Università degli studi di Pisa

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI

Corso di Laurea Specialistica in Informatica

Costruzione efficiente di indici per ripetizioni con caratteri wild card

Relatore: Dott. Nadia Pisanti .....

Correlatore: Prof. Roberto Grossi .....

Tesi di laurea di:
Davide Cangelosi
Matr. 224348

Anno Accademico 2006 - 2007

Indice

Capitolo 1

Introduzione

In questo capitolo si inquadra il problema algoritmico affrontato in questa tesi. A cominciare da una breve introduzione al problema dell'inferenza di motivi in informatica, affronteremo le principali problematiche dell'inferenza con wildcard e le applicazioni biologiche che la motivano. Esamineremo, inoltre, la necessità dell'approssimazione in applicazioni biologiche, introdurremo la formulazione combinatoria della soluzione proposta. Infine, riassumeremo i risultati raggiunti.

1.1 Inferenza di motivi

L'inferenza di pattern ripetuti è un'area di ricerca volta a sviluppare strumenti e metodi per trovare pattern non conosciuti *a priori*. Tali pattern sono spesso chiamati anche motivi. Quest'area di ricerca, nota anche come *pattern discovery* o *motif inference*, ha molti campi di applicazione. Nella compressione di testi, ad esempio, una delle attività dell'algoritmo di compressione mediante sostituzione di testi, è proprio la ricerca di stringhe che si ripetono nel file da comprimere. Tali ripetizioni vengono rimpiazzate da puntatori a copie contenute in una struttura dati, chiamata dizionario, costruita a par-

tire dal file. Da ciò risulta che, più sono le ripetizioni di sequenze nel file di input, maggiore è il grado di compressione raggiunta, in quanto i puntatori occupano in genere meno spazio delle stringhe rimpiazzate [ZL77]. Nei sistemi di data mining, pattern o *item* che occorrono insieme abbastanza spesso, detti *itemset frequenti*, sono alla base del problema della ricerca delle regole associative. In musica, l'identificazione di ripetizioni di segmenti o frasi nei pezzi musicali è una tecnica che rivela una struttura nelle registrazioni audio, che è molto utile per l'automatizzazione dei processi di analisi, ricerca e classificazione di musica a partire da una qualsiasi rappresentazione dell'audio [DH02]. Tra i nuovi campi di applicazione vi è quella della biologia molecolare, a cui ci interessiamo in questa tesi, e nell'ambito della quale pattern ripetuti, in proteine e sequenze di DNA, possono corrispondere ad elementi biologicamente rilevanti dal punto di vista funzionale o strutturale. La ricerca di motivi in quest'area si basa sull'assunzione che, siccome queste regioni occorrono più spesso di quanto ci si aspetti, esse sono state più conservate durante l'evoluzione e quindi potrebbero avere una particolare e rilevante funzione biologica. La ricerca dei motivi è uno dei problemi fondamentali della bioinformatica e trova applicazione in molti dei suoi campi, di cui sono esempi l'allineamento multiplo di sequenze, la caratterizzazione delle famiglie di proteine, l'identificazione di segnali promotori, e altre aree di ricerca.

L'inferenza di motivi, in qualsiasi campo essa sia applicata, ha a che fare, sia con ripetizioni identiche che con ripetizioni che non occorrono perfettamente uguali. In biologia, le ripetizioni identiche sono importanti perchè rappresentano oggetti estremamente conservati. Le ripetizioni con qualche differenza sono altrettanto ricorrenti e non meno importanti. Il motivo per cui sequenze che si somigliano possono rappresentare lo stesso oggetto biologico è che il DNA, e in generale le sequenze biologiche, sono suscettibili

ad alterazioni. Una mutazione, come viene generalmente chiamata un'alterazione del codice genetico, non modifica necessariamente la funzione dell'oggetto rappresentato dalla sequenza mutata (mutazioni silenti), quindi, sequenze non perfettamente identiche rappresentano lo stesso oggetto biologico. Algoritmicamente l'inferenza di motivi esatti è una sfida chiusa perchè oggigiorno esistono molte soluzioni efficienti, vedi [Ukk95], mentre l'inferenza di motivi approssimati rimane ancora un problema aperto. Per questo motivo, la ricerca delle ripetizioni con differenze, o inferenza di motivi approssimati, è l'argomento trattato in questa tesi.

1.1.1 Inferenza di motivi con wild card

I motivi approssimati eventualmente scoperti con la ricerca, rappresentano pattern diversi che in teoria sono lo stesso oggetto biologico. Le attuali forme di rappresentazione dei motivi approssimati catturano aspetti come la frequenza delle lettere per ogni posizione del motivo, quali sono le posizioni di differenza tra le occorrenze, ecc. Non esiste attualmente una rappresentazione di motivo che cattura pienamente, sia in termini formali che in termini biologici, tutti gli aspetti che caratterizzano le ripetizioni nelle sequenze biologiche. I principali tipi di rappresentazione dei motivi approssimati sono classificabili in base al modo con cui raggruppano le occorrenze. Questa classificazione distingue due grandi famiglie: le matrici (i cui componenti sono pesi determinati dalla frequenza delle lettere in ogni posizione del motivo, come le sequenze LOGO, Consensus e PSSW) e i pattern con wildcard (intesi come sequenze fatte sia da caratteri dell'alfabeto sia da caratteri wildcard). Un carattere wild card, denotato con il simbolo $\circ$, è un carattere speciale che non appartiene all'alfabeto della sequenza di input, ma che corrisponde a qualsiasi carattere in essa contenuta. In altre parole, ogni confronto tra

un carattere dell'alfabeto e il carattere wildcard restituisce lo stesso risultato del confronto tra due caratteri uguali dell'alfabeto. In questa tesi, scegliamo la rappresentazione di motivi come pattern con wildcard. L'insieme delle occorrenze rappresentate da un pattern con wildcard è composto da tutte le ripetizioni identiche con al più differenze nelle posizioni relative ai caratteri wildcard. Un esempio di pattern con wild card è il seguente.

Esempio 1. *Data una lunghezza* $L = 8$*, un numero* $k = 4$ *e una sequenza* $s = ABCABCABABBCBACABC \in \{A, B, C\}$*, alcuni dei motivi con wildcard per* s *sono:*

$m = AB \circ \circ AB \circ \circ$

m *occorre nelle posizioni {2,4} di* s*.*

$m1 = \circ \circ \circ \circ \circ \circ BC$

$m1$ *occorre nelle posizioni {5,10} di* s*.*

$m2 = BC \circ \circ C \circ B \circ$

$m2$ *occorre nelle posizioni {2,11} di* s*.*

Non esaminiamo tutte le possibili combinazioni di wildcard perchè li vedremo più in dettaglio nei prossimi capitoli.

Sulla base di quanto detto, definiamo il problema dell'inferenza di motivi con wildcard.

Problema 1 (Problema dell'inferenza dei Motivi con wildcard)**.** *Dati in input una sequenza di caratteri* $s \in \Sigma^n$*, una lunghezza* L*, un numero* k *e un quorum* q*, trovare tutti i motivi che occorrono in* s *con al più* k *wildcard e che occorrono almeno* q *volte.*

Nel problema appena definito, si intuisce che le posizioni dei wildcard, non sono prevedibili a priori. Non si può discriminare tra le posizioni, tranne nel caso in cui non si conosca già la struttura del motivo, quindi, la ricerca

dei motivi deve esaminare tutte le possibili combinazioni di k posizioni wildcard. Questo approccio è noto come approccio combinatorio e il numero di combinazioni da analizzare è esponenziale. Il motivo di questa complessità è che ogni parola di lunghezza L del testo ha 2^{L-k} possibili sottosequenze distinte, o combinazioni di k posizioni wildcard. In s si distinguono al più $n-L$ parole diverse. Quindi, potenzialmente, il numero di parole da esaminare è dell'ordine di $O(n * 2^n)$.

Uno degli obbiettivi di questa tesi, come vedremo, è risolvere il problema dell'inferenza dei motivi con wildcard riducendo sensibilmente lo spazio delle combinazioni da esaminare.

1.2 La Biologia e l'Informatica

La biologia e l'informatica, negli ultimi decenni, hanno stretto un importante legame di cooperazione per raccogliere la sfida che la scienza ha lanciato riguardo la comprensione dei complessi meccanismi che governano la vita. La bioinformatica è la disciplina che si occupa dello sviluppo e dell'integrazione delle applicazioni della scienza dell'informazione al servizio della ricerca scientifica in campo biotecnologico [Pro]. In particolare, l'impiego di modelli statistici validi per l'interpretazione dei dati che gli esperimenti di biochimica e biologia molecolare producono, lo sviluppo di nuovi modelli e strumenti matematici adatti all'analisi di sequenze di DNA, RNA e proteine e l'organizzazione e la gestione di basi di dati delle conoscenze acquisite sul genoma, sono solo alcuni dei compiti svolti dall'informatica. Il campo in cui si inquadra la nostra ricerca è lo sviluppo di nuovi modelli e strumenti matematici per l'analisi di sequenze di ogni tipo, ad esempio, di DNA, RNA e proteine.

Il contributo fornito dall'informatica alla biologia molecolare è, quindi, sostanziale.

Sul piano dell'informatica il DNA, gli RNA e le proteine sono delle sequenze di caratteri definite su un alfabeto, che per il DNA, l'RNA e le proteine sono rispettivamente:

- $\Sigma_{DNA} = \{A, C, T, G\}$
- $\Sigma_{RNA} = \{A, C, U, G\}$
- $\Sigma_{Proteina} = \{A, R, D, N, C, E, Q, G, H, I, L, K, M, F, P, S, T, W, Y, V\}$

1.3 Alcune nozioni di biologia molecolare

Il patrimonio genetico o genoma e tutto ciò che fa funzionare una cellula è codificato nel *DNA*. Il DNA è una lunga catena di piccole molecole, chiamati nucleotidi o basi, che possono assumere uno tra quattro possibili valori. Nella sua forma naturale, il DNA ha una struttura attorcigliata che, se estesa, da origine alla nota doppia elica. I due filamenti distinti che formano il DNA sono uno *complementare* all'altro: ciò significa che ogni coppia di molecole accoppiate nei due filamenti ha precise caratteristiche chimico-fisiche. Il funzionamento del DNA dipende dalla particolare affinità tra le basi; se prendiamo due sequenze complementari e li mettiamo vicine, esse tenderanno ad unirsi come una cerniera, compiendo quello che in biologia molecolare è chiamato *ibridizzazione*. L'espressione del DNA, cioè l'attività che permette al DNA di tradursi in istruzioni, si verifica per ibridizzazione di alcune zone del DNA con altre molecole come proteine e RNA. Notoriamente, si suddivide il DNA in zone *codificanti* e *non codificanti*: le zone codificanti sono zone specifiche del DNA adibite alla sintesi delle proteine, le zone non codificanti sono, invece, zone la cui implicazione nel processo di regolazione è stata accertata, ma che per la maggior parte, è composta dal cosiddetto *DNA spazzatura*, di cui non se ne è ancora capita la funzione.

All'interno della parte codificante del DNA, si individua il genoma composto interamente dai cosiddetti *geni*, cioè sequenze di DNA trascritte e tradotte nel processo di sintesi delle proteine, come vedremo più avanti. Ogni gene può contribuire alla sintesi di diverse proteine. L'individuazione dei geni coinvolti nella sintesi di una determinata proteina è una delle sfide che la biologia stà affrontando oggigiorno. La dinamica di una cellula, cioè l'attività che viene compiuta dentro una cellula, è un flusso continuo di interazioni tra piccole molecole e macromolecole come DNA, RNA e proteine [AC01]. Le proteine sono molecole fatte di amminoacidi unite da legami. Esse portano in sè il programma codificato dai geni è generalmente svolgono la loro funzione all'interno della cellula, interagendo con le altre molecole. Ad esempio, delle proteine chiamate *enzimi*, accelerano molte reazioni chimiche, altre si occupano dell'architettura della cellula (membrane), altre ancore regolano la trascrizione dei geni, ecc. [AC01]

L'attività principale del DNA è la sintesi delle proteine che è il processo con cui esse si creano a partire dal DNA. La sintesi proteica passa attraverso due tappe fondamentali: la trascrizione e la traduzione. La trascrizione sintetizza una molecola fatta da una catena di triplette di nucleotidi, chiamata mRNA, ottenuta con un complesso meccanismo, come si dice in termini tecnici, di co-espressione e co-regolazione dei geni coinvolti nel processo. In altre parole, il DNA funge da stampo per la trascrizione di alcune sue sequenze nella catena complementare che forma il mRNA. Una volta sintetizzato il mRNA, esso compie un piccolo viaggio verso una zona specifica della cellula, in cui sono presenti dei corpuscoli chiamati ribosomi. I ribosomi sono i responsabili della cosiddetta traduzione. Con la traduzione le triplette di nucleotidi sul mRNA vengono decifrate e tradotte nei corrispondenti amminoacidi che formano la proteina in costruzione. Nella formazione, man mano

che il mRNA scorre sul ribosoma, altre molecole chiamate transfer RNA (tRNA), che portano legate in modo specifico un amminoacido, si mettono in posizione e si attaccano tra loro con legami peptidici, formando alla fine la proteina codificata dal quel particolare mRNA.

1.4 L'inferenza di motivi con wildcard in biologia molecolare

L'inferenza di motivi con wildcard è applicabile ad esempio a pattern relativamente corti, a motivi con una struttura più o meno nota, a motivi per la cui conservazione della funzione biologica è possibile escludere a priori particolari posizioni, ecc. In questa sezione, affrontiamo in particolare, due applicazioni reali che sono di grande interesse per la bioinformatica, e che rientrano nelle categorie prima menzionate: i binding site e i microsatelliti.

1.4.1 Binding site e promotori

Durante la trascrizione di un gene, una o più molecole, chiamati *fattori di trascrizione* (TF), devono legarsi a regioni di DNA chiamate *binding site*. Un binding site si trova nella regione promotore di un gene, cioè la zona di DNA, vicina alla sequenza genica, contenente l'informazione necessaria (e sufficiente) per localizzare la sequenza da cui iniziare la trascrizione genica. Questa informazione è fondamentale, in biologia, non solo per trovare il gene corrispondente al gene regolato da quel particolare binding site, ma anche per studiare i meccanismi che controllano l'espressione e la regolazione del gene stesso. Un fattore di trascrizione può legarsi a più binding site, presenti nelle regioni promotrici dei diversi geni corrispondenti, e da questo si deduce che uno stesso TF regola la trascrizione di molti geni, che in tal caso sono detti

co-regolati. Le sequenze nucleotidiche dei binding site sono modellabili con pattern abbastanza corti (spesso non vanno oltre le decine di basi). Inoltre, i binding site possono occorrere con differenze che spesso sono localizzabili, e questo giustifica anche la loro possibile inferenza mediante i motivi con wildcard.

Come per i pattern delle proteine, i motivi nelle sequenze non codificanti del DNA possono essere usati per determinare la funzione delle sequenze nucleotidiche, ad esempio, per trovare tutti i promotori in un genoma, o per determinare siti funzionali specifici, come le regioni coinvolte nella regolazione dell'espressione genica. Le sequenze non-codificanti generalmente non sono ben conservate, quindi, la presenza di sequenze conservate nelle regioni promotrici spesso implica che queste regioni sono funzionalmente importanti. Per trovare tutti i promotori in sequenze genomiche vaste, occorre identificare caratteristiche che sono comuni a tutti i promotori, ma che non sono presenti in sequenze non promotrici. Spesso l'obbiettivo non è solo identificare i promotori, ma anche capire come sono regolati i geni sotto il controllo dei promotori. Ciò comporta l'identificazione di sequenze regolatrici specifiche nelle regioni promotrici che si legano a specifici fattori di trascrizioni in risposta ai segnali biologici. Motivi che sono agganciati da TF conosciuti possono essere modellati per trovare nuovi binding site in un genoma e per identificare geni che sono regolati nello stesso modo. Quando i TF che si legano ai motivi sono sconosciuti, questi possono essere trovati cercando elementi comuni nelle regioni promotrici che è noto che siano co-regolati.

Inoltre, la presenza di patologie degenerative spesso è associata alla co-regolazione di gruppi di geni specifici, la cui contemporanea attivazione è alla base delle condizioni patologiche. Quindi, l'identicazione dei motivi è il punto di partenza per ogni attività sperimentale volta a studiare la co-espressione

e la co-regolazione dei geni associate a specifiche patologie.

I binding site si presentano anche in forma strutturata, in [MS00], sono modellati come motivi composti da uno o più sottosequenze, chiamate *boxes*, che si trovano ad una certa distanza tra loro (distanza intesa come numero di nucleotidi che separano i boxes nella sequenza). Dal nostro punto vista ogni motivo con wild card è di fatto un motivo strutturato in cui i caratteri wildcard sono elementi separatori tra i boxes.

1.4.2 Microsatelliti

Non tutto il DNA codifica geni, il 98% del genoma umano è costituito da sequenze non codificanti che hanno altre funzioni, tra le quali attivare e disattivare i geni. Tuttavia, gran parte del DNA spazzatura non sembra avere alcuna utilità. Nel DNA spazzatura si identificano regioni note come DNA *satellite* [JDW05]. Si tratta di sequenze ripetute più volte in varie combinazioni. Negli ultimi anni, si è cominciato a scoprire che i cosiddetti *microsatelliti*, regioni contenenti sequenze ripetitive brevi (*simple sequence repeats*), costituiscono il 3% del genoma, eseguono numerose funzioni, e hanno un significato biologico molto importante.

Si è constatato che queste regioni del genoma sono caratterizzate da una forte variabilità in lunghezza, che può avere effetti sia positivi che negativi. In alcuni batteri patogeni, le sequenze ripetitive promuovono la comparsa di nuove caratteristiche che possono consentire loro di sopravvivere a modificazioni ambientali potenzialmente letali (si parla a tal proposito di (*zone ipervariabili*). Inoltre, si è ipotizzato che queste sequenze ripetute proteggano le zone codificanti vicine da mutazioni dannose.

È probabile che alcuni microsatelliti abbiano effetti notevoli anche sull'uomo dal momento che nel nostro genoma c'è ne sono almeno 100.000

[JDW05]. Finora, però, ai microsatelliti umani è stata assegnata solo una funzione negativa, e cioè essere la causa di malattie di natura neurologica. Per questo motivo, i microsatelliti sono usati per diagnosticare malattie neurologiche e per identificare soggetti a rischio. Inoltre, poichè la lunghezza dei microsatelliti può variare da un individuo ad un altro, si è iniziato ad usarli per l'identificazione dei criminali, o per determinare la paternità. I microsatelliti sono noti anche come *ripetizioni in tandem semplici* (STR), in quanto consistono sempre in sequenze ripetute in tandem (per esempio: nella sequenza ATGCCATGCCATGCC, la stringa ATGCC è ripetuta in tandem tre volte). Per sequenze ripetute in tandem si intende sequenze in cui una porzione è ripetuta più volte una di seguito all'altra. Anche per questo problema sono stati sviluppati molti algoritmi come quelli in [RKK03, WYKG04, GLM02, SM98].

