intervallo, si potrebbe senza dubbio asserire che tale valore codifica la stringa in oggetto, ma si può dimostrare che esso codifichi anche la stringa $\{aaaabcaacac\boldsymbol{c}\}$. In parole povere, la scelta di un singolo valore ci pone dinnanzi al fatto che esistono infiniti intervalli che possono contenerlo. C'è quindi la necessità d'imporre un limite nella ricerca della stringa, ad esempio fissando il numero dei simboli da codificare (nel caso precedente a 10) oppure introducendo un carattere di fine linea (*eof*) all'interno dell'alfabeto[11].

[11] Si sarebbe, ad esempio, potuto considerare un alfabeto del tipo $\mathcal{A} = \{a, b, c, eof\}$, con distribuzione di probabilità $\Pi = \{0{,}63\ ,\ 0{,}09\ ,\ 0{,}18\ ,\ 0{,}1\}$, e valutare la stringa in ingresso $\{a\ a\ a\ a\ b\ c\ a\ a\ c\ a\ eof\}$.

1.6. Efficienza della codifica

La scelta del valore da assegnare alla stringa, tra quelli possibili all'interno dell'intervallo, può essere effettuata in vari modi, ad esempio considerando il valore medio e valutando la più breve rappresentazione binaria dello stesso. Riprendendo l'esempio precedente, è infatti possibile scrivere gli estremi dell'intervallo finale in termini binari, ottenendo

$$0{,}189160384 \cong 0{,}001100000110110_2 \quad ; \quad 0{,}1894898012 \cong 0{,}001100001000001_2$$

Scegliere la più breve rappresentazione binaria del valore medio vuol dire considerare il primo bit dell'estremo inferiore che si differenzia rispetto a quelli relativi all'estremo superiore, portando poi ad uno il successivo bit "0". Nel caso in esame è quindi possibile scegliere di codificare la sequenza attraverso il valore binario $0{,}001100000\mathbf{0}111\mathbf{1}_2 = 0{,}189208984375$, e pertanto trasmettere direttamente 001100000111. Ovviamente il codice trasmesso sarà univocamente determinato, in quanto esso rappresenterà un numero all'interno dell'intervallo ottenuto, ma, se scelto in questa maniera, tale codice ha anche il pregio di rappresentare la più piccola sequenza di bit che risulti al contempo univocamente decodificabile. Supporre, per assurdo, che tale codice non fosse univocamente decodificabile, significherebbe infatti ammettere che esso possa rappresentare un *prefisso* per un altro codice. Per essere un prefisso valido per un'altra stringa in ingresso, il codice finale ottenuto dovrebbe tuttavia poter appartenere ad un intervallo diverso; in altre parole, nel nostro esempio, un codice del tipo $0{,}001100000111bbbbbbbbbb \cdots b_2$ potrebbe rappresentare un'altra stringa valida solo se il suo valore cadesse fuori dall'intervallo $[0{,}189160384\ ,\ 0{,}1894898012)$. Ovviamente ciò è impossibile poiché, se anche vi fossero infiniti bit successivi posti tutti ad "1", il limite superiore dell'intervallo ($0{,}0011000010_2$) resterebbe ugualmente irraggiungibile. È possibile dimostrare che la scelta del valore binario ottenuto nel modo mostrato equivale a troncare il valore medio ad una lunghezza in bit L_c maggiore al più di un solo bit rispetto all'approssimante superiore dell'autoinformazione della stringa in ingresso, ossia

$$L_c \leq \left\lceil \log_2 \frac{1}{\Pr\{s\}} \right\rceil + 1 \leq \log_2 \frac{1}{\Pr\{s\}} + 2$$

Calcolando tale valore per il nostro esempio, otteniamo che $\Pr\{s\} = 0{,}7^7 \cdot 0{,}1 \cdot 0{,}2^2$, per cui

$$L_c \leq \left\lceil \log_2 \frac{1}{\Pr\{s\}} \right\rceil + 1 = 13 \leq \log_2 \frac{1}{\Pr\{s\}} + 2$$

ed in effetti la lunghezza del nostro codice binario è di 12 cifre. Se avessimo considerato come ingresso la stringa $aaaaaaaaaa$, per la quale $\Pr\{s\} = 0{,}7^{10}$, si può dimostrare che il codice ottenuto sarebbe stato 0000001, la cui lunghezza è proprio pari ad

$$L_c = \left\lceil \log_2 \frac{1}{\Pr\{s\}} \right\rceil + 1 = 7$$

Allo stesso modo si può verificare che la stringa $bbbbbbbbbb$ (che rappresenta sicuramente la meno probabile) verrebbe codificata attraverso $11000111000111000111000111000011011$, ossia attraverso una parola codice di 34 bit, per la quale tuttavia risulta ancora verificata la disuguaglianza

$$L_c \leq \left\lceil \log_2 \frac{1}{\Pr\{s\}} \right\rceil + 1 = \left\lceil \log_2 \frac{1}{0{,}1^{10}} \right\rceil + 1 = 35$$

