Informazione e compattezza di un messaggio

Definizione operazionale (e incompleta) dell'informazione, indipendente dal contenuto:

è generata, trasmessa, ricevuta

L'informazione provoca una reazione

L'informazione è diversa dalla conoscenza

Come **misurare** l'informazione trasmessa in un messaggio (sequenza) costituito da *n* "eventi"?
Limitiamoci all'informazione formale, che i simboli contenuti negli eventi potrebbero rappresentare tenendo conto della loro frequenza e delle loro associazioni, e trascuriamo il significato dell'informazione vera (o falsa) che il messaggio potrebbe indurre in chi lo riceve.

A - Definizione probabilistica (Shannon)

Le notizie meno frequenti contengono più informazione in quanto provocano più "sorpresa".
Nei tre messaggi in codice binario

00000100 0100000 00000001

la cifra 1 sembra essere la più interessante, la più ricca di contenuto informativo.

Shannon ha definito l'informazione propria o intrinseca (*self-information*) $i(A_j)$ di un evento A_j appartenente ad un insieme ***A*** di *n* eventi possibili, come il logaritmo in base 2 dell'inverso della probabilità $p_j = p(A_j)$ che l'evento appaia:

$$i(A_j) = \log_2 \frac{1}{p_j}$$

L'informazione propria di due eventi indipendenti è la somma delle informazioni relative a ciascun evento.

L'unità di misura dell'informazione cosí definita è il *bit*

Il valore di attesa (*expectation*) di $i(A_j)$ sugli *n* eventi possibili è definito come "entropia" H e rappresenta l'informazione media per evento (o per simbolo)

$$H = E(i(A_j)) = \sum_{j=1}^{n} p_j \log_2 \frac{1}{p_j} = -\sum_{j=1}^{n} p_j \log_2 p_j$$

che ci si aspetta di ricevere.

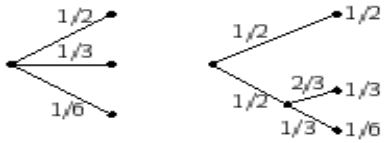
L'entropia H

>> è una funzione continua delle probabilità individuali;

>> se tutte le *n* probabilità sono uguali, è una funzione di *n* monotona e crescente;

>> è indipendente dall'ordine in cui gruppi di informazioni sono registrati

$$H(\tfrac{1}{2},\tfrac{1}{3},\tfrac{1}{6}) = H(\tfrac{1}{2},\tfrac{1}{2}) + \tfrac{1}{2} H(\tfrac{2}{3},\tfrac{1}{3})$$



Esempi

- Certezza:

$p_1 = 1 \qquad p_j = 0 \; per \; j \neq 1 \qquad H = 0$

Se l'evento corrispondente a p_1 è rappresentato dal simbolo a che ci si aspetta appaia ogni volta, la sequenza di eventi $aaaaaaaaaaaaa....$ non può portare nulla di nuovo.

- Moneta:

$P(testa) = 1/2 \qquad P(croce) = 1/2$
$i(testa) = 1 \, bit/simbolo \qquad i(croce) = 1 \, bit/simbolo$
$H = 1 \, bit/simbolo$ per esempio $testa = 0$ e $croce = 1$

Per rappresentare una sequenza di n eventi sono necessari n *bits*.

- ***n*** simboli equiprobabili:

$p = 1/n \qquad H = \log_2 n \quad [bit/simbolo]$

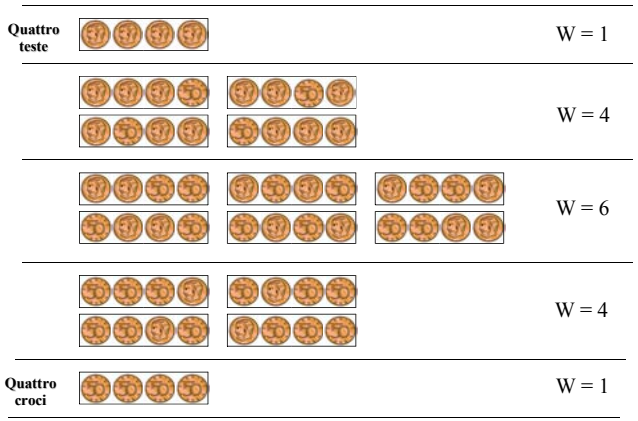
Massimo dell'informazione potenziale per simbolo

Analogia coll'entropia della meccanica statistica (Boltzmann), proporzionale, in equilibrio termodinamico, al logaritmo del numero W di stati elementari equiprobabili (per molecola).