1.5 L'approssimazione nell'inferenza dei motivi

Le principali motivazioni per l'approssimazione nell'inferenza dei motivi in sequenze biologiche sono: la presenza di errori nei dati rilevati da esperimenti condotti in laboratorio e le alterazioni subite dalla sequenza di DNA. Noi ci soffermiamo sulla seconda categoria. Se le mutazioni coinvolgono singole basi, allora si parla di alterazioni o mutazioni puntiformi, se riguardano parti di cromosomi sono dette cromosomiche, altrimenti sono dette genomiche [Bet]. In genere, si può pensare che le mutazioni abbiano effetti fatali o comunque dannosi e spesso questa credenza è vera: le mutazioni possono provocare malattie, ma in realtà è molto importante tenere in mente che le mutazioni del DNA sono anche alla base dell'evoluzione di tutti gli esseri viventi e quindi anche dell'uomo. Le mutazioni hanno come effetto sulla sequenza inserzioni, cancellazioni e sostituzioni di singoli nucleotidi. Noi siamo

interessati alle mutazioni che provocano soltanto sostituzioni. Le mutazioni con sostituzioni possono essere classificate in : senso, non senso, neutre e silenti. Una mutazioni di senso sostituisce un singolo amminoacido nella sequenza della proteina codificata; una mutazione non senso sostituisce un nucleotide che causa la sintesi di una proteina incompleta; la mutazione neutra forma una nuova tripletta che specifica un aminoacido diverso, ma la sostituzione non altera la funzione della proteina; la mutazione silente, invece, forma una nuova tripletta che specifica lo stesso aminoacido. Dunque, le mutazioni con sostituzione non hanno sempre effetti sull'attività cellulare, nè sulla codifica espressa dalle sequenze stesse. Ad esempio, se un binding site per una proteina subisce una mutazione silente, comunque la proteina continua la sua attività nella stessa maniera. Le ripetizioni che rappresentano lo stesso oggetto con la stessa funzione, come nel caso della co-regolazione o la co-espressione, che come abbiamo già accennato, possono subire, o aver subito, mutazioni. Di conseguenza, la ricerca di tali oggetti funzionali ha bisogno dell'approssimazione nell'inferenza di motivi. Infine, sul piano dell'informatica, gli errori sono stati modellati con una funzione di distanza tra stringhe introdotta da Levenshtein in [Lev66] nota anche come distanza di edit. Date due stringhe in ingresso, la distanza calcola il numero minimo di operazioni di cancellazioni, inserzioni e sostituzioni che trasformano una sequenza nell'altra. Esistono anche altre funzioni di distanza che vedremo nel prossimo capitolo.

1.6 Approcci per la ricerca dei motivi in sequenze biologiche

Il problema dell'inferenza dei motivi con differenze è molto difficile da risolvere a causa della vastità dello spazio delle soluzioni da esplorare. Esistono due approcci fondamentali alla risoluzione di questo problema.

L'*approccio combinatorio* di cui fanno parte gli *algoritmi esatti*, tra cui quello proposto in questa tesi, che si basano sulla ricerca esaustiva dei motivi e che quindi garantiscono di trovare la soluzione ottima. Gli *algoritmi euristici*, che riducono lo spazio di ricerca basandosi su euristiche e cercando i motivi nei sottoinsiemi di tutti i possibili pattern, con conseguente possibile perdita di una parte non sempre definibile della soluzione, e che quindi garantiscono solo il raggiungimento di ottimi locali [ABG98]. Infine, e l'*approccio statistico* al problema, di cui sono esempi *Gibbs sampler* [LW93] e *MEME* [BE94], in cui l'alta complessità del problema è contrastata semplicemente non effettuando la ricerca tra tutti i possibili motivi esistenti, ma riducendo lo spazio di ricerca facendo delle assunzioni probabilistiche.

Sebbene il tempo di esecuzione di questi algoritmi potrebbe essere esponenziale, i programmi esistenti usano tecniche di pruning per restringere lo spazio di ricerca e velocizzare la ricerca stessa, facendo uso di strutture dati quali grafi e alberi dei suffissi. Anche in questa tesi si utilizza una tecnica che usa una struttura di indicizzazione, restringendo in pratica lo spazio di ricerca.

Mostriamo adesso, due degli approcci che rientrano nella famiglia di soluzioni combinatorie al problema dell'inferenza dei motivi con differenze.

1.6.1 Approccio pattern driven

Una delle tecniche più semplici per risolvere il problema dell'inferenza dei motivi, è conosciuta come *Pattern Driven Approach* (PD) [PS00]. Gli algoritmi pattern driven enumerano ogni possibile pattern che soddisfa i vincoli, tipicamente la lunghezza, stabiliti dall'utente. Se la lunghezza dei pattern da cercare è ad esempio l, ci sono 4^l possibili pattern, utilizzando l'alfabeto del DNA. Gli approcci pattern driven confrontano ogni pattern con tutte le sottostringhe lunghe l presenti nella sequenza di input, incrementando il contatore associato ad ogni pattern nello spazio delle soluzioni che è presente, esattamente o con differenze per una o più basi, anche nella sequenza di input. Il vantaggio di questo metodo è che garantisce di trovare il pattern migliore. D'altro canto questo approccio è esponenziale nel tempo rispetto a l ed il problema diventa irrisolubile anche per moderati valori di l [Tom99]. Questo semplice metodo è infatti limitato a motivi corti che raramente superano una decina di basi.

1.6.2 Approccio sample driven

Per superare il problema dell'eccessivo tempo necessario per l'approccio PD, Waterman[MWG84] e Galas[DGW85] suggeriscono un algoritmo che riduce significativamente il tempo richiesto dall'approccio PD. La maggior parte dei 4^l pattern esaminati nell'approccio PD potrebbero anche non essere considerati visto che nè questi pattern nè loro simili compaiono nella sequenza di input. Da ciò segue che, essendo il tempo speso per esaminare tali pattern molto grande nell'approccio PD, si potrebbe velocizzare significativamente il processo eliminandoli dallo spazio di ricerca. Basandosi su questa osservazione venne sviluppato un algoritmo, detto [Esk02] *Sample Driven Approach*, che esplora solo le sottostringhe di lunghezza fissa l presenti ef-

fettivamente nella sequenza di input e quelle ad esse simili. L'approccio adottato in questa tesi è di questo tipo.

1.7 Obbiettivi della tesi e risultati raggiunti

L'obbiettivo di questa tesi è quello di concepire un nuovo metodo per indicizzare un testo, o un insieme di testi, al fine di individuarvi efficientemente pattern frequenti (o comuni ai vari testi) approssimati. Ci siamo focalizzati sull'approssimazione realizzata utilizzando il simbolo wild card per la descrizione del motivo ricorrente, e abbiamo analizzato la natura combinatoria dell'insieme di pattern di lunghezza fissa con tutti i possibili numeri e posizioni dei simboli wild card. Il nostro primo risultato è stato la definizione di una nuova relazione di equivalenza all'interno di questo insieme, che mascheri la degenerazione dei motivi; come mostriamo nel Capitolo 2, tale degenerazione è in generale motivo di grave inefficienza. Abbiamo successivamente strutturato le varie relazioni di equivalenza introducendo una relazione di ordine parziale tra motivi della stessa lunghezza aventi wild card in posizioni annidate, e l'abbiamo caratterizzata utilizzando il concetto di albero binomiale descritto nel Capitolo 3, associando un nodo di tale albero ad un insieme di posizioni dei simboli wild card. Abbiamo inoltre mostrato (Capitolo 4) come un indice di un tipo di pattern relativo ad un certo nodo possa essere efficientemente ricavato dall'indice del pattern corrispondente al nodo padre nell'albero binomiale. Tra questi indici, ovvero tra i nodi dell'albero, abbiamo effettuato una classificazione e definito un concetto di base generatrice di alberi o indici corrispondente ad un sottoinsieme dei nodi dell'albero binomiale, che porta alla riduzione dello spazio delle combinazioni del problema di indicizzare il testo per trovare tutti i pattern aventi un certo numero di wild card. Questo ci ha permesso di concepire un algoritmo efficiente per

la generazione gerarchica degli alberi per i motivi con k errori, e dunque ad un nuovo approccio combinatorio per la risoluzione efficiente del problema dell'inferenza di motivi con wildcard.

Il problema dell'inferenza di motivi di lunghezza fissa con wild card trova applicazione nella ricerca, in sequenze biologiche, dei binding site dei fattori di trascrizione dei geni. Date le dimensioni dei dati di ingresso in problemi di questo tipo, appare molto promettente un approccio basato sulla costruzione efficiente di un indice.

Capitolo 2

Relazioni di somiglianza e algoritmi

Gli alberi dei suffissi rappresentano un'efficiente struttura di indicizzazione per la ricerca di motivi esatti e per la ricerca di pattern con wildcard [Gus97]. Il loro impiego si estende anche a problemi come l'inferenza dei motivi con i modelli in [Sag98] e, per quanto ci riguarda, all'inferenza di motivi con wildcard nell'albero dei suffissi stesso. In questo capitolo si discutono le principali nozioni sugli alberi dei suffissi, gli algoritmi di costruzione più conosciuti ed efficienti e la dualità tra il concetto di distanza e la somiglianza tra sequenze. Ci soffermeremo, in particolare, sul modello di distanza di Edit e sulla distanza di Hamming. Si metterà in evidenza la somiglianza espressa come relazione tra parole o posizioni e dopo aver introdotto gli algoritmi combinatori per le principali relazioni, discuteremo la nostra idea di relazione e il suo rapporto con gli alberi.

2.1 Definizioni preliminari

Sia Σ un alfabeto finito di caratteri. Sia ε la stringa vuota, Σ^* l'insieme delle stringhe su Σ e Σ^+ quelle non nulle di $\Sigma^* \backslash \{\varepsilon\}$. Se $s = uwv$ con $u, w, v \in \Sigma^*$ allora u è il prefisso di s, w è una parola di s e v è il suffisso di s. Una parola w di s è *ramificata a sinistra* (o a destra) se esistono in s due caratteri distinti $a, b \in \Sigma$ tali che wa e wb (oppure aw e bw se ramificata a destra) sono parole di s. Un prefisso o un suffisso sono *annidati* in s se esiste un altro prefisso o suffisso in s che lo contiene.
Lungo il corso della tesi scriveremo i termini sequenza o stringa riferendoci allo stesso oggetto.

Adesso si riportano alcune definizioni di alberi su stringhe.

Definitione 1 (Σ^+-albero). *Un Σ^+-albero T in [GK97] è un albero radicato con le etichette degli archi in Σ^+ e per ogni carattere $a \in \Sigma$ ogni nodo f di T ha al più un arco etichettato $f \xrightarrow{aw} f'$.*

Denotiamo con $path(f)$ la stringa ottenuta con la concatenazione delle etichette degli archi del percorso che va dalla radice di T fino al nodo f. Se $path(f) = w$ allora w è unica e scriviamo il nodo f come $\overline{f}$. Infine, indichiamo con $T_{\overline{w}}$ il sottoalbero di T che ha $\overline{w}$ come radice.

Definitione 2 (Occorrenza di una parola in un Σ^+-albero). *In [GK97], dato un Σ^+-albero T, si dice che una stringa w occorre in T se e solo se esiste un nodo $\overline{wv}$ per qualche stringa $v \in \Sigma^*$.*

Le parole in un Σ^+-albero sono rappresentate come coppie.

Definitione 3 (Rappresentazione delle parole in un Σ^+-albero). *In [GK97], dato un Σ^+-albero T, l'insieme* word(T) *è l'insieme di tutte le stringhe in T. Per ogni $s \in$* word(T) *e $u \in \Sigma^*$, chiamiamo una coppia ($\overline{b}$,u)* coppia di

referenza *per s rispetto a T, se $\overline{b}$ è un nodo in T e $s = bu$. La coppia di referenza è detta* canonica *se $\overline{b}$ è il più lungo prefisso di s in T. La coppia di referenza canonica per una parola $s \in word(T)$ è unica.*

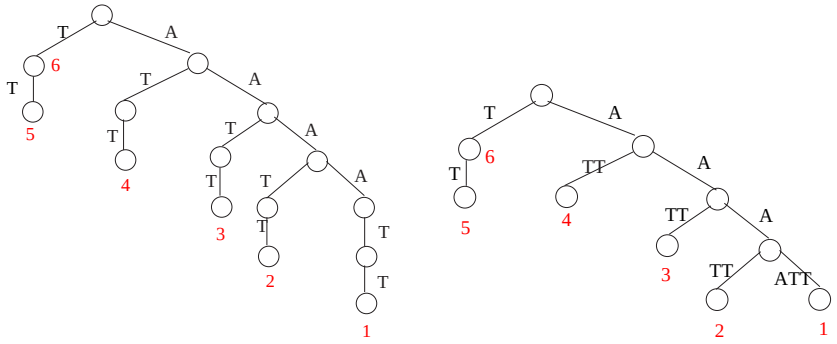
Una coppia canonica di referenza ($\overline{b}$,ε) è detta *esplicita* perchè fa riferimento al nodo $\overline{b}$ in T, mentre se espressa ($\overline{b}$,aw) è detta *implicita* perchè non fa riferimento ad un nodo ma ad una stringa che finisce all'interno di un arco $b \overset{awv}{\rightarrow} bawv$.

Definiamo un Σ^+-albero atomico o compatto nel modo seguente.

Definitione 4 (Σ^+-albero atomico e compatto)**.** *In [GK97], un Σ^+-albero T è detto* atomico *se ogni arco è marcato con un singolo carattere. T è, invece,* compatto *se le etichette degli archi possono essere anche stringhe.*

Un Σ^+-albero atomico è chiamato anche *Trie*. Vediamo un esempio di albero atomico e compatto dei suffissi.

Esempio 2. *Consideriamo la stringa $s = AAAATT$. La Figura 2.1 mostra gli alberi atw e ctw per s.*



(a) ast (b) cst

Figura 2.1: Albero atomico e albero compatto dei suffissi per la stringa AAAATT.

Un Σ^+-albero compatto dei suffissi di un testo s è anche chiamato albero dei suffissi di s.

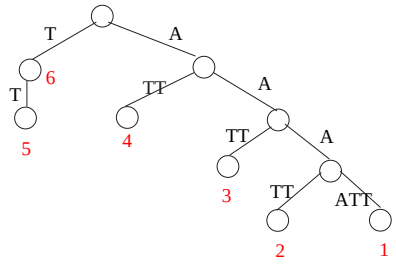
Definitione 5 (Albero dei suffissi). *Un* Albero dei suffissi *per una stringa* $s \in \Sigma^+$ *è un* Σ^+*-albero compatto* T*, tale che word(T)={*v *|* v *è un suffisso di* s*}*

Mostriamo un esempio più dettagliato di un albero dei suffissi per una stringa s.

Esempio 3. *Consideriamo la stringa* $s = AAAATT$*, la Figura 2.2 tutti i suffissi di* s *e l'albero dei suffissi di* s*.*

Posizione dei suffissi in s	word(T)
1	AAAATT
2	AAATT
3	AATT
4	ATT
5	TT
6	T

(a) Suffissi della stringa AAAATT



(b) Albero dei suffissi per la stringa AAAATT

Figura 2.2: Insieme dei suffissi e l'albero per i suffissi di s .

Un albero dei suffissi è profondo al più n, ma se si è interessati a parole lunghe L, allora ha senso troncare l'albero alla profondità L, poichè è sufficiente. Lo stesso vale per un Σ^+-albero atomico dei suffissi di s.

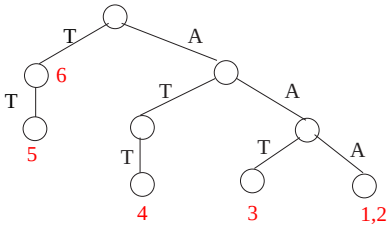
Definitione 6 (Σ^+-albero atomico dei suffissi troncato). *Dati una sequenza di caratteri* $s \in \Sigma^n$*, una lunghezza* $1 \leq L \leq n$*, un* Σ^+*-albero atomico dei suffissi* AT *è detto* troncato all'altezza L *se* AT *è tale che word(AT)={*v *|* vu *è un suffisso di* s *e* $|v| \leq L$*}.*

La definizione appena vista riguarda l'albero atomico con soli caratteri solidi; noi siamo interessati a un albero atomico che abbia anche caratteri wildcard.

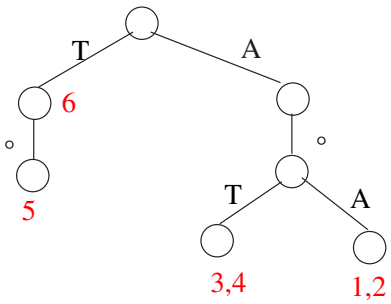
Definitione 7 (Σ^+-albero atomico dei suffissi troncato con wildcard)**.** *Dati una sequenza di caratteri $s \in \Sigma^n$, una lunghezza $1 \leq L \leq n$, un insieme di posizioni $l \subseteq \{0, \cdots, L-1\}$, un Σ^+-albero $ATW(s, L, l)$ è un* Σ^+-albero atomico dei suffissi troncato all'altezza L con wildcard in l *se $ATW(s, L, l)$ è un Σ^+-albero atomico dei suffissi troncato all'altezza L e $v_j = \circ$ per ogni $j \in \{0, \cdots, L-1\} \setminus l$.*

In sostanza ogni parola di ATW(s, L, l) ha il simbolo wildcard nelle posizioni non contenute in l.

Esempio 4. *Consideriamo ancora la stringa $s = AAAATT$, l'altezza $L = 3$ e l'insieme delle posizioni $l = \{2, 0\}$ in questo esempio mostriamo l'albero troncato all'altezza L e l'albero troncato all'altezza L con wildcard in l.*



(a) Albero troncato all'altezza 3



(b) Albero troncato all'altezza 3 con wildcard

Le due definizioni valgono anche se il Σ^+-albero è compatto; in questo caso, però, non è detto che un prefisso di lunghezza L sia esplicito nell'albero dei suffissi, quindi il troncamento è un pò più laborioso, occorre creare infatti, un nodo nell'albero troncato che esprima i prefissi impliciti di lunghezza L.

Scriviamo *ast(s)* per indicare l'*albero atomico dei suffissi* di una stringa s, *cst(s)* per riferirci all'*albero compatto dei suffissi* di s, ATW(s, L, l) e CTW(s, L, l) per indicare rispettivamente il Σ^+-albero atomico e compatto

dei suffissi troncato all'altezza L con wildcard nelle posizioni di L che non compaiono in l.

Nel corso della tesi useremo per semplicita solo gli alberi ATW.

Raffiniamo la nostra notazione indicando con *foglie(ATW(s,L,l))* l'insieme delle foglie dell'albero ATW(s,L,l).

Vediamo adesso alcuni concetti specifici che caratterizzano la costruzione degli alberi dei suffissi.

