



UNIVERSITÀ DEGLI STUDI DI URBINO "CARLO BO"

Facoltà di Scienze e Tecnologie
Corso di Laurea in Informatica Applicata

Tesi di Laurea

**ANALISI DEL GENOMA PER
L'INDIVIDUAZIONE DI MUTAZIONI PUTATIVE
RESPONSABILI DI TRASLOCAZIONE
NUCLEARE**

Relatore:
Chiar.mo Prof. Alessandro Bogliolo

Candidato:
Manuel Finotto

Correlatore:
Ing. Valerio Freschi

Anno Accademico 2007-2008

A Silvia, Daniele, Alice e Simone

Indice

1	Introduzione	1
1.1	Contesto e motivazioni	1
1.2	Contributo	2
1.3	Organizzazione della tesi	2
2	Background	4
2.1	Il genoma	4
2.1.1	I codoni	6
2.2	I meccanismi di mutazione e i tandem repeats	6
2.3	NES e NLS	8
2.4	Le banche dati	9
2.4.1	Il formato FASTA	10
2.4.2	Il formato GenBank	14
3	Approccio	17
3.1	Approccio diretto	17
3.2	Approccio inverso	19
4	Implementazione	21
4.1	Specifiche	21
4.2	Tool flow	23
5	Risultati e conclusioni	30
A	Organizzazione dei file	34
B	Script get_CDS	35
	Bibliografia	37
	Ringraziamenti	39

Elenco delle figure

2.1	Dal genoma alle proteine [5]	5
2.2	Poros nucleare	8
3.1	Approccio diretto. Flow chart con la previsione computazionale della generazione di NES putativi attraverso duplicazioni in tandem nel genoma umano. Il numero di sequenze che vengono processate ai passi 4 e 5 si riferiscono ad unità di ripetizione di lunghezza 3.	18
3.2	Approccio inverso. Sequenza logica delle operazioni per la localizzazione di segnali NES o NLS nel genoma.	20
4.1	Flusso dei file nell'applicazione	22
4.2	Esempio inserimento ripetizione in testa alla sequenza nucleotidica	25
4.3	Esempio di potenziale ottimizzazione dello spazio di ricerca . . .	25

Elenco delle tabelle

2.1	Associazione aminoacido-codone	7
2.2	Codici FASTA acidi nucleici	12
2.3	Codici FASTA aminoacidi	13
4.1	Parametri riga di comando	23

Capitolo 1

Introduzione

1.1 Contesto e motivazioni

Il presente lavoro va inquadrato nell'ambito della bioinformatica. La bioinformatica è una disciplina scientifica dedicata alla risoluzione di problemi biologici a livello molecolare con metodi informatici.

Numerose possono essere le applicazioni della bioinformatica. Qui citeremo solo un aspetto della medicina molecolare. Si ritiene che molte malattie siano associate ad una componente genetica. La malattia, infatti, può essere ereditaria (sono note circa 3000-4000 malattie genetiche come la fibrosi cistica, alcune forme di diabete, etc) oppure essere il risultato di fattori ambientali che causano alterazioni del genoma (tumori, malattie cardiache, ecc). Una branca della bioinformatica studia quali geni siano associati a diverse malattie per capirne più chiaramente le basi molecolari con lo scopo di migliorarne la prevenzione e la cura.

In questa ricerca si andranno ad individuare sequenze di aminoacidi (*segnali*) responsabili di traslocazione nucleare. Questo tipo di ricerca è di interesse in quanto si è scoperto, ad esempio, che c'è una proteina, la nucleofosmina, che nelle cellule sane si trova nel nucleo, laddove è contenuto anche il DNA. Nella leucemia mieloide acuta, invece, la nucleofosmina si trova nel citoplasma, ovvero fuori dal nucleo cellulare. Poiché quello non è il suo posto, la comparsa del tumore del sangue viene favorita. È stato recentemente scoperto che la traslocazione è determinata da una mutazione (più precisamente dalla duplicazione di 4 nucleotidi nella sequenza che codifica la proteina) che fa comparire un NES (nuclear export signal) dove non dovrebbe esserci.[4]

Le mutazioni che può subire il DNA sono molteplici, qualsiasi alterazione della catena nucleotidica può potenzialmente portare alla modificazione della sequenza polipeptidica da essa derivante. In questa tesi verranno trattate le

anomalie genetiche, denominate *tandem repeat*, costituite da regioni di DNA con sequenze di due o più nucleotidi ripetute una di seguito all'altra. È stato infatti dimostrato che le duplicazioni di regioni di DNA sono tra le mutazioni più probabili e giocano un ruolo fondamentale nell'evoluzione. Verranno inoltre presi in esame gli errori di decodifica, detti *frame shift*, potenzialmente indotti da fenomeni di duplicazione di un numero di nucleotidi non divisibile per 3. Entrambe le anomalie possono dare luogo a segnali di traslocazione nucleare erronei.

1.2 Contributo

Il problema che si vuole affrontare è la ricerca delle alterazioni genetiche, di tipo ripetizioni in tandem e frameshift, responsabili di traslocazione nucleare. Data la dimensione del genoma umano (v. 2.1) e delle possibili sequenze codificanti segnali di traslocazione nucleare (v. 2.3) il problema della complessità computazionale diviene critico.

Al fine di individuare i *segnali* su citati è stata sviluppata un'applicazione software in grado di alterare la sequenza nucleotidica di uno o più geni e di ricercare nelle derivate proteine le sequenze di aminoacidi responsabili di traslocazione nucleare. L'idea di base è realizzare un programma in grado di alterare il genoma umano e di analizzare le conseguenze di tale cambiamento, evitando però di generare esplosioni combinatorie che comprometterebbero l'efficienza e l'applicabilità di questa possibile soluzione.

Il problema delle esplosioni combinatorie è stato lo spunto per il presente lavoro in quanto sono già disponibili software che perseguono lo stesso obiettivo, ma sono affetti da importanti limitazioni dovute appunto ad una crescita della complessità esponenziale con il crescere delle lunghezze dei segnali da ricercare.

Il metodo qui sviluppato, denominato approccio inverso, risolve, capovolgendo gli approcci oggi disponibili, il problema della crescita verticale della complessità in relazione alle sequenze da ricercare e raggiunge gli obiettivi di identificazione proposti.

Sono stati sviluppati anche dei *tool* software di supporto per poter estrarre informazioni utili dalle banche dati (v. 2.4), per poterli poi utilizzare nell'applicazione principale.

1.3 Organizzazione della tesi

Il resto della tesi è organizzata in quattro capitoli.

Capitolo 2. Vengono fornite le conoscenze di base nell'ambito della biologia in modo da comprendere le terminologie, le problematiche e gli obiettivi del lavoro svolto.

Capitolo 3. Si descrivono due possibili approcci alla risoluzione dei problemi definiti come obiettivi della ricerca. Il primo approccio rappresenta una soluzione già sviluppata e non coperta in questo documento. Il secondo approccio rappresenta invece quello perseguito nel presente lavoro.

Capitolo 4. L'implementazione software dell'applicazione viene esposta in modo dettagliato. Si descrivono le specifiche di progetto e le soluzioni algoritmiche. Vengono illustrate le sequenze operative per l'utilizzo dell'applicazione.

Capitolo 5. Vengono riportati i risultati ottenuti e vengono fatte le considerazioni finali su quanto raggiunto.

Capitolo 2

Background

Come già anticipato, il presente lavoro è un'applicazione della bioinformatica alla medicina molecolare, ovvero a quel settore che studia i rapporti che intercorrono tra la patologia e il patrimonio genetico umano e quindi tutte le malattie dovute a modificazioni del codice genetico oppure a errori di trascrizione o di traduzione dell'informazione o, infine, a errori della regolazione dell'espressione genetica.

Una corretta informazione genetica è la condizione essenziale per il mantenimento di una costante composizione chimica dell'organismo, per la conservazione della sue normali strutture molecolari e per il regolare andamento di tutte le reazioni metaboliche che in esso si svolgono. Ogni alterazione dell'informazione genetica a livello dei geni strutturali o di regolazione modifica, perciò, questi equilibri e può tradursi in un errore metabolico più o meno dannoso per l'economia generale dell'organismo.

Alcune mutazioni non modificano la reattività dell'organismo all'ambiente e non sono quindi causa di malattie, altre sono così gravi ed estese da non essere compatibili con la vita. Tra questi due limiti sono note numerose deviazioni metaboliche capaci di determinare la comparsa di malattie, sia limitate a tessuti o a organi, sia interessanti l'intero organismo, ma compatibili con la vita almeno per un certo tempo. In queste malattie il guasto metabolico è praticamente irreversibile, al contrario di quanto si verifica per le malattie acquisite (infezioni, intossicazioni, etc.) in cui le modificazioni metaboliche hanno carattere transitorio e scompaiono con il cessare della causa che ha originato la malattia.

2.1 Il genoma

Il genoma rappresenta la totalità del patrimonio genetico di una specie ed è costituito da tutte le sequenze di DNA di cui l'organismo può essere dotato,

sia quelle codificanti per le proteine, che quelle non codificanti.

Nel caso dei procarioti, il genoma è rappresentato da un unico cromosoma circolare, mentre per gli eucarioti con il termine genoma nucleare si intende una serie completa di cromosomi che risiedono nel nucleo della cellula. Si parla di genoma nucleare perché gli eucarioti possiedono anche un genoma mitocondriale, e le piante presentano anche un genoma cloroplastico.