$S = k \ln W \quad [J/K]$
$k = 1.38 \times 10^{-23} \quad [J/K]$ costante di Boltzmann

Se si misurasse l'entropia di Shannon in unità termodinamiche, un *bit* corrisponderebbe a $k \ln 2 = 0{,}96 \times 10^{-23} \quad [J/K]$

Tutti i possibili stati elementari di 4 monete



- Sequenza di 16 eventi:

1 2 3 2 3 4 5 4 5 6 7 8 9 8 9 10

Ci sono 10 simboli

$P(1) = P(6) = P(7) = P(10) = 1/16$
$P(2) = P(3) = P(4) = P(5) = P(8) = P(9) = 2/16$

$H = 3.25 \, bits/simbolo$

Sono necessari $3.25 \times 16 = 52$ *bits* per rappresentare la sequenza

Se si rappresenta la sequenza di 16 eventi come variazione di un evento rispetto al precedente, si ottiene:

1 1 1 -1 1 1 1 1 -1 1 1 1 1 1 -1 1 1

Ci sono due simboli: 1 e -1
con probabilità $P(1)=13/16$ e $P(-1)=3/16$

$$H=0.70\,bits/simbolo$$

Sono necessari $0.70\times16=11.2$ *bits* per rappresentare la sequenza, ma **in più** bisogna conoscere la relazione tra gli elementi x_n della sequenza e i residui r_n : $x_n=x_{n-1}+r_n$

• Sequenza, di 20 eventi:

1 2 1 2 3 3 3 3 1 2 3 3 3 3 1 2 3 3 1 2

› se chiamiamo "simboli" 1, 2 e 3, la sequenza contiene 3 simboli ed $n=20$ eventi

$$P(1)=P(2)=5/20=1/4 \qquad P(3)=10/20=1/2$$

$$H=1.5\,bits/simbolo$$

Sono necessari $1.5\times20=30$ *bits* per rappresentare la sequenza

1 2 1 2 3 3 3 3 1 2 3 3 3 3 1 2 3 3 1 2

› se chiamiamo "simboli" le coppie "12" e "33", la sequenza contiene 2 simboli ed $n=10$ eventi

$$P(12)=5/10=1/2 \qquad P(33)=5/10=1/2$$

$$H=1\,bit/simbolo$$

Sono necessari $1\times10=10$ bits per rappresentare la sequenza

Le coppie esplicitano una regolarità e introducono un **modello** nella sequenza

▪ Alfabeto italiano:

$n=22$ simboli (21 lettere più lo spazio)

se fossero equiprobabili $H=\log_2 22=4.46\,bits/simbolo$

siccome le probabilità sono diverse (***e*** è la più probabile $p_e=0.1262$, ***q*** la meno probabile $p_q=0.0057$) $H\approx4\,bit/simbolo$

se si utilizzano altre caratteristiche del linguaggio (come gruppi di lettere frequenti, sequenze sintattiche etc.) , ossia se si introduce un **modello**: $H\approx1\,bit/simbolo$

(per l'alfabeto inglese $n=26$)

B – Definizione algoritmica (Chaitin)

Che cos'è una sequenza aleatoria? Osservando tre delle $2^{20}=1048576$ sequenze possibili costruite con 20 caratteri in codice binario

0101 0101 0101 0101 0101	369525
1001 1010 0101 1011 0010	632242
1011 0101 0000 0100 1111	741455

la prima non sembra aleatoria, è costruita ripetendo 10 volte la coppia "01"; la seconda e la terza forse si.

La complessità algoritmica di una sequenza (messaggio), ossia la descrizione più concisa che permette di riprodurla, per esempio attraverso un programma di computer che la genera, dà una misura del contenuto in informazione.

Per riprodurre una sequenza di lunghezza n con una struttura dove "01" è ripetuto $n/2$ volte, basta un breve programma che esegue $n/2$ volte un *loop* che stampa "01".

Per descrivere il contenuto di una sequenza aleatoria di lunghezza n sono necessari tanti *bits* quanto è lunga (cfr. moneta): il programma ha quindi lunghezza superiore a n *bits*.