Gli *Open edges, o archi aperti*, sono stati introdotti in [Ukk95] e sono una particolare rappresentazione delle etichette degli archi che conducono da un nodo ad una foglia di un albero dei suffissi. La rappresentazione delle etichette degli archi delle foglie è una coppia, della forma (i,|s|), dove i rappresenta l'indice di un suffisso della stringa s di lunghezza n e $|s|$ il numero di caratteri della stringa s di lunghezza n. Questo significa che se il suffisso attuale si estende verso destra nella costruzione, l'etichetta della foglia cresce di conseguenza in forma implicita e la foglia rappresenta ancora una volta un suffisso esteso di s.

Le nozioni di suffisso e prefisso attivo rappresentano un altro concetto di fondamentale importanza per l'efficienza degli algoritmi di costruzione che vedremo nella prossima sezione.

Definitione 8 (Suffissi e Prefissi Attivi)**.** *In [GK97], un suffisso di s è un* suffisso attivo *, denotato con $\alpha(s)$, se esso è il più lungo suffisso annidato in s. Un prefisso di s è un* prefisso attivo *, denotato con $\alpha^{-1}(s)$, se esso è il più lungo prefisso annidato in s.*

Se la coppia ($\overline{u}$,v) rappresenta il suffisso attivo dell'albero cst(s), allora un eventuale estensione verso destra di s nella costruzione e il conseguente aggiornamento della struttura dell'albero parte proprio da ($\overline{u}$,v). In maniera

analoga, si può dire che un prefisso attivo è il punto di partenza per un eventuale estensione verso sinistra della stringa s.

Esempio 5. *Consideriamo la stringa $s = AAAATT$ e creiamo una tabella che mostra la sequenza dei suffissi di s in ordine crescente e decrescente rispettivamente, evidenziando in grassetto i prefissi e i suffissi attivi per s.*

Suffissi di lunghezza crescente	*Suffissi di lunghezza decrescente*
A	$AATTAA$
$\boldsymbol{AA}$	$ATTAA$
TAA	$TTAA$
$TTAA$	TAA
$ATTAA$	$\boldsymbol{AA}$
$AATTAA$	A

In [E.M76], McCreight usa le funzioni *Head* e *Tail* per distinguere il prefisso attivo dal resto.

Definitione 9 (Head e Tail). *Sia $s = ut$ per qualche stringa $u, t \in \Sigma^*$,* head(t) *è il più grande prefisso x di t tale che x è un suffisso annidato di ux.* Tail(t) *è definito da $t = Head(t)Tail(t)$.*

Se un suffisso t di s è annidato non ha una foglia $\bar{t}$ nell'albero dei suffissi di s. Se si vuole che ad ogni suffisso dell'albero dei suffissi T di s corrisponda una foglia non devono esserci suffissi annidati. Per questo si aggiunge alla stringa s il carattere speciale $\$$ che non occorre in t, chiamato sentinella.

Spesso, assumere la non esistenza della sentinella semplifica molto le dimostrazioni e le costruzioni, quindi da questo momento in poi escludiamo la sentinella dalla nostra trattazione salvo dei casi in cui lo diremo esplicitamente.

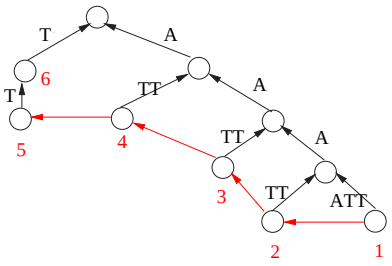
L'inserzione in tempo costante di un suffisso in un albero dei suffissi di una stringa s è una delle operazioni chiave dei principali algoritmi di costruzione

degli alberi dei suffissi. I link dei suffissi, introdotti in [Ukk95], sono archi aggiuntivi di un generico Σ-albero, definiti come segue.

Definitione 10 (Link dei suffissi)**.** *Consideriamo un Σ-albero T, sia $a \in \Sigma$, $w, v \in \Sigma^*$, $\overline{aw}$ un nodo in T e v il più lungo suffisso di w tale che $\overline{v}$ sia anch'esso un nodo in T. Un arco non etichettato $\overline{aw} \rightarrow \overline{v}$ è un* Link di suffisso*. Un link di suffisso $\overline{aw} \rightarrow \overline{w}$ è detto* atomico.

In questo esempio mostriamo i link dei suffissi per l'albero $cst(AAAATT)$.

Esempio 6. *I link dei suffissi dell'albero sono rappresentati con le frecce di colore rosso, le altre sono i link dei suffissi presenti lungo la costruzione.*



In [GK97] gli autori dimostrano che i link dei suffissi in un $ast(s)$ e in un $cst(s\$)$ per una stringa $s \in \Sigma^+$ sono tutti atomici. La verifica per un $ast(s)$ è semplice perchè tutti i nodi sono espliciti, in un $cst(s\$)$ i nodi non sono tutti espliciti ma se aw è un suffisso di $s\$$, allora poichè ogni suffisso ha un nodo esplicito nell'albero dei suffissi, essendo w un suffisso $\overline{w}$ sarà un nodo dell'albero dei suffissi di $s\$$.

2.2 Costruzione degli alberi dei suffissi

In questa sezione parliamo dei due principali algoritmi di costruzione di un albero dei suffissi di un testo $s \in \Sigma^+$ di lunghezza n, quello di Ukkonen

[Ukk95] e quello di McCreight [E.M76], cercando di evidenziare i loro aspetti comuni, le differenze e le novità introdotte. Nel corso della discussione chiameremo gli algoritmi di Ukkonen e di McCreight rispettivamente Ukk e Mcc.

2.2.1 L'algoritmo di costruzione di Ukkonen

Ukk esamina il testo s da sinistra verso destra, carattere dopo carattere, e incrementalmente costruisce un albero che ad ogni iterazione è l'albero dei suffissi per la stringa prefisso di s letta fino a quel momento. Durante la procedura di lettura dei caratteri di s, alcuni open edges crescono implicitamente, altri archi vengono divisi e altri ancora vengono aggiunti esplicitamente.

Analizziamo un caso reale, il passaggio dall'albero $cst(p)$ a $cst(pa)$, dove p è uno dei prefissi di s e a è il prossimo carattere di s a essere letto. Sia ta un suffisso di pa; i casi che possono capitare sono tre:

1. t non è un suffisso annidato di p, allora il nodo $\overline{t}$ è già una foglia per l'albero $cst(p)$. Quindi $|ta| > |\alpha(p)a|$. In tal caso il suffisso ta mantiene in $cst(pa)$ la stessa foglia per l'albero $cst(p)$, il corrispondente arco aperto è esteso implicitamente.

2. ta forma un nuovo suffisso. Cioè $|\alpha(p)a| \geq |ta| > \alpha(pa)$ e, in tal caso, si introduce la nuova foglia $\overline{ta}$ e si aggiorna l'insieme dei link dei suffissi per la nuova foglia.

3. $|\alpha(pa)| \geq |ta|$. In questa ipotesi ta occorre già in $cst(p)$ quindi ta è implicito.

Ogni suffisso in Ukk è rappresentato da una coppia di referenza canonica calcolata attraverso il link dei suffissi.

La procedura di costruzione si articola in n chiamate successive di una funzione che riceve in input quattro parametri:

- $cst(p) = T$.
- l'insieme L dei link dei suffissi per T.
- la coppia di referenza canonica $(\overline{b}, u)$ di $\alpha(p)$.
- la posizione i tale che $p = s_1 \cdots s_{i-1}$ dove s_i è il prossimo carattere di s da leggere.

è restituisce $cst(ps_i)$.

2.2.2 L'algoritmo di costruzione di McCreight

Sul piano tecnico, ciò che distingue Ukk da Mcc è il modo con cui avviene, la lettura dei caratteri e l'attraversamento dei link dei suffissi. Mcc inserisce interi suffissi di $s\$$ in un albero che inizialmente è vuoto; partendo dal suffisso più lungo di s\$ fino al più corto. Il metodo non è On-Line come per Ukk, gli alberi intermedi non sono alberi dei suffissi. Per ogni suffisso t di s\$, $T(t)$ denota il Σ-albero dei suffissi costruito per suffissi di lunghezza maggiore di t. La sequenza prodotta è, dunque, $cst(\varepsilon), T(s_1 \cdots s_n), T(s_2 \cdots s_n), \cdots, T(s_{n-1}s_n)$, $T(n) = cst(s)$. Solo il primo e l'ultimo albero sono alberi dei suffissi, mentre gli alberi intermedi sono Σ-alberi.

Il primo passo dell'algoritmo è banale $T(s) = T(s_1 \cdots s_n\$)$ è costruito a partire da $cst(\varepsilon)$ inserendo il più lungo suffisso di s\$. $T(s\$)$ è il Σ-albero con un solo arco $root \xrightarrow{s\$} \overline{s}$. Consideriamo adesso un suffisso di s at, supponiamo che $x = Head(at)$. Il passaggio da $T(at)$ a $T(t)$ si articola in tre fasi: nella prima, si calcola la coppia di referenza per $y = Head(t)$ e $Tail(t)$ a partire dalla coppia per $x = Head(at)$ e $Tail(at)$ seguendo il link dei suffissi

$\overline{x} \rightarrow \overline{y}$, nella seconda, si esamina il sottoalbero discendente da $\overline{y}$ per estendere eventualmente il prefisso attivo, nella terza, si procede con la ramificazione, se necessaria, o con l'inserzione di una foglia etichettata con $Tail(t)$.

Entrambi gli algoritmi di costruzione di un albero dei suffissi per una stringa s hanno complessità lineare in tempo e spazio. Per questo motivo sono gli algoritmi di riferimento nel calcolo degli alberi dei suffissi.

2.3 Distanza e Somiglianza

L'approssimazione, la distanza e la somiglianza tra sequenze sono strettamente legate tra loro. Il grado di somiglianza tra due stringhe si quantifica con la misura di una distanza. La distanza più opportuna da utilizzare dipende dalla specifica applicazione. Il modello di distanza più diffuso, nel confronto tra stringhe, è il modello di distanza di Edit. Essa misura la distanza tra stringhe in termini di operazioni di Edit, cioè, cancellazioni, inserzioni e sostituzioni di singoli caratteri. La distanza tra due stringhe si calcola determinando la sequenza di operazioni di edit che trasformano una stringa nell'altra che, una volta stabilito il costo di ogni operazione, minimizza la somma dei costi delle operazioni coinvolte. Tale sequenza può essere calcolata in tempo proporzionale al prodotto delle lunghezze delle due stringhe, usando la tecnica nota come programmazione dinamica. La distanza di edit, che vedremo in dettaglio nella Sezione 2.3.1, è una misura di somiglianza locale, in quanto il costo del confronto è altamente dipendente dalla posizioni che le singole lettere occupano nelle rispettive stringhe. Esistono anche altri modelli, tra i quali, il modello *maximal matches* e il modello a *q-gram*. L'idea del modello maximal matches è di contare il numero minimo di occorrenze di caratteri in una stringa tale che se questi caratteri sono cancellati, le sottostringhe rimanenti sono tutte sottostringhe dell'altra stringa. In questo

modo, stringhe con lunghe sottostringhe comuni hanno una distanza piccola. L'idea del modello q-gram è di contare il numero di occorrenze di comuni q-grams nelle due stringhe: stringhe con molti q-gram comuni hanno una distanza piccola. Un aspetto molto interessante di questi due modelli, come vedremo nella Sezione 2.3.2 anche per la distanza di Hamming, è che la distanza tra due stringhe è calcolata in tempo proporzionale alla somma delle lunghezze delle due stringhe [Gie02]. Quando si confrontano sequenze biologiche, il costo computazionale della distanza di edit è spesso troppo alto, e l'ordine dei caratteri della sequenza è importante.

In generale, una misura di somiglianza è una funzione che associa un valore numerico ad una coppia di sequenze. Un valore più grande non implica necessariamente una maggiore somiglianza tra due sequenze. La nozione di distanza è in qualche modo duale a quella di somiglianza. Essa tratta le sequenze come punti in uno spazio metrico. Una distanza è una funzione che associa, come per la somiglianza, un valore numerico a una coppia di sequenze, ma con la differenza che a un valore più grande di distanza corrisponde un valore più piccolo di somiglianza, e viceversa. La nozione di distanza soddisfa gli assiomi matematici sulle metriche. Una distanza è una metrica per delle sequenze se e solo se sono soddisfatte le proprietà:

1. $d(u, v) \geq 0$.
2. $d(u, v) = 0$ se u=v.
3. $d(u, v) = d(v, u)$.
4. $d(u, v) \leq d(u, w) + d(w, v)$ Disuguaglianza triangolare.

2.3.1 Il modello di distanza di Edit

La nozione di operazione di edit è la chiave del modello di distanza di edit.

Definitione 11. *Un* operazione di edit *è una coppia* $(\alpha, \beta) \in (\Sigma \cup \{\epsilon\}) \times (\Sigma \cup \{\epsilon\}) \backslash \{(\epsilon, \epsilon)\}$.

Un operazione di edit (α, β) è di solito scritta come $\alpha \to \beta$. Questo riflette la vista operazionale che considera le operazioni di edit come regole di riscrittura che trasformano una stringa di partenza in una stringa obiettivo, passo per passo. In particolare, ci sono quattro tipi di operazioni di edit:

- $\alpha \to \alpha$ che denota la corrispondenza di un carattere α,
- $\alpha \to \epsilon$ che denota la cancellazione di un carattere α,
- $\epsilon \to \beta$ che denota l'inserzione di un carattere β,
- $\alpha \to \beta$ che denota la sostituzione di un carattere α con un carattere β.

Nel confronto tra stringhe la misura di quanto le stringhe sono distanti è importante, ma spesso l'interesse è analizzare quali differenze distinguono le due stringhe. Il modo più diretto per stabilire le differenze tra due stringhe è l'allineamento.

Definitione 12. *Un* allineamento A *di* $u, v \in \Sigma^*$ *è una sequenza*
$(\alpha_1 \to \beta_1, \alpha_2 \to \beta_2, \cdots, \alpha_h \to \beta_h)$
di operazioni di edit tali che $u = \alpha_1 \cdots \alpha_h$ *e* $v = \beta_1 \cdots \beta_h$.

La nozione di allineamento ottimo richiede una certa funzione di score, ovvero una funzione di costo come segue:

Definitione 13. *Un funzione di costo* δ *assegna ad ogni operazione di edit* $\alpha \to \beta$, $\alpha \neq \beta$ *un costo reale positivo* $\delta(\alpha \to \beta)$. *Il costo* $\delta(\alpha \to \alpha)$ *di una operazione di edit* $\alpha \to \alpha$ *è 0. Se* $\delta(\alpha \to \beta) = \delta(\beta \to \alpha)$ *per tutte le operazioni* $\alpha \to \beta$ *e* $\beta \to \alpha$, *allora* δ *è simmetrica. Il costo* $\delta(A)$ *di un*

allineamento $A = (\alpha_1 \to \beta_1, \alpha_2 \to \beta_2, \cdots, \alpha_h \to \beta_h)$ *è la somma dei costi delle operazioni di edit in A. Più precisamente,*

$$\delta(A) = \sum_{i=1}^{h} \delta(\alpha_i \to \beta_i)$$

.

Mostriamo un esempio di allineamento tra due stringhe.

Esempio 7. *Date due stringhe* $u = AAAATT$ *e* $v = ATATAT$*. l'allineamento* $A = \{A \to A, T \to A, A \to A, A \to \varepsilon, T \to T, \varepsilon \to A, T \to T\}$*. Se prendiamo per semplicità la funzione di costo:*

$$\delta = \begin{cases} 0 & \text{se } \alpha, \beta \in \Sigma \text{ e } \alpha = \beta \\ 1 & \text{altrimenti} \end{cases}$$

Il costo dell'allineamento è:

$\delta(A) = \sum_{i=1}^{h} \delta(\alpha_i \to \beta_i) = 3$

L'allineamento, come si può facilmente verificare, trasforma effettivamente la stringa u nella stringa v.

Definitione 14. *La* distanza di edit *tra due stringhe u e v, denotata con* $dist_L(u, v)$*, è il minimo costo per un allineamento di u e v. Cioè,* $dist_L(u, v) = \min\{\delta(A) | A \text{ è un allineamento di } u \text{ e } v\}$.

Un allineamento A è *ottimo* se $\delta(A) = dist_L(u, v)$.

2.3.2 Il modello di distanza di Hamming

La distanza di Hamming è la più semplice delle nozioni di distanza tra sequenze. Introdotta in [Ham50], date due sequenze della stessa lunghezza, essa conta il numero minimo di sostituzioni per trasformare una stringa nell'altra. In molti casi la misura di Hamming è molto utile, ma in generale

non è abbastanza flessibile. Prima di tutto, le sequenze potrebbero avere differente lunghezza. Secondo, non c'è generalmente corrispondenza fissata tra le posizioni dei loro caratteri.
La natura del modello non presuppone un allineamento. L'allineamento si forma con lo slittamento di una sequenza rispetto all'altra, la distanza di Hamming conta solo le differenze tra i singoli caratteri delle sequenze. Questa caratteristica rappresenta sia la forza che il limite di questo modello. Un caso di inefficienza è quando la semplice cancellazione di un carattere in una delle due sequenze a confronto renderebbe le due sequenze identiche. Lo slittamento di una sola posizione porta ad un valore esagerato di distanza di Hamming quando in realtà le sequenze sono identiche senza lo slittamento. L'Esempio 8 mostra un caso reale in cui la distanza di Hamming non è abbastanza flessibile.

Esempio 8. *Date due sequenze di lunghezza sei* $u = AATTAA$ *e* $v = ATTAA\sqcup$*, la cancellazione del carattere A nella seconda posizione renderebbe le due sequenze uguali, ma il valore di distanza di Hamming è tre.*

Come per il modello di edit, la funzione di costo per la distanza di Hamming, denotata con $\gamma(a,b)$ per $a,b \in \Sigma$, è: $\gamma(a,b) = \begin{cases} 0 & \text{se } a = b \\ 1 & \text{se } a \neq b \end{cases}$

Consideriamo adesso, la definizione e un esempio di misura della distanza tra due sequenze.

Definitione 15. *Date due sequenze* $u, v \in \Sigma^*$*, con* $|u| = |v|$*, la* distanza di Hamming *tra* u *e* v *, denotata con* $dist_H(u,v)$*, è il minimo numero di sostituzioni per trasformare* u *in* v*. Alternativamente,* $dist_H(u,v) = \sum_{i=1}^{|u|} \gamma(u_i, v_i)$.

In altre parole, calcolare la distanza di Hamming tra due sequenze della stessa lunghezza è come contare, per ogni posizione, se il carattere in quella

posizione per entrambe le sequenze è lo stesso. Se così fosse si passa alla posizione successiva e si rifà il confronto, altrimenti, è diverso, quindi la distanza si incrementa di uno e solo dopo aver incrementato il valore della distanza si passa alla posizione successiva. Il costo del calcolo della distanza di Hamming tra due sequenze è la somma delle lunghezze delle sequenze stesse. L'Esempio 9 mostra il calcolo della distanza di Hamming tra due sequenze della stessa lunghezza.

