Università degli Studi "Roma Tre"

Facoltà di Ingegneria

Corso di Laurea Specialistica-Magistrale in
Ingegneria Gestionale e dell'Automazione

Metodi ed Algoritmi per il Problema di Tag SNP Selection

Laureanda
Serena D'Aguanno
matricola 252788

Relatore
Prof. Dario Pacciarelli

Correlatore
Dott.ssa Paola Bertolazzi

Anno Accademico 2005-2006

Indice

Introduzione

La *bioinformatica*, o *biologia computazionale*, ha lo scopo di utilizzare tecniche matematiche ed informatiche per risolvere problemi biologici. Una delle aree più significative della bioinformatica è rappresentata dall'analisi dei dati generati dal Progetto Genoma Umano, che ha permesso di identificare la codifica dell'intero genoma. Con il termine *genoma*, o *patrimonio genetico*, si intende l'insieme dei geni di un organismo vivente. L'informazione genetica è portata dalla molecola di Acido desossiribonucleico (DNA) che, associato a proteine, è il principale costituente dei cromosomi degli Eucarioti. Ogni gene è formato da un tratto di molecola di DNA, e contiene una sequenza di coppie di basi azotate. Il genoma umano è organizzato in 46 cromosomi, raggruppati in 23 coppie, ciascuna contenente un cromosoma di origine materna e uno di origine paterna. Ogni sottosequenza di cromosomi è detta *aplotipo*, invece una sottosequenza di entrambi è detta *genotipo*. Partendo dalla considerazione che più del 99% delle posizioni del genoma non varia da individuo ad individuo; inizialmente, l'obiettivo della ricerca genetica è stato quello di individuare quali posizioni del genoma umano risultano tipicamente varianti e quali invarianti. Generalmente, quando una variazione occorre almeno in una certa percentuale della popolazione, è considerata una variazione *comune*. Di solito tali variazioni comportano la sostituzione di un singolo nucleotide, e sono chiamate *Single Nucleotide Polimorphisms*(SNPs - pronunciato *snips*).
L'interesse corrente della ricerca genetica è focalizzato sull'individuazione di un'associazione tra variazioni di geni e malattie; che significa identificare quale variazione di DNA o quale insieme di variazioni di DNA è altamente correlato con una malattia specifica.
Nel caso di malattie dette complesse le mutazione dei geni risultano multiple, e questo implica la necessità di ottenere informazioni genetiche su larga-scala per individuare delle relazioni. I metodi tradizionali che i biologi hanno tutt'oggi a disposizione per estrarre delle sequenze aplotipo del genoma umano, sono molto lunghi e troppo costosi; invece è possibile conoscere direttamente le sequenze genotipo.

Recentemente, il metodo chiamato *haplotype analysis* è stato utilizzato con successo per identificare variazioni rilevanti di DNA in diverse malattie; ed è ora considerato il metodo più promettente per studiare l'associazione malattia-gene. Le principali procedure computazionali utilizzate nell'haplotype analysis sono: l'*Haplotype Phasing* e il *Tag SNP Selection*. Perciò, il problema affrontato in questa tesi, diventa quello di studiare una di queste procedure, in particolare il problema di Tag SNP Selection; ovvero il problema di determinare un sottoinsieme di SNPs che risulti sufficientemente informativo per determinare l'associazione malattia-gene ma piccolo abbastanza per ridurre il carico necessario a derivare l'informazione sui genotipi.
In particolare, il primo capitolo della tesi è dedicato alla descrizione di alcune nozioni basilari di genetica e dell'analisi computazionale per permettere una maggiore comprensione del problema. Nel secondo capitolo viene introdotto il problema di Tag SNP Selection fornendone una definizione analitica, analizzando e confrontando i principali approcci fin ora adottati per risolverlo. Nell terzo si chiariscono le caratteristiche fondamentali del problema e si arriva alla costruzione del modello matematico che è alla base di ciascun problema di ricerca operativa. Nel quarto capitolo sono descritti gli strumenti utilizzati per la creazione del programma software: Visual C++ e Ilog CPLEX. Grazie all'ambiente di lavoro Visual C è stato possibile realizzare un programma che sfruttando i principi della Ricerca Operativa implementa un algoritmo euristico, che, attraverso l'uso delle funzioni proprie di CPLEX, è in grado di eseguire l'ottimizzazione del problema e di modificare opportunamente la soluzione. All'interno del capitolo sono inoltre descritte le funzioni di CPLEX utilizzate nel programma definendo le loro caratteristiche, la loro sintassi e gli output che forniscono in uscita.
La seconda parte del testo, composta dagli ultimi tre capitoli, è orientata alla descrizione dettagliata del codice del programma realizzato e alla sua sperimentazione su insiemi di dati reali. Nel quinto capitolo è infatti descritto in maniera dettagliata ed esauriente il codice del programma realizzato, con i suoi file di input e i suoi risultati. Nel sesto è invece testata la validità del programma realizzato, riportando i risultati ottenuti lavorando su dati reali. Infine l'ultimo capitolo contiene alcune conclusioni sul lavoro svolto.

Capitolo 1

Background biologico

Le caratteristiche di un individuo, cioè le diversità dei suoi tratti esterni, sono diretta conseguenza della struttura del DNA che costituisce il materiale genetico della maggior parte degli organismi. Più in particolare, le differenze fra i diversi individui sono dovute a piccole variazioni nella sequenza dei rispettivi DNA; queste diversità sono oggetto dell'analisi genetica. Il loro studio ha portato all'individuazione degli elementi fondamentali per l'ereditarietà; i *geni* e le loro diverse varianti, gli *alleli*, e alla scoperta che questi corrispondono a tratti di DNA disposti linearmente sulle lunghe catene di DNA costituenti i cromosomi[1].
Il Progetto Genoma Umano, che ha fornito la sequenza del DNA, ha permesso di osservare che il DNA presenta identità genetiche per il 99% della struttura. E' quindi il rimanente 1% di esso che caratterizza e identifica le diversità fra gli individui, ed è in questa percentuale che si devono andare a cercare i motivi che possono portare allo sviluppo, in un individuo piuttosto che in un altro, di determinate malattie.

1.1 Struttura del DNA e ricombinazione genetica

La struttura del DNA può essere schematizzata come due filamenti con orientamento antiparallelo, avvolti l'uno sull'altro a doppia elica. Ogni filamento, o elica, è composto da una sequenza di *nucleotidi*. Un nucleotide può essere considerato come l'unità base del DNA ed è costituito da tre elementi: una base azotata, un gruppo fosfato e un desossiribosio (zuccheri). Ogni nucleotide può essere pensato come un singolo anello della catena di DNA.

[1] la descrizione del DNA, dei cromosomi e dei geni è rimandata al paragrafo seguente.

La struttura interna che permette ai due filamenti di essere uniti e di formare la molecola di DNA è costituita da un appaiamento tra basi, ognuna delle quali è orientata verso l'interno dell'elica. Le basi sono: l'Adenina (A), la Citosina (C), la Guanina (G) e la Timina (T); ed i legami che si creano tra le coppie di esse mantengono insieme la doppia elica. E' importante osservare che le quattro basi non possono unirsi fra loro in tutte le combinazioni possibili, bensì devono rispettare degli appaiamenti specifici secondo i quali gli unici accoppiamenti consentiti sono Adenina-Timina A-T e Citosina-Guanina C-G. Da questo consegue che se si conosce la sequenza di un filamento si può ricostruire anche quella dell'altro. Le coppie A-T e C-G si chiamano *coppie di basi complementari* (diagramma schematico di figura1.1).

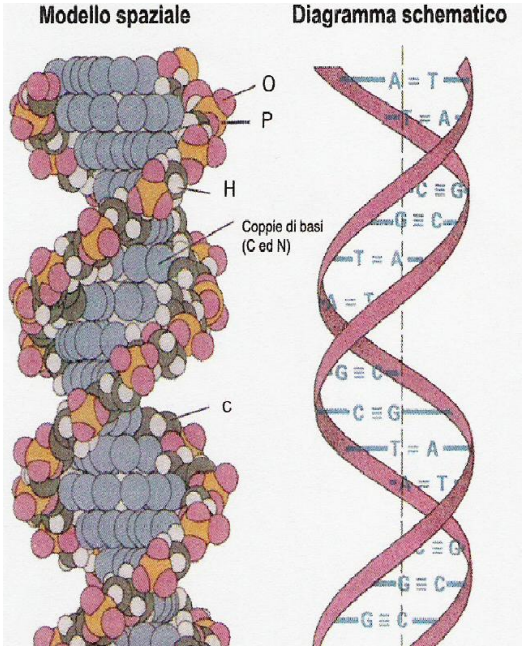


Figura 1.1: Struttura del DNA

Molecole di DNA, sono contenute, all'interno della cellula, in strutture lineari dette *cromosomi*. Nelle cellule *dipolidi* umane i cromosomi sono 46, organizzati in coppie di cromosomi omologhi; i cromosomi omologhi sono cromosomi identici nella disposizione dei geni (contengono alleli diversi dello stesso gene) e nella loro struttura visibile; uno dei due cromosomi è di origine materna e l'altro di origine paterna. Le cellule umane *aploidi* contengono una sola serie di 23 cromosomi[2].

[2]nell'uomo solo le cellule sessuali.

Riassumendo, ogni cromosoma è quindi costituito da una doppia elica di DNA, sulla quale è possibile individuare sottosequenze di DNA, i geni. In particolare, ai geni sono legati i caratteri ereditari, che rappresentano ed individuano un tipo di caratteristica. La mappa genetica illustra la posizione di ogni gene all'interno del DNA, il cosiddetto *locus genico*, ovvero la posizione specifica occupata da un gene lungo il cromosoma e le distanze genetiche tra i vari geni. Le forme alternative di un gene in un singolo locus genico sono dette *alleli*. In un organismo diploide possono verificarsi due situazioni differenti: se entrambi gli alleli di uno stesso locus sono uguali, l'organismo si dice *omozigote* rispetto a quel locus; se invece possiede due differenti alleli in uno stesso locus viene detto *eterozigote*. Per poter analizzare queste porzioni di DNA non è necessario considerare le quattro catene che formano le due eliche, ma è sufficiente analizzare una sola catena, nel caso in cui si voglia costruire la mappa dei geni (ovvero vedere dove inizia e dove finisce ogni porzione); oppure analizzarne due, nel caso in cui si debba studiare l'ereditarietà.
Un'altra nozione genetica di interesse è quella di *ricombinazione genetica*, che copre un ruolo di fondamentale importanza durante la meiosi[3]. Nella prima parte della meiosi avviene un fenomeno detto *crossing-over*, il quale si manifesta tra coppie di cromosomi materni e paterni producendo uno scambio fisico e reciproco fra tratti di cromosomi differenti in punti corrispondenti (figura1.2).

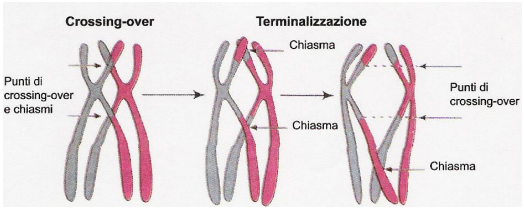


Figura 1.2: Crossing-over fra due cromosomi

Questo fenomeno permette alle cellule figlie di non possedere alleli unicamente paterni o materni, ma di ereditare alcuni caratteri dal DNA materno ed altri da quello paterno. Tale evento si verifica durante ogni meiosi, ma i cosiddetti punti di crossing-over, zone in cui i cromosomi si rompono e si scambiano una parte di materiale con altri, variano da una meiosi all'altra. Studi sulle popolazioni hanno mostrato che la probabilità di avere una ricombinazione in una determinata posizione non si mantiene uniforme lungo il cromosoma.

[3]processo durante il quale sono prodotte quattro cellule aploidi ottenute da una cellula diploide mediante una duplicazione dei cromosomi e due divisioni cellulari.

1.2 SNP, aplotipi e genotipi

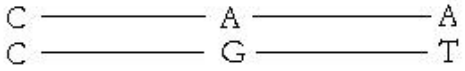
La diversità tra gli individui è legata alla presenza di piccole variazioni genetiche chiamate *polimorfismi*. A livello di DNA, un polimorfismo è una regione il cui contenuto varia tra individuo ed individuo. Lo *SNP* (Single Nucleotide Polymorphism) è il più piccolo polimorfismo di questo tipo, relativo ad una singola base. Ogni SNP si presenta solo con due dei quattro valori possibili (A, C, G, T).
Come già detto il patrimonio genetico di ogni individuo è organizzato in 46 cromosomi, raggruppati in 23 coppie. Considerando una di queste coppie, viene indicato con il termine *aplotipo* un piccolo frammento sequenziato di DNA relativo ad una singola copia di un cromosoma. Mentre per indicare l'informazione genetica contenuta in un frammento di entrambe le copie, si utilizza il termine *genotipo*. Ricostruire la mappa degli aplotipi ed individuare gli SNPs, causa fondamentale della diversità del patrimonio genetico, permette non solo di individuare i geni portatori di malattie ma anche di poter confrontare aplotipi di persone sane e malate con l'eventuale possibilità di evidenziare la malattia prima del suo sviluppo e di prevenirla. Per quanto riguarda gli SNPs, possono essere divisi in due tipi:

- *omozigoti*: entrambi gli aplotipi hanno la stessa variante (allele);
- *eterozigoti*: formati da due alleli differenti.

Considerando, ad esempio, due cromosomi di un organismo diploide, uno di eredità paterna ed uno materno del tipo:

cromosoma paterno: aggtccatagggCtaaatcgacgacctccAaaagggtcgActacg
cromosoma materno: aggtccatagggCtaaatcgacgacctccGaaagggtcgTctacg

Si supponga che sulla sequenza vi siano tre SNPs:



Di questi solo il primo è omozigote mentre gli altri due sono eterozigoti. I due aplotipi evidenziati sono perciò: CAA e CGT. Il genotipo associato a

questi due aplotipi è invece: C, {A,G}, {A,T}.
Il processo che i biologi hanno a disposizione per poter leggere le molecole che costituiscono il DNA di un individuo viene chiamato *sequenziamento*. Anche utilizzando i migliori strumenti, il sequenziamento non avviene mai in maniera perfettamente corretta ed errori di lettura nella sequenza degli aplotipi, durante questo processo, sono inevitabili. Inoltre, considerando una popolazione di individui, ottenere la sequenza degli aplotipi di ognuno di essi risulta un procedimento molto complesso poichè i metodi tutt'oggi a disposizione dei biologi sono molto lunghi e costosi. E'invece possibile ottenere, con tempi e costi molto ridotti, le sequenze dei genotipi della popolazione.
Oltre ad essere note le esatte posizioni degli SNPs nel DNA, i biologi conoscono anche la "fase" di ogni SNP, cioè qual'è l'allele più frequente fra i due possibili. Allora la fase viene codificata con il simbolo 0 e l'altra base con il simbolo 1. Se i due alleli, per uno SNP, hanno la stessa frequenza, allora la scelta dello 0 o dell'1 è arbitraria. Questo permette di rappresentare ogni aplotipo di lunghezza m con una stringa di lunghezza m di elementi nell'alfabeto $\sum = \{0, 1\}$. Nella tabella sottostante è riportato un esempio di codifica degli aplotipi.