Il problema sta a determinare se una sequenza è davvero aleatoria o no! Per esempio, la terza sequenza riportata sopra corrisponde alle venti cifre più significative di $\sqrt{2}$ in codice binario, non è aleatoria ed ha una rappresentazione concisa.

1011 0101 0000 0100 1111 $\qquad 741455/2^{19}=1.414213$

Commenti vari su **Informazione, Entropia, Conoscenza**

1 - **Cancellare** dell'informazione è un processo dissipativo.
Principio di Landauer: si genera entropia termodinamica la livello di almeno $k\ln 2=0{,}96\times10^{-23}\ \ [J/K]$ per ogni *bit* cancellato.

2 - Oltre all'informazione di Shannon e a quella algoritmica esiste anche un'**informazione quantica**.
Un registro classico è un sistema che può assumere due stati, 0 e 1, in modo esclusivo. Chiamiamolo **Cbit**. Fornisce fino a un *bit* di informazione quando è letto e non è perturbato dalla lettura.
Il sistema quantico corrispondente, **Qbit** o **qubit**, può essere in una sovrapposizione dei due stati 0 e 1, che costituiscono i vettori di base di uno spazio di Hilbert. Per quanto contenga informazione superiore a un *bit*, in lettura restituisce al massimo un *bit* ed è perturbato dal processo di misura (*collapse of the wave function*).

2a – Il processo di misura applicato a un sistema quantistico è un processo dissipativo

3 – In un sistema quantistico si definisce l'**entropia di von Neumann**, con analogie con l'entropia della meccanica statistica e coll'entropia di Shannon.
(**No one knows what entropy really is,** so in a debate you will always have the advantage (J. von Neumann))

4 – L'informazione provoca effetti e azioni che non sono determinati da forze ma da reazioni umane o di strumenti disegnati dall'uomo.

5 - L'informazione esiste solo quando e perché è interpretata dalle funzioni cerebrali? Nell'evoluzione dell'universo l'informazione aumenta automaticamente, come l'entropia di un sistema isolato, o aumenta perché è interpretata da esseri intelligenti?

Ridondanza e compressione

Una sequenza di n simboli contiene una quantità di informazione "vera" (utile, interessante....) che corrisponde ad una entropia H_{vera}, di cui, in generale, si possiede solo un **valore stimato**.

- Se gli n simboli sono equiprobabili il valore stimato più attendibile dell'informazione contenuta è $H = \log_2 n$.
- Se gli n simboli non sono equiprobabili $H < \log_2 n$: il valore stimato dipende dalle probabilità e dai modelli.
- Quando si realizza una **rappresentazione** della configurazione si ottiene

$$H_{vera} \leq H_{rappr}$$

La **ridondanza** della rappresentazione è

$$R = H_{rappr} - H_{vera} \; bits/simbolo$$

(la definizione di ridondanza varia secondo gli autori!)

Supponiamo di aver quattro simboli a_i

Stima: Se non sappiamo nulla della probabilità dei simboli o se sono equiprobabili sono necessari $\log_2 4 = 2$ bits/simbolo

Rappresentazione: Se li rappresentiamo con un codice ASCII usiamo 8 bits/simbolo. ma solo 7 sono utilizzati per l'informazione.

La **ridondanza** della rappresentazione ASCII (per rappresentare 4 simboli equiprobabili) è $7 - 2 = 5$ bits/simbolo.

Simbolo	*Probabilità*	*Codice*
a_1	0.25	00
a_2	0.25	01
a_3	0.25	10
a_4	0.25	11

Il codice riportato nella tabella precedente richiede 2 bits/simbolo; la ridondanza di questa rappresentazione (per rappresentare 4 simboli equiprobabili) è zero

Se i simboli hanno probabilità come nella tabella seguente, la rappresentazione stimata più efficiente richiede
$-(0.49\log_2 0.49 + 0.25\log_2 0.25 + 0.25\log_2 0.25 + 0.01\log_2 0.01) \approx 1.57$
bits/simbolo

Simbolo	*Probabilità*	*Codice punti neri*
a_1	0.49	1
a_2	0.25	01
a_3	0.25	000
a_4	0.01	001

Per questa distribuzione di probabilità la rappresentazione a due *bits* ha ridondanza $2 - 1.57 = 0.43$ *bits*/simbolo

Un codice a lunghezza variabile (**codice punti neri** nella tabella) richiede mediamente $1 \times 0.49 + 2 \times 0.25 + 3 \times 0.25 + 3 \times 0.01 \approx 1.77$ *bits*/simbolo
La ridondanza con questo codice è $1.77 - 1.57 = 0.2$ *bits*/simbolo.