La funzione primaria del materiale genetico è quella di contenere le informazioni, immagazzinate nella sequenza delle basi azotate del DNA, che sono necessarie per produrre i caratteri dell'organismo, codificati dai geni. La porzione di DNA non codificante negli eucarioti è spesso molto vasta ed arriva, nell'uomo, a rappresentare il 99% dell'intero genoma. La funzione di tali sequenze è al momento oggetto di discussione ed indagine scientifica.

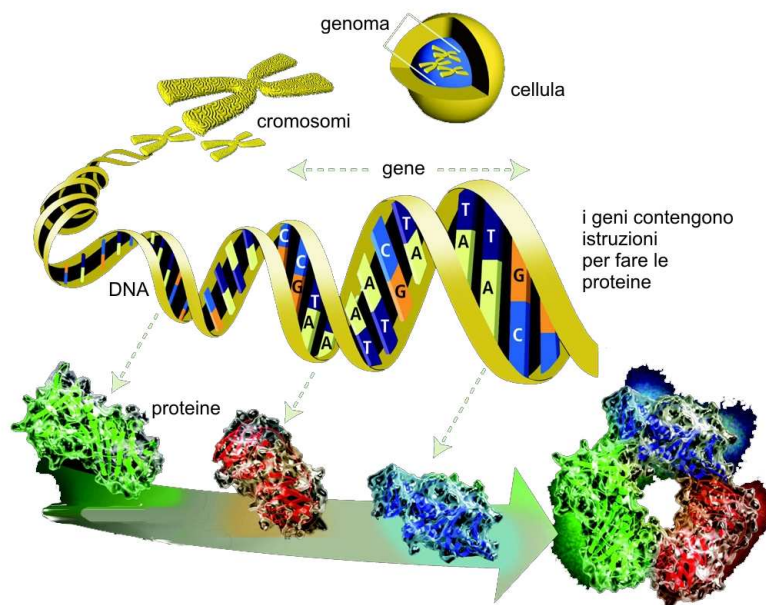


Figura 2.1: Dal genoma alle proteine [5]

Le specie eucariotiche contengono una o più serie cromosomiche, ciascuna delle quali è costituita da cromosomi differenti. La specie umana, per esempio, è costituita da due serie di 23 cromosomi ciascuna (che fanno un totale di 46 cromosomi). La quantità di DNA delle specie eucariotiche è molto più alta della quantità presente nelle specie procariotiche, e permette di codificare molti più geni. All'interno delle specie eucariotiche è presente una notevole diversità

nella dimensione dei genomi, che spesso non è correlata alla complessità della specie.

Grazie allo Human Genome Project, il genoma dell'uomo è stato completamente sequenziato nel 2002.

2.1.1 I codoni

Il codone (in inglese codon) viene definito come una sequenza specifica di 3 nucleotidi (tripletta) lungo l'mRNA che codifica l'informazione per l'inserimento di uno specifico amminoacido durante la sintesi proteica o per la fine della stessa (definito codone di Stop). Il codone è alla base del codice genetico.

Il codice esatto è costituito da triplette (il minimo sufficiente a coprire il set di amminoacidi). Se fossero state sequenze di 2 nucleotidi infatti non sarebbero state sufficienti, perché 4 nucleotidi presi 2 alla volta avrebbero originato $4^2 = 16$ combinazioni. Il codice triplo invece origina $4^3 = 64$ combinazioni, che sono tuttavia più che sufficienti per codificare tutti gli amminoacidi. Infatti molti di essi sono codificati da più di un codone. Questa ridondanza è definita come degenerazione del codice. Vi sono anche 3 codoni ai quali non corrisponde nessun amminoacido e in corrispondenza di essi la sintesi proteica cessa (codoni di stop).

In tabella 2.1 sono riportati i 64 codoni e gli amminoacidi corrispondenti ad ognuno di essi. Per il significato dei codici usati per gli amminoacidi vedere tabella 2.3.

2.2 I meccanismi di mutazione e i tandem repeats

A livello molecolare, un'alterata informazione genetica può condurre ad un errore nella sintesi delle proteine sia causando una modificazione della loro struttura primaria (mutazione a livello del gene strutturale), sia modificando la velocità di sintesi della molecola (mutazione a livello dei geni regolatori). Nel primo caso verranno prodotte proteine strutturalmente anomale; nel secondo caso ci sarà una variazione quantitativa di proteine a struttura normale.

Poiché una catena polipeptidica è formata da una sequenza di centinaia di amminoacidi e ognuno di essi può essere sostituito da un altro come risultato della mutazione genica, possono formarsi innumerevoli alterazioni strutturali della proteina. Le proprietà di queste proteine possono differire l'una dall'altra in funzione del particolare amminoacido che è stato sostituito o del particolare sito nella catena polipeptidica in cui si è verificata la sostituzione.

Aminoacido	Codone
A	GCT GCC GCA GCG
L	TTA TTG CTT CTC CTA CTG
R	CGT CGC CGA CGG AGA AGG
K	AAA AAG
N	AAT AAC
M	ATG
D	GAT GAC
F	TTT TTC
C	TGT TGC
P	CCT CCC CCA CCG
Q	CAA CAG
S	TCT TCC TCA TCG AGT AGC
E	GAA GAG
T	ACT ACC ACA ACG
G	GGT GGC GGA GGG
W	TGG
H	CAT CAC
Y	TAT TAC
I	ATT ATC ATA
V	GTT GTC GTA GTG
> (start)	ATG
< (stop)	TAG TGA TAA

Tabella 2.1: Associazione aminoacido-codone

In genetica e biologia molecolare si definiscono ripetizioni in tandem tutte le regioni di DNA costituite da sequenze di due o più nucleotidi ripetute una di seguito all'altra. Ad esempio la sequenza

ATTCGATTCGATTCGATTCG

è una ripetizione in tandem di ATTCG (ripetuta quattro volte).

Nel presente lavoro si andranno ad inserire artificialmente delle ripetizioni di nucleotidi di lunghezza variabile nel gene per poi tradurlo nella relativa proteina. Una volta ottenuta la proteina si andranno a cercare i segnali NES e NLS (v. 2.3).

Viene anche data la possibilità di introdurre dei *frameshift*. Le mutazioni frameshift (dall'inglese, spostamento dell'ordine di lettura) sono dette anche mutazioni da scivolamento. Sono causate dall'aggiunta (inserzione) o l'eliminazione (delezione) di uno o pochi nucleotidi. Nel caso di inserzione o delezione di nucleotidi in numero non multiplo di tre, viene alterato l'ordine di lettura

(in inglese frame) di tutti i codoni successivi nell'ordine a quello inserito o cancellato.

2.3 NES e NLS

Il nucleo è separato dal citoplasma tramite 2 strati di membrana unitaria perforati da una moltitudine di pori attraverso i quali possono passare una serie di molecole solubili. La struttura di questi pori è complessa e media un trasporto molecolare abbastanza selettivo.

Nei pori nucleari risiede uno tra i più grossi complessi molecolari conosciuti. Il complesso del poro nucleare (NPC) ha un diametro di circa 200nm, un peso molecolare di 125KDa ed è formato da una trentina di proteine. Ha una simmetria ottagonale, nel lato interno possiede otto prolungamenti uniti tra loro all'estremità distale dall'anello terminale a formare una struttura chiamata basket nucleare. Il complesso del poro nucleare è fissato alla membrana tramite due anelli, uno nucleare e uno citoplasmatico, da quest'ultimo partono 8 filamenti che sporgono nel citoplasma. Infine, al centro del poro c'è un tappo proteico responsabile della selettività che lascia uno spazio di circa 9nm.

Lo spazio che lascia il poro è sufficientemente ampio da consentire il passaggio di una vasta gamma di molecole, ma difficilmente permette transito di proteine foldate, queste devono prima essere srotolate.

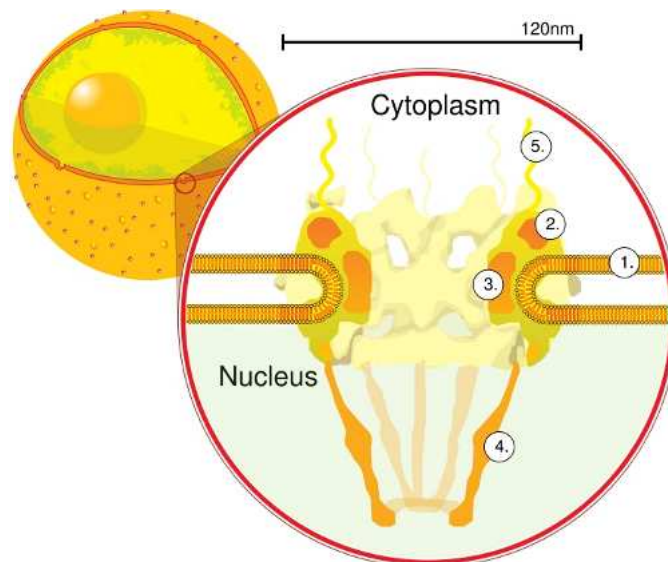


Figura 2.2: Poro nucleare

Il cargo che deve attraversare il poro necessita dell'intervento di altre proteine: le importine e le esportine. Per l'esportazione occorre che la proteina abbia una sequenza specifica detta **NES** (segnale di esportazione nucleare) che viene riconosciuta dall'esportina. Dopo il legame tra proteina ed esportina si unisce anche una GTPasi chiamata Ran, per formare il complesso necessario per l'esportazione.