Alleli	C/A	G/A	C/G	T/C	T/C	G/A	C/G	
h_1	C	G	C	T	T	A	C	0000010
h_2	C	G	G	C	C	G	G	0011101
h_3	A	A	C	T	T	A	C	1100010
h_4	C	G	C	T	T	G	C	0000000
h_5	A	A	C	T	T	G	C	1100000

I genotipi invece saranno codificati come stringhe ancora di lunghezza finita ma i cui elementi appartengono all'alfabeto $\sum' = \{0, 1, 2\}$. Con tale scelta infatti comparirà l'elemento 0 nel caso in cui nella stessa posizione di entrambi gli aplotipi compaia lo 0, l'elemento 1 nel caso opposto, mentre nel caso in cui i due aplotipi abbiano nello stesso locus simboli diversi il genotipo conterrà l'elemento 2; le posizione in cui compare tale valore sono dette **ambigue**, proprio per il loro duplice contenuto.

1.3 Analisi computazionale degli aplotipi

Molti metodi bio-molecolari sono in grado di identificare l'informazione degli aplotipi direttamente dai cromosomi, ma a causa del costo e dei lunghi tempi delle operazioni, vengono utilizzati principalmente per campioni di dimensioni moderate (tipicamente per diverse decine di individui). Per campioni di larga-scala (tipicamente da centinaia a migliaia di individui) per identificare gli alleli dei loci obiettivo sono impiegati metodi con alto throughput. La maggior limitazione di quest'ultimi risiede nell'incapacità di distinguere il cromosoma di origine di ogni allele.
Un'interessante caratteristica degli aplotipi, detta *linkage disequilibrium* (LD), è rappresentata dall'associazione non casuale tra gli SNPs compresi in ciascuno aplotipo. Teoricamente, però, la ricombinazione genetica può manifestarsi lungo i cromosomi un qualsiasi numero di volte. Perciò, l'origine di uno SNP su un cromosoma non risulta influenzata dall'origine degli altri SNPs. Questa caratteristica di indipendenza tra gli SNPs è chiamata *linkage equilibrium*.
In generale, gli SNPs fisicamente molto vicini sono considerati avere elevato LD; in quanto la probabilità di ricombinazione cresce con l'aumentare della distanza tra due SNPs. Come conseguenza, gli alleli tendono ad essere molto correlati tra loro ed il numero di aplotipi distinti composti dagli SNPs risulta molto più piccolo rispetto a quello atteso sotto linkage equilibrium.
Recentemente, degli studi su larga-scala hanno avvalorato l'ipotesi che vede il DNA partizionato in regioni discrete note come *blocchi*. Le ricombinazioni risultano molto rare all'interno dei blocchi e molto comuni tra di essi. Sebbene non ci sia unanimità sul metodo di definizione dei blocchi sul genoma, non esistono dubbi in merito alla struttura a blocchi dello stesso.
L'elevato LD tra SNPs vicini e il limitato numero di aplotipi, dovuto alla struttura a blocchi del genoma, hanno costituito le basi dell'analisi computazione degli aplotipi per lo studio di associazioni malattia-gene.
Haplotype Phasing e *Tag SNP Selection* sono le due procedure computazionali utilizzate nell'analisi degli aplotipi. La prima deriva l'informazione degli aplotipi a partire dai genotipi; la seconda seleziona un sottoinsieme di SNPs su un aplotipo che sia sufficientemente informativo per determinare l'associazione malattia-gene, ma piccolo abbastanza per ridurre il carico necessario a derivare l'informazione sui genotipi.In figura1.3 la procedura di analisi computazionale degli aplotipi è confrontata con quella tradizionale. Gli esperimenti bio-molecolari sono riportati in blocchi bianchi, mentre le procedure computazionali e statistiche in blocchi neri.

L'analisi computazionale è caratterizzata da Haplotype Phasing, Tag SNP Selection e Haplotype-Disease Association unitamente a due esperimenti sui

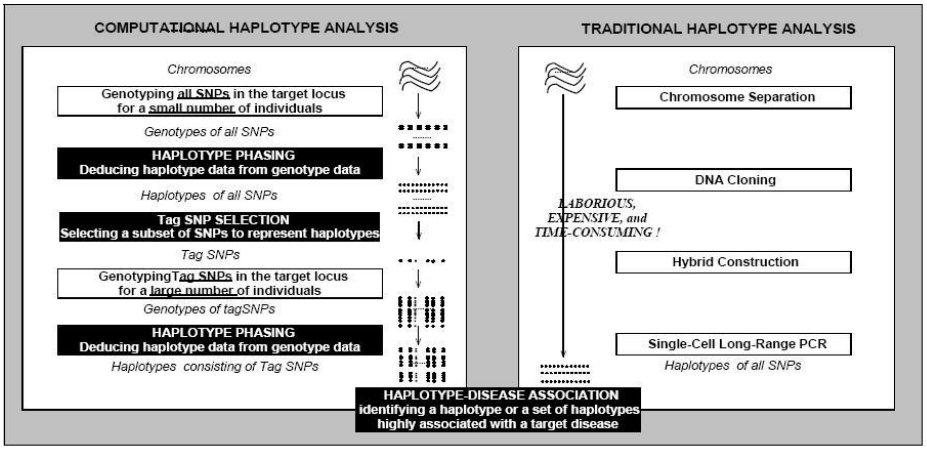


Figura 1.3: Confronto tra l'analisi computazionale degli aplotipi e quella tradizionale

genotipi. Inizialmente, vengono estratte informazioni dei genotipi di un numero relativamente piccolo di individui e ne vengono inferiti gli aplotipi attraverso gli algoritmi di Haplotype Phasing. In seguito, gli algoritmi di Tag SNP Selection selezionano un piccolo sottoinsieme di SNPs sugli aplotipi, tali da rappresentare gli aplotipi con una perdita di informazione ridotta. Utilizzando gli SNPs si ricava l'informazione dei genotipi di un numero maggiore di individui. Di nuovo, vengono impiegati algoritmi di Haplotype Phasing per inferire i nuovi aplotipi. Infine, sugli aplotipi viene eseguita l'Haplotype-Disease Association per identificare l'associazione tra un aplotipo, o insiemi di aplotipi, e una malattia specifica.
In contrasto con l'analisi computazionale, quella tradizionale si basa su esperimenti bio-molecolari per ottenere direttamente l'informazione sugli aplotipi. Perciò i risultati sono più accurati e, in un futuro prossimo, potranno diventare uno standard per l'analisi degli aplotipi. Comunque, fino ad allora, le due procedure computazionali, Haplotype Phasing e Tag SNP Selection, risultano più utili per gli studi di larga-scala.

Capitolo 2

Il Problema di Tag SNPs Selection

La maggior parte della variazione genetica tra gli individui può essere caratterizzata da *Single Nucleotide Polimorphisms*(SNPs), i quali rappresentano mutazioni relative ad una singola posizione all'interno di un nucleotide che occorrono durante la storia umana e che vengono trasmessi a livello ereditario. Per questo gli SNPs sono alla base degli studi genetici che mirano ad individuare un'associazione tra variazioni di geni e malattie complesse. Negli studi di larga-scala, generare tutti gli SNPs di una regione candidata risulta un'operazione estremamente lunga e costosa, pertanto l'obiettivo è selezionare un sottoinsieme di SNPs che risulti sufficientemente informativo per determinare l'associazione malattia-gene, ma piccolo abbastanza per ridurre il carico necessario a derivare l'informazione sui genotipi, tale processo è noto come *Tag SNPs Selection*.

2.1 Descrizione del problema

Negli studi di larga-scala, per ridurre i costi proibitivi necessari all'estrazione delle informazione degli aplotipi, sono state introdotte due metodologie:

- *Pilot Study*: viene selezionato un piccolo campione della popolazione e, considerando tutti gli SNPs, vengono estratti i genotipi. Attraverso metodi statistici vengono inferiti gli aplotipi e si seleziona l'insieme dei tag SNPs da utilizzare nel population study.
- *Population Study*: sulla base dei tag SNPs, vengono estratti i genotipi

e inferiti gli aplotipi della popolazione rimanente e, a partire da questi, si estrapola l'intero insieme degli aplotipi.

In figura 2.1 sono illustrate le fasi caratterizzanti pilot study e population study ed il ruolo che rivestono i tag SNPs al loro interno.

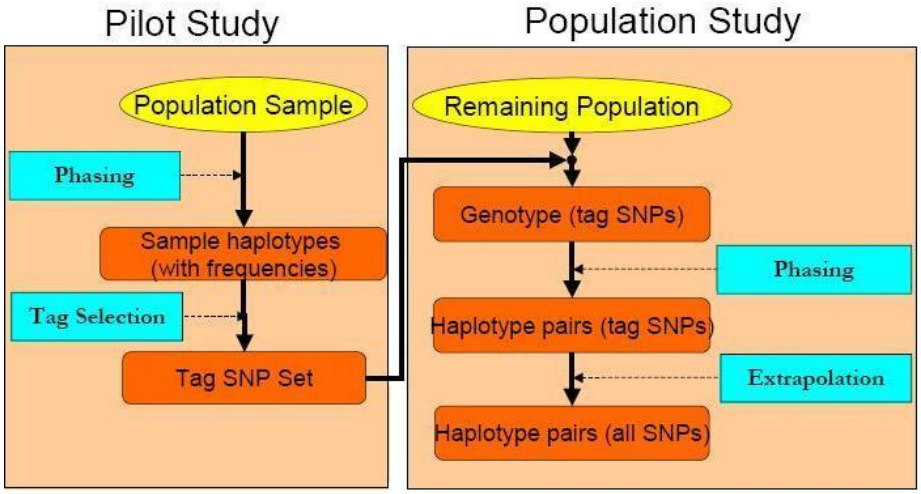


Figura 2.1: Flusso di selezione dei tag SNPs e della ricostruzione degli aplotipi

Formalmente, il problema di tag SNPs selection può essere definito come segue: Sia $S = \{s_1 ... s_m\}$ l'insieme di m SNPs in una regione candidata, e $D = \{h_1 ... h_n\}$ un insieme di n aplotipi composti da m SNPs, tale che h_i è un vettore i cui elementi sono uguali a 0 quando l'allele di uno SNP è *maggiore* e uguali ad 1 quando è *minore*. Supponiamo che il numero massimo di tag SNPs sia k e che una funzione $f(T', D)$ valuti quanto bene un sottoinsieme $T' \subset S$ rappresenta l'insieme iniziale D. Allora il problema di Tag SNP Selection può essere espresso come segue:

- Problema: Tag SNP Selection.
- Input: Un insieme di SNPs S, un insieme di aplotipi D, un numero massimo di tag SNPs k.
- Output: Un insieme di tag SNPs $T = \arg\max_{T' s.t. T' \subset S, |T'| \leq k} f(T', D)$.

In breve, per risolvere il problema di Tag SNP Selection, occorre trovare il sottoinsieme ottimo di SNPs, T, di dimensione $\leq k$ in base alla funzione di valutazione f, tra tutti i possibili sottoinsiemi di quello originale.
Inizialmente il Tag SNP Selection era motivato dal *linkage disequilibrium* (LD), infatti se esiste un elevato LD tra gli SNPs, l'informazione dei loro alleli potrebbe essere circa la stessa. Perciò, si può selezionare solo uno di questi SNPs ridondanti in modo che l'informazione in un aplotipo sia conservata anche in un sottoinsieme di quello originario. Comunque, quale sia la strategia migliore per selezione i tag SNPs è ancora un problema aperto.
I ricercatori hanno proposto una *varietà di misure* per rappresentare l'informazione degli aplotipi e hanno cercato di identificare un sottoinsieme di SNPs che ottimizzi tale misure. Le relazioni tra queste misure ed i loro effetti sulla selezione degli tag SNPs sono ancora oggetto di studi. Inoltre, è importante sottolineare che non esiste uno standard per valutare le performance dei diversi approcci; perciò il confronto tra questi risulta molto difficile.

2.2 Metodi per la selezione dei tag SNPs

L'approccio seguito è quello di considerare la selezione dei tag SNPs come un problema di ricostruzione dell'insieme originale degli aplotipi utilizzando solo un sottoinsieme di SNPs. Perciò l'obiettivo è selezionare un insieme di SNP che possa predire quelli non selezionati con il minimo errore. In pratica gli alleli degli SNPs da predire sono ottenuti utilizzando quelli dei tag SNPs, e l'associazione malattia-gene è condotta sulla base dell'intero insieme di aplotipi che è stato ricostruito.

2.2.1 La misura *Informativeness*

L'algoritmo seleziona un insieme informativo minimo di SNPs evitando ogni riferimento ai blocchi degli aplotipi. La selezione può essere suddivisa in tre step:

1. Determinare vicinati di LD: quali insiemi di SNPs possono essere utilizzati per inferire in modo significativo gli altri.
2. Stimare la qualità: determinare una misura di qualità che descriva quanto bene i tag SNPs inferiscono gli altri.
3. Ottimizzazione: minimizzare il numero di tag SNPs.

Il primo obiettivo è quello di ristringere la ricerca degli SNPs, in particolare l'attenzione deve essere focalizzata sugli SNPs che risultano in stretta vicinanza con i "target" che dovrebbero inferire. A causa della velocità di ricombinazione e della variabilità della densità degli SNPs; si nota che per un generico SNP s, il vicinato di SNPs $N(s)$ ad esso correlato (in LD), risulta altamente variabile. Per questo motivo come vicinato si utilizza l'unione di tutte le diverse decomposizioni dei blocchi di una regione in cui è contenuto lo SNP s. Comunque un'alternativa migliore, totalmente *block-free*, è rappresentata da mappe con metrica basata su LD, in base a cui solo gli SNPs al di sotto di una certa distanza sono da considerare correlati in modo significativo con gli altri.
La misura di qualità che stabilisce quanto bene uno SNP t può essere ricostruito sulla base del suo vicinato $N(t)$ è chiamata *informativeness*. Dato uno SNP s, gli aplotipi i, j e posto $D^s_{i,j}$ l'evento tale che gli aplotipi h_i e h_j abbiano un diverso allele in corrispondenza dello SNP s; l'"*informativeness*" dello SNP s rispetto allo SNP t è definita come:

$$I(s,t) = Pr_{i \neq j}(D^s_{i,j} \mid D^t_{i,j}),$$

dove i e j sono due aplotipi sorteggiati casualmente dall'insieme di tutte le diverse coppie di aplotipi. Osserviamo che $I(s,t) = 1$ implica completa prevedibilità. La definizione è facilmente estendibile ad un sottoinsieme di SNPs. Dato $S' \subseteq S$ e posto $D^{S'}_{i,j}$ l'evento tale che gli aplotipi h_i e h_j abbiano un diverso allele in corrispondenza di un qualche SNP $s \in S'$, allora l'*informativeness* è definita come:

$$I(S',t) = Pr_{i \neq j}(D^{S'}_{i,j} \mid D^t_{i,j}).$$

E' importante sottolineare che la definizione di *informativeness* assume la disponibilità della fase degli aplotipi; tale problema viene superato inferendo computazionalmente gli aplotipi su ogni vicinato.
L'ultimo step è quello che permette di ottimizzare il numero di tag SNPs; per semplicità si assume che la distanza tra due SNP sia semplicemente il numero di SNPs tra essi compresi e che la dimensione dei vicinati sia limitata una una certa costante w. Formalmente il problema può essere così definito:

- Input: Un insieme di S composto da n SNPs e un intero k, tale che $0 \leq k \leq n$
- Output:Un sottoinsieme $S' \subseteq S$ tale che $I(S',S) = \max_{R \subset S, |R| \leq k} I(R,S)$.