Esempio 9. *Date due sequenze di lunghezza sei* $u = AATTAA$ *e* $v = ACGCAA$*, il valore* $dist_H(u,v) = \sum_{i=1}^{|u|} \gamma(u_i, v_i) = 3$ *nelle posizioni 2,3,4.*

Il modello di distanza di Hamming rappresenta la metrica di base su cui si basa la relazione di somiglianza che proporremo nella prossima sezione.

2.3.3 Nozioni di somiglianza

La somiglianza tra due oggetti rappresenta una relazione tra gli oggetti in questione. Ogni relazione permette di raggruppare gruppi di oggetti. La somiglianza permette di creare, quindi, dei gruppi di oggetti simili. Gli oggetti della somiglianza, che intendiamo approfondire, sono le sequenze di caratteri. In [SW], si indagano diverse relazioni di somiglianza, come vedremo in dettaglio in questa sezione; ad esempio, la relazione identità, adottata da Karp, Miller e Rosenberg in [KMR72], raggruppa classi di ripetizioni identiche sparse in una stringa, in [SVC95], gli autori usano una relazione non transitiva tra le lettere dell'alfabeto per creare gruppi di ripetizioni, chiamati cricca, in [Sag98], Sagot usa invece, una relazione con errore basata sul concetto di modello per gruppi di ripetizioni con errori. La nostra idea è, invece, la definizione di una nuova relazione con differenze. In particolare, la relazione proposta è una relazione di equivalenza che raggruppa classi di

oggetti con differenze. Dimostreremo che una nozione di somiglianza costruita su una relazione non transitiva, crea il fenomeno della degenerazione (una singola parola appartiene a più insiemi) e una maggiore difficoltà nella verifica e nella riscrittura di un generico indice di suffisso appartenente ad un insieme.

Identità

L'identità è una relazione di equivalenza tra oggetti definiti su un alfabeto Σ. Si definisce una relazione di identità E_L tra parole, in funzione di una relazione di identità tra singole lettere di Σ.

Definitione 16 (Relazione E_L)**.** *Data una stringa $s \in \Sigma^n$ e due posizioni in s tali che $0 \leq i, j \leq n - L + 1$, allora i E_L j se e solo se s_{i+l} E s_{j+l} Per ogni l tale che $0 \leq l \leq L - 1$.*

Due posizioni i e j sono in relazione di equivalenza E_L se e solo se le parole di lunghezza L in s che iniziano alle posizioni i e j sono perfettamente identiche. Ogni classe di E_L di cardinalità almeno due rappresenta un insieme di parole ripetute esattamente in s.

Se tutte le parole di una classe per la relazione E_L sono identiche, quindi, la classe contiene tutte e sole le parole uguali, l'identificazione della classe a cui una posizione o una parola appartiene è immediata leggendo i caratteri che compongono la parola stessa.

Relazioni Non-Transitive

Trattando, per esempio, sequenze biologiche si deve sempre tenere in mente che dietro ogni simbolo, nucleotidi per il DNA e aminoacidi per le proteine, vi sono oggetti complessi con proprietà fisico-chimiche specifiche, ad esempio, polarità, dimensione, livello di acidità, ecc. Alcune di queste proprietà

possono essere condivise da due o più di questi oggetti. In [SVC95], hanno pensato di definire una relazione tra i simboli dell'alfabeto, che non sia necessariamente una relazione di equivalenza in quanto riflessiva e simmetrica, ma non transitiva.

La relazione viene rappresentata con un grafo in cui i nodi sono tutti i simboli dell'alfabeto Σ e dove l'arco tra due nodi evidenzia la condivisione di sufficienti caratteristiche fisico-chimiche tra gli elementi biologici.

Come nella precedente sottosezione, è stata definita la relazione R_L tra parole lunghe L di s come funzione della relazione R tra i singoli componenti dell'alfabeto.

Definitione 17 (Relazione R_L). *Data una stringa $s \in \Sigma^n$ e due posizioni in s tali che $0 \leq i, j \leq n - L + 1$, allora i R_L j se e solo se s_{i+l} R s_{j+l} per ogni l tale che $0 \leq l \leq L - 1$.*

Per la relazione R_L gli insiemi di oggetti possono non essere più classi di equivalenza, per via della non transitività della relazione. Tali insiemi sono conosciuti come insiemi *cricca* della relazione non transitiva. Una cricca è definita nel modo seguente.

Definitione 18 (Cricca su parole di una relazione R). *Dato un alfabeto Σ e una relazione non transitiva R su Σ, un insieme C di elementi di Σ è una* cricca *della relazione R se α R β per ogni $\alpha, \beta \in C$. Se C è una cricca e $C \cup \{\gamma\}$ non è una cricca per ogni $\gamma \in \Sigma \backslash C$, allora C è detto* cricca massimale.

La nozione di cricca può essere definita sulle parole, ma anche sulle posizioni.

Definitione 19 (Cricca di posizioni di una relazione R). *Data una stringa $s \in \Sigma^n$, un insieme C_L di posizioni in s è una cricca della relazione R_L se*

$i\ R_L\ j$ per ogni $i, j \in C_L$. Se C_L è una cricca e $C_L \cup \{l\}$ non è una cricca per ogni $l \in [1 \cdots n] \backslash C_L$, allora C_L è detto cricca massimale *di R_L.*

La cricca massimale di R_L ci offre un secondo modo per stabilire una relazione di somiglianza tra parole di lunghezza L in una stringa.

A differenza di quanto visto per la relazione identità, una posizione, ovvero una parola, può appartenere a più di un insieme cricca di s. Se una parola si trova in relazione con due parole diverse che tra loro non sono in relazione allora la parola in esame farà parte di due insiemi cricca distinti. Il numero di cricca di s a cui una parola appartiene è chiamato *grado di degenerazione* di una posizione o di una parola.

Nella prossima sezione si approfondirà la degenerazione e si mostrerà come la non transitività di una relazione aggiunge complessità algoritmica alla ricerca di oggetti simili in una sequenza.

Relazioni con differenze

Le relazioni, identità e non transitiva, rientrano nella categoria delle relazioni esatte. Le relazioni esatte sono quelle che non considerano errori o differenze tra gli oggetti. In questa sezione, si introduce una categoria di relazioni che considerano la somiglianza con differenze o errori. Le relazioni con differenze si basano, generalmente, sulla distanza di Hamming, le relazioni con errori usano, invece generalmente, la distanza di Edit. In questa sezione si assumono solo relazioni che usano la distanza di Hamming.

In concordanza con le definizioni di relazione date in precedenza, si tenta di definire la relazione con differenze indotta dalla distanza di Hamming, tra due parole di lunghezza L in una stringa s, o tra due posizioni i e j in s, nel modo seguente:

Definitione 20 (Primo tentativo di definizione della relazione H_L). *Data una stringa $s \in \Sigma^n$, una lunghezza k e due posizioni in s tali che $0 \leq i, j \leq n - L + 1$, allora i H_L j se e solo se $dist_H(s_i \cdots s_{i+L-1}, s_j \cdots s_{j+L-1}) \leq k$.*

Calcolare H_L non è semplice come calcolare E_L o R_L: sebbene le definizioni 16 e 17 coinvolgano coppie di posizioni nella stringa s, è possibile riscrivere le definizioni in modo che, data una posizione i in s e una lunghezza k, risulta immediato identificare la classe o le cricche di s a cui i appartiene, leggendo semplicemente i suoi componenti $s_i \cdots s_{i+L-1}$.

Nel caso dell'identità la posizione i appartiene alla classe con etichetta $s_i \cdots s_{i+L-1}$. Mentre per la relazione non transitiva tra i simboli di Σ, chiamiamo C l'insieme delle cricche di R e denotiamo con $cricca_R(\alpha)$ le cricche di R a cui il carattere α appartiene. La posizione i appartiene alle cricche massimali di R_L con etichetta calcolabile dall'espressione regolare $cricca_R(s_i) \cdots cricca_R(s_{i+L-1})$. La differenza sostanziale tra le due relazioni stà nella verifica della massimalità dell'insieme a cui l'indice o la parola i appartiene, che, per la relazione non transitiva, deve essere verificata, mentre per la relazione identità ciò non è necessario in quanto ogni classe è massimale.

La riscrittura e la verifica di una parola o di una posizione non sono così semplici nel caso della Definizione 20 di H_L. Se volessimo costruire la nozione di somiglianza su quella di cricca di H_L dovremmo confrontare i componenti $s_i \cdots s_{i+L-1}$ con i componenti delle parole di tutte le cricche di H_L. Per questo motivo in [SW] si riscrive la definizione di H_L come relazione non transitiva con errori o con differenze basata sul concetto di modello. In tal modo, si propone un modo per avere una riscrittura e una verifica immediata di una parola. La riscrittura e la verifica è fatta confrontando le lettere che compongono la parola considerata con tutti i modelli per s. La definizione

di modello non ha, però, eliminato la degenerazione delle parole.
In [SW] viene proposta la seguente definizione di relazione basata sul concetto di modello sulle parole di una sequenza s.

Definitione 21 (Relazione H_L con la nozione di modello per le parole)**.** *Data una stringa $s \in \Sigma^n$, una lunghezza k e due posizioni in s tali che $0 \leq i, j \leq n-L+1$, allora i H_L j se e solo se $\exists m \in \Sigma^L$ tale che $dist_H(m, s_i \cdots s_{i+L-1}) \leq k$ e $dist_H(m, s_j \cdots s_{j+L-1}) \leq k$.*

In [SW], dimostrano con un controesempio che gli insiemi di parole costruiti con la relazione H_L basata sui modelli non sono nè classi e nè cricche, rappresentano degli insiemi di parole in relazione H_L con il modello che etichetta l'insieme. Tali insiemi sono denotati come *group* di s.

La definizione di relazione H_L con la nozione di modello è analoga per le posizioni in s.

Definitione 22 (Relazione H_L con la nozione di modello per le posizioni)**.** *Un insieme S_L di posizioni in s rappresenta un insieme di parole in s di lunghezza L tutte tra loro simili se e solo se esiste almeno una stringa $m \in \Sigma^L$ tale che, per ogni $i \in S_L, dist_H(m, s_i \cdots s_{i+L-1}) \leq k$ e, per ogni $j \in [1 \cdot \cdot n] \backslash S_L, dist_H(m, s_j \cdots s_{j+L-1}) > k$*

Data una posizione i di s e una lunghezza L, la definizione della relazione H_L permette di ottenere le etichette dei group di s a cui i appartiene confrontando la parola corrispondente a i in s con tutti i modelli $m \in \Sigma^L$ tali che $dist_H(m, s_i \cdots s_{i+L-1}) \leq k$.

In [SW], si definiscono *modelli* le etichette di questi gruppi. L'etichetta di una cricca massimale, di una classe o di un modello, serve a dare un nome e una facile identificazione a tutti gli oggetti simili tra loro. Chiamiamo *motivo* o pattern questa etichetta.

Definire una relazione tra parole o posizioni di una stringa s serve proprio per raggruppare insieme sotto un'unica (o poche) etichetta (etichette), tutte le parole tra loro simili secondo una relazione.

La novità apportata dalla relazione appena introdotta e dal concetto di modello, sta nella conciliazione, tra una relazione non transitiva su un alfabeto e la possibilità di errori in un'unica definizione che ammette una facile verifica e riscrittura di una qualsiasi parola o posizione in s. Un modello non è necessariamente una parola inclusa in s. Come mostrato nel prossimo esempio.

Esempio 10. *Consideriamo la stringa $s = ATATACAA$, un errore $k = 1$, il modello $m = ACAT$ è un modello per la relazione H_4, è etichetta di un group di s che ha le sottostringhe $ATAT$ e $ACAA$ in s, ma non appartiene a s.*

2.4 Algoritmi per le relazioni

In questa sezione trattiamo alcune classi di algoritmi combinatori che risolvono problemi inerenti l'individuazione delle ripetizioni. Ogni classe di algoritmi crea gli insiemi di occorrenze, basandosi su una delle relazioni di somiglianza introdotte nella sezione precedente. Partendo dalla relazione identità, trattiamo le caratteristiche principali di ogni classe, i pregi e i difetti delle soluzioni proposte.

Infine, ci soffermeremo sulla nostra relazione di equivalenza e introdurremo i concetti che saranno parte integrante della nostra risoluzione del problema dell'inferenza dei motivi con differenze.

2.4.1 KMR per la relazione identità

Il primo algoritmo per la ricerca esatta delle ripetizioni in una stringa, chiamato KMR, è stato elaborato da Karp, Miller e Rosemberg nel 1972 in [KMR72]. Data una sequenza s, KMR risolve i seguenti problemi:

Problema 2 (Problema della ricerca di ripetizioni esatte di lunghezza fissa)**.** *Identificare le posizioni di tutte le parole di una fissata lunghezza L che appaiono ripetute in s.*

Problema 3 (Problema della ricerca di ripetizioni esatte di lunghezza massima)**.** *Trovare la lunghezza L_{max} della più lunga parola ripetuta in s, e risolvere il problema 2 per $L = L_{max}$.*

In KMR si usa la definizione E_L data nella sezione 2.3.3. Sulla base di questa relazione il problema 2 si riformula come il problema di trovare le classi di equivalenza della relazione E_L che si possono costruire a partire da s, il problema 3, invece, propone di trovare il massimo valore di L tale che E_L non definisce più classi di equivalenze in s.

Per risolvere i due problemi, l'algoritmo KMR costruisce iterativamente delle classi o partizioni di E_l, con $l < L$ e l potenza di due. Quindi ogni iterazione costruisce un insieme di classi di equivalenza che si riferiscono a parole di s lunghe il doppio rispetto all'iterazione precedente. Formalmente l'idea di base di KMR è espressa dal Lemma 1 nel modo seguente:

Lemma 1. *Dati due interi a, b con $1 \leq b \leq a$, due posizioni i, j in una stringa s di lunghezza n, tale che $i, j \leq n - (a + b) + 1$, allora $i\ E_{a+b}\ j$ se e solo se $i\ E_a\ j$ e $(i + b)\ E_a\ (j + b)$.*

In particolare l'algoritmo KMR applica il lemma 1 con $b = a$ per il maggior numero possibile di volte. Quindi costruisce ripetizioni di lunghezza $2a$

usando gli indici delle parole nelle classi per le ripetizioni di lunghezza a. La composizione di due indici di lunghezza a fa in modo che un indice sia il prefisso ed l'altro il suffisso di una parola di lunghezza $2a$. Se L non è una potenza di 2, si usa il Lemma 1 con $b < a$ così che si possano costruire effettivamente le classi per E_L anche se L non è potenza di due. Per quanto riguarda la ricerca di L_{max}, l'algoritmo applica il Lemma 1 con $b < a$, avanzando di una posizione ogni passo a partire dal valore di L potenza di 2, che chiamiamo 2^p, per cui $E_{2^{p-1}}$ è identità e E_{2^p} non è più identità.

Un altro modo di vedere la costruzione delle partizioni o classi di E_a è quello di un'operazione di intersezione iterativa di insiemi.

Per risolvere il Problema 2 l'algoritmo KMR impiega tempo $O(n \log L)$ e spazio O(n) perchè, costruisce ad ogni passo pattern lunghi il doppio del passo precedente, quindi, compie $\log L$ passi e ogni passo costa n perchè n possono essere i caratteri da leggere nella sequenza s di lunghezza n. Per il Problema 3 arriva fino a $O(n \log n)$ in quanto cercare il valore k_{max} potrebbe significare provare qualsiasi lunghezza nella sequenza s, anche, $k_{max} = n$.

Mostriamo un esempio di esecuzione dell'algoritmo KMR su una sequenza s.

Esempio 11. *Dato un alfabeto $\Sigma = \{A, B, C, D\}$, una lunghezza $L = 4$ e una sequenza $s = ABCDABCD$. Le posizioni di s vanno da uno a otto. Cerchiamo tutte le parole di s di lunghezza L che si ripetono almeno due volte.*

Il primo passo è per $L = 1$ e le classi generate sono:

$A = \{1, 5\}$ $B = \{2, 6\}$ $C = \{3, 7\}$ $D = \{4, 8\}$

Il secondo passo è per $L = 2$ e le classi generate sono:

$AB = \{1, 5\}$ $BC = \{2, 6\}$ $CD = \{3, 7\}$

Il terzo passo è per $L = 4$ e l'unica classe generata è:

$ABCD = \{1, 5\}$

L'algoritmo ritorna proprio $ABCD = \{1, 5\}$. *Una classe con almeno due occorrenze di lunghezza* $L = 4$.

Infine, essendo ogni classe la rappresentazione di una parola che si ripete in s, concludiamo dicendo che il numero di classi non supera mai n.

2.4.2 KMRC per la relazione non transitiva senza errori

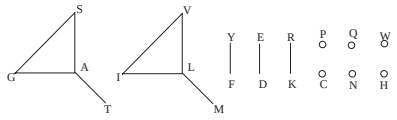
L'algoritmo, chiamato KMRC, è stato elaborato nel 1995 in [SVC95]. L'idea di base è stata di adattare KMR alle relazioni non transitive R. I problemi risolti sono rimasti gli stessi di KMR. L'algoritmo applicato è sostanzialmente lo stesso dell'algoritmo KMR con la sola differenza che il Lemma 1 è stato riformulato per la relazione R. Come si vede nell'enunciato seguente:

Lemma 2. *Dati due interi a,b con* $1 \leq b \leq a$, *e i,j due posizioni in una stringa s di lunghezza n, tale che* $i, j \leq n - (a + b) + 1$, *allora* $i\ R_{a+b}\ j \Leftrightarrow i\ R_a\ j$ *e* $(i + b)\ R_a\ (j + b)$.

L'algoritmo è stato chiamato KMRC perchè, si basa su KMR e per via del fatto che, gli insiemi di posizioni o di parole di s in relazione R_L, non formano classi bensì cricche (la C sta per cricca). Una posizione non è inclusa esclusivamente in una classe, che è massimale per definizione, ma può appartenere a due o più cricche distinte di R_L. Visto che alcuni insiemi possono non essere massimali, in quanto costruiti ad un passo intermedio e potenzialmente inclusi in altri, è stato necessario aggiungere una procedura che ad ogni passo elimina gli insiemi inclusi in altri, in modo da avere solo cricche massimali. Come abbiamo accennato all'inizio di questa sezione, la non transitività della relazione R_L genera il fenomeno della degenerazione di una parola o di una posizione. La degenerazione avviene quando una parola i

è in relazione R_L con più di una parola di s e tali parole non necessariamente sono in relazione R_L tra loro, quindi, si trovano in cricche distinte di R_L. In questo caso, la parola i sarà presente in ognuna di queste cricche distinte. Nel caso delle classi di equivalenza, come abbiamo visto per KMR, il loro numero è lineare nella lunghezza della sequenza s, mentre, con le cricche di una relazione non transitiva R_L e la degenerazione, il loro numero non è più lineare come per KMR ma esponenziale. Prima di passare allo studio della complessità dell'algoritmo KMRC e alla dimostrazione di quanto appena affermato, mostriamo un esempio di relazione non transitiva R tra i simboli dell'alfabeto Σ e le cricche massimali su R.