Il meccanismo di importazione è leggermente più complicato rispetto a quello di esportazione. Anche le proteine che devono essere importate possiedono una sequenza di riconoscimento, che in questo caso si chiama **NLS** (sequenza di localizzazione nucleare), data da una serie di lisine. A questa sequenza si legano un'importina- α ed un'importina- β che permettono l'importazione nel nucleo. L'importina- β ha una struttura caratteristica, data da una serie di α -eliche disposte a zig-zag come una sorta di fisarmonica, che conferiscono una notevole flessibilità alla molecola.

Le sequenze NES e NLS sono quelle che verranno cercate dal software sviluppato in questa tesi. Anticipiamo che in realtà il software è in grado di ricercare una qualsiasi sequenza che venga fornita, ma per i fini specifici di questa ricerca verranno fornite solo sequenze NES e NLS.

2.4 Le banche dati

Una delle attività principali dei bioinformatici consiste nella progettazione, costruzione e uso (come nel presente lavoro) di banche dati di interesse biologico. Una banca dati raccoglie dati e informazioni derivati da esperimenti di laboratorio, da esperimenti in silico (cioè utilizzare il dato informatico come punto di partenza per gli esperimenti in vitro. Si dice in silico, in quanto i processori dei calcolatori sono costituiti da silicio) e dalla letteratura scientifica.

Le banche dati sono progettate come contenitori costruiti per immagazzinare dati in modo efficiente e razionale al fine di renderli facilmente accessibili a tutti gli utenti: ricercatori, medici, studenti, etc.

Una banca dati è costituita da voci (in inglese entry) ciascuna contenente informazioni sull'oggetto caratteristico della banca dati (ad esempio: sequenze nucleotidiche o referenze bibliografiche) insieme a tutte le altre informazioni che si riferiscono a quella entry in particolare).

Una entry di una banca dati di sequenze nucleotidiche potrebbe contenere, oltre alla sequenza di una molecola di DNA, il nome dell'organismo cui la sequenza appartiene, la lista degli articoli che riportano dati su quella sequenza, le caratteristiche funzionali (cioè si tratta di un gene o di una sequenza non

codificante) e ogni altra informazione ritenuta di interesse.

Le banche dati possono essere di due tipi:

- primarie
- specializzate

Le banche dati primarie contengono informazioni e annotazioni delle sequenze nucleotidiche e proteiche, strutture del DNA e proteine e dati sull'espressione di DNA e proteine.

Le principali banche dati primarie sono: la EMBL datalibrary, la GenBank e la DDBJ. La EMBL datalibrary è la banca dati europea costituita nel 1980 nel laboratorio Europeo di Biologia Molecolare di Heidelberg (Germania). La GenBank è la corrispondente banca americana costituita nel 1982 e la DDBJ è la corrispondente Giapponese. Fra le tre banche dati è stato stipulato un accordo internazionale per cui il contenuto dei dati di sequenza presenti nelle tre banche dati è quasi del tutto coincidente in quanto gli aggiornamenti quotidiani apportati in ciascuna banca dati vengono automaticamente trasmessi alle altre due.

Le banche dati specializzate si sono sviluppate successivamente e raccolgono insieme di dati omogenei dal punto di vista tassonomico e/o funzionale disponibili nelle Banche dati Primarie e/o in Letteratura, o derivanti da vari approcci sperimentali, rivisti e annotati con informazioni di valore aggiunto.

I dati utilizzati in questa tesi provengono principalmente da GenBank.

2.4.1 Il formato FASTA

In bioinformatica il formato FASTA (noto anche come formato di Pearson) è un formato basato su file di testo. Possono essere rappresentate sequenze di acidi nucleici o sequenze peptidiche in cui i nucleotidi o gli aminoacidi sono rappresentati utilizzando codici a lettere singole. Il formato consente anche ai nomi della sequenza o a commenti di precedere le sequenze.

La semplicità del codice FASTA rende facile la manipolazione e la scansione delle sequenze usando applicazioni software per file di testo.

Una sequenza in formato FASTA comincia con una singola riga di descrizione, seguita dalle righe di sequenze di dati. La riga di descrizione si distingue dalla sequenza di dati perché come carattere nella prima colonna ha il simbolo '>' (maggiore di). La parola che segue il simbolo '>' è l'identificatore della sequenza, il resto della riga è la descrizione (entrambe sono facoltative).

Non è consentito nessuno spazio tra il simbolo '>' e la prima lettera dell'identificatore. E' raccomandato che tutte le righe del testo siano più corte di 80 caratteri. La sequenza termina se un'altra riga che inizia con '>' viene incontrata; questo indica l'inizio di un'altra sequenza. Qui di seguito un semplice esempio di una sequenza in formato FASTA:

```
>gi|5524211|gb|AAD44166.1| cytochrome b [Elephas maximus maximus]
LCLYTHIGRNIYYGSYLYSETWNTGIMLLITMATAFMGYVLPWQMSFWGATVITNLFSaipyi
GTNLVEWIWGGFSVDKATLNRFFAFHFILPFTMVALAGVHLTFLHETGSNNPLGLTSDSDKIPFH
PYYTIKDFLGLLILLLLLLLLLLALLSPDMLGDPDNHMPADPLNTPHLIKPEWYFLFAYAILRSVPN
KGGVLALFLSIVILGLMPFLHTSKHRSMMLRPLSQALFWTLTMDLLTLTWIGSQPVEYPYTIIG
QMASILYFSIILAFLPIAGXIENY
```

Le sequenze di dati possono essere sequenze di aminoacidi o sequenze di acidi nucleici e possono contenere vuoti o caratteri di allineamento. Le sequenze devono essere rappresentate con codici nello standard IUB/IUPAC per gli aminoacidi e gli acidi nucleici, le eccezioni sono:

- vengono accettate le lettere minuscole e sono mappate in lettere maiuscole
- il singolo trattino può essere usato per rappresentare un carattere vuoto
- nella sequenza di aminoacidi, U e * sono lettere accettate

I caratteri numerici non sono permessi, ma sono utilizzati in alcuni database per indicare la posizione nella sequenza.

I codici per gli acidi nucleici sono riportati in tabella 2.2, mentre i codici per gli aminoacidi sono riportati in tabella 2.3.

Codice	Significato
A	Adenosina
C	Citosina
G	Guanina
T	Timina
U	Uracile
R	G A (puRina)
Y	T C (pirimidina)
K	G T (chetone)
M	A C (gruppo amino)
S	G C (interazione forte)
W	A T (interazione debole)
B	G T C (non A)(B viene dopo A)
D	G A T (non C)(D viene dopo C)
H	A C T (non G)(H viene dopo G)
V	G C A (non T, non U)(V viene dopo U)
N	A G C T (qualsiasi)
X	mascherata
-	vuoto di qualsiasi lunghezza

Tabella 2.2: Codici FASTA acidi nucleici

Codice	Significato
A	Alanina
B	Acido aspartico o Asparagina
C	Cisteina
D	Acido aspartico
E	Acido glutamminico
F	Fenilalanina
G	Glicina
H	Istidina
I	Isoleucina
K	Lisina
L	Leucina
M	Metionina
N	Asparagina
O	Pirrolisina
P	Prolina
Q	Glutammina
R	Arginina
S	Serina
T	Treonina
U	Selenocisteina
V	Valina
W	Triptofano
Y	Tirosina
Z	Acido Glutamminico o Glutammina
X	qualsiasi
	fine traduzione
-	lacuna di lunghezza indeterminata

Tabella 2.3: Codici FASTA aminoacidi

2.4.2 Il formato GenBank

Vedremo più avanti, nel capitolo 4, che i dati contenuti nel database GenBank ci offrono informazioni utili al fine di ottimizzare le ricerche. Verrà sfruttato il fatto che non tutta la sequenza nucleotidica del gene è codificante. In biologia molecolare si definisce DNA non codificante ogni sequenza di DNA in un genoma non soggetta a trascrizione in RNA o rimossa dall'mRNA prima della traduzione (introni) e che viene perciò considerata, allo stato attuale delle conoscenze nel campo, apparentemente priva di funzione.

Nei database GenBank è possibile trovare le informazioni riguardanti la parte del gene codificante, detta *coding sequence*.