Il principale svantaggio di questo metodo è rappresentato dalla complessità computazionale della programmazione dinamica, pari a $O(nk2w)$, dove w rappresenta la prossimità fisica tra SNPs e tag SNPs, che risulta esponenziale e, per questo, andrebbe ridotta.

2.2.2 Il Metodo BEST - Best Enumeration of SNP Tags

Al contrario di altri metodi, BEST è un metodo esatto, analitico e senza perdita, che trae vantaggio da una peculiarità del genoma (limitata diversità degli aplotipi) per confinare lo spazio della ricerca.
Sia $S = \{s_1, .., s_m\}$ l'insieme di m SNPs, il metodo BEST partiziona ricorsivamente tale insieme in due gruppi H e D tali che $S = H - D$, dove H rappresenta il minimo insieme di SNPs necessari e sufficienti a derivare tutti gli altri SNPs e D rappresenta l'insieme di (m-k) SNPs derivabili da H; uno SNP d_j è derivabile da H se il suo valore in ogni aplotipo può essere espresso da una funzione Booleana $f(.)$ di elementi di H.
Il primo passo dell'algoritmo è generare l'insieme H_1 di tag SNPs non derivabili da ogni sottoinsieme di S, identificato esaminando la dipendenza Booleana di ogni s_j sull'insieme di SNPs $S \setminus s_j$ e assegnando ad H_1 ogni s_j non derivabile. Gli elementi di H_1 determinano una partizione dei rimanenti SNPs in $S \setminus H_1 = C_1 \cup D_1$. L'insieme D_1 contiene gli SNPs derivabili da H_1 e C_1 gli SNPs non derivabili, e quindi candidabili ad essere tag SNPs. In seguito è applicata una procedura che sposta uno o più elementi da C_1 a H_1 e da C_1 a D_1. Quando l'insieme di SNPs candidati è vuoto, se la procedura restituisce uno o più insiemi necessari e sufficienti di tag SNPs l'algoritmo termina. Se non viene trovato un insieme, la procedura viene ripetuta su ogni insieme H.

2.2.3 Il Metodo STAMPA - Selection of Tag SNPs to Maximize Prediction Accuracy

Dato un campione di genotipi, il metodo STAMPA (Selection of Tag SNPs to Maximize Prediction Accuracy) trova un insieme di tag SNPs a partire da un insieme di individui non collegati. L'informazione sulla fase viene ricavata solo per un insieme di training di riferimento al fine di selezionare i tag SNPs e, conseguentemente, predire il valore degli altri SNPs.
Un altro problema cruciale riguarda la definizione di un'adeguata misura della qualità di predizione, in questo caso viene proposta una misura, detta *prediction accuracy*, che direttamente valuta la qualità media nella predizione degli SNPs.

Informalmente, l'algoritmo di predizione è una funzione f che utilizza il valore dei genotipi dei tag SNPs per predire quello dei restanti SNPs. Per un dato vettore $q \in \{0,1,2\}^t$ di tag SNPs, sia $f_j(q)$ la j-esima componente di tale vettore e sia $z_t : \{0,1,2\}^m \rightarrow \{0,1,2\}^t$ la restrizione del genotipo (di lunghezza m) alle posizioni dei tag SNPs; l'obiettivo è minimizzare la seguente espressione:

$$\eta = \sum_{j=1}^{m} Pr[f_j(z_T(g)) \neq g(j)]$$

Il problema principale nel raggiungere tale obiettivo è che non si conoscono le frequenze dei genotipi; perciò si utilizza un insieme di genotipi di training per apprenderne la distribuzione. Come conseguenza l'espressione precedente è minimizzata da genotipi selezionati in modo casuale dall'insieme di training, formalmente: dato $X_T = |\{(i,j)|g_{i,j} \neq f_j(z_T(g_i))\}|$, dove $g_{i,j}$ è il j-esimo SNP di g_i, si vuole determinare un insieme T di tag SNPs tale da minimizzare X_T. L'algoritmo di predizione si basa sul fatto che la correlazione tra SNPs tende a decadere col crescere della distanza fisica. Perciò la funzione di predizione deduce il valore di uno SNP utilizzando il valore dei due tag SNPs ad esso più vicini. La procedura `Predict(`i, j_1, j_2, a_1, a_2`)` utilizza il voto a maggioranza al fine di determinare quale valore appare con maggior probabilità in posizione i posto che le posizioni j_1 e j_2 hanno valore $a_1 \in \{0,1,2\}$ e $a_2 \in \{0,1,2\}$ rispettivamente.
Supponiamo che il nostro campione consista di sei aplotipi con cinque SNPs, e che gli SNPs 1 e 2 siano selezionati come tag SNPs e che si voglia costruire un nuovo campione come mostrato in figura 3.2.a).

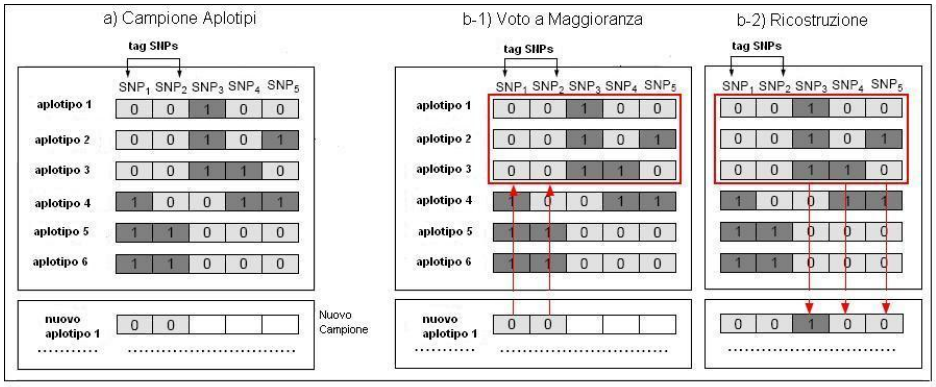


Figura 2.2: Voto a Maggioranza per la predizione degli SNPs

Per ricostruire gli SNPs, per prima cosa, si individuano gli aplotipi i

cui alleli dei tag SNPs non sono cambiati. In figura 3.2.b-1) questi aplotipi sono evidenziati in blocco. Infine, nel nuovo aplotipo, per ogni SNP da predire si assegna l'allele che occorre più spesso negli aplotipi identificati precedentemente, come in figura 3.2.b-2). Come conseguenza, la funzione basata sul voto a maggioranza tende ad assegnare ai nuovi aplotipi gli alleli comuni piuttosto che quelli rari.
Per la selezione dei tag SNPs vengono proposti due algoritmi:

- *algoritmo esatto*: per ogni coppia di SNPs calcola tre funzioni ausiliarie di punteggio che calcolano il numero di errori in una subregione (insieme contiguo di SNPs), dato un insieme parziale di tag SNPs. Viene poi definita una funzione f utilizzata nel processo ricorsivo della programmazione dinamica per minimizzare X_T. (la complessità è $O(m^3n)$, il che rende l'algoritmo molto dispendioso se l'insieme di SNPs è ampio. In molti casi pratici si usa un bound c sulla massima distanza negli SNPs situati tra due tag SNPs vicini).

- *random sampling*: utilizzato nel caso in cui si è interessati a trovare velocemente un numero ristretto di SNPs per predire gli altri. Si generano casualmente 100 insiemi di tag SNPs $T_1, T_2, ..., T_{100}$, si calcola X_{T_i} per i=1,2,..,100 e si sceglie l'insieme che minimizza X_{T_i}.

2.2.4 Metodi basati sulla Riduzione Lineare

Il metodo proposto è basato sull'algebra lineare, ed è già stato utilizzato con successo per il problema di inferenza degli aplotipi. Per misurare la qualità dell'approccio si è calcolato direttamente il numero di predizioni corrette. Per definizione i tag SNPs sono k siti $t_1, t_2, ..., t_k$ che permettono di ricostruire un intero aplotipo h a partire dal valore del vettore $h_k = (h_{t_1}, ..., h_{t_k})$. Al fine di descrivere formalmente la ricostruzione si utilizza un vettore $f = (f_1, ..., f_m)$ chiamato *reconstruction function*, dove $f_j = (x_1, ..., x_k)$ è una funzione k-variabile pari al j-esimo sito dell'aplotipo ricostruito $\overline{h} = (f(h_k))$.
Il problema è quindi determinare i k siti e la funzione di ricostruzione che minimizzano la distanza di Hamming attesa tra un qualsiasi aplotipo $h \in P$ (insieme della popolazione) e la sua predizione $\overline{h}$.
Per ottimizzare la selezione l'approccio standard è quello che prevede di prendere, in modo casuale, un sottoinsieme H della popolazione P, estrarre gli SNPs, trovare la funzione di ricostruzione e, infine, controllare l'accuratezza della predizione sull'insieme $P \backslash H$. I risultati vanno poi mediati su diverse scelte casuali di H.
Di solito, nelle sequenze genetiche derivanti dagli aplotipi umani, il numero

di siti *equivalenti* è molto elevato. Due siti sono detti equivalenti se le corrispondenti colonne 0-1 sono uguali o complementari; perciò una volta selezionato uno di questi siti, non ha senso considerare gi altri, dal momento che non apportano nessuna informazione additiva. Bisogna osservare che per restaurare una colonna eliminata non è possibile applicare direttamente combinazioni lineari di altre colonne, perchè colonne equivalenti sono linearmente indipendenti. Per superare questo ostacolo si sostituiscono gli 0 con -1. Il vantaggio della notazione $(-1, 1)$ è che due siti sono equivalenti se e solo se sono collineari (linearmente dipendenti). Il metodo di selezione è basato sulla sola selezione di SNP linearmente indipendenti come tag SNPs.
L'implementazione è molto efficiente, infatti utilizzando l'eliminazione di Gauss (complessità $O(n^2m)$) è possibile trasformare la matrice H, in una matrice ridotta H' con esattamente r righe non nulle, dalle quali si estraggono gli r tag SNPs. Il problema dell'eliminazione di Gauss è che sceglie i primi tag SNPs linearmente indipendenti, senza considerare che l'informazione di un aplotipo è diffusa su tutta la lunghezza dello stesso. Per tale motivo è stata sviluppata una variante al metodo LR, denominata Randomized LR (RLR), in cui gli SNPs vengono permutati in modo casuale. Nel caso in cui alcuni SNPs non siano risolti si applica il metodo RLR con Postprocessing (RLRP), attraverso il quale agli SNPs non risolti viene assegnato il valore -1 se il valore della predizione è negativo e 1 altrimenti.

2.2.5 Metodi basati sulla Regressione Lineare Multipla

Il metodo proposto è basato sulla regressione lineare multipla (MLR). Per un generico SNP s da predire esistono, nel caso degli aplotipi, solo due possibili risoluzioni, dette s_0 e s_1, in cui il valore dello SNP sconosciuto risulta rispettivamente 0 oppure 1. Per i genotipi esistono, invece, tre possibili risoluzioni, dette s_0, s_1 e s_2, corrispondenti rispettivamente al valore 0, 1 o 2. Il metodo di predizione assume che la risoluzione più probabile dovrebbe essere quella "più vicina" all'insieme dei tag SNPs T. La distanza tra la risoluzione dello SNP e T è misurata attraverso s e la sua proiezione sullo spazio vettoriale $span$(T). Computazionalmente tale distanza è calcolata come $dist(T, s_i) =\mid T \cdot (T^t \cdot T)^{-1} \cdot T^t \cdot s_i - s_i \mid$. La figura 2.3 mostra come il valore predetto per lo SNP sia 1 dal momento che la distanza tra s_1 e la sua proiezione s_1^T è più breve rispetto a s_0 e a s_2. La complessità dell'algoritmo è $O(kn^2)$) e, invece di utilizzare la notazione $\{0, 1, 2\}$, viene impiegata la codifica $\{-1, 1, 0\}$, in cui gli 0 son rimpiazzati dai -1 e i 2 dagli 0; in quanto degli esperimenti hanno evidenziato che quest'ultima necessita del 30% di tag SNPs in meno per raggiungere la stessa accuratezza nella previsione dell'altra.

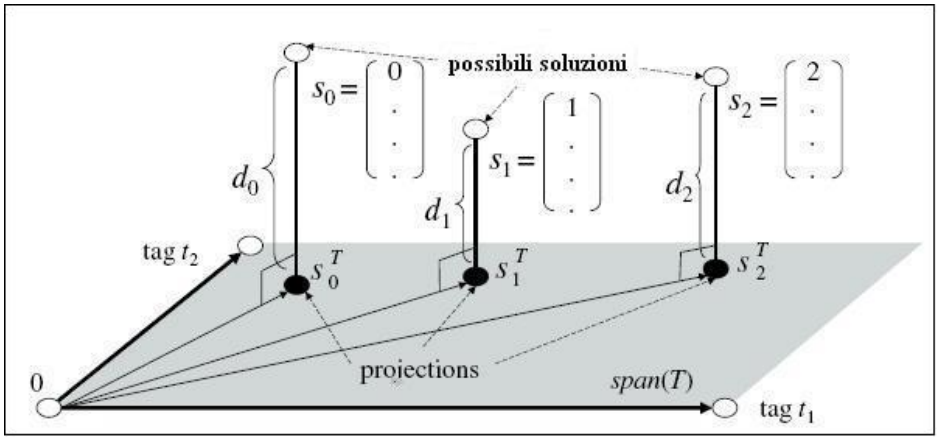


Figura 2.3: Algoritmo MLR

Per la selezione dei tag SNPs su cui applicare il metodo di predizione, occorre tenere in considerazione il fatto che un algoritmo esatto può risultare troppo lento, per questo sono state proposte due euristiche:

1. *Stepwise Tagging algorithm* (STA),
2. *Local-Minimization Tag Selection algorithm* (LMT).

Il primo algoritmo (il più veloce) inizia con il miglior tag SNP, detto t_0 (quello che minimizza l'errore di predizione) e continua aggiungendo t_1, la migliore estensione all'insieme $\{t_0\}$. La procedura continua finchè non si raggiunge il numero di tag SNPs prefissato (k). La complessità è pari a $O(kmnT)$ dove T rappresenta il runtime dell'algoritmo di predizione.
Il secondo algoritmo comincia con l'insieme dei tag SNPs selezionato da STA e, iterativamente, rimpiazza ogni singolo tag SNP con la migliore possibile scelta senza modificare tutti gli altri. Tale rimpiazzamento viene ripetuto finchè il miglioramento nella qualità di predizione cessa di essere significativo.