Per valutare l'efficienza della nostra codifica dovremmo, come noto, confrontare la lunghezza media della stringa codificata con l'entropia della sorgente estesa. Come dovrebbe essere noto, per una sorgente senza memoria, con simboli mutuamente esclusivi, l'entropia della sorgente estesa può essere misurata considerando semplicemente il numero di simboli, ossia

$$H(\Pi^m) = mH(\Pi)$$

Nel nostro caso, quindi, poiché $H(\Pi) \cong 1{,}157\ bit/simbolo$, l'entropia della sorgente estesa può semplicemente essere calcolata come $H(\Pi^{10}) = 10 \cdot H(\Pi) \cong 11{,}57\ bit/simbolo$, e pertanto codificare la stringa con una lunghezza di soli 12 bit rappresenta senz'altro un ottimo risultato. D'altra parte, se L_{ci} rappresenta la lunghezza in bit della codifica per la stringa i-esima, è possibile valutare la media delle lunghezze, per la codifica aritmetica di una stringa di m simboli, come

$$L_{E^m} = \sum_{i=1}^{n^m} \Pr\{s_i\} \cdot L_{ci} \leq \sum_{i=1}^{n^m} \Pr\{s_i\} \cdot \left(\left\lfloor \log_2 \frac{1}{\Pr\{s_i\}} \right\rfloor + 1 \right) \leq \sum_{i=1}^{n^m} \Pr\{s_i\} \cdot \left(\log_2 \frac{1}{\Pr\{s_i\}} + 2 \right)$$

per cui

$$L_{E^m} \leq \sum_{i=1}^{n^m} \Pr\{s_i\} \cdot \log_2 \frac{1}{\Pr\{s_i\}} + 2 \sum_{i=1}^{n^m} \Pr\{s_i\} = H(\Pi^m) + 2$$

La lunghezza media per simbolo $L_E = L_{E^m}/m$ sarà quindi compresa tra

$$\frac{H(\Pi^m)}{m} \leq L_E \leq \frac{H(\Pi^m)}{m} + \frac{2}{m} \qquad \text{che equivale a dire che} \qquad H(\Pi) \leq L_E \leq H(\Pi) + \frac{2}{m}$$

Incrementando la lunghezza m della sequenza di simboli considerati ci si può quindi spingere sempre più vicino all'entropia, raggiungendo così il massimo che si possa ottenere[12].

1.7. Gli svantaggi della codifica aritmetica

Finora si è implicitamente assunto che il codificatore possa lavorare inizialmente con cifre ad infinita precisione, ricercando solo in seguito la frazione binaria che debba rappresentare la codifica. Ovviamente, nella realtà, i codificatori lavoreranno direttamente con un numero finito di cifre binarie, che quindi non permetterà di considerare sequenze di simboli lunghe a piacere. Il principale svantaggio della codifica aritmetica è tuttavia quello di dover necessariamente attendere che la sorgente generi l'intera sequenza di m simboli, prima di poter iniziare ad inviare qualcosa. L'intervallo finale è infatti univocamente determinato da tutti i simboli che compongono la sequenza, e sebbene per quanto visto questo sia un pregio irrinunciabile, esso conduce inevitabilmente all'impossibilità di conoscerne il valore prima della generazione di tutti i simboli.