Un esempio di *record* appartenente al database GenBank è il seguente:

```
LOCUS      SCU49845      5028 bp      DNA      PLN      21-JUN-1999
DEFINITION Saccharomyces cerevisiae TCP1-beta gene, partial cds,
            and Axl2p(AXL2) and Rev7p(REV7) genes, complete cds.
ACCESSION  U49845
VERSION    U49845.1  GI:1293613
KEYWORDS   .
SOURCE     Saccharomyces cerevisiae (baker's yeast)
  ORGANISM Saccharomyces cerevisiae
            Eukaryota; Fungi; Ascomycota; Saccharomycotina;
            Saccharomycetes; Saccharomycetales;
            Saccharomycetaceae; Saccharomyces.
REFERENCE  1 (bases 1 to 5028)
  AUTHORS  Torpey,L.E., Gibbs,P.E., Nelson,J. and Lawrence,C.W.
  TITLE    Cloning and sequence of REV7, a gene whose function
            is required for DNA damage-induced mutagenesis in
            Saccharomyces cerevisiae
  JOURNAL  Yeast 10 (11), 1503-1509 (1994)
  PUBMED   7871890
REFERENCE  2 (bases 1 to 5028)
  AUTHORS  Roemer,T., Madden,K., Chang,J. and Snyder,M.
  TITLE    Selection of axial growth sites in yeast requires
            Axl2p, a novel plasma membrane glycoprotein
  JOURNAL  Genes Dev. 10 (7), 777-793 (1996)
  PUBMED   8846915
REFERENCE  3 (bases 1 to 5028)
  AUTHORS  Roemer,T.
  TITLE    Direct Submission
  JOURNAL  Submitted (22-FEB-1996) Terry Roemer, Biology, Yale
            University, New Haven, CT, USA
```

FEATURES	Location/Qualifiers
source	1..5028 /organism="Saccharomyces cerevisiae" /db_xref="taxon:4932" /chromosome="IX" /map="9"
CDS	<1..206 /codon_start=3 /product="TCP1-beta" /protein_id="AAA98665.1" /db_xref="GI:1293614" /translation="SSIYNGISTSGLDLNNGTIADMRQLGIVES YKLKRAVSSASEAAEVLLRVDNIIRARPRTANRQHM"
gene	687..3158 /gene="AXL2"
CDS	687..3158 /gene="AXL2" /note="plasma membrane glycoprotein" /codon_start=1 /function="required for axial budding pattern of S. cerevisiae" /product="Axl2p" /protein_id="AAA98666.1" /db_xref="GI:1293615" /translation="MTQLQISLLLTATISLLHLVVATPYEAYPI GKQYPPVARVNESFTFQISNDTYKSSVDKTAQITYNCFDLPSWL SFDSSSRTFSGEPSSDLLSDANTTLFNVILEGTDSDSTSLNN TYQFVVTNRPSISLSSDFNLLALLKNYGYTNGKNALKLDPNEVF NVTFRSMFTNEESIVSYGRSGLYNAPLPNWLFFDSGELKFTG TAPVINSIAIPETSYFVVIATDIEGFSAVEVEFELVIGAHQLT TSIQNSLIINVTDGTGNVSYDLPLNYVYVLDLDDPISSDKLGSINLL DAPDWVALDNATISGSVPDELLGKNSNPANFSVSIYDYTGVDIY FNFEVSTTDLFAISSLPNINATRGEWFSYYFLPSQFTDYVNTN VSLEFTNSSQDHDWVKFQSSNLTLAGEVPKNFDKLSLGLKANQG SQSQELYFNIIGMDSKITHSNHSANATSTRSSHSTSTSSYTSS TYTAKISSTSAAATSSAPAALPAANKTSSHNKKAVAIACGVAIP LGVILVALICFLIFWRRRRRENPDENLPHAISGPDLNPNANKPN QENATPLNPFDDDASSYDDTSIARRLAALNTLKLDNHSATESD ISSVDEKRDSLGMNTYNDQFQSQSKEELLAKPPVQPPESPFFD PQNRSSSVYMDSEPAVNKSWRYTGNLSPVSDIVRDSYGSQKTVD TEKLFDELAPEKEKRTSRDVTMSSLDPWNSNISPSVPRKSVTPS PYNVTKHRNRHLQNIQDSQSGKNGITPTTMSSTSSDDFVVPKDG ENFCWVHMEPDRRPSKKRLVDFSNKSNVNVGQVKDIHGRIPEM L"

```

gene          complement(3300..4037)
              /gene="REV7"
CDS           complement(3300..4037)
              /gene="REV7"
              /codon_start=1
              /product="Rev7p"
              /protein_id="AAA98667.1"
              /db_xref="GI:1293616"
              /translation="MNRWVEKWLRVYLKCYINLILFYRNVYPPQ
SFDYTTYQSFNLPQFVPINRHPALIDYIEELILDVLSKLTHVYR
FSICIIINKNDLCIEKYVLDVDFSELQHVDKDDQIITETEVDFEFR
SSLNSLIMHLEKLPKVNDDTITFEAVINAELELGHKLDNRNRV
DSLEEKAEIERDSNWVKCQEDENLPDNGFQPPKIKLTSLVGSD
VGPLIIHQFSEKLI SGDDKILNGVYSQYEEGESIFGSLF"

ORIGIN
    1 gatcctccat atacaacggt atctccacct caggtttaga tctcaacaac
    61 cgcacatgag acagttaggt atcgtcgaga gttacaagct aaaacgagca
   121 ctgcatctga agccgctgaa gttctactaa ggggtgataa catcatccgt
   181 gaaccgcaa tagacaacat atgtaacata ttaggatata acctcgaaaa
   241 ccacactgtc attattataa ttagaaacag aacgcaaaaa ttatccacta
   301 agacgcgaaa aaaaaagaac aacgcgtcat agaacttttg gcaattcgcg
   361 attttgcaa cttatgtttc ctcttcgagc agtactcgag ccctgtctca
   421 aatacccatc gtaggtatgg ttaaagatag catctccaca acctcaaagc
   481 gagtcgccct ctttgcga gtaattttca cttttcatat gagaacttat
   541 ttactctca catcctgtag tgattgacac tgcaacagcc accatcacta
   601 acaattactt aatagaaaaa ttatatcttc ctgaaacga tttcctgctt
   661 cgtatatcaa gaagcattca cttacatga cacagcttca gatttcatta
   721 ctactatata actactccat ctagtagtgg ccacgcccta tgaggcatat

...

//

```

Una descrizione dettagliata del significato dei campi presenti nei record Genbank può essere trovata nel web all'indirizzo:

<http://www.ncbi.nlm.nih.gov/Sitemap/samplerecord.html#LocusNameB>

Capitolo 3

Approccio

Al fine di esporre chiaramente le specifiche del problema che viene affrontato si precisa quanto segue:

- Anche i geni non mutati (wild-type) possono avere delle sequenze di nucleotidi che codificano una sequenza di aminoacidi tale da rappresentare dei segnali NES o NLS
- I geni possono essere soggetti a mutazioni geniche come ad esempio *tandem repeat* o *frameshift* le quali, alterando le sequenze nucleotidiche, possono codificare una sequenza di aminoacidi tale da rappresentare dei segnali NES o NLS
- L'obiettivo della ricerca è lo sviluppo di un'applicazione software per l'identificazione delle mutazioni del genoma responsabili di generazione di segnali NES e NLS. Le mutazioni geniche prese in considerazione sono i *tandem repeat* ed i *frameshift*.

Come spesso accade un problema può essere affrontato in modi diversi. Qui di seguito vengono presentati due approcci denominati *diretto* e *inverso*. L'approccio diretto è già stato sviluppato da un lavoro[6] che precede il presente e viene descritto al fine di evidenziare i vantaggi qui sviluppati tramite l'approccio inverso.

3.1 Approccio diretto

Al fine di esplorare le ipotetiche duplicazioni potenzialmente causa di noti segnali NES nel genoma umano è stato usato un approccio computazionale (figura 3.1) basato su di un software sviluppato ad hoc.

Come primo passo si accede al database[7] delle sequenze NES disponibile in rete (<http://www.cbs.dtu.dk/databases/NESbase>).

Per prevenire un'esplosione combinatoria durante la codifica inversa (retrotraduzione) indotta dalla ridondanza del codice genetico sono state utilizzate sequenze lunghe al più 15 aminoacidi.

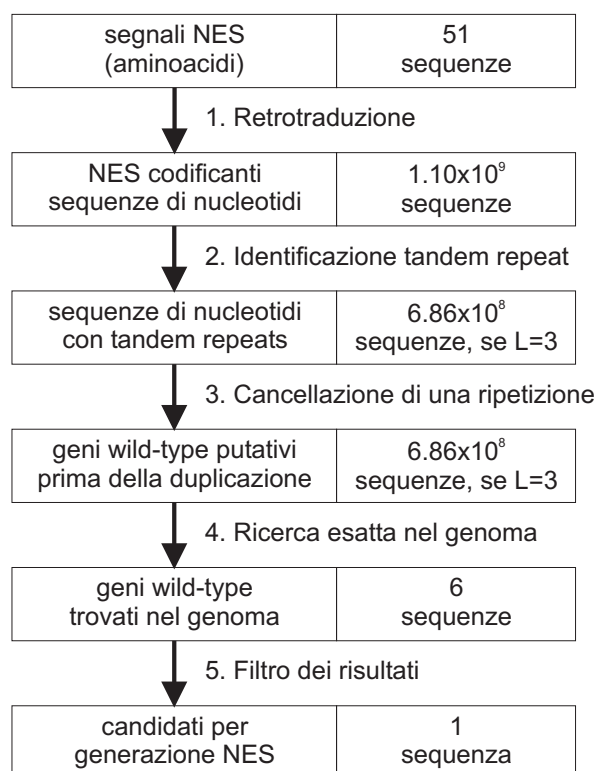


Figura 3.1: Approccio diretto. Flow chart con la previsione computazionale della generazione di NES putativi attraverso duplicazioni in tandem nel genoma umano. Il numero di sequenze che vengono processate ai passi 4 e 5 si riferiscono ad unità di ripetizione di lunghezza 3.