2.2.6 Metodi basati sulla PLI

Il metodo proposto si basa sulla formulazione di programmazione lineare intera riportata di seguito:

- Input: un insieme di aplotipi $h_1, h_2, .., h_m$ ognuno di n SNPs

- Output: un insieme di tag SNPs tale che ogni coppia di aplotipi differisce almeno di un tag SNP

$$\min \sum_{j=1}^{n} x_j$$

$$\sum_{j:h_i(j)\neq h_{i'}(j)} x_j \geq 1, \quad 1 \leq i \leq i' \leq m$$

dove x_j è una variabile binaria tale che:

$$x_j = \begin{cases} 1, & \text{se lo SNP } j \text{ è selezionato come tag SNP;} \\ 0, & \text{altrimenti.} \end{cases}$$

Nella fase di ricostruzione degli aplotipi sono usuali errori dovuti ad una estrapolazione non corretta, ad inferenze sbagliate, o all'incapacità di ricostruzione. Per tale motivo un secondo problema da risolvere è quello che ha come obiettivo la massimizzazione del numero di coppie di aplotipi distinguibili attraverso i tag SNPs selezionati (formulazione ILP1):

$$\max \sum_{1\leq i\leq i'\leq m} y_{i,i'}$$

$$\sum_{j:h_i(j)\neq h_{i'}(j)} x_j \geq y_{i,i'}, \quad 1 \leq i \leq i' \leq m$$

$$\sum_{j=1}^{n} x_j = K$$

dove K è il numero massimo di SNP selezionabili e $y_{i,i'}$ è una variabile binaria tale che:

$$y_{i,i'} = \begin{cases} 1, & \text{sse gli aplotipi } h_{i'} \text{ e } h_i \text{ differiscono almeno di un tag SNP;} \\ 0, & \text{altrimenti.} \end{cases}$$

Basandosi su tale formulazione la precisione della ricostruzione può essere migliorata considerando le frequenze degli aplotipi p_i e $p_{i'}$ stimate dall'iniziale

insieme campione (formulazione ILPf - ILP with frequency). In questo caso l'obiettivo è trovare i K tag SNPs che massimizzano la probabilità totale di distinguere coppie di aplotipi.
In figura 2.4 sono riportate le prestazioni dell'algoritmo su uno stesso insieme di dati variando il numero di tag SNPs. Per il confronto viene anche riportato il risultato derivante da selezioni casuali degli stessi (RAND).

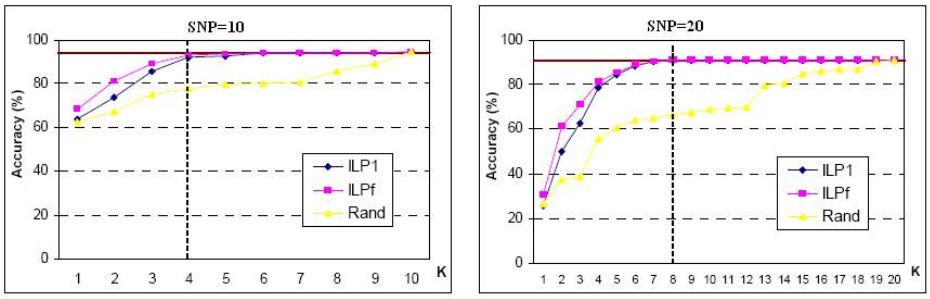


Figura 2.4: Confronto delle prestazioni con un numero di tag SNPs diverso

Risulta evidente che l'impiego delle frequenze degli aplotipi migliora la qualità della ricostruzione rispetto alla selezione casuale e alla formulazione ILP1.

2.3 Confronti e considerazioni

L'interesse principale dell'attuale ricerca genetica è legata all'associazione malattia-gene, che significa identificare quale variazione di DNA, o quale insieme di variazioni, è altamente associato con una specifica malattia. Per condurre studi su vasta scala si ricorre all'analisi computazionale degli aplotipi,e in modo più specifico, alla procedura di *Tag SNP Selection* la quale permette di raggiungere performance poco costose, veloci e relativamente accurate. Restano, però, dei limiti che devono essere superati: qual'è la strategia migliore per selezionare i tag SNPs ?
I confronti tra i metodi illustrati in precedenza mettono in evidenza tale aspetto.

L'approccio basato sulla misura Informativeness mostra diversi vantaggi rispetto agli altri. (1) non è basato sulla partizione a blocchi; (2) utilizza la

relazione *multi*-SNP; (3) non necessita di informazioni sul fenotipo. Nonostante ciò, attualmente le performance di questo approccio sono limitate dalla complessità computazionale delle procedure di programmazione dinamica; per tale ragione si potrebbe pensare di applicare altre tecniche di machine learning per migliorare le performance, in particolare approcci probabilistici.

Un importante risultato di BEST è che riesce ad identificare l'insieme minimo di tag SNPs perfino in insiemi di aplotipi che non sarebbe possibile esaminare con la ricerca esaustiva. Inoltre gli approcci correnti per l'individuazione di tag SNPs si basano sul linkage disequilibrium e su misure stocastiche di diversità degli aplotipi, ma la selezione analitica dei tag SNPs in regioni arbitrarie del genoma è più efficiente.
Nonostante questi vantaggi alcuni esperimenti hanno evidenziato che i metodi basati sulla PLI [1] risultano più veloci di BEST.

Il metodo STAMPA ha il vantaggio di utilizzare l'informazione dei genotipi, più spesso reperibile rispetto a quella degli aplotipi. Inoltre non si basa sulla partizione in blocchi dl genoma, il che rende gli algoritmi più semplici.
Rispetto ad altri metodi, sfrutta le dipendenze *multi*-SNP per selezione i tag SNPs e questo comporta una riduzione del numero di tag SNPs, infatti questo limita a 2 il numero di tag SNPs per ogni SNP da predire. Inoltre, tutti i metodi di programmazione dinamica garantiscono di trovare un ottimo globale rispetto alla misura data. Comunque, l'efficenza della predizione è ancora limitata da alcune restrizioni come il numero fisso di tag SNPs.
Non ultimo, utilizza la fase solo per l'insieme di training e questo lo rende più veloce di altri algoritmi di selezione dei tag SNPs.

Altre considerazioni sulla qualità della soluzione data dalla selezione degli SNPs e dai metodi di predizione possono trarre dall'esperimento *leave-one-out*. Uno alla volta, ogni individuo è rimosso dal campione e vengono estratti i tag SNPs utilizzando solo gli individui rimasti. L'individuo "left out" è ricostruito sulla base dei tag SNPs e dei rimanenti individui nel campione. La qualità di predizione è rappresentata dalla percentuale di SNPs correttamente predetti rispetto a tutti gli individui. La tabella sottostante mostra il numero di tag SNPs necessari a raggiungere una qualità di predizione dell'80 % e del 90% nel caso del metodo MLR/STA [2, 3], STA combinato con MLR, e del metodo STAMPA [4] su sei insiemi di dati diversi. Ne risulta che il primo metodo richiede un numero di tag SNPs minore ed è più veloce.

Acc.	Algorithm	ENm013	ENr112	ENr113	STEAP	TRPM8	5q31
80%	MLR	2	6	4	1	1	1
	STAMPA	5	9	11	2	3	2
90%	MLR	6	14	10	1	4	5
	STAMPA	12	17	18	2	6	6

Figura 2.5: Confronto tra i metodi MLR/STA e STAMPA

Dal momento che l'algoritmo di predizione MLR può essere applicato anche agli aplotipi, può essere confrontato anche con la misura Informativeness [5, 6] e con il metodo LR [7], in questo caso, MLR/STA risulta migliore rispetto agli altri due per entrambi gli insiemi di dati considerati (figura2.6).

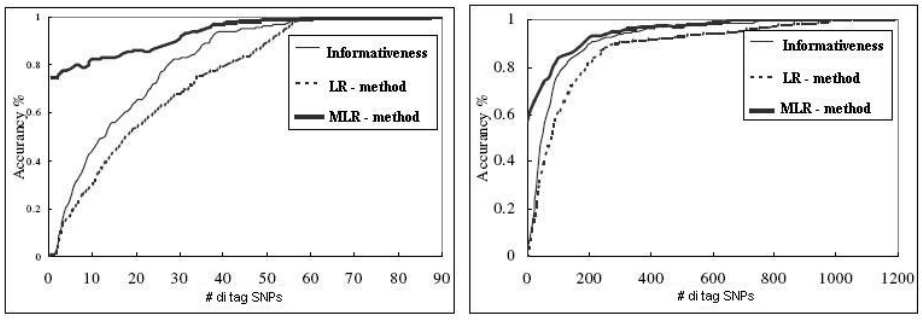


Figura 2.6: Confronto tra tre metodi di predizione basati sugli aplotipi

Un altro confronto interessante è quello tra la misura Informativeness [5, 6], basato sulla definizione dei vicinati, ed il metodo RLRP [7, 8], che ricostruisce ogni SNP sulla base di tag SNPs che possono essere anche molto lontani da esso. La figura sottostante (fig.2.7) mette in evidenza che, esaminando due diversi insiemi di dati ed utilizzando come tag SNPs il 10% dei SNPs, il metodo RLRP raggiunge l'80% di precisione mentre l'altro metodo solo il 20%.

Come si evince dall'analisi appena conclusa, le misure proposte per rappresentare l'informazione degli aplotipi sono numerose, e non di meno sono i metodi sviluppati per identificare un sottoinsieme di SNPs che le ottimizzi. Senza dubbio gli approcci basati sulla predizione degli SNPs hanno apportato miglioramenti significativi rispetto ai precedenti metodi di selezione dei

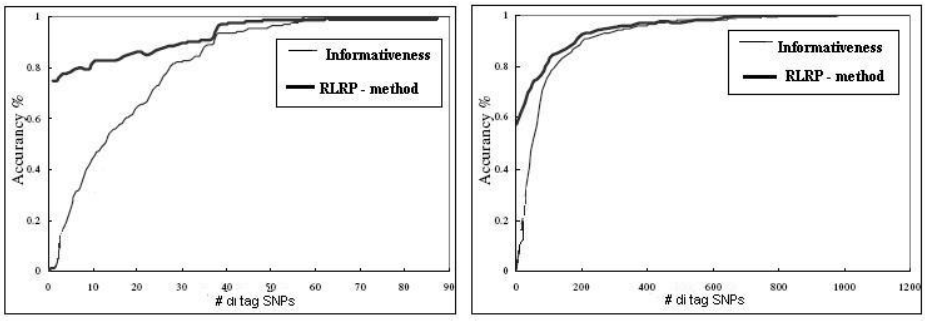


Figura 2.7: Confronto tra la misura *Informativeness* e il metodo RLRP

tag SNPs. Quello che manca è uno standard per valutare in modo univoco le performance dei diversi approcci esistenti; nonchè uno studio più approfondito sulle funzioni di ricostruzione da utilizzare, in alternativa al criterio del voto a maggioranza, nella predizione degli SNPs.
Gli studi futuri dovrebbero essere volti, innanzitutto, alla risoluzione di tali problematiche; per poi giungere alla formulazione di una strategia che risulti inequivocabilmente migliore delle altre.

Capitolo 3

Formulazione del modello

Dal momento che si trattano dati binari, il problema di selezionare un sottoinsieme SNPs di dimensione minima, che garantisca di distinguere tutte le possibili coppie di aplotipi dell'insieme di input, può essere formulato come un Problema di Programmazione Lineare Intera, più precisamente come un Problema di Set Covering.
Ciò nonostante, è stato impiegato anche un algoritmo di Clustering per studiarne il comportamento e per confrontarne i risultati col Set Covering in fase di ricostruzione degli aplotipi dell'insieme di input.

3.1 Il Set Covering

Dati un insieme $I = \{1, 2, ..., m\}$ di m oggetti e un insieme $J = \{1, 2, ..., n\}$ di n indici, siano $P_j, j \in J$, n sottoinsiemi di I, a ciascuno dei quali è associato un costo c_j . Il problema di Set Covering consiste nel determinare una copertura minima J^* dell'insieme I.
La classica formulazione del Set Covering è quindi la seguente:

$$\min \sum_{j \in J} c_j x_j$$

$$\sum_{j \in J} a_{ij} x_j \geq 1, \quad i \in I$$

$$x_j \in \{0, 1\}, \quad j \in J$$

dove a_{ij} è una variabile binaria tale che:

$$a_{ij} = \begin{cases} 1, & \text{se l'elemento i appartiene al sottoinsieme } P_j; \\ 0, & \text{altrimenti.} \end{cases}$$

e x_j è una variabile decisionale tale che:

$$x_j = \begin{cases} 1, & \text{se l'indice j appartiene a } J^*; \\ 0, & \text{altrimenti.} \end{cases}$$

3.1.1 Set Covering per il tag SNP selection

Una definizione formale del problema di selezione dei tag SNPs è stata presentata nel capitolo precedente. L'input è un insieme di SNPs $\{s_1...s_m\}$, un insieme di aplotipi $\{h_1...h_n\}$ e un numero massimo di tag SNPs β. Uno SNP s_k differenzia (copre) una coppia di aplotipi $\{h_i, h_j\}$ se l'allele (i,k) ha un valore diverso dall'allele (j,k). Il problema di selezionare un sottoinsieme di SNPs di dimensione minima, che garantisca di distinguere tutte le possibili coppie di aplotipi può essere formulato come segue:

$$\max \alpha$$

$$\sum_{k=1}^{m} a_{ijk} x_k \geq \alpha, \quad \forall (i,j), i \neq j$$

$$\sum_{k=1}^{m} x_k = \beta$$

$$x_k \in \{0,1\}, \quad k = 1...m$$

$$\alpha \in \{0, \beta\}$$

dove β rappresenta il numero massimo di SNPs selezionabili come tag e α una variabile intera che quantifica le coppie di aplotipi distinguibili attraverso i tag SNPs.
Il fatto che α possa assumere valori compresi nell'intervallo $(0; \beta)$ può sembrare un' inutile ridondanza, dal momento che basterebbe avere $\alpha = 1$ per essere certi che $\forall (i,j), i \neq j$ le coppie $\{h_i, h_j\}$ di aplotipi siano distinguibili; ma, al contempo, questa scelta mette al sicuro dai eventuali problemi, come l'errata lettura di uno SNP, che potrebbero compromettere i risultati ottenuti. L'obiettivo è quindi massimizzare le coppie di aplotipi distinguibili attraverso un singolo tag SNP.

x_k è una variabile binaria tale che:

$$x_k = \begin{cases} 1, & \text{se lo SNP } k \text{ è selezionato come tag SNP;} \\ 0, & \text{altrimenti.} \end{cases}$$

e a_{ijk} è una variabile binaria tale che:

$$a_{ijk} = \begin{cases} 1, & \text{sse lo SNP k differenzia la coppia di aplotipi } (h_i, h_j); \\ 0, & \text{altrimenti.} \end{cases}$$

In alternativa a quello appena presentato, è stato pensato anche un modello "simmetrico", che viene riportato di seguito:

$$\min \beta$$

$$\sum_{k=1}^{m} a_{ijk} x_k \geq \alpha, \quad \forall (i,j), i \neq j$$

$$\sum_{k=1}^{m} x_k = \beta$$

$$x_k \in \{0,1\}, \quad k = 1...m$$

$$\alpha \in \{0, \beta\}$$

In questo caso l'obiettivo è minimizzare β, il numero di tag SNPs, fissato il valore di α. ovvero il numero delle coppie di aplotipi distinguibili attraverso un tag SNP.