Se c'è ridondanza, i dati possono essere compressi

I **modelli** intervengono tanto nella scelta di una rappresentazione efficiente quanto nella definizione dei metodi di compressione. Tengono conto di:

proprietà fisiche, es. l'immagine in un fax è in prevalenza bianca

probabilità dei simboli, es. nell'alfabeto q è poco probabile

correlazioni col passato recente (Markov), es. in *"lezione"*, dopo *"...ion"* ci sono probabilmente *"e"* ovvero *"i"*

La scelta della rappresentazione è legata a fattori storici o di convenienza, es. numerazione in base 10 o caratteri ASCII.
Quindi non è necessariamente la più efficiente.
Di qui l'importanza dei metodi di **compressione**.

Per compressione si intendono entrambi i processi: **ridurre** il volume dei dati d'origine e **restituire** i dati in una forma vicina all'originale.

Compressione senza perdite di informazione (***lossless***) necessaria per trasmettere, per esempio, un programma di computer.

Compressione con perdite di informazione (***lossy***) quando l'utente non ha bisogno di tutti i dettagli, come nella trasmissione di immagini di televisione.

Definizioni:

$$rapporto\ di\ compressione = \frac{volume\ dati\ dopo}{volume\ dati\ prima}$$

$$fattore\ di\ compressione = \frac{volume\ dati\ prima}{volume\ dati\ dopo}$$

Esempi di compressione *lossless*

Fax (*facsimile*)

Le immagini sono prodotte con uno *scanning* per righe.
Lungo la riga si leggono 8.05 punti/mm (*pixels, pels*) che possono essere solo bianchi o neri; la distanza tra le righe è scelta tra 3.85, 7.7 o 15.5 righe/mm. Una riga contiene dell'ordine di 1000 punti e una pagina qualche milione.

La maggior parte dei punti sono bianchi.

Invece di indicare individualmente ciascun punto bianco (b) o nero (n), si riporta il numero di punti bianchi, seguito dal numero di punti neri, seguito dal numero di punti bianchi e cosí via fino alla fine della linea.

Per convenzione, si aggiunge un punto bianco all'inizio della linea anche se non c'è e un **EOL** (***End Of Line***) alla fine della linea.

Questo metodo di codificazione si chiama **RLE** (***Run Length Encoding***)

nnnbbbbbbbnnnbbbbbbbbnnnnbbbbbbbbbbbbbb
1b 3n 7b 3n 8b 4n 15b EOL
1 3 7 3 8 4 15 EOL

Inoltre si tiene conto del fatto che le sequenze più frequenti di punti neri sono corte e si attribuisce un codice corto alle sequenze nere corte, come in tabella.

Il codice usato per i punti neri è un esempio di codice a dimensione variabile (*Variable Size Coding*) che tiene conto delle frequenze dei simboli: si tratta di un codice di **Huffman** (modificato).

Numero di punti	*Codice punti bianchi*	*Codice punti neri*
1	000111	010
2	0111	11
3	1000	10
4	1011	011
5	0110	0011
6	1110	0010
7	1111	00011
8	10011	000101
9	10100	000100
10	00111	0000100
11	01000	0000111
12	001000	0000100
13	000011	00000111
...	...	...
EOL	0000000001	

MWPC

In un sistema di camere a fili le informazioni si leggono per piani di fili. Un piano può aver più di 256 fili: per assegnare un indirizzo a ciascun filo un *byte* può non essere sufficiente. Se ci sono 256 piani di 256 fili, due *bytes* bastano per tutto il sistema.

Di solito pochi fili hanno un segnale e spesso fili con segnale sono contigui (*cluster*).

La situazione si presta ad applicare un ***run lengh encoding*** come per il fax, partendo da un'estremità del primo piano, alla fine del quale bisogna anche generare elettronicamente un EOL, più complicato se i piani hanno numeri di fili differenti. Il passaggio al piano successivo è identificato dall'EOL ovvero da un indirizzo predeterminato.

Un altro modo di lettura consiste ad assegnare un indirizzo ad ogni filo del sistema, a leggere tutti i fili e a conservare solo gli indirizzi dei fili che hanno prodotto un segnale (***zero suppression***).