Un totale di 28 sequenze NES sono state retrotradotte, considerando tutte le possibili combinazioni di codoni. Oltre a queste 28 sequenze NES da database, vengono anche analizzati i termini COOH di 23 noti mutanti leucemici NPM1 descritti da Falini[8].

Dopo la retrotraduzione, ogni segnale è stato processato per la ricerca di tandem repeat con periodo di 3, 4, 5, o 6 nucleotidi come anche duplicazioni multiple esatte, ad esempio 'TCTG TCTG TCTG'.

A fronte dell'identificazione la sua ultima duplicazione viene eliminata e la sua sequenza di DNA viene ricercata nelle sequenze codificanti del genoma umano (NCBI repository: ftp://ftp.ncbi.nih.gov/genomes/H_sapiens/rna.fa.gz)

nel tentativo di trovare i geni wild-type (cioè i geni non mutati) con il potenziale di generare NES dopo degli eventi di duplicazione.

L'insieme risultante dei geni viene filtrato per escludere falsi riscontri, ad esempio riscoprire i NES originali.

3.2 Approccio inverso

Come si può vedere nella descrizione dell'approccio diretto il limite principale è l'esplosione combinatoria, durante la retrotraduzione, indotta dalla ridondanza del codice genetico, infatti sono state utilizzate sequenze lunghe al più 15 aminoacidi.

Un'osservazione importante riguarda la struttura formale delle sequenze NLS rispetto alle NES. I segnali NLS, oltre ad essere più numerosi dei NES, possono essere espressi anche tramite *espressioni regolari*. Le espressioni regolari (in inglese regular expression, che può trovarsi abbreviata in regex, rex o RE) sono sintassi attraverso le quali si possono rappresentare insiemi di stringhe.

Essendo l'identificazione di segnali NLS parte degli obiettivi di questa tesi, l'utilizzo dell'approccio diretto comporterebbe, per via delle espressioni regolari, un'esplosione combinatoria tale da rendere impercorribile questa strada.

L'approccio inverso non pone i limiti presenti nell'approccio diretto in termini di esplosioni combinatorie e relativo carico computazionale.

In figura 3.2 viene riportata la sequenza logica con cui si procede nell'approccio indiretto. Nella realtà l'applicazione software processa un gene alla volta, quindi i passi da 1 a 3 vengono eseguiti gene per gene scorrendo l'intero genoma o un sottoinsieme di esso.

Dal punto di vista algoritmico e computazionale il passo 3 rappresenta quello più critico, sia per i problemi insiti nello sviluppo di algoritmi di ricerca efficienti sia per la necessità di eseguire ricerche di espressioni regolari.

Il passo 2 riveste un ruolo importante in quanto è responsabile di minimizzare la lunghezza delle sequenze affette dalle mutazioni geniche. Il passo numero 2 ha un impatto diretto sul carico computazionale demandato al passo 3.

Per finire il passo 1 aggiunge semplicemente dei tandem repeat nella sequenza nucleotidica dei geni, i frameshift vengono considerati nel momento

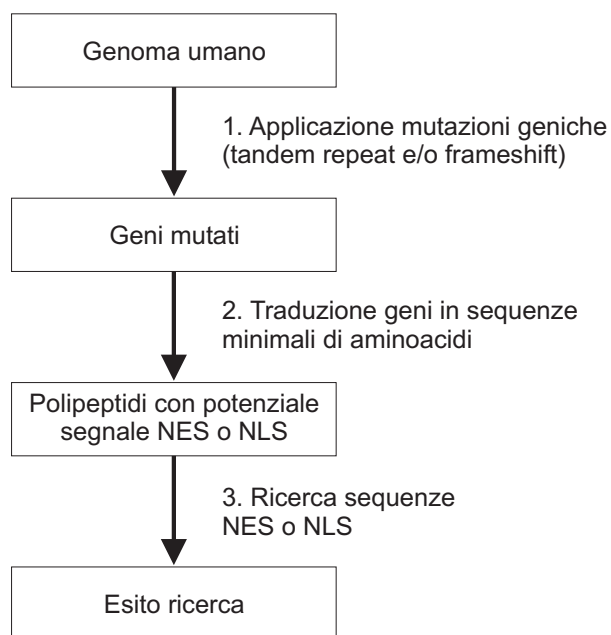


Figura 3.2: Approccio inverso. Sequenza logica delle operazioni per la localizzazione di segnali NES o NLS nel genoma.

della traduzione da codoni ad aminoacidi.

Si ribadisce il fatto che nell'approccio inverso non viene dato nessun limite alla lunghezza dei segnali da ricercare, ovvero a differenza di quanto accade per l'approccio diretto non c'è un'analogia esplosione combinatoria.

Nel capitolo 4 viene descritta nel dettaglio l'implementazione dell'approccio inverso.

Capitolo 4

Implementazione

Il software applicativo è stato sviluppato e implementato per ambiente Linux OS in linguaggio C.

Nel dettaglio la distribuzione linux utilizzata è stata la OpenSUSE 10.2(i586)[9] con kernel Linux 2.6.18.2 e compilatore *GNU gcc* 4.1.2[10]. E' stato scritto anche uno script *GNU awk*[11] (v. appendice B) per l'estrazione delle *coding DNA sequence* da file in formato GenBank[12].

4.1 Specifiche

L'applicazione sviluppata ha un'interfaccia a console e i comandi vengono passati tramite parametri sulla riga di comando.

I file in input all'applicazione sono:

- File in formato FASTA (v. sezione 2.4.1) contenente uno o più geni, possibilmente anche l'intero genoma. Input obbligatorio.
- File in formato proprietario (v. appendice B) contenente le *coding sequence* dei geni. Input opzionale.
- File in formato testo con lista di espressioni regolari riconoscibili da *GNU grep*. Contiene i segnali NES, NLS o più in generale qualsiasi tipo di pattern. Input obbligatorio.

I file di output:

- File in formato testo simile al formato fasta dove vengono riportati, solo in caso di riscontro, il gene mutato con l'istanza dell'espressione regolare che ha generato il match e il relativo pattern in forma di espressione regolare.

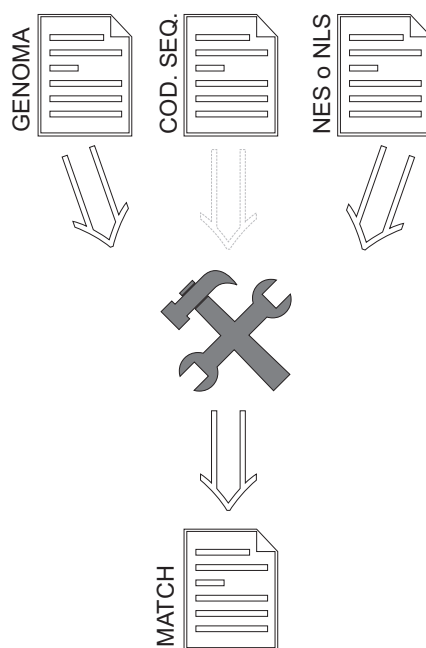


Figura 4.1: Flusso dei file nell'applicazione

Segue un esempio di file di output. La sequenza di aminoacidi è stata tagliata (sono stati inseriti tre punti ...) perché non necessaria alla spiegazione.

```
>gi|52426734|ref|NM_001148.3| Homo sapiens ankyrin 2, neuronal (ANK2),
transcript variant 1, mRNA, [FS2-RL3-RP5160]
PNC SK<<TKMQLRKATVERSSTAVVRGEKDPRSLTAMQASSVLPEQATWTKLWNI<RGA<TSIPAIRMDS
TL SIWLPRKATWGWCRSCWEEGPLWILPLRREIPLFTLHLWLDKQKLSKFLLRKEPILMHSLRMALLLYT
WLPKRITLML<NICWKME...
HRKYEAPRHPRHHKRPRKR
[>NLS:18 ] HRKYEAPRH.\{6\}PRKR
```

Sulla prima riga viene riportata la descrizione del gene trovata nel file FASTA di partenza. In coda a tale descrizione viene aggiunta dal programma una stringa, nell'esempio [FS2-RL3-RP5160]. Tale stringa definisce la mutazione che è stata applicata al gene wild-type, nel caso dell'esempio un frameshift di 2 nucleotidi, un tandem repeat di lunghezza 3, la posizione del tandem repeat è 5160. Sulla riga successiva alla descrizione viene prodotta l'intera sequenza di aminoacidi derivata dalla traduzione del gene mutato. Segue l'istanza del pattern che ha generato il match, in questo caso HRKYEAPRHPRHHKRPRKR. Per finire viene riportato il nome del pattern come descritto nel file di input dei pattern e la relativa espressione regolare.

Come anticipato l'applicazione viene parametrizzata tramite riga di comando, in tabella 4.1 vengono riportati i parametri possibili.