3.2 Il Clustering

Il Clustering è il problema di riunire un insieme di dati in gruppi, il cui significato dipende dall'applicazione. Non c'è un metodo migliore per clusterizzare la scelta di un metodo rispetto ad un altro dipende soprattutto dal caso specifico. Gli algoritmi di Clustering possono essere divisi in due grandi classi:

- algoritmi di Clustering che prevedono la conoscenza a priori del numero di gruppi in cui i dati vanno riuniti.
- algoritmi di Clustering che non prevedono la conoscenza a priori del numero di gruppi in cui i dati vanno riuniti.

Tra gli algoritmi in cui si prevede la conoscenza a priori del numero di Cluster, possiamo annoverare il metodo *K-Means*.
K-Means [MacQueen,1967] è uno dei più semplici algoritmi che risolvono il problema del Clustering. L'idea fondamentale consiste nel definire k centroidi, dove k è il numero di Clusters scelto a priori. Dato un insieme di vettori d'ingresso, ogni vettore è di dimensione n, tale insieme deve essere raggruppato in k Clusters. Un centroide è un vettore di dimensione n che rappresenta il centro del Cluster. L'algoritmo consiste nei seguenti passi:

1. Posizionare i k centroidi nello spazio a n dimensioni. Tale posizionamento rappresenta la scelta iniziale per i centri dei Clusters.
2. Assegnare ad ogni vettore da clusterizzare al cluster che ha il centroide più vicino alla propria posizione.
3. Una volta che tutti i vettori sono stati assegnati ad un Cluster, calcolare nuovamente la posizione dei k centroidi.
4. Ripetere i punti 2 e 3 finché i centroidi non si muovono più.

I principali svantaggi dell'algoritmo K-Means sono i seguenti:

- Non è specificato il modo in cui si possono inizializzare i centroidi. Un metodo popolare è quello di inizializzarli con valori casuali.
- Il risultato prodotto dipende dai valori d'inizializzazione scelti per i centroidi. Una soluzione standard di questo problema è provare con un certo numero di inizializzazioni differenti.
- Il risultato dipende dalla scelta del numero di Clusters k.

Lo svantaggio di maggior rilievo è l'ultimo visto che non c'è un metodo per trovare il numero ottimo di Cluster, che valga in generale.
Una semplice soluzione del problema può essere quella di eseguire l'algoritmo con diversi valori di k e scegliere il miglior valore di k secondo certi criteri che definiscono la qualità della clusterizzazione. Bisogna stare molto attenti con questo approccio perché ,aumentando il numero dei Clusters k , la funzione di errore diminuisce per definizione ma aumenta il rischio di overfitting. Per fare un esempio, se il numero di elementi da clusterizzare è pari al numero di Cluster, un risultato dell'algoritmo di Clustering che manda la funzione di errore a zero può essere quello in cui ogni elemento è centroide di un Cluster. Tale soluzione manda la funzione di errore a zero, ma il risultato di questa clusterizzazione risulta inutilizzabile.

3.2.1 Clustering per il tag SNP selection

Nella selezione dei tag SNPs, il metodo *K-Means* è stato impiegato sugli SNPs, per cui, in base alla struttura dell'algoritmo, si è scelto β come numero di Cluster da creare. Come centroidi iniziali sono stati scelti β SNP scelti casualmente dall'insieme di input. Mentre per l'assegnamento degli SNPs ai Clusters è stata utilizzata una metrica basata sulla *Distanza di Hamming*[1]; in particolare per ogni SNP è stata dapprima calcolata la sua distanza di Hamming rispetto ad ogni centroide e, poi, lo si è assegnato al Cluster il cui centroide fosse a distanza minima.
Terminato l'assegnamento occorre calcolare nuovamente la posizione dei β centroidi. Tale operazione viene svolta effettuando un *Voto a Maggioranza*[2] tra tutti gli SNPs assegnati a quel Cluster così come mostrato in figura 3.1; si contano le occorrenze di 0 ed 1 e si prende il valore con più occorrenze, nel caso di parità viene scelto il valore 1.

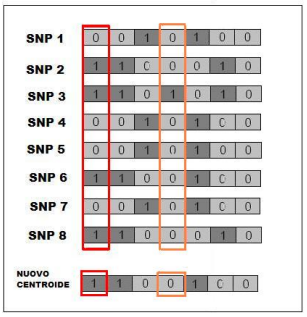


Figura 3.1: Ricalcolo del centroide

Dopo aver ricalcolato i centroidi viene ripetuta la procedura di assegnamento degli SNPs illustrata in precedenza. Il processo viene riiterato fin quando gli SNPs cessano di spostarsi da un Cluster all'altro.
Terminata questa fase, in ogni Cluster viene selezionato uno SNP, per un totale di β SNPs che costituirà l'insieme dei tag SNPs. La scelta viene fatta sempre sulla base della distanza di Hamming: lo SNP più vicino al centroide

[1] la distanza di Hamming tra due stringhe di ugual lunghezza è il numero di posizioni nelle quali i simboli corrispondenti sono diversi. In altri termini, misura il numero di sostituzioni necessarie a convertire una stringa nell'altra, o il numero di errori che hanno trasformato una stringa nell'altra.

[2] la scelta vincitrice è quella per cui il numero di occorrenze è maggiore della metà del totale

del proprio Cluster viene scelto come rappresentante. Può sembrare strano non scegliere i centroidi come tag SNPs, in realtà questa scelta è dovuta al fatto che il ricalcolo dei centroidi previsto dall'algoritmo può portare alla generazione di centroidi che non corrispondono a nessun SNP dell'insieme di input. Una tale eventualità si è rivelata penalizzante in fase di ricostruzione e, quindi, scartata.

3.3 La Ricostruzione

Il miglior modo per valutare l'efficacia dell'insieme di SNPs selezionati come tag è testarlo nella ricostruzione degli aplotipi dell'insieme di input.
Per ogni aplotipo viene calcolato, secondo la distanza d Hamming, il blocco di righe ad esso più vicino in funzione delle posizioni dei tag e, su questo blocco di righe, viene fatto il voto a maggioranza per determinare il valore da assegnare alle posizioni non-tag. Anche in questo caso, come per il ricalcolo dei centroidi, nel caso di ugual numero di occorrenze il valore predominante sarà l'uno.
Ad esempio, in figura 3.2 sono mostrate tutte le fasi relative alla ricostruzione dell'aplotipo 1. Dal momento che gli SNPs selezionati come tag sono lo SNP 1 e lo SNP 2, le posizioni 1 e 2 dell'aplotipo 1 sono note ed hanno valore [0 0]. Di conseguenza, il blocco-righe relativo all'aplotipo 1 sarà composto da tutte quelle righe che presentano il valore [0 0] rispettivamente in posizione 1 in posizione 2. Se non dovessero esistere delle righe con valore [0 0], si procede scegliendo tra quelle a distanza di hamming minima.

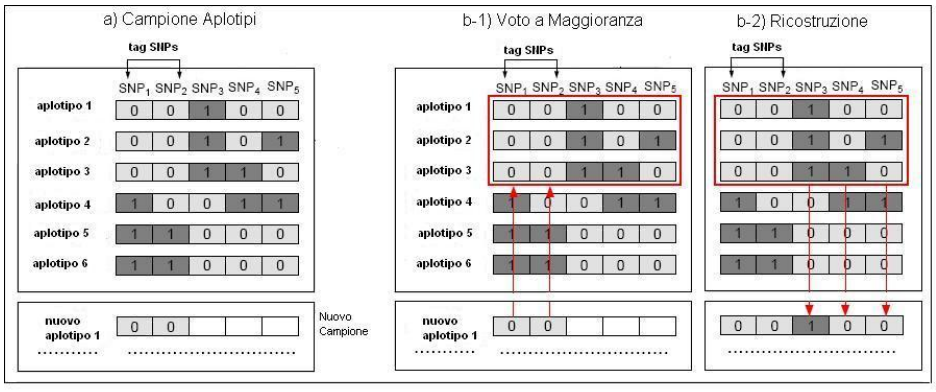


Figura 3.2: Voto a Maggioranza

Per l'aplotipo 1, le righe più vicine sono tre: l'aplotipo 2, l'aplotipo 3, e lo stesso aplotipo 1; tutte a distanza di hamming pari a zero. Su di esse

viene effettuato il voto a maggioranza per determinare il valore da inserire nelle posizioni 3, 4 e 5 dell'aplotipo 1. In questo esempio tali posizioni assumeranno i valori [1 0 0].
Terminata la ricostruzione è facile verificare eventuali errori commessi confrontando, riga per riga, aplotipi originari ed aplotipi ricostruiti. L'obiettivo deve essere rimanere sotto la soglia del 5% d'errore. In riferimento all'esempio precedente, si può notare (figura 3.3b) che non tutti gli aplotipi sono stati ricostruiti correttamente, in particolare si fa riferimento agli aplotipo 2 e 3, evidenziati in figura.

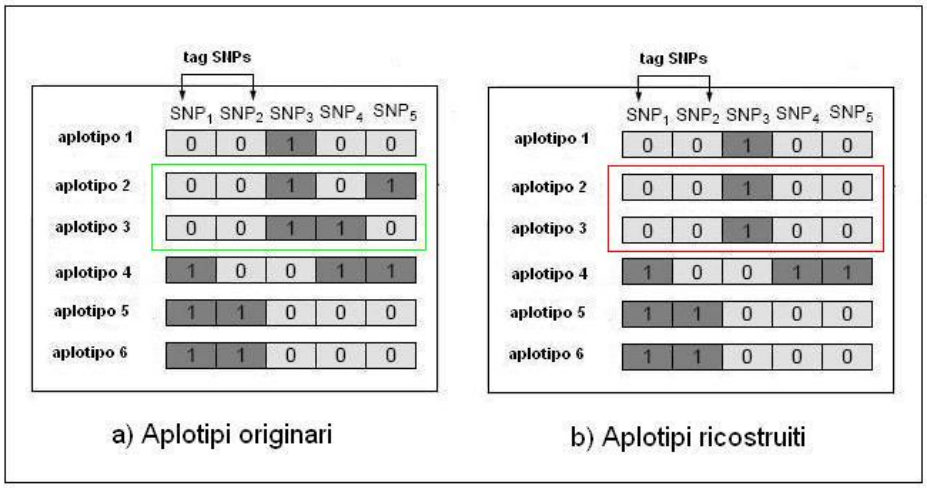


Figura 3.3: Valutazione delle prestazioni (1)

L'errore supera addirittura la soglia del 5%, questo sta ad indicare che la scelta degli SNPs 1 e 2 come tag non si è rivelata premiante. Si può facilmente osservare che scegliendo altri SNPs come tag i risultati cambiano notevolmente.
Ad esempio, selezionando gli SNPs 3 e 4 come tag solo l'aplotipo 1 risulta mal ricostruito (figura 3.4), e ancora, semplicemente aumentando di uno il numero di tag SNPs e scegliendo gli snp 3, 4 e 5 non si comme nessun errore di ricostruzione.

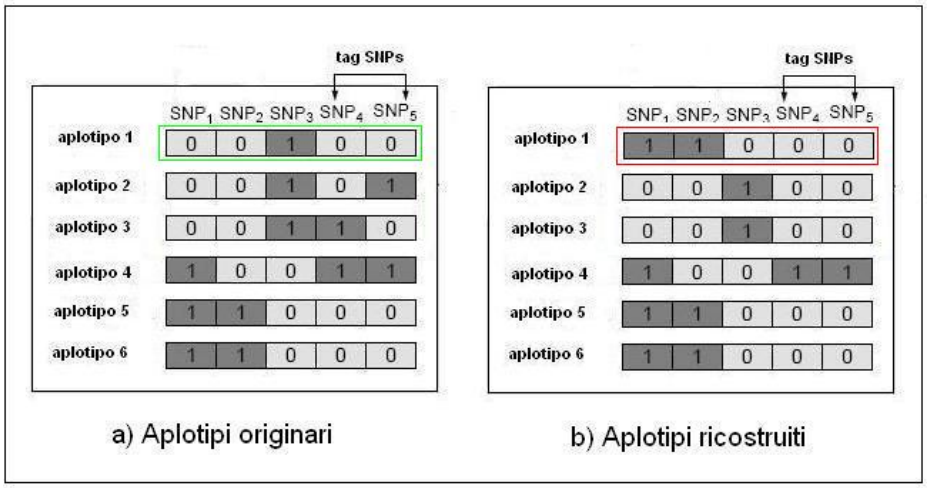


Figura 3.4: Valutazione delle prestazioni (2)

Capitolo 4

Strumenti utilizzati per la risoluzione del problema

4.1 Il programma ILOG - CPLEX

Ilog CPLEX è un software per la programmazione lineare mista molto usato sia in ambito industriale che di ricerca. CPLEX deve la sua fama sia alle sue elevate prestazioni che alla stabilità della sua implementazione dell'algoritmo del simplesso. Infatti, fino a qualche anno fa, l'ostacolo principale alla soluzione di istanze difficili era l'instabilità numerica; questa era dovuta sia alle dimensioni delle matrici che ai rapporti fra i coefficienti stessi. Capitava spesso, soprattutto in ambito di ricerca, che un Branch&Bound non potesse essere portato a termine per l'insolubilità di alcuni nodi. CPLEX offre la possibilità di essere esteso con dei moduli esterni chiamati *callbacks*: questi consistono di parti di codice scritte in C, C++ o Java che implementano funzionalità come la scelta della variabile di branching, la separazione di una variabile frazionaria etc, che vengono eseguite al momento opportuno.

4.1.1 Presentazione e caratteristiche di funzionamento

Il software commerciale ILOG - CPLEX è uno strumento dedicato alla risoluzione di problemi di programmazione lineare (intera e non) della forma:

max / min

$$c_1x_1 + c_2x_2 + ... + c_nx_n$$

subject to

$$\begin{aligned} a_{11}x_1 + a_{12}x_1 + ... + a_{1n}x_n &\sim b1 \\ a_{21}x_1 + a_{22}x_1 + ... + a_{2n}x_n &\sim b2 \\ &..... \\ a_{m1}x_1 + a_{m2}x_1 + ... + a_{mn}x_n &\sim bm \end{aligned}$$

bounds

$$\begin{aligned} l_1 \leq x_1 \leq u_1 \\ l_2 \leq x_2 \leq u_2 \\ l_n \leq x_n \leq u_n \end{aligned}$$

dove $\sim$ può essere $\leq$, $\geq$, o $=$, ed i limiti superiori u_i ed inferiori l_i possono essere positivi infiniti, negativi infiniti o un qualsiasi numero reale. I dati forniti come input a CPLEX sono:

- i coefficienti della funzione obiettivo $(c_1, ..., c_n)$,
- i coefficienti dei vincoli $(a_{11}, ..., a_{mn})$,
- i termini noti a secondo membro dei vincoli $(b_1, ..., b_m)$,
- i limiti superiori $(u_1, ..., u_n)$ delle variabili (upper bounds),
- i limiti inferiori $(l_1, ..., l_n)$ delle variabili (lower bounds).