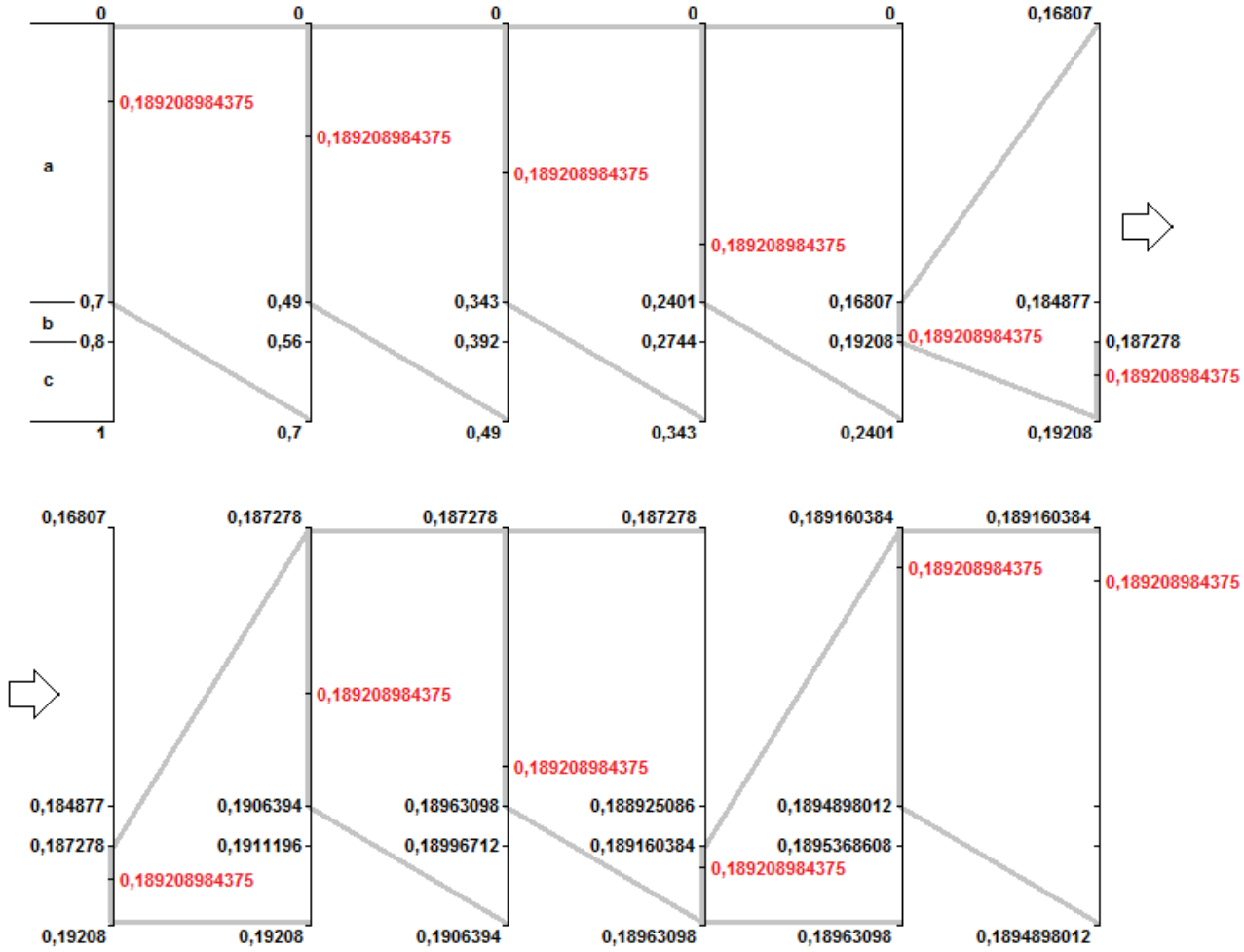
[12] È bene precisare che ciò è vero se le probabilità a priori considerate siano effettivamente prossime alla frequenza dei simboli generati dalla sorgente, altrimenti l'entropia valutata risulterebbe falsata da tali incongruenze. Inoltre è possibile verificare che per la codifica di Huffman si ottiene una limitazione migliore, in quanto risulta

$$H(\Pi) \leq L_E \leq H(\Pi) + \frac{1}{m}$$

ma si sono già discussi i motivi per i quali tale codifica, sebbene teoricamente migliore, risulti in pratica inaccettabile.

1.8. Il processo di decodifica

Evitando di soffermarsi sul processo binario, ovvero sul come i vari valori binari siano adoperati all'interno dei calcolatori, il processo di decodifica risulta estremamente banale. Ad ogni passo, infatti, il decodificatore non dovrà far altro che considerare l'attuale intervallo, suddividerlo in maniera proporzionale alle probabilità (assegnando agli estremi dei sottointervalli il valore della *cdf* nello stesso modo visto in precedenza) e valutare in quale sottointervallo cada il numero rappresentato dalla codifica. Un esempio, che riprende il valore $0{,}001100000111_2 = 0{,}189208984375$ precedente, è rappresentato nella figura seguente.



Come si può vedere nell'ultimo passo, il processo di decodifica potrebbe continuare, ma per quanto detto in precedenza supponiamo che la nostra decodifica imponga un limite (nel nostro caso pari a 10) ai simboli che devono essere decodificati.

BIBLIOGRAFIA

[1] K. Sayood, *Introduction to Data Compression*, 3rd ed., Morgan Kaufmann, 2006

[2] David Salomon, *Data Compression – The complete reference*, 4th ed., Springer, 2007

[3] J. J. Rissanen, G. G. Langdon, *Arithmetic Coding*, IBM Journal on Reserch and Development, 20, no. 3, pp. 146-162, Marzo 1979