Switch	Significato
-c <rev.database>	crea il database inverso-complementare partendo dal database di input
-f <frame_shift>	introduce un frameshift. Valori possibili 1,2,3 (def: 1)
-g <CDS_file>	utilizza il file per caricare le coding sequence
-h	stampa a video l'help
-i <database>	nome del database di input in formato FASTA (def: dna.fas)
-j	aggiunge le ripetizioni solo se vengono trovate ripetizioni native
-l <max_len>	lunghezza massima della riga nel file di output (def: 80)
-m <match_file>	nome del file di match di output (def: match.fas)
-n <max_pattern_len>	lunghezza massima del pattern per cui cercare il match (def: 0=nessun limite)
-o <output_file>	nome del file di output dove scrivere le sequenze polipeptidiche. File di appoggio. (def: aminogen.out)
-p <pattern_file>	nome del file con i pattern per cui cercare i match (def: nls.regex)
-r <repetition>	lunghezza della ripetizione (def: 0)
-s	termina la sequenza di output degli aminoacidi se viene trovato il carattere '<'
-v	output verbose
<i>Nota</i> -l 0	per rimuovere il limite alla lunghezza delle righe di testo del file di output

Tabella 4.1: Parametri riga di comando

Il nome dato all'applicazione è **aminogen**.

4.2 Tool flow

In questa sezione vengono descritti i passi computazionali sviluppati per l'implementazione dell'approccio inverso come da figura 3.2.

Quello che viene descritto per un singolo gene può essere applicato ad un gruppo di geni o al genoma intero, semplicemente l'elaborazione verrà reiterata

per ogni gene fornito.

La procedura è la seguente:

1. Data una sequenza N di nucleotidi lunga n , tipicamente un gene, viene inserita una ripetizione R , lunga m , di nucleotidi in testa ad N ottenendo una sequenza $Q_{[1..n+m]} = R_{[1..m]} + N_{[1..n]}$.
2. La sequenza $Q_{[1..n+m]}$ viene tradotta nella sequenza $A_{[1..(n+m)/3]}$ di aminoacidi, considerando eventuali frame shift.
3. I segnali NES o NLS (o potenzialmente qualsiasi pattern) vengono ricercati nella sequenza A . Eventuali match vengono registrati.
4. La ripetizione R viene fatta avanzare di una posizione creando una nuova sequenza di nucleotidi $K_{[1..n+m]} = N_{[1..i]} + R_{[1..m]} + N_{[i+1..n]}$. Se non è più possibile avanzare perchè si è raggiunta la fine della sequenza N allora il ciclo di ricerca termina.
5. Viene calcolato il sottoinsieme S derivato da K . L'idea è che K , rispetto alla prima sequenza Q , è molto simile e quindi lo sarà anche la relativa traduzione nella sequenza di aminoacidi. Da qui si deduce che è inutile eseguire la ricerca su tutta la sequenza, è invece sufficiente utilizzare solo l'intervallo che introduce una variazione rispetto alla ricerca precedente. Dato che la ricerca non viene eseguita direttamente su Q o K , ma sulle sequenze tradotte di aminoacidi è necessario calcolare l'intervallo **minimo** S che introduce una variante rispetto a quanto generato da Q . Questa ottimizzazione è possibile solamente se si definisce una lunghezza massima del pattern da ricercare.
6. La sequenza $S_{[1..o]}$ viene tradotta nella sequenza di aminoacidi $A_{[1..o/3]}$, considerando eventuali frame shift
7. Si torna al punto 3

Segue una rappresentazione grafica d'esempio, dove la lunghezza della ripetizione è 5 nucleotidi e la lunghezza del pattern da cercare è 3 aminoacidi. Si assume nessun frameshift. I colori utilizzati hanno il seguente significato:

- | | | |
|---|--|--|
| Ripetizione | Nucleotidi wild-type | Aminoacidi invariati |
| Aminoacidi mutati per la ripetizione | Pattern da ricercare | |

In figura 4.2 si può vedere che la ripetizione lunga 5 nucleotidi viene inserita in testa alla sequenza nucleotidica, come descritto nel punto 1 della procedura. La sequenza di aminoacidi ottenuti dalla traduzione viene analizzata completamente per cercare il pattern, in questo esempio lungo 3 aminoacidi.

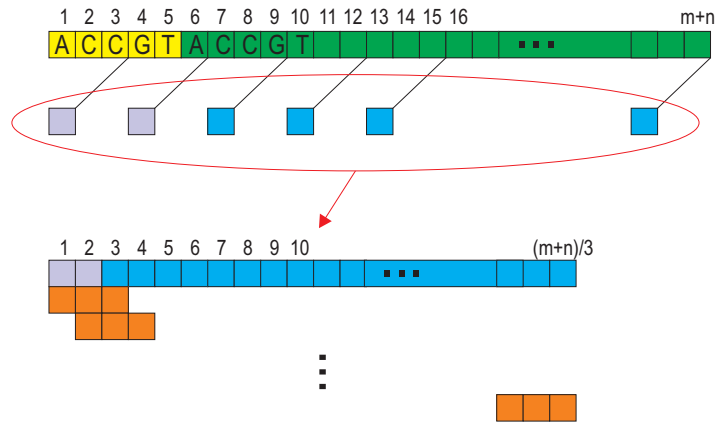


Figura 4.2: Esempio inserimento ripetizione in testa alla sequenza nucleotidica

Come descritto al punto 5 della procedura è possibile ottimizzare lo spazio di ricerca. In figura 4.3 si può vedere che la sequenza di aminoacidi ottenuti dalla traduzione hanno solamente due elementi influenzati dalla ripetizione. Se, ad esempio, prendiamo un pattern lungo 3 aminoacidi si può vedere che è sufficiente ricercare tale pattern tra i primi 6 aminoacidi perchè la rimanente sequenza di aminoacidi è stata analizzata nel corso della prima scansione.

Tale ottimizzazione è possibile se si prende in considerazione una lunghezza massima del pattern da ricercare, in teoria ci potrebbero essere delle espressioni regolari che non hanno una lunghezza definita, questa eventualità dovrà essere valutata dal ricercatore che utilizza l'applicazione.

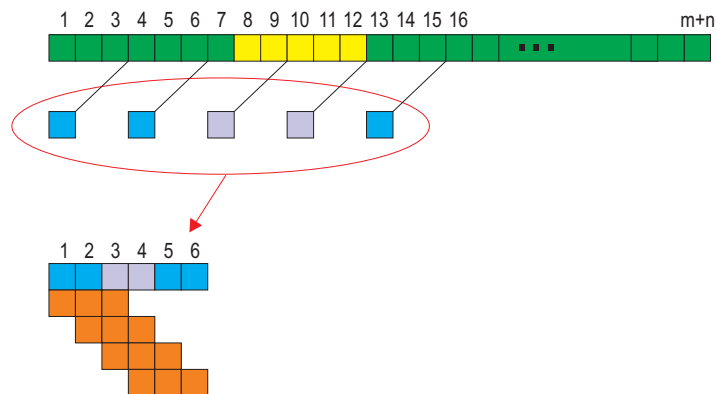


Figura 4.3: Esempio di potenziale ottimizzazione dello spazio di ricerca

Qui di seguito viene listato il frammento di codice sorgente che determina l'intervallo minimo da utilizzare:

```

/*insert the repetition and compute amino interval
   affected by the repetition*/
for (k=0;k<rep_len;k++)
{
gene[j-rep_len+k] = gene[j+k]; //copy nucleotide
if (k == 0)
{
//compute start position in the aminoacid sequence
start = (j-rep_len - frame_shift)/3 - (max_pattern_len - 1);
if (start < 0)
start = 0;

//compute stop position in the aminoacid sequence
stop = (j-rep_len - frame_shift)/3 + (max_pattern_len - 1);
if (stop >= (rep_len+gene_len-frame_shift)/CODON_LEN)
//trim if in out of bounds
stop = (rep_len+gene_len-frame_shift)/CODON_LEN - 1;
}
else if ((j-rep_len+k-frame_shift)%3 == 0)
{
//compute new stop position in the aminoacid sequence
stop = (j-rep_len + k -frame_shift)/3 + (max_pattern_len - 1);
if (stop >= (rep_len+gene_len-frame_shift)/CODON_LEN)
//trim if in out of bounds
stop = (rep_len+gene_len-frame_shift)/CODON_LEN - 1;
}
}
}

```

Un'altra possibilità di ottimizzazione deriva dal fatto che non tutta la sequenza nucleotidica del gene è codificante. In biologia molecolare si definisce DNA non codificante ogni sequenza di DNA in un genoma non soggetta a trascrizione in RNA o rimossa dall'mRNA prima della traduzione (introni) e che viene perciò considerata, allo stato attuale delle conoscenze nel campo, apparentemente priva di funzione.

Nei database GenBank (v. sezione 2.4) è possibile trovare le informazioni riguardanti la parte del gene codificante, detta *coding sequence*. Lo *script* descritto in appendice B estrae tali informazioni dal database e le prepara per essere utilizzate dall'applicazione al fine di minimizzare lo spazio di ricerca.

Una parte importante, ai fini delle performance, nello svolgimento della procedura è sicuramente quella descritta al punto 3, ovvero la ricerca dei pattern. Il problema assume il nome di *pattern matching* con il quale, in ambito

informatico, si intende una corrispondenza tra stringhe. Diversi sono gli algoritmi sviluppati per affrontare la problematica, con approcci diversi a seconda della specificità del problema da affrontare. Al fine di evitare di reinventare la ruota è stato scelto di utilizzare un programma disponibile *liberamente* denominato *grep*.

Grep è un comando dei sistemi Unix e Unix-like, e più in generale dei sistemi POSIX, che ricerca in uno o più file di testo le linee che corrispondono ad uno o più modelli specificati con espressioni regolari o stringhe letterali, e produce un elenco delle linee (o anche dei soli nomi di file) per cui è stata trovata corrispondenza.