Come output CPLEX restituisce i valori delle variabili in corrispondenza di una soluzione ottima $x^* = (x_1^*, ..., x_n^*)$ ed il valore della funzione obiettivo $f(x^*)$.

CPLEX può essere usato per risolvere:

- Problemi di Mixed Integer Programming (MIP), dove alcune o tutte le variabili sono sottoposte all'ulteriore vincolo di assumere solo valori interi.
- Problemi di Quadratic Programming (QP), dove la funzione obiettivo può anche contenere termini quadratici.
- Problemi cosiddetti Network Flow, un caso speciale di LP che CPLEX può risolvere molto più velocemente sfruttando la struttura del problema.

CPLEX può essere utilizzato in tre forme differenti:

- Il CPLEX Interactive Optimizer è un programma eseguibile che legge un problema in modo interattivo o da file (in certi formati standard), risolve il problema e mostra la soluzione in modo interattivo o in file di testo.
- La Concert Tecnology è un insieme di librerie che permettono al programmatore di includere gli ottimizzatori CPLEX in un'applicazione C++.
- La CPLEX Callable Library è una libreria C che permette al programmatore di includere gli ottimizzatori CPLEX in applicazioni scritte in C, Visual Basic, Java e Fortran.

4.1.2 Le funzioni di CPLEX utilizzate

Nel lavoro di tesi da me svolto, alcune funzioni di CPLEX sono state richiamate all'interno del programma software. Tra queste si possono ricordare:

- CPXopenCPLEX

Sintassi:

```
CPXENVptr CPXPUBLIC CPXopenCPLEX(int *status)
```

Descrizione:
La funzione `CPXopenCPLEX` inizializza un ambiente CPLEX environment. Deve essere la prima funzione CPLEX chiamata.

Parametri della funzione:
`*status` puntatore ad un intero dove è memorizzato un codice di errore.

Output:
La funzione restituisce il puntatore all'ambiente CPLEX . Nel caso in cui si verifica un errore (inclusi problemi di licenza), il valore restituito è NULL. Se la funzione termina con successo il valore di ritorno è 0.

- CPXcreateprob

Sintassi:
`CPXLPptr CPXPUBLIC CPXcreateprob(CPXCENVptr env, int *status, const char *probname)`

Descrizione:
La funzione `CPXcreateprob` crea un CPLEX problem object nell'ambiente CPLEX. Il CPLEX problem object esiste finchè non viene chiamata la funzione `CPXfreeprob`.

Parametri della funzione:
`env` puntatore al'ambiente CPLEX restituito da `CPXopenCPLEX`.
`*status` puntatore ad un intero dove è memorizzato un codice di errore.
`probname` stringa di caratteri che specifica il nome del problema creato.

Output:
La funzione restituisce un puntatore che può essere passato ad un'altra funzione CPLEX per identificare il problem object che è stato creato. Se si verifica un errore ,il valore restituito è NULL, e un codice di errore è riportato nella variable `*status`. Se la funzione termina con successo il valore di `*status` è 0.

- CPXsetlogfile

Sintassi:
`int CPXPUBLIC CPXsetlogfile(CPXENVptr env, CPXFILEptr lfile)`

Descrizione:
La funzione `CPXsetlogfile` modifica il log file in cui vengono scritti i messaggi provenienti da CPLEX.

Parametri della funzione:
`env` puntatore al'ambiente CPLEX restituito da `CPXopenCPLEX`.
`lfile` CPXFILEptr al log file. Imposta lfile come file puntatore al log file corrente. NULL è il valore di default.

Output:
La funzione restituisce 0 in caso di successo e non zero in caso di errore.

- CPXmipopt

Sintassi:
```
int CPXPUBLIC CPXmipopt(CPXCENVptr env, CPXLPptr lp)
```

Descrizione:
La funzione `CPXmipopt` può essere utilizzata, in ogni momento dopo la creazione di un problema intero misto, per cercare una soluzione a quel problema.

Parametri della funzione:
`env` puntatore all'ambiente CPLEX restituito da `CPXopenCPLEX`.
`lp` puntatore al CPLEX problem object restituito da `CPXcreateprob`.

Output:
La funzione restituisce 0 in caso di successo e non zero in caso di errore.

- CPXgetnodecnt

Sintassi:
```
int CPXPUBLIC CPXgetnodecnt(CPXCENVptr env, CPXCLPptr lp)
```

Descrizione:
La funzione `CPXgetnodecnt` è usata per accedere al numero di nodi utilizzati per risolvere il problema intero misto.

Parametri della funzione:
`env` puntatore all'ambiente CPLEX restituito da `CPXopenCPLEX`.
`lp` puntatore al CPLEX problem object restituito da `CPXcreateprob`.

Output:
La funzione restituisce 0 in caso di successo e non zero in caso di errore.

- CPXgetstat

Sintassi:
```
int CPXPUBLIC CPXgetstat(CPXCENVptr env, CPXCLPptr lp)
```

Descrizione:
La funzione `CPXgetstat` è usata per accedere allo stato della soluzione del problema dopo che quest'ultimo è stato ottimizzato.

Parametri della funzione: `env` puntatore all'ambiente CPLEX restituito da `CPXopenCPLEX`.
`lp` puntatore al CPLEX problem object restituito da `CPXcreateprob`.

Output:
Se la soluzione esiste, la funzione restituisce il numero di nodi, altrimenti 0.

- CPXgetmipobjval

Sintassi:
```
int CPXPUBLIC CPXgetmipobjval(CPXCENVptr env, CPXCLPptr lp, double
*objval)
```

Descrizione:
La funzione `CPXgetmipobjval` è usata per accedere al valore della soluzione obiettivo del problema intero misto.

Parametri della funzione:
`env` puntatore all'ambiente CPLEX restituito da `CPXopenCPLEX`.
`lp` puntatore al CPLEX problem object restituito da `CPXcreateprob`.
`objval` puntatore alla posizione in cui è stato memorizzato il valore della soluzione obiettivo.

Output:
La funzione restituisce 0 in caso di successo e non zero in caso di errore. Se non viene trovata una soluzione intera, è riportato il valore CPXERR_NO_INT_SOLN.

- CPXgetmipx

Sintassi:
```
int CPXPUBLIC CPXgetmipx(CPXCENVptr env, CPXCLPptr lp, double *x,
int begin, int end)
```

Descrizione:
La funzione `CPXgetmipx` è usata per accedere ai valori del range delle soluzioni.

Parametri della funzione:
`env` puntatore all'ambiente CPLEX restituito da `CPXopenCPLEX`.
`lp` puntatore al CPLEX problem object restituito da `CPXcreateprob`.

`x` array che contiene i valori delle variabili primali del problema.
`begin` intero che indica l'inizio del range delle variabili.
`end` intero che indica la fine del range delle variabili.

Output:
La funzione restituisce 0 in caso di successo e non zero in caso di errore. Se non viene trovata una soluzione intera, è riportato il valore CPXERR_NO_INT_SOLN.

- CPXfreeprob

Sintassi:
```
int CPXPUBLIC CPXfreeprob(CPXCENVptr env, CPXLPptr *lp)
```

Descrizione:
La funzione `CPXfreeprob` è usata per rimuovere lo specifico CPLEX problem object dall'ambiente CPLEX e liberare la memoria utilizzata internamente da CPLEX.

Parametri della funzione:
`env` puntatore all'ambiente CPLEX restituito da `CPXopenCPLEX`.
`lp` puntatore al CPLEX problem object restituito da `CPXcreateprob`.

Output:
La funzione restituisce 0 in caso di successo e non zero in caso di errore.

- CPXcloseCPLEX

Sintassi:
```
public int CPXcloseCPLEX(CPXENVptr *env)
```

Descrizione:
La funzione `CPXcloseCPLEX` libera tutte le strutture dati associate a CPLEX.

Parametri della funzione:
`env` puntatore ad una variabile contenente il puntatore all'ambiente CPLEX restituito da `CPXopenCPLEX`.

Output:
La funzione restituisce 0 in caso di successo e non zero in caso di errore.

4.2 Il programma `ms`

`ms` è un programma che genera insiemi campioni. L'obiettivo del programma è di permettere a chi lo utilizza di analizzare le proprietà polimorfiche dei campioni. Il tipico data set è ottenuto attraverso uno studio di risequenziamento, in cui lo stesso segmento omologo è sequenziato in diversi individui, presi come campioni, di una popolazione. Il classico esempio di un tale insieme di dati è lo studio di Kreitman del gene Adh (1983) in cui sono sequenziate 11 copie del gene Adh del Drosophila melanogaster. In questo caso, le 11 copie sono state isolate da 11 diversi sforzi del D. melanogaster collezionati in regioni sparse del globo. Il programma `ms` può essere utilizzato per generare molti insiemi indipendenti sotto una varietà di assunzioni sulla migrazione, sulla velocità di ricombinazione e sulla dimensione della popolazione per migliorare l'interpretazione di tali studi polimorfici. Gli insiemi sono generati utilizzando lo standard dell'approccio coalescente, nel quale la genealogia del campione viene dapprima generata in modo casuale e successivamente vengono inserite delle mutazioni. Questo progrmma è predisposto per essere utilizzato sotto Unix, o sitemi operativi Unix-like, come Linux o MacOsX.

4.2.1 La linea di comando base

`ms nsam mreps -t` θ

La linea sovrastante permette di creare un campione di dimensione costante, con vincoli di ricombinazione. In questo caso ci sono tre argomenti:

- `nsam` il numero di copie del loco in ogni campione
- `nreps` il numero di campioni indipendenti che si vuole generare
- l'opzione `-t` seguita dal paramentro θ

dove θ rappresenta il parametro di mutazione pari a $4N_0\mu$, dove N_0 è la dimensione della popolazione e μ la velocità di mutazione per l'intero loco. Ad esempio, digitando la riga

```
ms 4 2 -t 5.0
```

si otterranno in output 2 campioni, ognuno composto da 4 cromosomi, generati assumendo che $\theta = 5.0$. La figura sottostante mostra in dettaglio la struttura della'output.

```
ms 4 2 -t 5.0
27473 36154 10290


//
segsites: 4
positions: 0.0110 0.0765 0.6557 0.7571
0010
0100
0000
1001


//
segsites: 5
positions: 0.0491 0.2443 0.2923 0.5984 0.8312
00001
00000
00010
11110
```

La prima linea dell'output è la linea di comando. La seconda linea mostra il valore del numero casuale di semi necessari a generare nuovamente lo stesso insieme di campioni. Di seguito si possono osservare un insieme di linee per ogni campione. Ognuno è preceduto da una linea con solo "`//`", segue "`segsites:`" ed il numero di siti polimorfici presenti nel campione, "`positions:`", seguito dalle posizioni di ogni sito polimorfico, su una scala $(0, 1)$. Infine vengono forniti gli aplotipi relativi ad ognuno dei cromosomi campioni sotto forma di stringhe di zero e uno. Quando un campione non ha siti polimorfici, la linea con le posizioni egli aplotipi del campione sono omessi.

4.2.2 Fissare il numero di siti polimorfici

Nel caso in cui si desidera generare campioni in modo tale che ognuno abbia lo stesso numero di siti polimorfici è possibile farlo utlizzando l'opzione `-s`.

```
ms nsam mreps -s segsites
```

Anche in questo caso gli argomenti sono tre:

- `nsam` il numero di copie del loco in ogni campione
- `nreps` il numero di campioni indipendenti che si vuole generare
- l'opzione `-s` seguita dal paramentro `segsites`

dove `segsites` rappresenta il numero di siti polimorfici.
Ad esempio digitando la riga

```
ms 10 3 -s 7
```

si otterranno in ouput 3 campioni ognuno composto da 10 cromosomi e da 7 siti polimorfici. È da sottolineare che questo metodo di generare campioni non è equivalente a specificare il valore di θ e, di seguito, porre condizioni sul numero fisso di siti polimorfici. Piuttosto, questi campioni sono realizzati posizionando in modo casuale un numero fissato di mutazioni per ogni genealogia generata. Se entrambe le opzioni `-t` e `-s` vengono utilizzate ci sarà una linea di output addizionale per ogni campione. Tale linea comincia con "`prob:`" ed è seguita dalla probabilità dello specificato numero di siti polimorfici data la storia genealogica del campione e il valore di θ specificato, come mostra la figura 4.2.2.

Almeno una tra le opzioni `-t` e `-s` devono essere utlizzate. Inoltre dopo `nsam` e `npreps` ognuna delle altre opzioni con i suoi parametri può essere letta da file utlizzando `-f filename`.

4.2.3 Crossing over

Un'altra opzione disponibile è -r

`ms nsam nreps -t` θ `-r` ρ `nsites`

Gli argomenti in questo caso sono quattro:

- `nsam` il numero di copie del loco in ogni campione
- `nreps` il numero di campioni indipendenti che si vuole generare
- l'opzione `-t` seguita dal paramentro θ
- l'opzione `-r` seguita dal paramentro ρ

```
ms 5 2 -t 3.0 -s 5
3579 27011 59243


//
segsites: 5
prob: 0.145142
positions: 0.0803 0.1516 0.2276 0.8501 0.8636
01110
10000
00001
00001
10000


//
segsites: 5
prob: 0.0709507
positions: 0.0856 0.1591 0.2881 0.2901 0.6601
11000
11010
01001
01001
00100
```

dove ρ rappresenta il parametro di crossing-over (ricombinazione) pari a $4N_0r$, dove r è la probabilità di crossing-over per generazione tra le estremità del loco che è stato simulato.
Per esempio, per simulare campioni di dimensione 15 per un loco di lunghezza pari a 2,501 coppie di basi con una probabilità di cross-over tra basi adiacenti pari a 10^{-8} per generazione, e assumendo $N_0 = 10^6$ occorre digitare:

```
ms 15 1000 -t 10.04 -r 100.0 2501
```

In questo caso $\rho = 100$, dal momento che la probabilità di cross-over tra due estremità del loco è $10^{-8}(2501 - 1) = 2.5 \times 10^{-5}$ e perciò $\rho = 4 \times 10^6 \times 2.5 \times 10^{-5} = 100$. Mentre θ è ottenuto assumendo che la velocità di mutazione sia pari a 10^{-9} per sito.

Capitolo 5

Implementazione software

Entrambi i modelli presentati nel Capitolo 3 sono stati implementati in `C` sotto il sistema operativo `Linux` e, per la risoluzione del Set Covering, è stato impiegato il risolutore Cplex.

Il file `sc.c`

In questo file è stato implementato il modello di Set Covering, descritto nel Capitolo 3, nelle sue due varianti. Il modello viene passato a Cplex sotto forma di file `.lp` per la selezione dei tag SNPs. Quest'utlimi sono, infine, utilizzati per ricostruire l'insieme di input e testare la qualità della soluzione trovata. Di seguito sono descritte le strutture e le principale funzioni utilizzate per raggiungere tale obiettivo.