Esempio 12. *Sia* $\Sigma = \{A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y\}$ *un alfabeto per gli aminoacidi e* R *la relazione non transitiva tra questi simboli espressi con la seguente figura: Le cricche massimali di* R *sono gli in-*



siemi: $\{A, S, G\}$, $\{A, T\}$, $\{I, L, V\}$, $\{L, M\}$, $\{F, Y\}$, $\{D, E\}$, $\{K, R\}$, $\{C\}$, $\{P\}$, $\{N\}$, $\{Q\}$, $\{H\}$, $\{W\}$. *Come si può notare dagli insiemi* $\{A, S, G\}$, $\{A, T\}$ *e* $\{I, L, V\}$, $\{L, M\}$, *il simbolo* A *è presente in più di una cricca, così come il simbolo* L. *In questo caso si parla di alfabeto degenerato.*

Per calcolare la complessità dell'algoritmo KMRC, abbiamo bisogno di definire il parametro g che misura il grado di non transitività della relazione R.

Definitione 23. *Data la relazione non transitiva* R, *chiamiamo* g *il massimo numero di cricche di* R *a cui un simbolo può appartenere, cioè:* $g = \max\{g_a \mid a \in \Sigma, g_a =$ *Numero di cricche a cui a appartiene*$\}$. *Chiamiamo* $\overline{g}$

il valore medio di g_a per $a \in \Sigma$, cioe: $\overline{g} = \frac{\Sigma_a g_a}{n_c}$, dove n_c è il numero di cricche di R.

La complessità è dell'ordine di O(nlog Lg^L) se ogni posizione i in s può appartenere al massimo g^L cricche di R_L. Se poi aggiungiamo la complessità introdotta per fare il test di inclusione si arriva a O(n log Ln^2g^{2L}). Per lo spazio invece è O(ng^L).

In particolare, il fattore esponenziale g^L deriva dal numero di parole di lunghezza L con cui una parola i può essere in relazione R_L. Nel caso di una relazione identità la parola i è in relazione E_L solo con parole lunghe L identiche a egli stesso, con la relazione R_L ogni simbolo della parola i può trovarsi in relazione R con un numero g di simboli dell'alfabeto, le parole in relazione R_L con i sono le g^L combinazioni di g simboli per ogni posizione di L, .

2.4.3 Albero dei suffissi per le relazioni identità e non transitive

Come abbiamo visto nella prima parte di questo capitolo, un albero dei suffissi è usato nell'indicizzazione di un testo, ogni suffisso possiede una foglia etichettata con l'indice del suffisso stesso nel testo e i suffissi che condividono un prefisso, hanno in comune anche la parte del percorso che dalla radice arriva al nodo di ramificazione per quel prefisso. Trovare le ripetizioni identiche di lunghezza L in un testo s con l'albero dei suffissi è un operazione semplice, in quanto, è necessario solo visitare tutti i percorsi relativi ai prefissi di lunghezza L e calcolare i suffissi che hanno in comune quei percorsi. La complessità di questo algoritmo naive è lineare.

L'algoritmo appena visto calcola gli stessi indici delle ripetizioni identiche calcolati dal'algoritmo KMR per la relazione identità.

L'albero dei suffissi non è solo usato per la ricerca di ripetizioni identiche. L'algoritmo proposto in [SW], usa l'albero dei suffissi per la generazione dei modelli di una relazione non transitiva con errori.

Il problema che i modelli cercano di rivolvere è il seguente:

Problema 4 (Problema della ricerca delle ripetizioni con la distanza di Hamming e di Edit)**.** *Data una stringa $s \in \Sigma^+$, un intero $k \geq 0$ e un quorum q, trovare tutti i modelli m tale che m è valido, cioè, è presente almeno q volte in s, ogni volta con al più k errori.*

Per generare i modelli di s, l'algoritmo visita l'albero dei suffissi ST di s. Ad ogni passo, si estende un modello di una unità e l'insieme delle occorrenze del nuovo modello m viene creato dall'insieme delle occorrenze del modello m' di lunghezza $\mid m \mid -1$ che è suo prefisso, aggiornando tale insieme in base a ciò che segue nell'albero dei suffissi. Quindi per ogni modello m considerato bisogna tenere gli insiemi delle occorrenze di tutti i modelli prefisso del modello m. Le occorrenze di un modello sono nodi dell'albero (nodi occorrenza) rappresentati con una coppia (i,d) dove i è una posizione in s e d è la misura della distanza dal modello in esame.

L'algoritmo applica ricorsivamente il Lemma 3, dove $\overline{x}$ denota un nodo dell'albero, *padre($\overline{x}$)* il suo più stretto antenato e d il numero di errori tra il prefisso x e il modello m.

Lemma 3. *($\overline{x}$,d) è un nodo-occorrenza di $m' = m\alpha$ con $m \in \Sigma^L$ e $\alpha \in \Sigma$ se e solo se $d \leq k$ e una delle condizioni seguenti è verificata:*

(uguaglianza) *(padre($\overline{x}$),d) è un nodo-occorrenza di m e l'etichetta dell'arco che va da padre($\overline{x}$) a $\overline{x}$ è proprio α;*

(sostituzione) *(padre($\overline{x}$),d-1) è un nodo-occorrenza di m e l'etichetta dell'arco che va da padre($\overline{x}$) a $\overline{x}$ è $\beta \neq \alpha$;*

(cancellazione) *($\overline{x}$,d-1) è un nodo-occorrenza di m;*

(inserzione) *(padre($\overline{x}$),d-1) è un nodo-occorrenza di mα;*

Per riuscire a verificare la validità di un modello è necessario aggiungere per ogni nodo $\overline{x}$ di ST, il numero di foglie del sottoalbero T_x. Se il numero è superiore al quorum q allora il modello è valido, altrimenti non lo è.

Prima di dare la complessità al caso pessimo è necessario introdurre la nozione di parole vicine o *word neighbourhood*.

Definitione 24. *Data una parola di lunghezza L definita su Σ^L, $\mathcal{V}(k, L)$, è il numero di parole a distanza (Hamming o Edit) al più k da essa.*

La vicinanza misura il numero di modelli $m \in \Sigma^L$ di cui una posizione i in s può essere un'occorrenza.

Il numero di modelli validi nel caso pessimo è allora $O(n\mathcal{V}(k, L))$. $\mathcal{V}(k, L)$ è limitato dal valore $L^k|\Sigma|^k$ [Sag98].

Per completezza, in [Sto00], il concetto di modello è stato esteso all'inferenza di motivi strutturati che rappresentano motivi per cui è possibile definire formalmente una struttura fatta di motivi singoli distanziati opportunamente l'uno dall'altro. La loro applicazione è stata indirizzata verso lo studio dei binding site.

2.5 La relazione di equivalenza con differenze H_L^l

La relazione R_L non transitiva con errori basata sui modelli, se si considera la definizione che usa la distanza di Hamming, è anche una relazione con differenze. La non transitività che la caratterizza è, però, cruciale per questa relazione. Il problema della ricerca delle ripetizioni con errore o con differenza che si basa sulle nozioni di distanza di Hamming o di Edit, conduce

inevitabilmente, per quanto abbiamo visto, ad una relazione non transitiva. Uno dei nostri obbiettivi è di proporre una definizione di relazione con differenze che sia una relazione di equivalenza. L'idea è venuta partendo proprio dalla ricerca di una relazione transitiva che riuscisse a fondere insieme, nella stessa definizione, la distanza di Hamming e la transitività. Comprendere la sottigliezza su cui si basa la nostra idea significa comprendere perchè, così come sono definite, la transitività e la distanza di Hamming coesistono difficilmente. La distanza di Hamming, sappiamo che, conta le differenze tra le sequenze a confronto. Questa definizione esclude in partenza la transitività tra le sequenze perchè, se una sequenza $s1$ ha k differenze con un altra sequenza $s2$ e se quest'ultima ha k differenze con una terza sequenza $s3$, allora, non è detto che il numero di differenze tra $s1$ e $s3$ sia lo stesso k. I casi che possono capitare sono due; se l'insieme delle posizioni di differenza tra $s1$ e $s2$, e tra $s2$ e $s3$ è diverso, allora l'insieme delle posizioni di differenza tra $s1$ e $s3$ è diverso e il numero di differenze è almeno $k+1$ e al più $2*k$, se invece, $s1$, $s2$ e $s3$ hanno differenze nello stesso insieme di posizioni allora la transitività è sempre verificata. La definizione proposta di relazione con differenza è una relazione di equivalenza che costruisce la nozione di somiglianza tra stringhe sulle posizioni di differenza tra le stringhe stesse. In sostanza due stringhe sono simili se, fissando un insieme di posizioni l, sono identiche nelle posizioni non scritte in l e differiscono al più nelle posizioni di l.

Formalmente, dato una lunghezza L fissiamo un insieme di posizioni $l \subseteq \{0, \cdots, L-1\}$, l'idea è di mettere insieme in una classe di equivalenza, tutte le parole identiche di s che differiscono al più nelle posizioni non scritte in l.

Partiamo dalla definizione di H_L con le posizioni fissate.

Definitione 25 (Relazione H_L con le posizioni fissate)**.** *Date due stringhe $u, v \in \Sigma^L$ e un numero k,*

u H_L *v se e solo se* $\exists l \subseteq \{0, \cdots, L-1\}$ *tale che*

- $|l| \leq k$,
- u_i *E* v_i *per ogni* $i \in \{0, \cdots, L-1\} \backslash l$,
- $dist_H(u_i, v_i) \leq 1$ *per ogni* $i \in l$.

La relazione H_L così definita, non è transitiva. Vediamo un controesempio che dimostri questa osservazione.

Esempio 13. *Dati un errore k=1 e tre stringhe u=abc, v=dbc e w=dgc. Le posizioni di una stringa sono in ordine decrescente (2, 1, 0). La proprietà transitiva di una relazione* H_3 *si scrive u* H_3 *v, v* H_3 *w* $\nRightarrow$ *u* H_3 *w.*

abc H_3 *dbc* *per* $l = \{2\}$

dbc H_3 *dgc* *per* $l = \{1\}$

abc H_3 *dgc* *per* $l = \{2, 1\}$

Visto che $|l| = 2 > k = 1$ *la relazione* H_3 *non è transitiva.*

Generalizzando, se la relazione H_L non è transitiva, allora non è una relazione di equivalenza. Cerchiamo una definizione di H_L per cui sia una relazione di equivalenza. La definizione che segue è la definizione corretta, che fissa veramente l'insieme delle posizioni di differenza.

Definitione 26 (Relazione H_L^l)**.** *Date due stringhe* $u, v \in \Sigma^L$*, un numero k e un insieme* $l \subseteq \{0, \cdots, L-1\}$ *e* $|l| \leq L - k$.
u H_L^l *v se e solo se*

- $u_i = v_i$ *per ogni* $i \in l$

Dimostriamo adesso che, la relazione H_L^l è una relazione di equivalenza.

Lemma 4. *Fissato L e un l,*H_L^l *è una relazione di equivalenza.*

Dimostrazione. Una relazione di equivalenza è una relazione tale che:

1. $u\ H_L^l\ u$(riflessiva)
2. $u\ H_L^l\ v \Rightarrow v\ H_L^l\ u$(simmetrica)
3. $u\ H_L^l\ v$ e $v\ H_L^l\ w \Rightarrow u\ H_L^l\ w$(transitiva)

La riflessività è immediata con l={}. La simmetria è verificata immediatamente per qualsiasi insieme l e la transitività è verificata per qualsiasi insieme l. Visto che l è fissato, se u e v sono identici tranne al più nelle posizioni di l e v e w sono identici tranne al più nelle posizioni di l, allora u e w sono identici tranne al più nelle posizioni di l. □

La definizione di H_L^l è immediata se consideriamo anche coppie di posizioni piuttosto che parole in s.

Definitione 27 (Relazione H_L^l per le posizioni). *Sia $s \in \Sigma^n$, un numero k, date due posizioni $i, j \in s$ e un insieme $l \subseteq \{0, \cdots, L-1\}$ tale che $|l| \leq k$. $i\ H_L^l\ j$ se e solo se*

- $s_{i+indice}\ E\ s_{j+indice}$ *per ogni indice* $\in \{0, \cdots, L-1\} \backslash l$,
- $dist_H(s_{i2}, s_{j2}) \leq 1$ *per ogni* $i2, j2 \in l$.

Data una posizione i in s, una lunghezza L e un insieme l, per identificare la classe di appartenenza di i leggendo semplicemente i suoi componenti $s_i \cdots s_{i+L-1}$, basta trovare la classe con etichetta $s_i \cdots s_{i+L-1}$ con wildcard nelle posizioni di non espresse in l.

La relazione H_L^l è una relazione di equivalenza. Una relazione H_L^l sulle parole di una stringa s crea una famiglia di classi di occorrenze di s. Un indice i, all'interno di una famiglia, appartiene al più ad una e una sola classe. Se osserviamo ogni famiglia di classi di occorrenza, un indice i di

s, compare in ogni famiglia creata a partire da una relazione H_L^l. Questa proprietà deriva dal fatto che ogni relazione esamina le stesse stringhe delle altre relazioni selezionadone la sottosequenza specificata nell'insieme l, per questo motivo l'indice i di una parola compare in ogni H_L^l.

Questa forma di degenerazione è diversa da quella vista finora, in quanto ogni relazione H_L^l con il trucco delle posizioni fissate non ha localmente (in ogni famiglia) degenerazione degli indici delle parole, ma solo a livello di tutte le relazioni. Vedremo nel Capitolo 4 come questa diversità o mascheramento ci permette di costruire gli indici delle parole in maniera efficiente.

Fissare l'insieme l con $|l| \leq k$ rende necessario comunque calcolare la relazione H_L^l per ogni possibile combinazione di posizioni l. La complessità esponenziale è intrinseca al problema della ricerca di ripetizioni con errore o differenze, di conseguenza avere le relazioni di equivalenza ci riduce lo spazio delle combinazioni da verificare, come vedremo nel Capitolo 4, ma non ci risolve il problema. Infatti, le posizioni delle differenze tra le occorrenze, essendo imprevedibili, sono calcolabili solo provando tutte le possibili combinazioni di posizione di errore.

Adesso si puntualizza il legame tra la relazione H_L^l e gli alberi $ATW(s, L, l)$. Gli indici dei suffissi associati alle foglie di un $ATW(s, L, l)$ o di un $ctw(s, L, l)$ sono in relazione H_L^l tra loro.

Lemma 5. *Dati una lunghezza* $1 \leq L \leq n$, *un insieme di posizioni* $l \subseteq \{0, \cdots, L-1\}$ *e un* $ATW(s, L, l)$, *per ogni foglia* $\overline{f} \in foglie(ATW(s, L, l))$ *tale che* $|f|=L$ *vale i* H_L^l j, *per ogni coppia* $i, j \in I_f$, *con* $I_f = \{i \in \{1, \cdot\cdot, n\} | i$ *è associato alla foglia di* $\overline{f}\}$.

Dimostrazione. Si sonsiderino due indici $i, j \in I_f$, per una generica foglia $\overline{f}$ di $ATW(s, L, l)$ tale che $|f| = L$, le parole di s in i e j condividono lo stesso

prefisso in $ATW(s, L, l)$ con al più $dist_H(s_{i2}, s_{j2}) \leq 1$ per ogni $i2, j2 \in \{0, \cdots, L-1\}$. Di conseguenza $i\ H_L^l\ j$. □

Il lemma 5 assicura che nelle foglie di un albero $ATW(s, L, l)$ siano raggruppate effettivamente le occorrenze delle classi della famiglia di una relazione H_L^l.

Nel Capitolo 3 si analizza la generazione delle combinazioni di posizioni e la struttura di indicizzazione delle combinazioni, nel Capitolo 4 si dimostrano le dipendenze tra gli indici e i problemi della formalizzazione di una gerarchia tra gli indici, infine, nel Capitolo 5 si mostrano tre algoritmi di risoluzione del problema dell'inferenza di motivi con wildcard e si analizza la loro complessità.

Capitolo 3

La generazione di tutte le combinazioni

La scelta degli elementi dell'insieme l per una relazione H_L^l non è determinabile a priori, non si può prevedere dove una mutazione avviene o è avvenuta e per questo motivo le selezioni vanno compiute considerando tutte le possibili scelte. In questo capitolo introduciamo la nozione di combinazione di posizioni e discutiamo alcune modi per rappresentarle. Inoltre, discutiamo la generazione delle combinazioni secondo un opportuno ordinamento e introduciamo la struttura di indicizzazione per le combinazioni.

3.1 Definizione e rappresentazione delle combinazioni

Una scelta di k posizioni di errore in una parola con L posizioni per una relazione H_L^l rappresenta un modo per selezionare un sottoinsieme di k oggetti da un insieme di cardinalità L. Notoriamente questo sottoinsieme è conosciuto con il nome di *k-combinazione* di L elementi. Il numero di *k-combinazioni*

di un insieme di cardinalità L è denotato con $\binom{L}{k}$, per definizione risulta:

$$\binom{L}{k} = \frac{L!}{k!(L-k)!} \tag{3.1}$$

La formula (3.1) è simmetrica per k e L-k:

$$(x+y)^L = \sum_{k=0}^{L} \binom{L}{k} x^k y^{L-k}. \tag{3.2}$$

Ciò significa che il numero di combinazioni di k oggetti selezionati tra L è uguale al numero di $L-k$ oggetti non selezionati t. Enfatizziamo questa simmetria dicendo che

$$L = k + t \tag{3.3}$$

Nel corso della nostra discussione ci riferiremo ad una k-combinazione di L oggetti come ad una (k,t)-combinazione. Una (k,t)-combinazione è un modo per suddividere una collezione di k + t oggetti in due insiemi di cardinalità k e t.