Algoritmi più veloci di quelli attualmente utilizzati in grep sono stati sviluppati per la ricerca di stringhe. Ad esempio l'algoritmo di Boyer e Moore [13] che inizia comparando la coda del pattern da cercare per poi procedere in senso inverso. Sebbene è stato dimostrato che questa tecnica migliora sensibilmente le prestazioni nella ricerca di stringhe, la relazione di Boyer e Moore indica che è 'sconsigliato' l'utilizzo dell'algoritmo per pattern di ricerca non espliciti come ad esempio quelli forniti dalle espressioni regolari. Questo è dovuto parzialmente al fatto che la ricerca opera in senso inverso e parzialmente perchè c'è il bisogno di saltare porzioni della stringa di ingresso lunghe tanto quanto la stringa da cercare. Evidentemente un'espressione regolare può rappresentare stringhe di diversa lunghezza ed è qui quindi che si evidenzia la non applicabilità.

GNU grep, qui utilizzato, è basato su un verificatore di corrispondenze lazy-state deterministico veloce (circa il doppio del grep standard di Unix), ibridato con una ricerca di Boyer-Moore-Gosper per una stringa fissa, il che fa in modo che il testo impossibile non venga considerato nella corrispondenza dell'espressione regolare senza dovere per forza verificare ogni carattere. Il risultato è di solito molto più veloce di grep o egrep di Unix.

L'estratto dal codice sorgente che segue evidenzia come viene invocato grep:

```
/*now in my output file I have all the proteins generated
   from the different position of the repetition in the gene.
   I look for the a match of the patterns stored in the
   pattern_table. I use grep for the pattern matching, the
   output is redirected (in append mode) to the matchings
   output file*/
for (j=0;j<pattern_no;j++)
{
    /*read the match output file size*/
    stat(fmatch_name,&stbuf);
```

```

file_size = stbuf.st_size;

/*prepare the grep command*/
sprintf(grep_command,"grep -B 1 '%s' %s >>%s ",
        pattern_table[j].pattern,
        fout_name,
        fmatch_name);
system(grep_command);

/*read the match output file size, so I check if
there was a match*/
stat(fmatch_name,&stbuf);
if (file_size != stbuf.st_size)
{ /*there was a match*/
/*print in the match file only the matching string
(not the whole line)*/
sprintf(grep_command,"grep -o '%s' %s >>%s ",
        pattern_table[j].pattern,
        fout_name,
        fmatch_name);
system(grep_command);

/*print the matching pattern*/
sprintf(grep_command,"echo '[%s] %s' >>%s ",
        pattern_table[j].pattern_name,
        pattern_table[j].pattern,
        fmatch_name);
system(grep_command);
}
}

```

Come indicato in tabella 4.1, sono possibili diverse parametrizzazioni da riga di comando. Si fa notare la possibilità di generare un database in formato FASTA contenente la versione inversa e complementare del database in ingresso (negli organismi viventi, il DNA non è quasi mai presente sotto forma di singolo filamento, ma come una coppia di filamenti). Una sequenza di DNA è definita *senso* se la sua sequenza è la stessa del relativo mRNA. La sequenza posta sul filamento opposto è invece detta *antisenso*. Quindi partendo dal senso si può ottenere l'antisenso e viceversa.

Altra opzione è la possibilità di inserire le ripetizioni solo nella posizione in cui una ripetizione, della lunghezza data, sia già nativamente presente nel gene wild-type. Qui di seguito il codice che esegue il controllo:

```

/*if requested to insert repetition just in case of the repeated

```

sequence is already repeated in the original seq. then I perform the check and if negative I skip to the next position and I check it again, and so on... */

```
if (native_rep)
{
  while (j <=gene_len-rep_len)
  {
    if (memcmp(gene+j,gene+j+rep_len,rep_len) == 0)
      break; //match found. I can insert my additional repetition
    else
    { //skip by one position
      gene[j-rep_len] = gene[j];
      j++;
    }
  }//while

  if (j > gene_len-rep_len)
    break; //no more possibilities to find a native repetition
}
```

Capitolo 5

Risultati e conclusioni

In questa tesi sono stati raggiunti gli obiettivi attesi, ovvero sviluppare un'applicazione software in grado di ricercare nel genoma umano le possibili mutazioni di tipo *tandem repeat* e *frameshift* responsabili di traslocazione nucleare. Il lavoro fatto assume una valenza in quanto ha ideato un approccio, alternativo a quelli disponibili come l'approccio diretto, capace di evitare le esplosioni combinatorie. La diretta conseguenza dell'approccio proposto è poter eseguire ricerche con tempi di elaborazione contenuti. Si è potuto ricercare segnali di traslocazione nucleare senza nessuna limitazione nella complessità degli stessi, cosa non possibile nei software oggi a disposizione per tale scopo.

In ambito software il concetto di risultato è spesso legato a quello di correttezza dello stesso. La fase di verifica e di validazione serve ad accertare che il software rispecchi i requisiti e che li rispetti nella maniera dovuta.

La verifica serve a stabilire che il software rispetti i requisiti e le specifiche, quindi ad esempio che non ci siano requisiti mancanti, mentre la validazione serve ad accertare che i requisiti e le specifiche siano anche rispettati nella maniera giusta. Questa fase, infatti, è molto delicata in quanto, si può ottenere un software perfettamente funzionante, senza errori, ma del tutto inutile in quanto non conforme alle specifiche iniziali.

Diversi test sono stati effettuati al fine di:

1. verificare e validare l'applicazione
2. poter constatare le prestazioni dell'approccio proposto (v. sezione 3.2)

Il primo obiettivo è stato perseguito inizialmente facendo delle prove utilizzando sottoinsiemi molto ridotti sia di segnali da ricercare che numero di geni su cui eseguire la ricerca. Con un insieme di dati ridotto è possibile verificare i risultati attesi anche 'manualmente'.

Tipicamente vengono preparati degli insiemi di segnali e geni dove è possibile calcolare i risultati attesi, successivamente viene fatta eseguire l'applicazione e si confrontano i risultati di quest'ultima da quelli attesi. Nel caso in cui quanto atteso non si verifichi si intraprende un'attività di *debugging* (ricerca dell'errore) e si ripete il test da capo.

Appena la quantità di dati aumenta la possibilità di poter prevedere in modo manuale i risultati attesi diventa nella pratica non sostenibile. Rimane possibile, anche se non agevole, andare a verificare se i risultati generati dall'applicazione sono corretti, ma ciò non garantisce che tutti i match siano stati trovati, piuttosto si può affermare che quelli trovati sono corretti.

Altro aspetto importante nel processo di verifica dei risultati è ben esemplificato da quanto affermato da Edsger Dijkstra nella sua frase:

testing can only show the presence of errors, not their absence

Diventa importante in quest'ultimo scenario poter incrociare risultati provenienti da metodi automatici distinti. L'approccio diretto, già sviluppato e collaudato, rappresenta, seppur limitato in termini di tipologia e lunghezza dei segnali, un efficace mezzo di verifica dei risultati forniti dall'approccio inverso.

A seguito dei test condotti si può ragionevolmente affermare che, pur non potendo escludere la presenza di errori, l'applicazione oggetto di questa tesi è stata verificata e validata.

L'aspetto più interessante di questo sviluppo è sicuramente l'aspettativa sul piano delle prestazioni in confronto con l'approccio diretto.

Al fine di ottenere delle misurazioni concrete dei tempi di elaborazione sono stati preparati un file con dei segnali NLS e un file con dei geni. Per motivi di praticità è stato scelto di non eseguire i test sull'intero genoma umano e sull'intero database di segnali NLS, in modo tale da poter eventualmente rieseguire i test senza dover aspettare diverse ore per attendere i risultati. Il file dei geni utilizzato per il test rappresenta circa l'1% dell'intero genoma umano ed è costituito da 374 geni. Nel file dei segnali NLS sono stati selezionati 95 pattern di cui 2 sono espressioni regolari e non stringhe costanti, il pattern più lungo è composto da 42 caratteri.

Se si prende in considerazione, ad esempio, il primo segnale NLS utilizzato nel test è possibile fare delle considerazioni preliminari. Il segnale è il seguente:

```
[LF] [STK] [VIQM] [KR]R[QMVI] [STK]L
```

essendo un'espressione regolare i termini tra parentesi quadre vanno interpretati come caratteri mutuamente esclusivi. Ad esempio [LF] significa o L o F, quindi alcune delle stringhe possibili possono essere:

LSVKRQSL
 LTIKRMTL
 ...
 FKMRRIKL

Facendo un calcolo rapido abbiamo che le stringhe ottenibili da tale espressione regolare sono

$$2 * 3 * 4 * 2 * 4 * 3 = 576 \quad (5.1)$$

Utilizzando l'approccio diretto si dovrebbero retrotradurre le stringhe ottenute in modo da ottenere le sequenze nucleotidiche. Si ricorda che i codoni costituiscono un codice ridondante rispetto agli aminoacidi, infatti a fronte di 64 possibili codoni gli aminoacidi sono solamente 22 (v. sezione 2.1.1). Considerando mediamente 4 diversi codoni associati ad ogni aminoacido si ottiene:

$$4^{lunghezza_{stringa} * n_{stringhe}} = 4^8 * 576 \approx 37x10^6 \quad (5.2)$$

A questo punto il metodo diretto cerca in questi 37 milioni di sequenze nucleotidiche i tandem repeat. Una volta localizzati, possono essere più numerosi delle sequenze nucleotidiche stesse, vengono usati come sequenza per la ricerca all'interno del genoma umano.