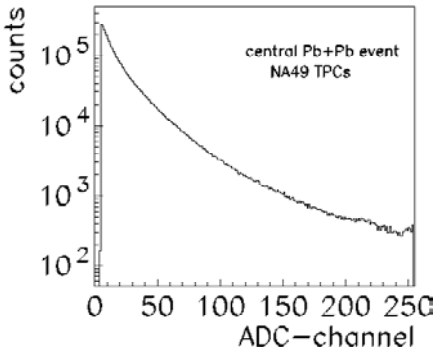
Per sistemi in cui il numero di fili toccati è dell'ordine di qualche percento i due sistemi di lettura precedenti producono compressioni comparabili.

Considerazioni analoghe si applicano alla lettura di altri rivelatori a soglia (con risposta **0** ovvero **1**) come le *pixels*.

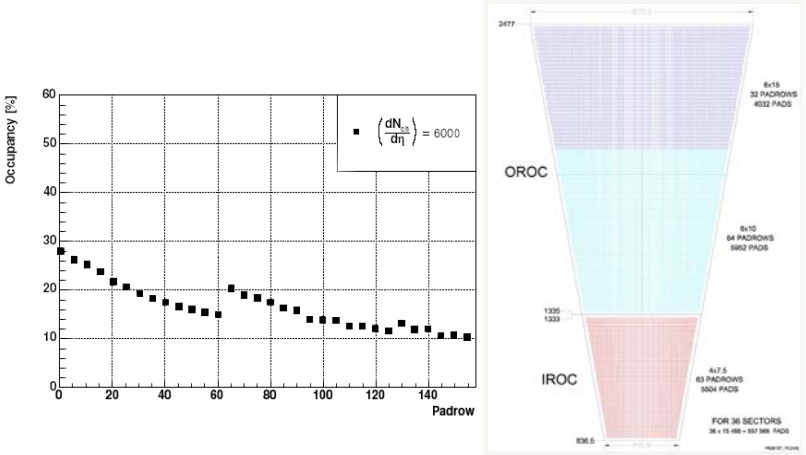
Huffman *coding* e la TPC

1 - La frequenza dei valori degli ADC è più elevata per valori piccoli. Anche qui si può applicare un codice di Huffman.

Compressione intorno al 30%.



2 - In una TPC ci sono regioni più popolate vicino alla zona di interazione e gli indirizzi di questi punti sono più frequenti. Assegnado codici brevi agli indirizzi frequenti si comprimono i dati. Compressione intorno al 10%.



Dictionary methods

Se una sequenza di caratteri si ripete frequentemente può essere sostituita da un codice breve.
Si costruisce un dizionario delle sequenze, definto dal modo (statico o dinamico) con cui è costruito, dalla lunghezza delle sequenze registrate e dal loro numero massimo
Esempi: UNIX *compress*, ZIP e GIF.

Applicabile ai dati con compressioni fino a 20%

Esempi di compressione *lossy*

Campionamento (*Quantization*)

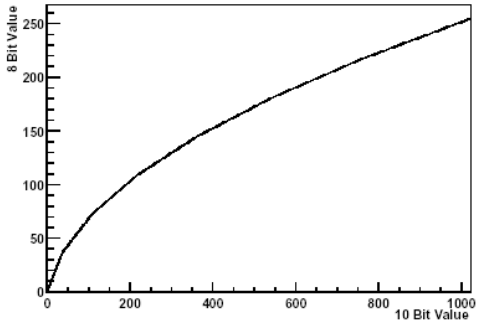
Data una funzione continua se ne conservano i valori solo in un numero limitato di punti. In questi punti i valori sono acquisiti con una certa precisione che può essere ridotta per comprimere ulteriormente.

Dopo decompressione i punti hanno un errore più grande
Esempio: ridurre il numero di *bits/pixel* in un'immagine.

Nella TPC i segnali nella direzione di *drift* sono campionati al ritmo dell'orologio e con la precisione intrinseca dell'ADC.

Si può cambiare la scala dell'ADC in modo non lineare, modificando di poco i valori piccoli e sostanzialmente i valori grandi.
Compressione dell'ordine del 10%.

Dopo decompressione l'errore per i valori grandi è aumentato.