Esistono due modi principali di rappresentare le (k,t)-combinazioni: possiamo scrivere la lista $c_k \cdots c_1$ degli elementi selezionati, o possiamo lavorare con le stringhe binarie $a_{k+t-1} \cdots a_0$ per cui vale la seguente equazione

$$a_{k+t-1} + a_{L-2} + \ldots + a_0 = k \tag{3.4}$$

La stringa binaria si può definire anche così:

$$a_i = \begin{cases} 1 & \text{se selezionato} \\ 0 & \text{se non selezionato} \end{cases} \tag{3.5}$$

L'ultima rappresentazione ha k 1 e t 0 (elementi selezionati e non se-

lezionati). Nella rappresentazione con la lista $c_k \cdots c_1$, gli elementi appartengono all'insieme $\{0, \ldots, L-1\}$ e i valori degli elementi sono ordinati in ordine decrescente:

$$L > c_k > c_{k-1} > \ldots > c_0 \geq 0 \tag{3.6}$$

La notazione binaria permette di collegare entrambe le rappresentazioni in maniera semplice, mostrando che le due rappresentazioni sono equivalenti. Possiamo, per esempio, trasformare una nell'altra. Infatti, la rappresentazione $c_k \cdots c_1$ può essere scritta come:

$$2^{c_k} + 2^{c_{k-1}} + \cdots + 2^{c_0} = \sum_{i=0}^{k+t-1} a_i 2^i = (a_{k+t-1} \ldots a_0)_2 \tag{3.7}$$

Con la notazione $(a_{k+t-1} \ldots a_0)_2$ intendiamo la rappresentazione binaria di una sequenza. Lo stesso tipo di riduzione può essere applicato anche alla lista delle posizioni non selezionate $b_t \cdots b_1$ in un insieme. La rappresentazione $b_t \cdots b_1$ ha valori in $\{0, \ldots, L-1\}$ e i componenti sono ordinati in ordine decrescente. Ciò significa che:

$$L > b_t > \ldots > b_1 \geq 0 \tag{3.8}$$

Un insieme di posizioni $b_t \cdots b_1$ soddisfa la condizione (3.8) se e solo se $c_k \cdots c_1$ soddisfa la (3.6). Le rappresentazioni viste sopra sono tutte equivalenti fra loro. In [Knu98], esistono anche altre forme per modellare le combinazioni. La Tabella 3.1 mostra $\binom{4}{2} = 6$ combinazioni alla luce delle tre forme di rappresentazione viste prima, in cui k=t=2.

L'ordinamento delle combinazioni nella Tabella 3.1 è di tipo lessicografico: i valori delle rappresentazioni $c_k \cdots c_1$ e $a_{k+t-1} \cdots a_0$ sono elencati in ordine decrescente. L'ordinamento di solito suggerisce il modo con cui gener-

$a_3a_2a_1a_0$	b_2b_1	c_2c_1
0011	32	10
0101	31	20
0110	30	21
1001	21	30
1010	20	31
1100	10	32

Tabella 3.1: Tabella per le tre principali rappresentazioni delle combinazioni

are le combinazioni. Generalmente, ordinamenti diversi hanno applicazioni diverse, costi algoritmici diversi e proprietà diverse [Knu98].

Come vedremo nel prossimo capitolo, la scelta di un ordinamento è importante nello studio delle dipendenze tra le combinazioni e gli indici generati a partire da essi. Tra tutte le forme di ordinamento e rappresentazione delle combinazioni scegliamo l'ordinamento lessicografico e la lista delle posizioni selezionate $c_k \cdots c_1$. Esse sono tra le più usate e godono di un gran numero di studi teorici e risultati accertati.

3.2 L'albero Binomiale e l'organizzazione delle combinazioni

In [Cor90], un albero binomiale rappresenta tutti i possibili sottoinsiemi di un insieme di cardinalità n in un albero seguendo l'ordinamento lessicografico. L'albero binomiale è definito ricorsivamente.

Definitione 28 (Albero binomiale)**.** *Data una lunghezza $L > 0$, un albero Binomiale B_L è un albero ordinato secondo una strategia. L'albero B_0 consiste di un solo nodo. L'albero B_L si compone di due alberi binomiali B_{L-1} collegati assieme. La radice di uno dei due alberi è il figlio più a sinistra della radice dell'altro (come mostrato in Figura 3.1).*

Mostriamo adesso, le principali proprietà degli alberi binomiali, riportando anche la loro dimostrazione.

Lemma 6 (Proprietà note di un albero binomiale)**.** *Sia B_L un albero binomiale, valgono le seguenti proprietà:*

1. *i nodi dell'albero sono* 2^L.
2. *l'altezza dell'albero è* L.
3. *i nodi a profondità* i *sono* $\binom{L}{i}$, per i= 0,..,L , *e*
4. *la radice dell'albero ha grado* L. *Il grado della radice è maggiore del grado di ogni altro nodo; inoltre se* L, L-1, L-2, ...,0 *è un enumerazione, da sinistra verso destra , dei figli della radice, allora il figlio numero* i *è la radice di un sottoalbero binomiale* B_i.

Riporto la dimostrazione delle proprietà principali di un albero binomiale da [Cor90] perchè sono importanti per la comprensione degli algoritmi di generazione delle combinazioni che vedremo nel Capitolo 5.

Dimostrazione. La dimostrazione è per induzione su L. Per ognuna delle proprietà la base dell'induzione è l'albero binomiale B_0. La verifica della validità delle proprietà per B_0 è immediata.

Per dimostrare i passi induttivi, assumiamo che il lemma sia verificato per l'albero binomiale B_{L-1}.

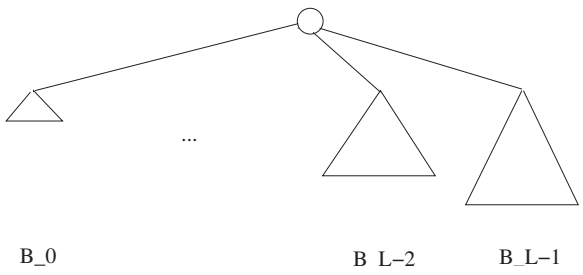


Figura 3.1: Albero binomiale B_L

1. L'albero binomiale B_L consiste di due copie di B_{L-1}, quindi B_L ha $2^{L-1} + 2^{L-1} = 2^L nodi$.

2. L'albero B_L è costituito da due copie di alberi B_{L-1}, collegate assieme, pertanto la profondità massima di un nodo nell'albero B_L si ottiene sommando *1* alla profondità massima di B_{L-1}. Per l'ipotesi induttiva, la massima profondità dell'albero B_L, risulta essere *(L-1)+1= L*.

3. Sia *D(L,i)* il numero di nodi a profondità i dell'albero binomiale B_L. Per costruzione l'albero binomiale B_L è costituito da due copie di B_{L-1} collegate assieme, pertanto un nodo a profondità i in B_{L-1} nell'albero B_L compare una volta a profondità i e l'altra volta a profondità *i+1*. In altri termini, abbiamo che il numero di nodi a profondità i nell'albero B_L è dato dalla somma del nummero di nodi a profondità i di B_{L-1} con il numero di nodi a profondità *i-1* in B_{L-1}. Quindi $D(L,i) = D(L-1,i) + D(L-1,i-1) =$

$$\binom{L-1}{i} + \binom{L-1}{i-1} = \binom{L}{i}. \tag{3.9}$$

   La seconda uguaglianza segue direttamente dall'ipotesi induttiva, e la terza è ovvia.

4. Il solo nodo con un grado più alto in B_L rispetto a quello che aveva in B_{L-1} è il nodo radice che ha un figlio in più di quanti ne aveva B_{L-1}. Dato che la radice di B_{L-1} ha grado *L-1*, segue segue che la radice di B_L ha grado *k*. Per l'ipotesi induttiva, e come è mostrato nella figura Albero binomiale B_L, i figli della radice di B_{L-1} sono, da sinistra a destra, le radici di $B_{L-2}, B_{L-3}, ..., B_0$. Pertanto, quando si collegano

assieme i due alberi B_{L-1} abbiamo che i figli della radice risultante diventano $B_{L-1}, B_{L-2}, ..., B_0$.

□

Mostriamo adesso, un esempio reale di albero binomiale e le sue proprietà.

Esempio 14. *Consideriamo un albero binomiale completo con:*

1. $\binom{4}{0} + \binom{4}{1} + \binom{4}{2} + \binom{4}{3} + \binom{4}{4} = 2^4$ *nodi,*
2. *altezza 4,*
3. *la radice ha grado 4 maggiore di ogni altro nodo.*

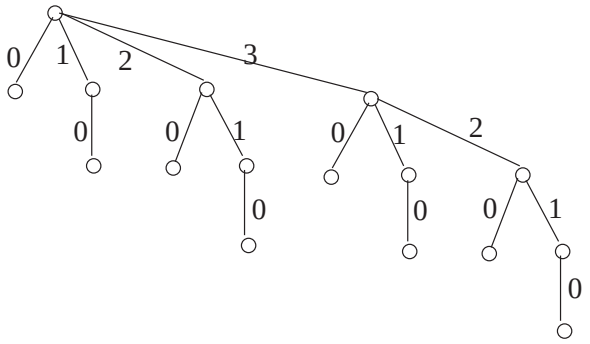


Figura 3.2: Albero binomiale B_4

Come si può notare dalla figura, il figlio più a destra della radice ha come sottoalbero un sottoalbero che è uguale all'albero bonomiale B_4 senza il sottoalbero considerato.

Capitolo 4

La gerarchia tra indici

Una gerarchia è un ordine di dipendenza tra gli elementi di un gruppo. Un indice è una struttura dati che permette di accedere in maniera efficiente alle informazioni contenute in un testo. Se una gerarchia si riferisce ad un gruppo di indici, allora essa è un ordine di dipendenza tra indici. Nel Capitolo 2 abbiamo visto come ottenere un indice da una sequenza s di lunghezza n e come modificare opportunamente tale indice per creare un altro indice, che è anch'esso un albero e che abbiamo chiamato ATW; in questo capitolo mostriamo come ottenere una particolare gerarchia tra indici, partendo da un esempio illustrativo.

4.1 Un esempio di ordinamento tra indici

Consideriamo una sequenza s di n caratteri su un alfabeto Σ, un intero $L < n$ che rappresenta la lunghezza delle parole da cercare e un albero binomiale B_L, nell'Esempio 15 si mostra un ordinamento lessicografico di indici, di tipo alberi ATW(s, L, l), dove l rappresenta un *insieme combinazione* di B_L. L'insieme l è formato dalle etichette del percorso di B_L che va dalla radice a un certo nodo di B_L.

Esempio 15. *Sia* $s = AAAATTACCCCATAGT$ *la sequenza di lunghezza* $n = 16$ *definita su* $\Sigma = \{A, T, C, G\}$, *l'intero* $L = 4 < n$ *e* B_4 *l'albero binomiale. Il trie per i suffissi della sequenza* s *è mostrato nella Figura 4.1:*

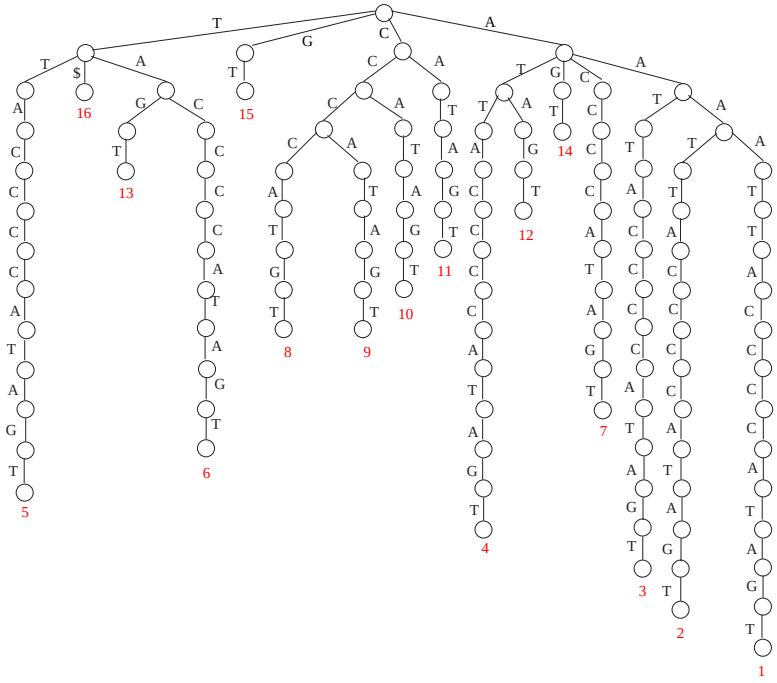


Figura 4.1: Trie dei suffissi per s

La generazione lessicografica delle combinazioni è ottenibile con una visita a ventaglio di B_4 *in Figura 3.2. La sequenza di insiemi combinazione generata con una visita a ventaglio da sinistra verso destra è la seguente:*
$\{0\}\{1\}\{2\}\{3\}\ \{1,0\}\{2,0\}\{2,1\}\{3,0\}\{3,1\}\{3,2\}\ \{2,1,0\}\{3,1,0\}\{3,2,0\}\{3,2,1\}\{3,2,1,0\}$

Gli alberi $ATW(s, L, l)$ *creati a partire da questi insiemi combinazione sono mostrati nell'Esempio C.*

L'ordinamento degli indici dell'Esempio 15 mostra solo un modo per creare un indice per ogni possibile combinazione presente in B_4. Una generazione ordinata, non è una gerarchia. Una gerarchia tra gli indici si costituisce nel momento in cui si definiscono delle dipendenze tra gli indici stessi.

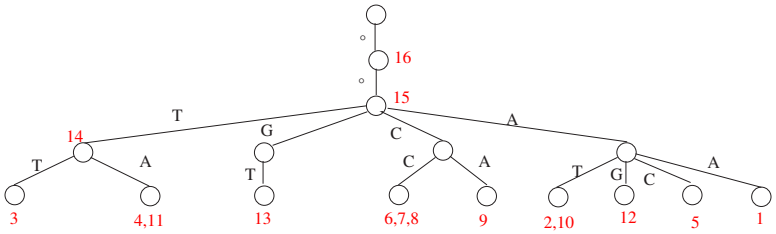


Figura 4.2: ATW(s,4,{1, 0})

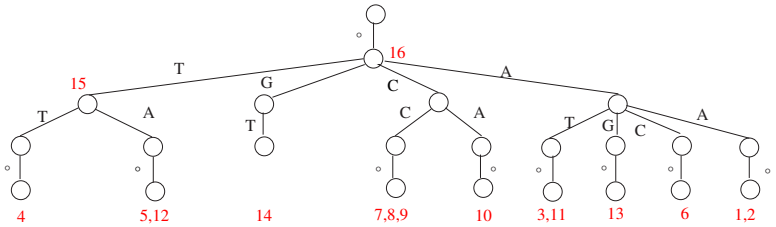


Figura 4.3: ATW(s,4,{2, 1})

L'Esempio 15, visto sopra, non definisce nè evidenzia nessuna dipendenza, ma solo un ordine di generazione; il nostro obbiettivo è dimostrare che esistono specifiche dipendenze tra indici ottenuti a partire da combinazioni con lo stesso numero di elementi e tra indici ottenuti a partire da combinazioni con un numero diverso di elementi.

Per capire le relazioni tra gli indici cerchiamo prima di tutto di evidenziare con delle osservazioni alcuni fatti che riguardano una parte degli indici visti nell'Esempio 15. A questo scopo consideriamo gli alberi ATW delle Figure A.5, A.7, A.10: Osservando gli insiemi associati alle foglie nelle Fig-

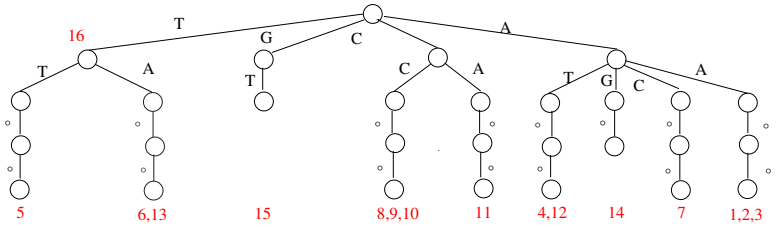


Figura 4.4: ATW(s,4,{3, 2})

ure 4.2, 4.5, 4.8, si nota come gli indici dei suffissi siano in relazione tra di loro. La Tabella 4.1 raggruppa questi insiemi di occorrenze di suffissi, che denotiamo singolarmente con *occ*, evidenziando questa relazione in particolare per gli insiemi omologhi. Rappresentiamo, inoltre, per convenzione l'insieme $\{-1, 0, 1\}$ con l'insieme $\{1\}$ e l'insieme $\{0, 1, 2\}$ con l'insieme $\{1, 2\}$. L'osservazione che segue riassume la relazione tra gli insiemi di occorrenze

Figura 4.2	*Figura* 4.5	*Figura* 4.8
{3}	{4}	{5}
{4,11}	{5,12}	{6,13}
{13}	{14}	{15}
{6,7,8}	{7,8,9}	{8,9,10}
{9}	{10}	{11}
{2,10}	{3,11}	{4,12}
{12}	{13}	{14}
{5}	{6}	{7}
{-1,0,1}	{0,1,2}	{1,2,3}

Tabella 4.1: Tabella per gli insiemi di indici di suffissi omologhi nelle Figure 4.2, 4.5, 4.8

di suffissi.

Osservazione 1. *Esiste una corrispondenza biunivoca tra gli elementi delle colonne che si esprime considerando due interi* $d \neq 0$ *e un* $d' \neq 0$ *tali che, per ogni insieme occ i corrispondenti omologhi nella riga* occ' *e* occ'' *sono fatti in modo che per ogni* $i \in occ'$ $i = j+d$ *con* $j \in occ$ *e per ogni* $i \in occ''$ $i = j + d'$ *con* $j \in occ$.

In altre parole, ogni insieme di occorrenza di indici di suffissi di uno degli alberi può essere calcolato da un omologo insieme di indici di suffissi di un altro albero del gruppo, aggiungendo o sottraendo, ad ogni indice dell'insieme un intero d che è lo stesso per tutti gli insiemi di occorrenze dell'albero ATW . Il segno di d dipende dall'insieme combinazione preso come base di confronto.

Nell'esempio, $d = 1$ tra gli insiemi delle Figure 4.2 e 4.5 e tra 4.5 e 4.8; $d = 2$ per 4.2 e 4.8. Se il confronto avesse preso come base, ad esempio, 4.8 e 4.2 invece che 4.2 e 4.8, il valore di d sarebbe stato $d = -2$.

Di fatto, l'osservazione evidenzia e definisce una relazione tra alcuni indici. La relazione rappresenta una dipendenza tra gli alberi ATW considerati. Nel caso degli indici visti nelle Figure 4.2, 4.5, 4.8 se la base del confronto è, per nostra scelta, l'indice rappresentato dall'albero ATW in Figura 4.8, allora gli indici nelle Figure 4.2, 4.5 si dice che sono generati, o dipendenti, dall'indice ATW in Figura 4.8. La dipendenza biunivoca non è solo una relazione tra gli alberi ATW ma anche degli insiemi combinazione da cui essi sono calcolati. In tal caso, l'osservazione 1 si può estendere anche agli insiemi combinazione l, l' e l'' degli alberi ATW(s, L, l), ATW(s, L, l') e ATW(s, L, l'') per le figure viste nell'Osservazione 1. Nel nostro caso, $l = \{3, 2\}$, $l' = \{2, 1\}$, e $l'' = \{1, 0\}$.

Osservazione 2. *Esiste una corrispondenza biunivoca tra gli insiemi combinazione relativi gli ATW visti nella Tabella 4.1, che si esprime considerando tre interi* $d \neq 0$, $d' \neq 0$ *e* $d'' \neq 0$ *tali che per ogni* $i \in l'$ $i = j + d$ *con* $j \in l$, *per ogni* $i \in l''$ $i = j + d'$ *con* $j \in l'$ *e per ogni* $i \in l''$ $i = j + d''$ *con* $j \in l$.