Risulta evidente che l'approccio diretto genera un'esplosione combinatoria molto pesante per le attuali potenze di calcolo disponibili. Si consideri inoltre che l'esempio è relativo ad un singolo pattern da ricercare.

L'approccio inverso sviluppato in questa tesi evita l'esplosione combinatoria perché parte dal genoma per arrivare alle corrispondenze dei pattern nelle catene polipeptidiche. Nell'esempio qui sopra, se si utilizza l'approccio inverso, si parte dal generare sequenze di aminoacidi ottenute inserendo tandem repeat e/o frameshift nei geni di partenza. E' stato sviluppato un algoritmo per la riduzione della lunghezza delle sequenze di aminoacidi (v. sezione 4.2). A questo punto si procede ricercando i 576 segnali nelle sequenze ottenute. In questo caso la crescita è lineare o, grazie all'algoritmo di ottimizzazione, sublineare.

Un'altra importante considerazione è relativa alla lunghezza massima del pattern da ricercare. Per motivi di esplosione combinatoria, rapportata alla potenza di calcolo disponibile, è stato possibile per i ricercatori che hanno utilizzato l'approccio diretto la ricerca di segnali con lunghezza massima di 15 caratteri. Nel test qui eseguito sono stati utilizzati pattern con lunghezza fino a 42 caratteri, ma sono stati fatti altri test anche con lunghezze ben maggiori

senza rendere la ricerca improponibile per l'approccio inverso.

E' stata utilizzata la funzione di sistema *time* disponibile in Linux per il calcolo del tempo di esecuzione del programma. La piattaforma usata è un PC con processore Pentium IV, 2.4 GHz e 1GB di RAM. Il sistema operativo Suse Linux 10 è stato fatto eseguire in una macchina virtuale VMware (situazione sfavorevole in termini di prestazioni a confronto di un'installazione nativa).

La riga di comando da shell per l'esecuzione del programma è:

```
time ./aminogen -i rna_1p100.fa -p nls.tesi_finotto.geq7 -n 42 -r 5
```

I tempi risultanti sono i seguenti:

```
real    76m30.170s
user    5m35.921s
sys     26m57.265s
```

Si può vedere che il tempo di esecuzione del programma è di circa 32 minuti (user+sys). Il tempo di 76 minuti (real) è dovuto al fatto che il sistema host, in questo caso windows XP, occupava una certa parte delle risorse della CPU. Volendo proiettare, in modo lineare, il tempo impiegato per l'1% del genoma umano sull'intero genoma si ottiene un tempo di esecuzione di circa 54 ore.

Facendo eseguire il programma che implementa l'approccio diretto sullo stesso insieme di dati è stato deciso, dopo alcuni giorni di esecuzione, di terminare il processo senza attenderne il completamento.

Dopo i test eseguiti si può concludere che l'approccio inverso apre la strada ad un'esplorazione più ampia del genoma umano alla ricerca di segnali NLS, NES o potenzialmente di qualsiasi altra categoria. L'assenza di un'esplosione combinatoria collegata alla lunghezza dei pattern rende la soluzione qui trattata preferibile all'approccio diretto.

Al fine di abbassare ulteriormente i tempi di esecuzione delle ricerche è auspicabile un'evoluzione dell'applicazione verso concetti di *parallel computing* oltre all'utilizzo di piattaforme hardware sempre più potenti.

Appendice A

Organizzazione dei file

I file che compongono l'applicazione sono i seguenti:

- **aminogen**. File eseguibile dell'applicazione
- **function.c**. File sorgente con l'implementazione degli algoritmi descritti nel capitolo 4
- **function.h**. Header file del file `function.c`
- **get_CDS**. Script awk che estrae da tutti i file indicati di tipo GenBank le informazioni relative alle *coding sequence* (v. appendice B)
- **main.c**. File sorgente contenente la funzione `main()` per la gestione della riga di comando e invocazione delle funzioni contenute in `function.c`
- **Makefile**. Makefile dell'applicazione
- **structures.h**. Header file delle strutture dati usate nel programma

Appendice B

Script `get_CDS`

Lo script `get_CDS` è sviluppato nel linguaggio interpretato da GNU awk. `Get_CDS` estrae da tutti i file indicati di tipo GenBank le informazioni di interesse ovvero

- identificativo gene (VERSION)
- coding sequence (CDS)

A valle del programma si ottiene un file singolo di output con appunto le informazioni su citate.

Questo file potrà esser fornito come input al programma principale.

La funzione espletata dallo script `get_CDS` poteva essere implementata anche all'interno dell'applicazione principale, ma tenendo le cose separate si avrà il vantaggio di una maggiore flessibilità. Infatti essendo il file di output un file di testo è possibile modificare il contenuto anche manualmente a seconda delle esigenze, ad esempio utilizzare le coding sequence solamente per certi geni e per altri no ecc.

Passando all'aspetto operativo dell'utilizzo dello script ecco qui di seguito una riga di comando di esempio, si assume che i file del database GenBank abbiano estensione `.gb`:

```
# awk -f get_CDS *.gb|sort >esempio.cds
```

E' conveniente eseguire anche il `sort` in pipe con `awk` perché potrebbe essere un file di migliaia di righe e se l'applicazione riceve una lista ordinata può eseguire delle ricerche in modo più performante evitando di dover far lui l'ordinamento.

Il formato del file di output è:

<VERSION> <CDS_start> <CDS_stop>

ecco un esempio:

```
NM_002444.2 199 1932
U49845.1 687 3158
NM_152695.4 4 6
```

Il programma principale allocherà dinamicamente una tabella e poi la dovrà consultare per ogni gene letto nel file di input in formato FASTA.

Bibliografia

- [1] A. Bogliolo: Corso di Bioinformatica, Università di Urbino, Anno Accademico 2007/08
- [2] Wikipedia homepage, <http://www.wikipedia.it>
- [3] sito del Dipartimento di Biochimica e Biologia Molecolare dell'Università degli studi di Ferrara, <http://web.unife.it/progetti/biologiamolecolaredellemostasi/index.html>
- [4] Falini B, Bolli N, Shan J, Martelli MP, Liso A, Pucciarini A, Bigerna B, Pasqualucci L, Mannucci R, Rosati R, Gorello P, Diverio D, Roti G, Tiacci E, Cazzaniga G, Biondi A, Schnittger S, Haferlach T, Hiddemann W, Martelli MF, Gu W, Mecucci C, Nicoletti I. Both carboxy-terminus NES motif and mutated tryptophan(s) are crucial for aberrant nuclear export of nucleophosmin leukemic mutants in NPMc+ AML. *Blood*. 2006;107:4514-4523.
- [5] genome programs of the U.S. Department of Energy Office of Science, <http://genomics.energy.gov>
- [6] Liso A, Bogliolo A, Freschi V, Martelli MP, Pileri SA, Santodirocco M, Bolli N, Martelli MF, Falini B In human genome, generation of a nuclear export signal through duplication appears unique to nucleophosmin (NPM1) mutations and is restricted to AML
- [7] la Cour T, Gupta R, Rapacki K, Skriver K, Poulsen FM, Brunak S. NES-base version1.0: a database of nuclear export signals. *Nucleic Acids Res*. 2003;31:393-396.
- [8] Falini B, Nicoletti I, Martelli MF, Mecucci C. Acute myeloid leukemia carrying cytoplasmic/mutated nucleophosmin (NPMc+ AML): biologic and clinical features. *Blood*. 2007;109:874-885.
- [9] OpenSUSE homepage, <http://www.opensuse.org>
- [10] GCC, the GNU Compiler Collection, <http://gcc.gnu.org>
- [11] Gawk homepage, <http://www.gnu.org/software/gawk/>

- [12] GenBank Flat File Format,
<http://www.ncbi.nlm.nih.gov/Sitemap/samplerecord.html#OriginB>
- [13] R. S. Boyer, J. S. Moore. A fast string searching algorithm. 1977 Comm. ACM 20: 762-772
- [14] Charles L. A. Clarke and Gordon V. Cormack. On the use of regular expressions for searching text. ACM Transactions on Programming Languages and Systems, 19(3):413-426, 1997.
- [15] Free Software Foundation, Inc. GNU grep Documentation, 2002.
- [16] Tony Abou-Assaleh, Wei Ai. Enhancing GNU grep. Dalhousie University, 2004.

Ringraziamenti

Vorrei ringraziare i professori che mi hanno seguito nello sviluppo di questa tesi, in particolare il Prof. Bogliolo e l'Ing. Freschi. Grazie anche alla Dott.ssa Pigliapoco che nel corso di questi tre anni si è sempre dimostrata disponibile e competente.

Ringrazio l'Università di Urbino per aver messo a disposizione dei corsi di laurea pensati anche per gli studenti lavoratori come me.

Un grazie all'azienda per cui lavoro, la multinazionale Moog, per avermi sempre facilitato, per quanto possibile, la convivenza tra lavoro e studio.

Infine il mio grazie più grande alla mia famiglia che per tre anni ha dovuto dividere un marito ed un padre con le esigenze dello studio. Senza il loro supporto non sarei riuscito a raggiungere questo risultato, sicuramente questo successo è anche loro.