Compressione di immagini: trasformata DCT

Nella compressione di immagini si sfrutta il fatto che *pixels* adiacenti sono correlate e che errori di riproduzione dopo decompressione possono essere tollerati, anche se qualche volta si vedono.

L'immagine è digitizzata in *pixels* di una certa dimensione spaziale e "profondità" (~ toni di grigio, colore).

Le correlazioni sono limitate a gruppi di 8 x 8 *pixels*. A questi gruppi si applica una DCT (*Discete Cosine Transform*) in due dimensioni.

La DCT è simile alla trasformata di Fourier, solo coseni e discreta. E' usata in compressione di immagine e di suono, JPEG, MPEG.

Video

La compressione video digitale si basa su due osservazioni:

- ogni immagine (*frame*) ha grande ridondanza spaziale
- immagini successive contigue cambiano di poco

La "prima" immagine è trattata secondo i metodi di compressione basati su DCT.

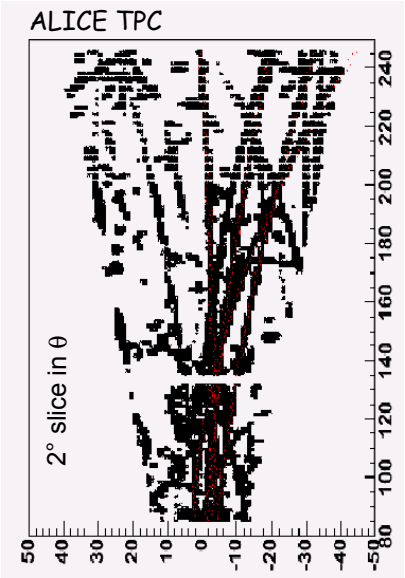
Si calcolano le differenze tra immagini successive contigue e queste differenze sono compresse come un'immagine che contiene poca informazione.

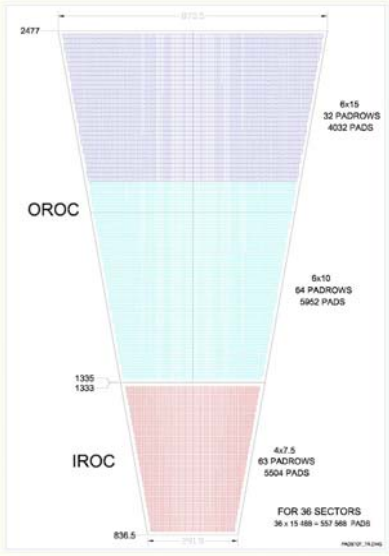
Gli standards MPEG seguono questa filosofia.

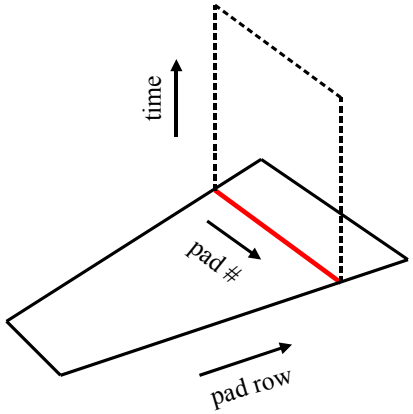
Applicazione alla TPC

Si potrebbe pensare alla TPC come a un televisore che produce sul piano dei catodi immagini successive correlate e applicare metodi di compressione video.
L'idea non funziona perchè l'immagine presente a un certo istante non ha abbastanza ridondanza per permettere una buona compressione, **senza perdere informazione essenziale**. Inoltre le correlazioni tra immagini successive non sono sufficienti a produrre una compressione significativa.

Per ottenere una buona compressione è necessario ricostruire le tracce, almeno in certe regioni.
Una ricostruzione completa richiede di associare punti vicini nel piano e nel tempo (*clusters*) per definire **punti nello spazio** e misurare la **carica totale**.
L'operazione (*cluster finding*) può essere effettuata online e produce compressione.