Per gli alberi ATW della Tabella 4.1 i valori di d, d' e d'' sono $d = d' = 1$ e $d'' = 2$. Nella prossima sezione mostreremo che non è un caso che i valori $d, d' ed''$ sono gli stessi sia per la prima che per la seconda osservazione. Inoltre, il segno degli interi d, d' e d'' dipende dalla combinazione di base del confronto.

Come abbiamo accennato prima, la relazione non è valida per ogni scelta di indici, infatti, ci sono degli specifici insiemi combinazione, per i quali la relazione non vale. Le Figure 4.5 e 4.6, mostrano un esempio di scelta indici tra cui la relazione non vale:

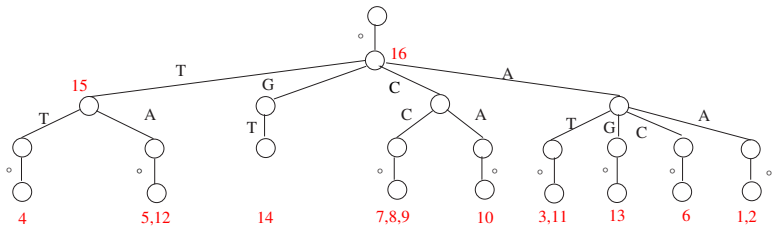


Figura 4.5: ATW(s,4,{2, 1})

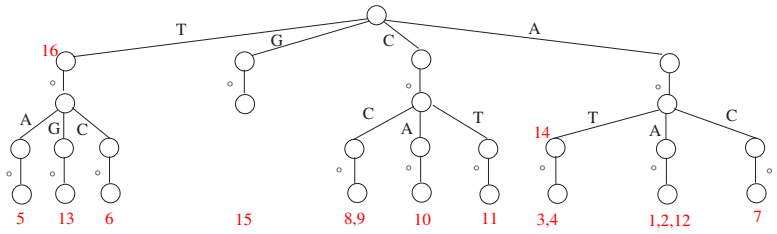


Figura 4.6: ATW(s,4,{3, 1})

Come per le Figure A.5, A.7, A.10, consideriamo raggruppiamo in una tabella gli insiemi associati alle foglie delle Figure 4.5 e 4.6. Come è evidente,

Figura 4.5	*Figura* 4.6
{4}	{13}
{5,12}	{5,6}
{14}	{15}
{7,8,9}	{8}
{10}	{19,11}
{3,11}	{10}
{13}	{1,4}
{6}	{12}
{1,2}	{7}
{/}	{2,3}

Tabella 4.2: Tabella per gli insiemi di indici di suffissi omologhi nelle Figure 4.5, 4.6

le due figure non hanno nè lo stesso numero di foglie nè un valore d intero che li renda simili.

4.2 Base di alberi e le combinazioni

In questa sezione definiamo e dimostriamo formalmente ciò che abbiamo anticipato con le osservazioni della sezione precedente, introducendo la nozione di relazione di dipendenza reciproca tra insiemi.

Definitione 29 (Dipendenza Reciproca di ordine d tra insiemi)**.** *Dati due insiemi* $c = \{c_k, \cdots, c_1\}$ *e* $c' = \{c'_k, \cdots, c'_1\}$ *con lo stesso numero di elementi si dice:*

$c \mathcal{R}_d c'$ *se e solo se* $c'_i = c_i + d$ *per ogni* $i = \{k, \cdots, 1\}$*. La relazione* $\mathcal{R}_d$ *è detta anche relazione della* reciproca dipendenza *tra due insiemi combinazione.*

E' facile mostrare che la relazione $\mathcal{R}_d$ è una relazione riflessiva, simmetrica e non transitiva. La non transitività non è necessaria perchè come vedremo nel raffinamento la relazione è strettamente applicata o verificata tra coppie. Se due insiemi non godono della dipendenza reciproca di ordine d allora si dicono *indipendenti* per l'ordine d.

Mostriamo nell'Esempio 16 alcuni insiemi in relazione $\mathcal{R}_d$ e altri insiemi indipendenti per $\mathcal{R}_d$.

Esempio 16. *L'insieme* $C = \{\{1,0\},\{2,1\}\}$ *è un esempio di insieme di insiemi in relazioni* $\mathcal{R}_1$*. Dove per* C *l'insieme* $\{2,1\}$ *è* c' *e* $\{1,0\}$ *è* c*. Mentre l'insieme* $C' = \{\{2,0\},\{2,1\}\}$ *è un esempio di insieme di insiemi indipendenti per* $\mathcal{R}_1$*, perchè l'insieme* $\{2,1\}$ *e* $\{2,0\}$ *hanno il componente alla prima posizione uguale a 2 e non differiscono per 1, come richiede la relazione.*

La dipendenza reciproca di ordine d tra insiemi combinazione coinvolge anche gli indici ATW creati a partire da tali insiemi combinazione.

Ampliamo la nostra notazione indicando con $OCC(s, L, l)$ l'insieme degli insiemi *occ* di un indice ATW(s, L, l). Diamo, di seguito, una definizione

formale della reciproca dipendenza tra gli indici costruiti da due insiemi combinazione in relazione $\mathcal{R}_d$.

Definitione 30 (La reciproca dipendenza o corrispondenza biunivoca tra ATW)**.** *Dati due alberi ATW(s, L, l) e ATW(s, L, l'), si dice che ATW(s, L, l) e ATW(s, L, l') godono della* reciproca dipendenza *di ordine d tra indici se e solo se per ogni occ $\in OCC(s, L, l)$ esiste un unico occ$' \in OCC$(s,L,l') tale che occ $\mathcal{R}_d$ occ$'$.*

In altre parole, due ATW relativi a due insiemi combinazione diversi si dicono in relazione reciproca se hanno tutti gli insiemi di indici dei suffissi in relazione $\mathcal{R}_d$ tra loro. La relazione che lega gli insiemi combinazione in relazione $\mathcal{R}_d$ e i relativi indici ATW costruiti a partire da tali combinazioni è molto forte. Il Lemma 7 chiarisce questo legame dimostrando che se due indici sono costruiti a partire da due insiemi combinazione in relazione di ordine d tra loro allora anche gli indici saranno in relazione per lo stesso ordine d.

Lemma 7. *Dati due alberi ATW(s, L, l) e ATW(s, L, l'), l $\mathcal{R}_d$ l' se e solo se ATW(s, L, l) e ATW(s, L, l') godono della reciproca dipendenza di ordine d tra ATW.*

Dimostrazione. Dati due insiemi combinazione l e l' tali che l $\mathcal{R}_d$ l', allora se ATW(s, L, l) e ATW(s, L, l') non godessero della reciproca dipendenza tra ATW allora per la definizione 30 d non sarebbe tale che per ogni $occ \in OCC(s, L, l)$ esiste $occ' \in OCC(s, L, l')$ tale che occ $\mathcal{R}_d$ occ'. Questo significa anche che esiste almeno un indice $i \in occ'$ tale che $i \neq j+d$ per qualche indice $j \in occ$. Ma allora nè il suffisso j-esimo contiene il suffisso i-esimo nè viceversa. Questo può capitare solo se l non è in relazione $\mathcal{R}_d$ con l'. Il che non è possibile per l'ipotesi. Dall'altro lato, dati due ATW(s, L, l) e ATW(s, L, l') che

godono della reciproca dipendenza tra ATW allora d è tale per cui per ogni $occ \in OCC(s, L, l)$ esiste un unico $occ' \in OCC(s, L, l')$ tale che $occ \; \mathcal{R}_d \; occ'$. Se l non fosse in relazione $\mathcal{R}_d$ con l' allora per quanto dimostrato prima ATW(s, L, l) e ATW(s, L, l') non dovrebbero essere in reciproca dipendenza è questo è impossibile.

□

Una volta definita la reciproca dipendenza la gerarchia tra indici si costituisce scegliendo l'indice di base del confronto. Chiamiamo questo indice per ogni insieme di indici in reciproca dipendenza *indice rappresentante* dell'insieme. Se l'insieme è composta da un solo indice quindi da un solo insieme combinazione, allora il rappresentante è l'unico indice ATW dell'insieme.

A questo punto, le dipendenze tra gli indici e la scelta del rappresentante permettono di definire il concetto di base di indici o di combinazioni. Tale concetto è di fondamentale importanza perchè la base ha lo scopo di generare gli altri elementi, siano essi indici o insiemi combinazione. La base è strettamente legata alla nozione di livello. Un livello è un insieme di elementi con lo stesso numero di elementi. In particola, quando si riferiamo ad un livello intendiamo l'insieme delle combinazioni con lo stesso numero di elementi, quindi alla stessa profondità nell'albero binomiale B_L.

Per ogni livello di B_L l'insieme dei rappresentanti per quello specifico livello forma l'insieme di combinazioni chiamato *base*. È ovvio dire che esistono tante basi quanti sono i livelli di profondità dell'albero binomiale B_L.

La scelta dei rappresentanti è arbitraria, chiaramente all'interno di ogni insieme di combinazioni in relazione. Vediamo quali combinazioni abbiamo scelto per formare una base.

Sia C^L l'insieme di tutti i 2^L insiemi combinazione dell'albero binomiale B_L.

Distinguiamo i sottoinsiemi C_i^L, per $i = \{0, \ldots, L\}$, di C^L come tutte le $\binom{L}{i}$ combinazioni di lunghezza i. Questi insiemi costituiscono il livello i-esimo di B_L.

Classifichiamo le combinazioni di un insieme C_L^i in tre categorie distinte. La classificazione serve per due motivi: per distinguere le combinazioni della base dal resto delle combinazioni e per distinguere le combinazioni rappresentanti per le classi con più di un elemento da quelle che ne hanno solo uno, all'interno della base.

Definitione 31 (Combinazione di tipo A). *Un* ***combinazione*** $c_i \ldots c_1 \in C_L^i$ *è di* ***tipo*** $\mathcal{A}$ *se e solo se* $c_i = L - 1$ *e* $c_1 > 0$.

Definitione 32 (Combinazione di tipo B). *Un* ***combinazione*** $c_i \ldots c_1 \in C_L^i$ *è di* ***tipo*** $\mathcal{B}$ *se e solo se* $c_i < L - 1$.

Definitione 33 (Combinazione di tipo C). *Un* ***combinazione*** $c_i \ldots c_1 \in C_L^i$ *è di* ***tipo*** $\mathcal{C}$ *se e solo se* $c_i = L - 1$ *e* $c_1 = 0$.

L'insieme delle combinazioni di tipo A e di tipo C di un livello C_L^i formano per nostra scelta la base per quello specifico livello.

La classificazione delle combinazione nei tipi A, B e C coinvolgono tutte le combinazioni di un livello. Cioè una combinazione generica $c_i \ldots c_1 \in C_L^i$ deve necessariamente appartenere a una delle tre classi. Se la combinazione, avendo i componenti in ordine decrescente, il suo componente c_i può assumere il valore massimo $L - 1$ o un valore minore. Nel primo caso rientra nelle classi A e C nell'altro appartiene alla classe B.

La figura di seguito mostra un esempio delle differenti classi di combinazione di C_4^1 all'interno dell'albero binomiale B_4.

Come si nota anche nell'esempio 17 per ogni livello dell'albero binomiale le basi sono tutte concentrate nel sottoalbero di B_L che raggruppa tutte le

Esempio 17 (Un esempio sulle classi A, B, C).

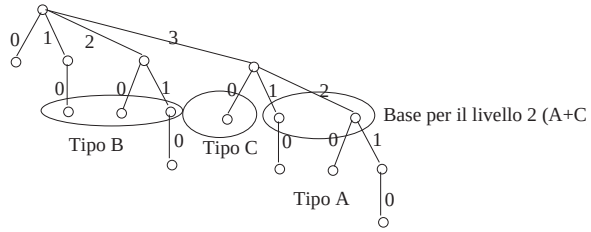


Figura 4.7: Nodi di classe A,B,C.

combinazioni che hanno come componente c_k il valore massimo $L-1$. Chiamiamo questo sottoalbero il *sottoalbero delle basi* di B_L. Le combinazioni di classe B, invece, sono tutte concentrate negli altri sottoalberi.

La nozione di *base* di un gruppo di alberi , come sappiamo, è relativo solo agli indici costruiti su combinazioni con lo stesso numero di elementi. Nella prossima sezione dimostriamo che esiste una relazione tra combinazioni che hanno un numero diverso di elementi.

Adesso quantifichiamo il numero di combinazioni di tipo B di un generico livello.

Lemma 8. *Il numero di combinazioni di tipo B, denotato con $\mathcal{B}_i$, è:*

$$\mathcal{B}_i = \binom{L-1}{i}. \tag{4.1}$$

Dimostrazione. Dalla Formula 3.9 sappiamo che $\binom{L-1}{i} + \binom{L-1}{i-1} = \binom{L}{i}$. Dove $\binom{L-1}{i}$. In altri termini, abbiamo che il numero di nodi a profondità i nell'albero B_L è dato dalla somma del nummero di nodi a profondità i di B_{L-1} con il numero di nodi a profondità *i-1* in B_{L-1}. Ma i nodi a profondità i di B_{L-1} sono i tutti i nodi delle combinazioni di tipo B, quindi, $\binom{L-1}{i}$ è il numero di combinazioni di tipo B per il livello i dell'albero binomiale B_L. □

Il numero di combinazioni che formano la base di un generico livello è il seguente.

Lemma 9. *Il numero di combinazioni della base(tipo A + tipo C) di un C_L^i, indicato con C_i, è:*

$$\mathcal{C}_i = \binom{L-1}{i-1}. \tag{4.2}$$

Dimostrazione. Dalla Formula 3.9 sappiamo che $\binom{L-1}{i} + \binom{L-1}{i-1} = \binom{L}{i}$. Dalla Formula 8 il numero di combinazioni di tipo B di un C_L^i sono $\binom{L-1}{i}$, il numero $\binom{L}{i}$ conta tutte le combinazioni di tipo A + B + C. Allora il numero di combinazioni di tipo A+C che rappresentano la base per C_L^i sono: $\binom{L}{i} - \binom{L-1}{i} = \binom{L-1}{i-1}$. □

Alla luce della nozione di base dimostriamo che per ogni insieme combinazione c di tipo B di un livello qualsiasi di un albero binomiale esiste un insieme combinazione c' di tipo A nella base di quel livello tale che c è in reciproca dipendenza di ordine d con c'.

Lemma 10. *Dato un insieme C_L^i per un albero binomiale B_L, per ogni insieme combinazione c di tipo B per C_L^i esiste un insieme combinazione c' di tipo A per C_L^i tale che c' è in reciproca dipendenza $\mathcal{R}_d$ con c.*

Dimostrazione. Un insieme combinazione c di tipo B per un C_L^i è un insieme di i elementi $\{c_i \cdots c_1\}$ con $c_i \neq L-1$. Un insieme combinazione c' di tipo A per un C_L^i è un insieme di i elementi $\{c'_i \cdots c'_1\}$ con $c_i = L-1$ e $c_i \neq 0$. Se c' non fosse in relazione $\mathcal{R}_d$ per un qualche d con c allora non varrebbe $c'_i = c_i + d$ per ogni $j = i, \cdots, 1$. Questo vuol dire che se costruiamo $c'_j = c_j + d'$ per un intero $d' > 0$ e per ogni $j = i, \cdots, 1$ tale che $c_i + d = L-1$ e $c'_1 > 0$, allora c' non è di tipo A, questo è impossibile per ipotesi. □

Un esempio esplicativo del lemma appena definito si può osservare in Figura 4.10.

Un altro aspetto importante da modellare è legato al numero di combinazioni di tipo B generabili da una combinazione di tipo A.

Definitione 34. *Date una lunghezza L e un insieme C_L^i di combinazioni B_L e un insieme combinazione $c = c_L \dots c_i$ di tipo A in C_L^i, le combinazioni di tipo B generabili da c sono le combinazioni $c' = c'_L \dots c'_i$ in C_L^i tali che $c'_L = c_L - d$ per ogni $d = \{1, \cdots, c_i\}$. Il loro numero è quindi c_i.*

La Definizione 34 è importante come vedremo per l'algoritmo con raffinamento nel prossimo capitolo.

La relazione tra insiemi combinazione con lo stesso numero di elementi è importante ma non è l'unica relazione tra le combinazioni. Come dimostriamo adesso ogni insieme combinazione della base per C_L^{i+1} di B_L può essere costruito a partire da una combinazione di tipo B dell'insieme C_L^i dell'albero binomiale B_L aggiungendo l'elemento $L-1$.

Lemma 11. *Dato un insieme C_L^{i+1} per un albero binomiale B_L, per ogni insieme combinazione c di tipo A o C per C_L^{i+1} esiste un unico insieme combinazione c' di tipo B di C_L^i di B_L, tale che, $c = c' \cup p$ per $p = L-1$.*

Dimostrazione. Data una combinazione $c \in C_L^{i+1}$, $c = c_L \cdots c_i$ essendo di tipo A o C ha il componente $c_L = L-1$ per definizione, consideriamo le componenti $c_{L-1} \cdots c_{i-1}$ se non esistesse un unico insieme combinazione $c' = c'_L \cdots c_i$ di tipo B di C_L^i di B_L, tale che, $c = c' \cup p$ per $p = L-1$, allora non dovrebbe esistere anche la combinazione di tipo A con cui ogni c' di tipo B è in relazione di reciproca dipendenza. Questo è impossibile perchè la combinazione c'' di tipo A con cui c' è in relazione è fatta così: $c'' = c'_L + d \cdots c'_i + d$ per $d = (L-1) - c_L$. Quindi esiste c' di tipo B tale che $c = c' \cup p$ per $p = L-1$. □

Il Lemma 11 dimostra la relazione tra combinazioni con un numero diverso di elementi. Questo risultato è molto importante perchè rappresenta l'idea di base di un nuovo algoritmo di generazione per tutte le possibili combinazioni di un albero binomiale B_L che vedremo in particolare nel prossimo capitolo.

4.3 La gerarchia e il raffinamento

La nuova idea di generazione delle combinazioni data con il Lemma 11 ha degli sviluppi importanti anche nella costruzione degli indici ATW. Infatti, l'ordine di generazione delle combinazioni permette, come dimostreremo nel prossimo capitolo, di creare gli indici ATW in maniera incrementale, cioè creando un indice ATW(s, L, l) a partire da un altro indice ATW(s, L, l') tale $|l'| = |l| + 1$. In generale, ogni insieme di occorrenze nelle foglie di ATW(s, L, l'), subisce una divisione in sottoinsiemi nell'indice successivo ATW(s, L, l'). Questa divisione è chiamata *raffinamento*. Il raffinamento è stato creato da noi per modellare questa relazione tra gli indici ATW generati con il procsso di generazione delle combinazioni data con il Lemma 11. Mostriamo adesso due indici ATW presi dall'Esempio 15 in cui è evidente la relazione che ha portato al raffinamento.