Le strutture

Durante l'implementazione è stato necessario costruire tre `struct` per tenere traccia della struttura dell'input originario anche a seguito dei cambiamenti (ad esempio eliminazione di righe e colonne) che si verificano durante l'esecuzione del programma. La prima di queste strutture è la `struct aplotipi` definita di seguito:

```
   struct aplotipi
{
int* indici;
char** elementi;
int* indiciEliminati;
char** elementiEliminati;
```

```
int* indiciRidondanti;
int Na;
int Na2;
int Na3;
};
```

Come si può notare, tale struttura è caratterizzata da otto campi, che rappresentano rispettivamente un puntatore agli indici degli aplotipi non ridondanti (1), un puntatore di puntatore agli aplotipi non ridondanti (2), un puntatore agli indici degli elementi eliminati (3) , un puntatore di puntatore agli aplotipi eliminati (4), un puntatore agli indici degli elementi ridondanti (5), e tre interi contenenti il numero di aplotipi rimasti, il numero di quelli eliminati ed il numero di quelli totali (6, 7 e 8).
La seconda struttura è la `struct snp` definita di seguito:

```
   struct snp
{
int* indici;
char** elementi;
int* indiciEliminati;
char** elementiEliminati;
int* indiciRidondanti;
int Ns;
int Ns2;
int Ns3;
};
```

Anche questa struttura è composta da otto campi che fanno riferimento, rispettivamente, agli indici degli SNPs non con tutti zero o uno (1), ai valori degli SNPs non con tutti zero o uno (2), agli indici degli SNPs con tutti zero o uno o, comunque, ridondanti (3), ai valori degli SNPs con tutti zero o uno o, comunque, ridondanti (4), agli indici degli elementi ridondanti (5), e tre interi contenenti il numero di SNPs rimasti, il numero di quelli eliminati, ed il numero di quelli totali (6, 7 e 8).

L'ultima è la `struct sol` definita di seguito:

```
   struct sol
{
int* idx;
int n;
```

};

Tale struttura contiene un puntatore agli indici degli SNP componenti la soluzione (*tag SNPs*) (1) e il numero di tali elementi (2).

Le funzioni

Le funzioni implementate possono essere divise in tre gruppi dal momento che il programma stesso può essere suddiviso in tre fasi:
la prima va dalla lettura dell'input alla costruzione del file .lp, la seconda va dalla chiamata a CPLEX per la risoluzione del file .lp alla costruzione della matrice dei tag SNPs e, la terza, comincia da qui per arrivare alla ricostruzione dell'insieme iniziale.
La prima fase fa riferimento alle seguenti strutture:

- `char** LeggiMatrice(char* nomefile, int *N, int *M);`
- `void AllocaVariabili();`
- `void StampaMatrice(char* fn, char** matrice, int N);`
- `void ShellSort(char** matrice, int N);`
- `int CreaStructAplotipi(char** matrice, int N);`
- `int OrdinaStructAplotipi(char** matrice, int N);`
- `int int EliminaRidondanze(char** matrice, int N);`
- `void StampaStruttura(char* fn, struct aplotipi *a);`
- `int MatriceTrasposta(char** matrice, char** trasposta, int N, int M);`
- `int CreaStructSNP(char** matrice, int N);`
- `int OrdinaStructSNP(char** matrice, int N);`
- `int ContaZeri(char *riga, int N);`
- `int EliminaSNP(char** matrice, int N, int M);`
- `void StampaStruttura2(char* fn, struct snp *s);`
- `int ScriviSC();`

La funzione `LeggiMatrice` prende il file di input e lo memorizza in un puntatore di puntatori di dimensione $(N \times M)$; in `AllocaVariabili` viene riservata l'area di memoria necessaria ad ogni variabile utlizzata nel programma; `StampaMatrice` stampa su file la matrice passata come parametro in figura 5.1 è riportato un esempio, in particolare la stampa della matrice degli aplotipi.

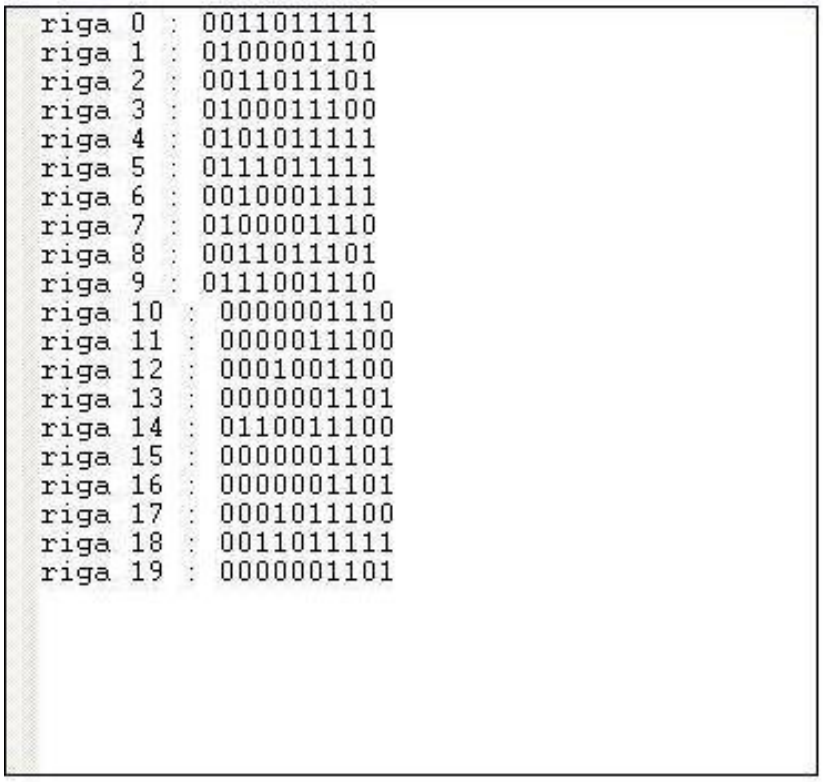


Figura 5.1: La stampa della matrice degli aplotipi

Con `CreaStructAplotipi` e `CreaStructSNP` le omonime strutture vengono inzializzate (indici e valori degli elementi).
La funzione `ShellSort` esegue un ordinamento del file di input. Lo Shell sort è simile all'Insertion sort, ma funziona spostando i valori di più posizioni per volta man mano che risistema i valori, diminuendo gradualmente la dimensione del passo sino ad arrivare ad uno. Alla fine, lo Shell sort esegue un Insertion sort, ma per allora i dati saranno già piuttosto ordinati. Il vantaggio dello Shell sort sta nel fatto che i valori si muoveranno usando passi di grosse dimensioni, cosicché un valore piccolo andrà velocemente nella sua

posizione finale con pochi confronti e scambi. L'idea dietro lo Shell sort può essere illustrata nel seguente modo:

1. sistema la sequenza dei dati in un array bidimensionale(con un numero h di colonne)
2. ordina i valori presenti all'interno di ciascuna colonna dell'array
3. ripeti dal punto 1 con un diverso numero h (minore del precedente) fino a portare h ad 1

L'effetto finale è che la sequenza dei dati viene parzialmente ordinata. La procedura viene eseguita ripetutamente, ogni volta con un array più piccolo, cioè, con un numero di colonne h più basso. Nell'ultima passata, l'array è composto da una singola colonna(h=1) trasformando di fatto questo ultimo giro in un insertion sort puro e semplice. Ad ogni passata i dati diventano sempre più ordinati, finché, durante l'ultima lo diventano del tutto. Comunque, il numero di operazioni di ordinamento necessarie in ciascuna passata è limitato, a causa dell'ordinamento parziale ottenuto nelle passate precedenti. Un esempio è riportato di seguito (figura 5.2, figura 5.3, figura 5.4).
Poniamo che

Figura 5.2: sequenza da ordinare

sia la sequenza da ordinare. Prima, viene organizzata in un array con 7 colonne (sinistra), poi le colonne vengono ordinate (destra):

Figura 5.3: prima passata

Gli elementi 8 e 9 sono ora arrivati in fondo alla sequenza, ma lì c'è anche un elemento piccolo (2) che non dovrebbe esserci. Nella prossima passata, la sequenza viene organizzata su tre colonne, che vengono nuovamente ordinate:

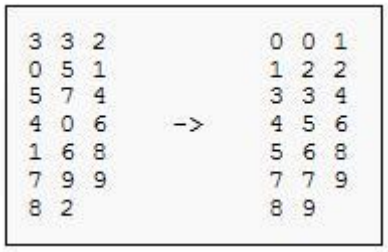


Figura 5.4: seconda passata

Ora la sequenza è quasi completamente ordinata. Una volta organizzata su una sola colonna durante l'ultima passata, sono solamente un 6, un 8 ed un 9 che devono essere spostati leggermente per arrivare a destinazione.
In realtà, i dati non vengono inseriti in un array bidimensionale, ma vengono tenuti in un array monodimensionale indirizzato opportunamente. Per esempio, i dati alle posizioni 0, 5, 10, 15, etc. formerebbero la prima colonna di un array a cinque colonne. Le colonne ottenute con questo indirizzamento vengono ordinate tramite l'Insertion sort, dal momento che questo metodo è piuttosto veloce con sequenze abbastanza ordinate.
`OrdinaStructAplotipi` e `OrdinaStructSNP` ordinano secondo lo Shell sort le omonime strutture (indici e valori degli elementi).
La funzione `EliminaRidondanze` prende in input la matrice degli aplotipi ordinata in precedenza ed elimina tutte le righe uguali e, al contempo, tali cambiamenti vengono memorizzati anche nella `struct aplotipi`. La stampa di tale struttura è realizzata da `StampaStruttura`, il cui risultato è riportato in figura 5.5.

`MatriceTrasposta` è una funzione ausiliaria che permette di calcolare la matrice trasposta di quella passata come parametro; mentre `EliminaSNP` prende in input la matrice degli SNPs ed elimina tutte le righe nulle o con tutti gli elementi pari ad uno (individuati attraverso `ContaZeri`), e quelli uguali, memorizzando tali cambiamenti anche nella `struct snp`. La stampa di tale struttura è realizzata da `StampaStruttura2`, il cui risultato è riportato in figura 5.6.

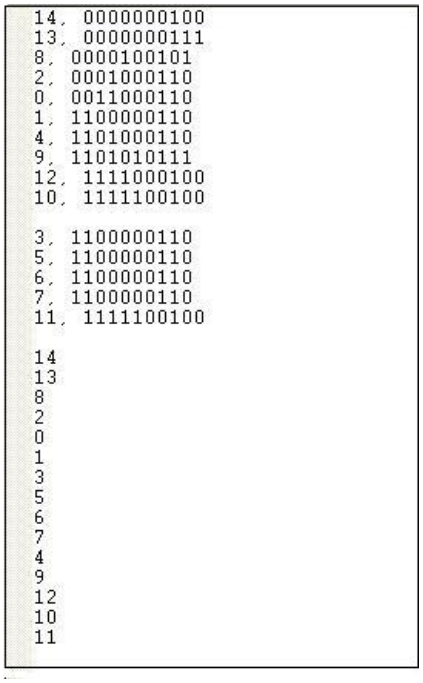


Figura 5.5: La struttura aplotipi

Infine la funzione `ScriviSC` permette di scrivere il file `.lp` da passare a CPLEX per la risoluzione del problema di Set Covering, l'output è riportato di seguito (figura 5.7).

Alla seconda fase invece appartengono le seguenti funzioni:

- `int TrovaTagSNP();`
- `void StampaSoluzione(char* fn, struct sol *soluzione);`
- `int RicostruzioneIndici(char** result);`

La funzione `TrovaTagSNP` contiene le chiamate a CPLEX necessarie per risolvere il problema di Set Covering costruito nel file `.lp`, mentre `StampaSoluzione` realizza la stampa (figura 5.8) su file delle variabili selezionate come soluzione da CPLEX, ovvero gli indici dei tag SNPs.

`RicostruzioneIndici`, invece, permette di risalire al valore degli SNPs a partire dagli indici ottenuti da CPLEX. Il risultato di tale processo viene memorizzato in una matrice.

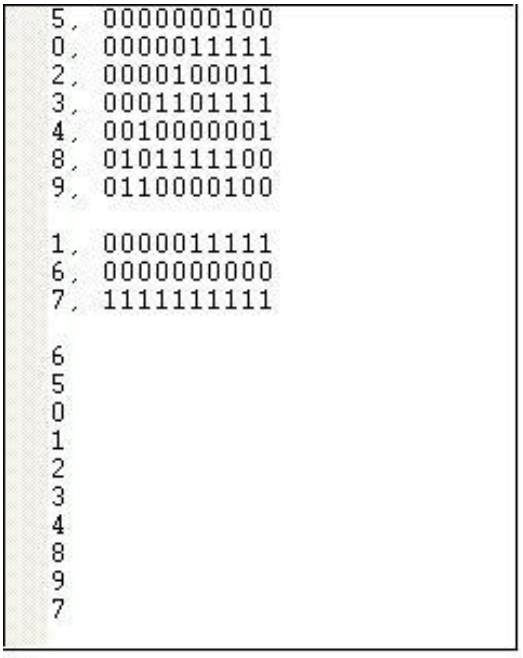


Figura 5.6: La struttura snp

E, infine alla terza fase appartengono le funzioni riportate di seguito:

- `int DistanzaHamming(char* riga1, char* riga2, int N);`
- `int CalcolaBlocco(char** matrice, char **result, char* riga, int N);`
- `int VotoMaggioranza(char* result, char** matrice, int N, int M);`
- `char** Ricostruzione(int N);`
- `char** CopiaMatrice(char** matrice, int N, int M);`
- `int InserisciRidondanze(char** matrice);`
- `int InserisciSNP(char** matrice);`
- `int ValutaRicostruzione(char** matrice1, char** matrice2, int N, int M);`

```
problem name: Set Covering (beta = 2)
maximize
obj: alfa
Subject To
c0:  + x_1 + x_2 + x_3 - alfa >= 0
c1:  + x_0 + x_1 + x_2 + x_3 - alfa >= 0
c2:  + x_0 + x_1 + x_2 + x_4 - alfa >= 0
c3:  + x_0 - alfa >= 0
c4:  + x_0 + x_3 + x_4 - alfa >= 0
c5:  + x_3 + x_4 - alfa >= 0
c6:  + x_0 + x_1 + x_2 + x_3 + x_4 <= 2
Bounds
0 <=  x_0 <= 1
0 <=  x_1 <= 1
0 <=  x_2 <= 1
0 <=  x_3 <= 1
0 <=  x_4 <= 1
0 <=  alfa <= 1
Binary
x_0 x_1 x_2 x_3 x_4
End
```

Figura 5.7: Il file `.lp`

```
Sono stati selezionati 10 Tag SNPs.
x_0
x_1
x_4
x_6
x_7
x_9
x_11
x_13
x_14
x_17
```

Figura 5.8: La soluzione

La principale funzione di questa fase è `Ricostruzione`, dal momento che in essa vengono richiamate tutte le altre funzioni, a partire da `CalcolaBlocco` che, per ogni riga del Training Set (aplotipo), provvede a selezionare il blocco di righe ad essa più vicine in prossimità delle posizioni dei tag e sulla base della distanza di Hamming (`DistanzaHamming`). Sui vari blocchi viene poi eseguito il voto a maggioranza (`VotoMaggioranza`) per determinare il valore da assegnare ad ogni allele dell'aplotipo.
Al termine di questa procedura vengono reinseriti gli SNP e gli aplotipi tolti in precedenza (`InserisciSNP` e `InserisciRidondanze`) e si effettua il calcolo degli errori commessi in fase di ricostruzione (`ValutaRicostruzione`).
Le informazioni acquisite e modificate durante l'esecuzione del programma sono riassunte nel file chiamato `logFile` (figura 5.9).