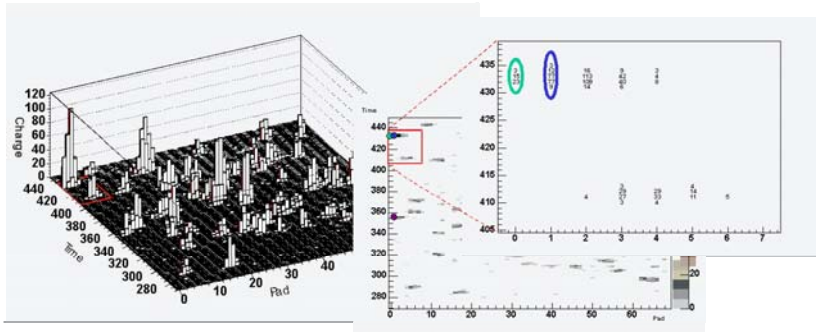






I segnali relativi a un segmentino di traccia sono distribuiti su *pads* adiacenti appartenenti alla stessa riga (*row*) e su cicli di orologio successivi; il numero della riga (*padrow #*) dà la terza coordinata.

4 views of clusters in the ALICE TPC



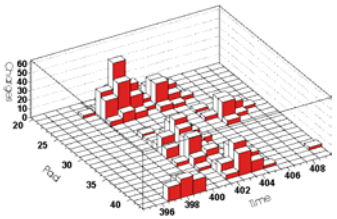
Gaute Grastveit, UoB

Quando le tracce sono ben separate i *clusters* (nel piano *time-pad*) hanno una forma a campana che permette un buon calcolo delle coordinate e della carica totale.

Se le tracce sono vicine, *clusters* adiacenti si sovrappongono e presentano un profilo con due gobbe: è necessaria una deconvoluzione per separarli con precisione.

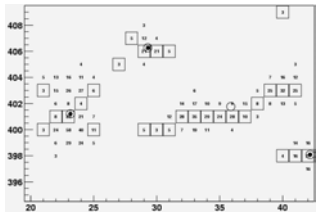
La forma di *clusters* sovrapposti dipende molto dall'inclinazione delle tracce: una deconvoluzione grossolana si ottiene separando i *clusters* al minimo tra le gobbe ma una deconvoluzione precisa richiede di ricostruire le tracce. In quest'ultimo caso, per ridurre il tempo di calcolo nel *fit* dei parametri di una gaussiana in due dimensioni, si fa una ricostruzione approssimata e locale delle tracce (per esempio con la trasformata di Hough e tracce elicoidali), usando in seguito i risultati come valori iniziali nella deconvoluzione.

ALICE TPC **Deconvolution 1**

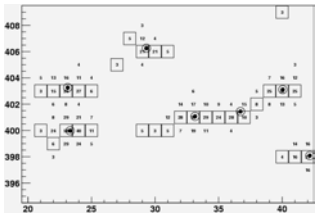


Simplified implementation,
almost for free – splits at minima
in both directions (time and pad)

off

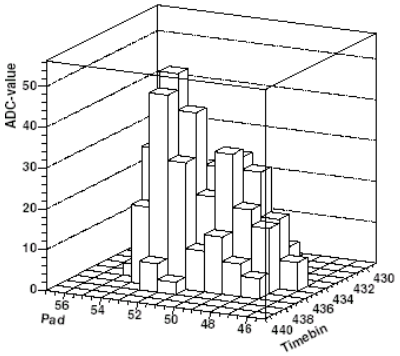


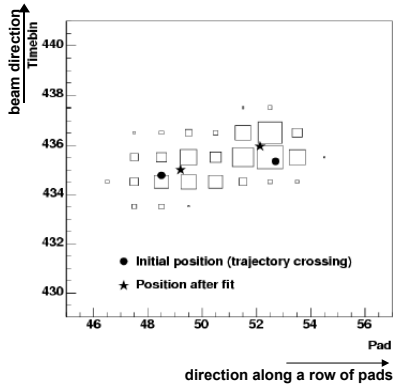
on



Gaute Grastveit, UoB

ALICE TPC **Deconvolution 2**





Initial position of cluster center = crossing of pad plane by tracks (helix) obtained with raw ADC data and Hough transform
Position after fit = space point by fitting cluster center

Conservando i parametri dei *clusters* riferiti alle coordinate spaziali si ottiene una buona compressione (un fattore 3 in ALICE).

Cluster parameters	Size (Bit)
Pad coordinate	14
Time coordinate	16
σ_{pad}	8
σ_{time}	8
Total charge	12
Total	58

Raw-data parameters	Size (Bit)
ADC-values	3×3×8=72
Pad-numbers	3×8=24
Time information	3×32=96
Total	192

A. Vestbø, UoB

Si può far meglio (fattore intorno a 5 in ALICE) riferendo i clusters alle tracce ricostruite, delle quali, ovviamente, si devono conservare i parametri. Che fare dei *clusters* "orfani" che non appartengono a nessuna traccia?

La precisione con cui si conservano i dati e il trattamento riservato ai *clusters* "orfani" influiscono sul fattore di compressione e sull'efficienza di ricostruzione offline.

Compressione attraverso una ricostruzione completa

La ricostruzione completa di un evento, o di una porzione di evento limitata ad una regione dello spazio, rappresenta un metodo efficiente di compressione dati.
Tale ricostruzione avviene per realizzare certe funzioni di trigger elaborato (**H**igh-**L**evel **T**rigger, **HLT**), le quali, **scartando eventi "senza interesse"**, producono una compressione (*lossy?*) molto significativa.

Se l'evento non è scartato si possono verificare tre situazioni:

1. Conservare i dati originali e le informazioni di trigger (HLT compreso);
2. Conservare solo l'evento (sub evento) ricostruito e le informazioni di trigger;
3. Conservare i dati originali, l'evento (sub-evento) ricostruito e le informazioni di trigger.

Solo nel secondo caso si ottiene una compressione, che può essere grande, superiore a un fattore 10.

Referenze

C. E. Shannon, A Mathematical Theory of Communications
The Bell System Technical Journal, 27 (1948), pp. 379–423,623–656.
http://cm.bell-labs.com/cm/ms/what/shannonday/shannon1948.pdf

G. J. Chaitin, Algoritmic Information Theory, IBM Journal of Research and Development, 21 (1977), pp.350-359, 496
http://www.cs.auckland.ac.nz/CDMTCS/chaitin/ibm.pdf
http://www.research.ibm.com/journal/rd/214/ibmrd2104G.pdf

N. D. Mermin, From Cbits to Qbits: Teaching Computer Scientists Quantum Mechanics, Am. J. Phys. 71 (1) (2003),
http://arxiv.org/PS_cache/quant-ph/pdf/0207/0207118.pdf
http://www.ccmr.cornell.edu/~mermin/qcomp/CS483.html

K. Sayhood, Introduction to Data Compression, 2nd ed., Morgan Kaufmann (Academic Press) 2000

D. Salomon, Data Compression, The Complete Reference, 2nd ed., Springer 2000

Appendice

Esempio di DCT in una dimensione

Supponiamo di aver diviso l'intervallo di definizione in 8 parti uguali e che la funzione "quantizzata" negli 8 intervalli sia rappresentata dal vettore

$$p_t = (12,10,8,10,12,10,8,11)$$

Calcoliamo i coefficienti della DCT

$$G_f = \frac{1}{2} C_f \sum_{t=0}^{7} P_t \cos\left(\frac{(2t+1) f \pi}{16}\right)$$

$$C_f = \frac{1}{\sqrt{2}} \; per \; f = 0 \qquad C_f = 1 \; per \; f = 1, ..., 7$$

e otteniamo gli 8 numeri

28.6375, 0.571202, 0.46194, 1.757,
3.18198, -1.72956, 0.191342, -0.308709.

Questi 8 numeri possono essere usati per ricostruire il vettore d'origine, salvo gli errori dovuti alla precisione limitata del calcolatore.

Lo scopo è quello di comprimere i dati, perciò quantizziamo i coefficienti una prima volta:

28.6, 0.6, 0.5, 1.8, 3.2, -1.8, 0.2, -0.3,

e applichiamo la DCT inversa trovando

12.0254, 10.0233, 7.96054, 9.93097,
12.0164, 9.99321, 7.94354, 10.9989.

Quantizziamo ancora di più:

28, 1, 1, 2, 3, -2, 0, 0,

e applichiamo la DCT inversa ottenendo

12.1883, 10.2315, 7.74931, 9.20863,
11.7876, 9.54549, 7.82865, 10.6557.

Finalmente quantizziamo a

28, 0, 0, 2, 3, -2, 0, 0,

e otteniamo con la DCT inversa la sequenza:

11.236, 9.62443, 7.66286, 9.57302,
12.3471, 10.0146, 8.05304, 10.6842,

cfr con la sequenza di ingresso: 12, 10, 8, 10, 12, 10, 8, 11.

Lo scarto più grande tra un valore d'origine (12) e un valore calcolato (11.236) è 0.764 (ossia 6.4% di 12).