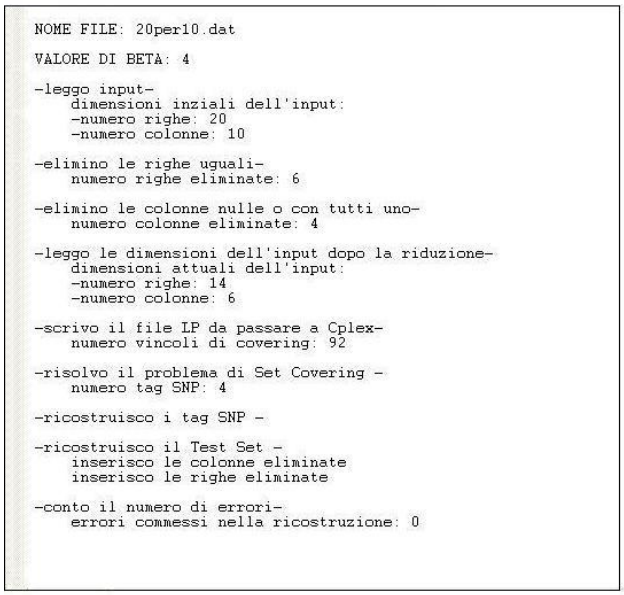


Figura 5.9: Il file di log

Il file `clustering.c`

In questo file è stato implementato il modello di Clustering descritto nel Capitolo 3. La procedura iterativa di assegnamento degli SNPs e di ricalcolo dei centroidi viene ripetuta finchè non si verifica il criterio di arresto. Solo allora i tag SNPs vengono estrapolati da ogni Cluster, utilizzati per ricostruire l'insieme di input e testare la qualità della soluzione trovata. Di seguito sono descritte le strutture e le principale funzioni utilizzate per raggiungere tale obiettivo.

Le strutture

Anche nel caso del Clustering, come per il Set Covering è stato necessario definire delle `struct`. Le prime due `struct aplotipi` e `struct snp` sono state già definite nel paragrafo precedente; oltre a queste è stata costruita la `struct cluster` definita di seguito:

```
   struct cluster
{
int* indici;
char* centroide;
char** elementi;
```

```
int Nc;
};
```

Tale struttura è caratterizzata da quattro campi, che rappresentano rispettivamente un puntatore agli indici degli SNPs componenti il Cluster (1), un puntatore al centroide del Cluster (2), un puntatore agli SNP componenti il Cluster (3) e un intero contenente il numero di SNPs presenti nel Cluster (4). L'ultima struttura definita è la `struct listacluster`:

```
   struct listaCluster
{
struct cluster* tipoElementi;
struct listaCluster *prossimo;
};
```

Questa struttura è stata utilizzata per costruire una lista di Clusters. Presenta due campi, un puntatore ad una struct di tipo `cluster`, gli elementi della lista (1), e un altro puntatore ad una struct di tipo `listaCluster`, prossimo elemento della lista (2).

Le funzioni

Anche in questo caso, come per il Set Covering, le funzioni implementate possono essere divise in tre gruppi dal momento che il programma stesso può essere suddiviso in tre fasi:
la prima va dalla lettura dell'input alla costruzione della lista iniziale dei Clusters, la seconda che ripete la procedura di assegnazione degli SNPs ai Clusters e di ricalcolo dei centroidi finchè non si verifica una condizione di arresto e, la terza, comincia da qui per arrivare alla ricostruzione dell'insieme iniziale.
La prima fase fa riferimento alle seguenti strutture:

- `char** LeggiMatrice(char* nomefile, int *N, int *M);`
- `void AllocaVariabili();`
- `void StampaMatrice(char* fn, char** matrice, int N);`
- `void ShellSort(char** matrice, int N);`

- `int CreaStructAplotipi(char** matrice, int N);`
- `int OrdinaStructAplotipi(char** matrice, int N);`
- `int EliminaRidondanze(char** matrice, int N);`
- `void StampaStruttura(char* fn, struct aplotipi *a);`
- `int MatriceTrasposta(char** matrice, char** trasposta, int N, int M);`
- `int CreaStructSNP(char** matrice, int N);`
- `int OrdinaStructSNP(char** matrice, int N);`
- `int ContaZeri(char *riga, int N);`
- `int EliminaSNP(char** matrice, int N, int M);`
- `void StampaStruttura2(char* fn, struct snp *s);`
- `int DistanzaHamming(char* riga1, char* riga2, int N);`
- `struct listaCluster* CreaLista(int N, int M);`
- `void StampaLista(char* fn, struct listaCluster *lista);`
- `int CalcolaDistanzaSNP(int* result, struct listaCluster *lista);`
- `int AssegnaSNP(struct listaCluster *lista);`
- `int RicalcolaCentroidi(int N, struct listaCluster *lista);`

Molte di queste funzioni sono già state presentate nel paragrafo precedente, pertanto verranno presentati i dettagli solo di quelle non presenti nel file `sc.c`. Tra queste vi è la funzione `CreaLista` che permette di costruire la lista iniziale dei Clusters, i cui centroidi saranno degli SNPs selezionati casualmente tra tutti quelli a disposizione. Si procede con l'assegnamento degli SNPs ai vari Clusters (`AssegnaSNP`) sulla base della loro vicinanza dai vari centroidi, misurata con la distanza di Hamming (`CalcolaDistanzaSNP`). Dopo l'assegnamento vengono ricalcolati i centroidi (`RicalcolaCentroidi`) effettuando un voto a maggioranza sugli elementi appartenenti ad ogni Cluster. L'esito di tali funzioni è visibile attraverso la stampa su file della lista, mediante `StampaLista` come visibile in figura 5.10.

Alla seconda fase invece appartengono le seguenti funzioni:

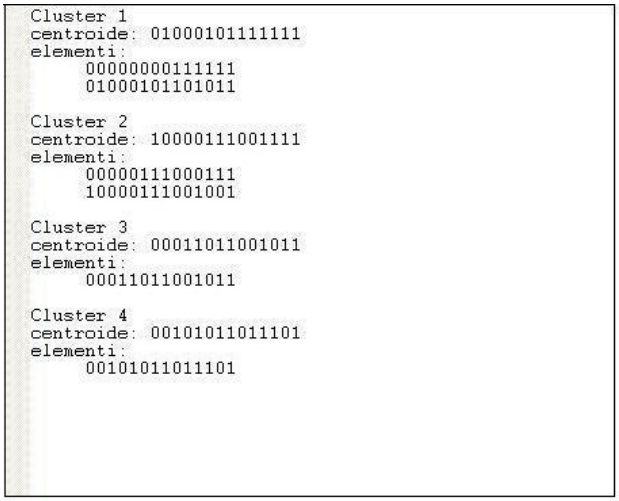


Figura 5.10: La lista dei Clusters

- `int ControlloSpostamenti(struct listaCluster *lista);`

- `struct listaCluster* Clustering(struct listaCluster *lista);`

La funzione principale di questa fase è ovviamente `Clustering`, l'altra (`ControlloSpostamenti`) viene richiamata al suo interno insieme ad altre funzioni della prima fase (`CalcolaDistanzaSNP`, `AssegnaSNP`, `RicalcolaCentroidi`). Ripetutamente la funzione modifica la lista sostituendo i centroidi calcolati al passo precedente, assegna gli SNPs ad ogni Cluster e ricalcola i centroidi; si arresta quando è verificata la condizione di uscita, ovvero quando gli SNPs smettono di spostarsi da un Cluster all'altro (`ControlloSpostamenti`).
E, infine, alla terza fase appartengono le funzioni riportate di seguito:

- `int TrovaSNP(char** result);`
- `int CalcolaBlocco(char** matrice, char **result, char* riga, int N);`
- `int VotoMaggioranza(char* result, char** matrice, int N, int M);`
- `char** Ricostruzione(int N);`
- `char** CopiaMatrice(char** matrice, int N, int M);`

- `int InserisciRidondanze(char** matrice);`
- `int InserisciSNP(char** matrice);`
- `int ValutaRicostruzione(char** matrice1, char** matrice2, int N, int M);`

Anche in questo caso, i dettagli relativi alle funzioni utilizzate in questa fase sono già stati presentati nel paragrafo precedente, per cui non verranno ripetuti, una sola nota aggiuntiva va fatta per `TrovaSNP`, che permette di selezionare i tag SNPs. Per prima cosa viene calcolata, per ogni centroide, la distanza minima dagli SNPs che appartengono al Cluster, in seguito, tra quelli a distanza minima, per ogni Cluster, viene selezionato il primo SNP trovato.

Infine, le informazioni acquisite e modificate durante l'esecuzione del programma sono riassunte nel file chiamato `logFile` (figura 5.11).

```
NOME FILE: 20per10.dat

VALORE DI BETA: 4

-leggo input-
    dimensioni inziali dell'input:
    -numero righe: 20
    -numero colonne: 10

-elimino le righe uguali-
    numero righe eliminate: 6

-elimino le colonne nulle o con tutti uno-
    numero colonne eliminate: 4

-leggo le dimensioni dell'input dopo la riduzione-
    dimensioni attuali dell'input:
    -numero righe: 14
    -numero colonne: 6

-creo la lista con i centroidi iniziali-

-risolvo il problema di clustering-
    numero iterazioni svolte: 1

-ricostruisco i tag SNP -

-ricostruisco il Test Set -
    inserisco le colonne eliminate
    inserisco le righe eliminate

-conto il numero di errori-
    errori commessi nella ricostruzione: 0
```

Figura 5.11: Il file di log

Capitolo 6

Test del programma software realizzato

In questo capitolo saranno presentati i risultati ottenuti dall'impiego del software realizzato su alcuni insiemi di dati reali. Tali dati sono stati reperiti dall' *International HapMap Project*, che si occupa di identificare e catalogare somiglianze e differenze genetiche negli esseri umani; al fine di permettere ai ricercatori di individuare i geni che influenzano salute, malattia, risposta ai farmaci, etc. Il Progetto nasce da una collaborazione tra scienziati e fondazioni appartenenti a Giappone, Gran Bretagna, Canada, Cina, Nigeria e Stati Uniti.

6.1 L'insieme di dati

Di seguito viene fornita una descrizione dettagliata per ognuno dei campioni utilizzati nella fase di test. Non sono però riportate informazioni di tipo fenotipico, per cui non c'è modo di sapere quali fossero le condizioni mediche dei campioni. I campioni selezionati appartengono alle seguenti popolazioni:

- *Yoruba in Ibadan, Nigeria* (abbreviazione: **YRI**)

Questi campioni sono stati collezionati in una particolare comunità di Ibadan, Nigeria, da individui che si distinguevano per l'avere tutti i nonni appartenenti all' etnia Yoruba.

- *Japanese in Tokyo, Japan* (abbreviazione: **JPT**)

Questi campioni sono stati collezionati nell'area metropolitana di Tokyo, da persone che provenivano (o i cui progenitori provenivano) da diverse parti del Giappone. Perciò, questo insieme di campioni può essere visto come rappresentativo della maggiora parte della popolazione giapponese.

- *Han Chinese in Beijing, China* (abbreviazione: **CHB**)

Questi campioni sono stati collezionati da individui che vivono nella comunità residenziale della Beijing Normal University che si distinguevano per l'avere almeno tre nonni appartenenti alla dinastia cinese Han.

- *CEPH (Utah Residents with Northern and Western European Ancestry)* (abbreviazione: **CEU**)

Questi campioni sono stati collezionati da persone che vivono nello Utah con progenitori provenienti dal nord e dall'ovest europeo. Il termine CEPH sta per il *Centre d'Etude du Polymorphisme Humain*, l'organizzazione che ha collezionato questi campioni nel 1980.

6.2 Risultati

Conclusioni

Bibliografia

[1] J. Jun and I. Mandoiu. Optimal Tag SNP Selection for Haplotype Reconstruction. *13th Annual International Conference on Intelligent Systems for Molecular Biology*, 2005.

[2] J. He and A. Zelikovsky. MLR-Tagging: Informative SNP Selection for Unphased Genotypes Based on Multiple Linear Regression. *Bioinformatics*, 22: 2558-2561, 2006.

[3] J. He and A. Zelikovsky. Tag SNP Selection Based on Multivariate Linear Regression. *Bioinformatics*, 2006.

[4] E. Halperin, G. Kimmel, and R. Sharmir. Tag SNP Selection in Genotype Data for Maximizing SNP Prediction Accuracy. *Bioinformatics*, 21(Suppl.1): i195-i203, 2005.

[5] V. Bafna, B. V. Halldórsson, R. Schwartz, A. G. Clark, and S. Istrail. Haplotypes and Informative SNP Selection Algorithms: Don't Block Out Information. *In Proceedings of the 7th International Conference on Computational Molecular Biology*, pages 19–26, 2003.

[6] B. V. Halldórsson, V. Bafna, R. Lippert, R. Schwatz, F. M. De La Vega, A. G. Clark, and S. Istrail. Optimal Haplotype Block-Free Selection of Tagging SNPs for Genome-Wide Association Studies. *Genome Research*, 14: 1633-1640, 2004.

[7] J. He, K. Westbrooks, and A. Zelikovsky. Linear Reduction Method for Predictive and Infomative Tag SNP Selection. *International Journal of Bioinformatics Research and Applications*, 1(3):249–260, 2005.

[8] J. He and A. Zelikovsky. Linear Reduction Methods for Tag SNP Selection. *Engineering in Medicine and Biology Society*, 2:2840– 2843, 2004.

[9] P. Sebastiani, R. Lazarus, S. T. Weiss, L. M. Kunkel, I. S. Kohane, and M. F. Ramoni. Minimal Haplotype Tagging. *Proceedings of the National Academy of Sciences*, 100:9900–9905, 2003.

[10] C. S. Carlson, A. Eberle, M. J. Rieder, L. Kruglyak, and D. A. Nickerson. Selecting a Maximally Informative Set of Single-Nucleotide Polymorphisms for Association Analyses Using Linkage Disequilibrium. *The American Society of Human Genetics*, 74: 106-120, 2004.

[11] P. Hyoun Lee. *Computational Haplotype Analysis: An overview of computational methods in genetic variation study.* PhD thesis, Queen's University, School of Computing, 2006.

[12] Ilog CPLEX. *www.ilog.com.*

[13] HapMap Project. *www.hapmap.org.*