Esempio 18. *In questo esempio le figure si riferiscono agli indici ATW costruiti a partire da due combinazioni* $l' = \{3, 2\}$ *e* $l = \{3, 2, 1\}$*, tali che,* $3 = |l| = |l'| + 1 = 2 + 1$*,* $\{3, 2\}\ \mathcal{R}_1 \{2, 1\}$ *e il componente* $l_L = L - 1 = 3$ *con* $L = 4$*.*

Nell'Esempio 18 è evidente come gli insiemi di occorrenze $\{13, 6\}$, $\{8, 9, 10\}$, $\{4, 12\}$ e $\{1, 2, 3\}$ della Figura 4.8 si dividono nei rispettivi insiemi $\{13\}$, $\{6\}$, $\{8, 9\}$, $\{10\}$, $\{4\}$, $\{12\}$, $\{1, 2\}$ e $\{3\}$.

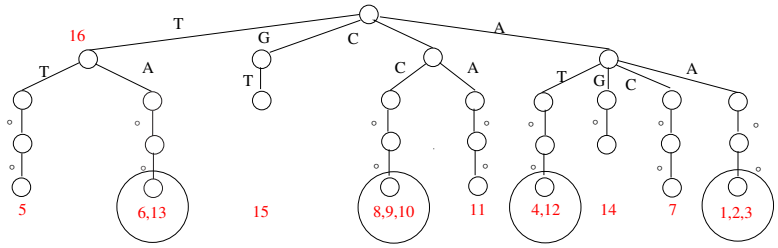


Figura 4.8: ATW(s,4,{3, 2})

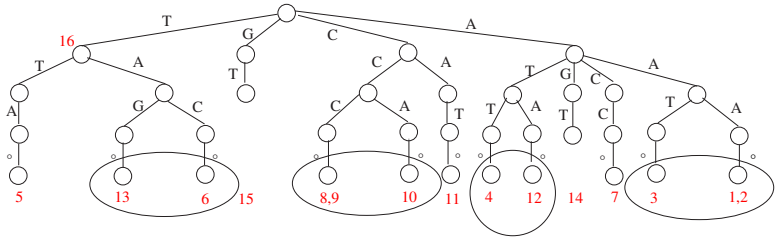


Figura 4.9: ATW(s,4,{3, 2, 1})

La generazione delle combinazioni individua anche un ordinamento tra le combinazioni. Quindi, seguendo questo ordinamento il raffinamento parte da un unico grande insieme di indici di suffissi del testo relativi ad un albero ATW fatto da soli caratteri wildcard.

Nel processo di raffinamento che segue l'ordinamento della generazione delle combinazioni, ad ogni iterazione una posizione dell'albero ATW in costruzione, generalmente la prima, si trasforma da posizione wildcard a solida. Questa trasformazione crea nuovi prefissi nell'albero ATW in costruzione con la conseguente divisione di quello che era o erano gli insiemi di indici di suffissi di partenza. Le dipendenze stabilite tra gli indici ATW nel raffinamento costituisce la gerarchia di indici di cui parlavamo ad inizio capitolo.

Rimane solamente un aspetto da modellare del raffinamento. Nel caso della generazione di una base specifica non è necessario generare tutte le combinazioni ma solo quelle che effettivamente generano gli alberi per la

base obbiettivo. Questa intuizione si formalizza identificando tutte e solo le combinazioni che nel sottoalbero delle basi sono annidate in un percorso che va dalla radice ad un nodo relativo ad una combinazione della base obbiettivo. L'esempio che segue mostra quali nodi dell'albero binomiale B_L è necessario generare per raffinamento della base obbiettivo.

Ipotizziamo di inferire tutti i motivi con $k = 1$ differenze.

Esempio 19 (Le combinazioni necessarie)**.** *Prendiamo l'albero binomiale B_5 e mostriamo la gerarchia delle combinazioni di B_5 che concorrono alla generazione della base obbiettivo per $L - k = 4$.*

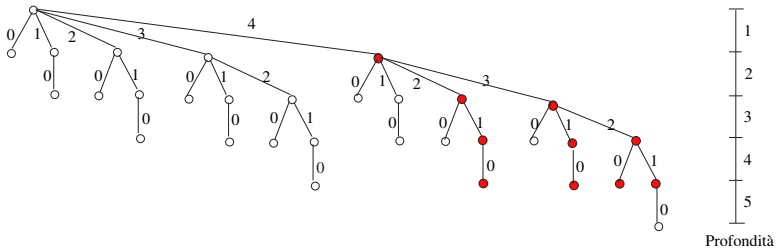


Figura 4.10: Nodi relativi alle combinazioni di classe A,C da calcolare.

L'algoritmo di generazione mirata così come lo abbiamo chiamato, è discusso e approfondito nel prossimo capitolo.

Capitolo 5

Algoritmi proposti per l'inferenza di motivi con wildcard

In questo capitolo porteremo avanti due possibili soluzioni che migliorano molto la complessità algoritmica dell'approccio naive per l'inferenza di motivi con wildcard. La prima soluzione consiste nella generazione diretta degli alberi ATW a partire dalle combinazioni della base obbiettivo e la seconda si basa su un idea di generazione alternativa alla precedente e una costruzione degli alberi ATW con la tecnica del raffinamento. Il primo algoritmo che esaminiamo è l'algoritmo naive per l'inferenza di motivi con wildcard. Il problema dell'inferenza di motivi con wildcard è stato introdotto nella Definizione 1, l'uso degli indici ATW e la nuova relazione di equivalenza H_L^l hanno permesso di inferire i motivi con wildcard direttamente negli insiemi di occorrenze associati alle foglie degli ATW. In altre parole, i motivi del problema sono rappresentati dalle etichette dei percorsi che vanno dalla radice alle foglie degli indici ATW, le occorrenze sono i rispettivi insie-

mi di indici di suffissi del testo associati a tali foglie e le occorrenze con al più k wildcard sono definite considerando tutte le possibili combinazioni di posizioni wildcard per ogni parola del testo. Chiamiamo l'insieme delle combinazioni di $L-k$ posizioni e i rispettivi alberi ATW, combinazioni obbiettivo e alberi obbiettivo.

5.1 Algoritmo naive

L'algoritmo naive per l'inferenza dei motivi con wildcard costruisce un nuovo ATW per ogni combinazione obbiettivo. Le combinazioni obbiettivo sono quelle restituite dalla visita lessicografica condotta sell'albero binomiale B_L. Esaminiamo l'Algoritmo 1 che genera lessicograficamente le combinazioni obbiettivo del problema dell'inferenza di motivi approssimati per i parametri L e k. L'idea di base dell'Algoritmo 1 è di calcolare lessicograficamente per ogni combinazione c tutte le possibili combinazioni che hanno c come prefisso.

Il numero di combinazioni obbiettivo restituite dall'Algoritmo 1 è uguale a tutte le $\binom{L}{k}$ possibili combinazioni di k wildcard su L posizioni. L'algoritmo genera tutte le combinazioni che nell'albero B_L sono lessicograficamente precedenti alle combinazioni obbiettivo. Il numero delle combinazioni visitate è $\sum_{i=1}^{L-k} \binom{L}{i} = 2^{L-k}$.

L'ordine di generazione delle combinazioni è mostrato in dettaglio nell'Esempio 20, in cui è indicata la sequenza di nodi combinazione dell'albero binomiale B_5 visitati con l'Algoritmo 1. Ogni figura mostra anche i nodi già generati al momento della generazione della combinazione obbiettivo a cui si riferisce la figura stessa.

Esempio 20.

Algorithm 1 Algoritmo per la generazione naive delle combinazioni

```
1: procedure GenAllCombination(L, k)
2:     t ← L
3:     c_t ← Null
4:     coda ← push(c)                          ▷ coda è una normale coda FIFO
5:     pa ← −1                                 ▷ pa rappresenta la profondità attuale
6:     while coda ≠ Null do
7:         c ← pop(coda)    ▷ push e pop sono le operazioni di inserzione e
   estrazione dalla coda
8:         if |c| > pa then                    ▷ |c| è la lunghezza di c
9:             pa ← |c|
10:            t ← t − 1
11:        end if
12:        if c_{t+1} = 0 then
13:            N ← −1
14:        else
15:            N ← c_{t+1} − 1
16:        end if
17:        for j ← 0, N do
18:            c' ← c
19:            c'_t ← j
20:            if |c'| = L − k then
21:                Stampa c'
22:            else if |c'| < L − k then
23:                coda ← push(c')
24:            end if
25:        end for
26:    end while
27: end procedure
```

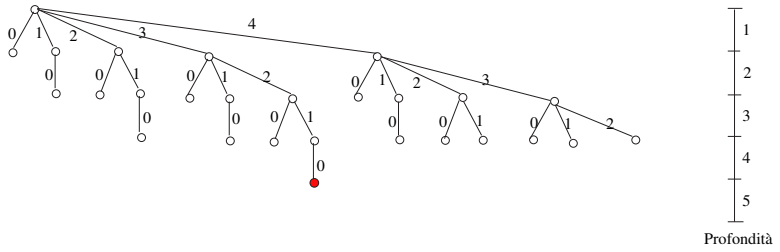


Figura 5.1: Visita del nodo combinazione 3210

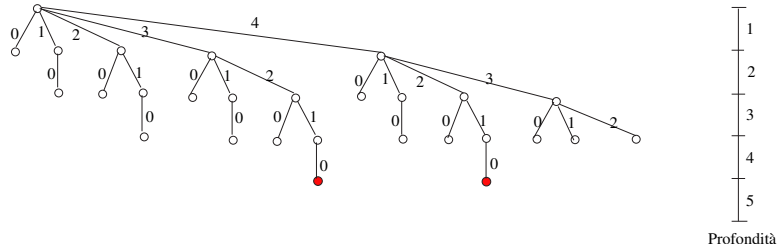


Figura 5.2: Visita del nodo combinazione 4210

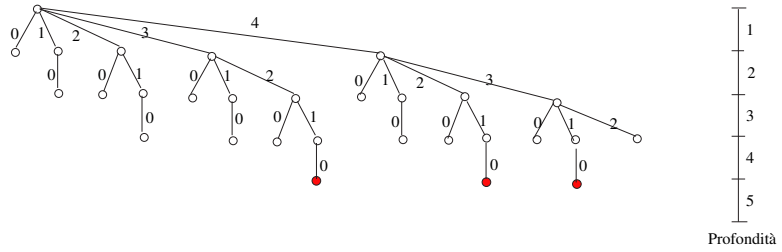


Figura 5.3: Visita del nodo combinazione 4310

L'inferenza di motivi con wildcard si ricava dall'Algoritmo 1 sostituendo la stampa della combinazione c' con la chiamata alla procedura che costruisce l'albero ATW(s, L, c') a partire dall'albero dei suffissi di s troncato all'altezza L con wildcard nelle posizioni in c'. La complessità di questa costruzione è lineare per ogni possibile ATW(s, L, c') considerando l'Algoritmo di Ukkonen visto nel Capitolo 2.

Chiamiamo combinazioni obbiettivo le combinazioni ad altezza $L - k$ dell'albero binomiale B_L. La complessità dell'algoritmo naive è calcolabile considerando il costo per la generazione delle combinazioni visitate prima di generare le combinazioni obbiettivo che è, al caso pessimo, 2^{L-k-1} e il costo per la costruzione di ogni indice ATW per ogni combinazione obbiettivo, che è rappresentato da $n * \binom{L}{k}$. La complessità al caso pessimo risulta quindi $O(2^{L-k-1} + n * \binom{L}{k})$.

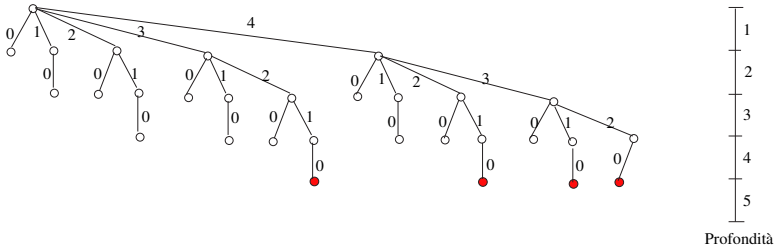


Figura 5.4: Visita del nodo combinazione 4320

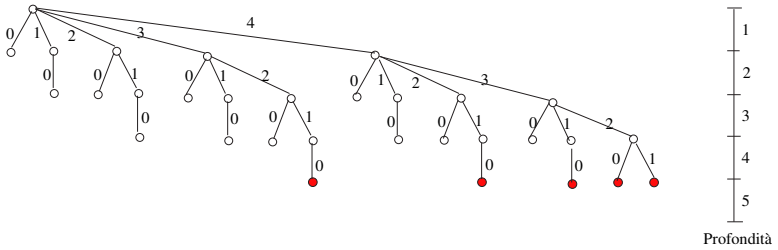


Figura 5.5: Visita del nodo combinazione 4321

5.2 Soluzione con generazione mirata delle combinazioni

Come abbiamo visto, l'Algoritmo 1 visita tutte le che lessicograficamente precedono le combinazioni obbiettivo, questo è fonte di inefficienza perchè ai fini della generazione delle combinazioni della base obbiettivo (insieme delle combinazioni della base per un C_L^{L-k}) non è necessario calcolarle tutte. La soluzione con la generazione mirata propone un algoritmo che già al livello di generazione calcola in maniera mirata le sole combinazioni necessarie per generare ogni combinazione della base obbiettivo. Al livello degli indici, la generazione mirata costruisce solo gli alberi ATW(s, L, l) strettamente necessari. L'idea su cui si basa l'algoritmo di generazione che proponiamo sfrutta la reciproca dipendenza tra gli insiemi combinazione di un insieme C_L^{L-k}. Secondo cui, come sappiamo, tutte le combinazioni possibili sono calcolabili o generabili a partire da un insieme di rappresentanti chiamato

base. Per questo approccio, non è necessario calcolare ex novo un albero ATW per ogni possibile combinazione delle $\binom{L}{k}$, ma solo per le combinazioni della base di C_L^{L-k}.

Il Lemma 10 ci garantisce che per ogni combinazione di tipo B esiste una combinazione di tipo A nella base con cui la combinazione di tipo B è in reciproca dipendenza.

L'Algoritmo 2 genera le combinazioni della base obbiettivo in B_L senza visitare tutte le $\sum_{i=1}^{L-k}\binom{L}{i}$ possibili combinazioni, ma esaminando solo un sottoinsieme di combinazioni di B_L. Nello specifico tutte le combinazioni che condividono, con uno dei nodi combinazione della base obbiettivo, una parte del loro percorso .

A partire da una combinazione $c = c_L \cdots c_i$, l'Algoritmo 2 esamina solo le combinazioni che hanno il componente $c_{i-1} \geq d$ dove $d = L - k - 1 - |c|$ ($|c|$ è la lunghezza della combinazione c). Il valore d così calcolato esclude fin dalla prima iterazione tutte le combinazioni corrispondenti a sottoalberi di B_L che non raggiungono la profondità $L - k$ della base obbiettivo.

L'Algoritmo 2 che genera lessicograficamente le combinazioni obbiettivo è mostrato di seguito.

L'Algoritmo 2 usa una coda per raccoglie le combinazioni man mano esaminate. Ad ogni iterazione del ciclo while, si estrae una combinazione dalla coda, se la lunghezza della combinazione estratta dalla coda è maggiore della profondità attuale della generazione, allora significa che si è appena iniziata la generazione per un nuovo livello dell'albero binomiale, quindi un nuovo componente deve essere aggiunto alle nuove combinazioni. Il componente delle nuove combinazioni ha un valore che può andare dal valore del componente precedente meno 1 fino al valore di d. Ogni nuova combinazione generata in questo modo viene inserita nella coda e così via fino a che la

Algorithm 2 Algoritmo per la generazione mirata delle combinazioni

```
1: procedure GenRidCombination(L, k)
2:     t ← L
3:     c_t ← L − 1
4:     coda ← push(c)                          ▷ coda è una normale coda FIFO
5:     pa ← 0                                  ▷ pa rappresenta la profondità attuale
6:     while coda ≠ Null do
7:         c ← pop(coda)    ▷ push e pop sono le operazioni di inserzione e
   estrazione dalla coda
8:         if |c| > pa then
9:             pa ← |c|
10:            t ← t − 1
11:            d ← L − k − 1 − |c|
12:        end if                                ▷ |c| è la lunghezza di c
13:        N ← c_{t+1} − 1
14:        for j ← d, N do
15:            c' ← c
16:            c'_t ← j
17:            if |c'| = L − k then
18:                Stampa c'
19:            else if |c'| < L − k then
20:                coda ← push(c')
21:            end if
22:        end for
23:    end while
24: end procedure
```

profondità attuale non è superiore all'altezza della base obbiettivo.

Il numero di combinazioni della base per quanto dimostrato nella Formula 9 è $\binom{L-1}{i-1}$. Il numero di combinazioni visitate è stabilito dalla congettura $\binom{L}{k+1}$. $\binom{L}{k+1}$ è per noi una congettura che lasciamo come lavoro futuro.

L'ordine di generazione delle combinazioni è mostrato in dettaglio nell'Esempio 21, in cui è indicata la sequenza di nodi combinazione dell'albero binomiale B_5 visitati con l'Algoritmo 2. Ogni figura mostra anche i nodi già generati al momento della generazione della combinazione obbiettivo a cui si riferisce la figura stessa.

L'inferenza di motivi con wildcard si ricava dall'Algoritmo 2 sostituendo la stampa della combinazione c' con la chiamata alla procedura che costruisce l'albero ATW(s, L, c'). La procedura costruisce un ATW(s, L, c') a partire dall'albero dei suffissi di s troncato all'altezza L con wildcard nelle posizioni in c'. La complessità di questa costruzione è lineare per ogni possibile ATW(s, L, c') considerando l'Algoritmo di Ukkonen visto nel Capitolo 2. La costruzione è la stessa vista nella Sezione 5.1.

La complessità dell'algoritmo di inferenza di motivi approssimati con la generazione mirata è calcolabile considerando il costo per la generazione delle combinazioni visitate prima di generare le combinazioni obbiettivo che è, al caso pessimo, $\binom{L}{k+1} - \binom{L-1}{k-1}$ e il costo per la costruzione di ogni ATW per ogni combinazione della base obbiettivo che è $n * \binom{L-1}{k-1}$. La complessità al caso pessimo è quindi $O((\binom{L}{k+1} - \binom{L-1}{k-1})) + n * \binom{L-1}{k-1}))$.

5.3 Soluzione per raffinamento

Rispetto alla soluzione con la generazione mirata delle combinazioni, la soluzione con raffinamento introduce due novità: la prima, consiste nella generazione di tutte le $\binom{L}{k+1}$ combinazioni e non solo di quelle della base

obbiettivo, la seconda, consiste nella nuova costruzione incrementale degli indici ATW.

5.3.1 Metodo alternativo per la generazione mirata

In letteratura, esistono molti algoritmi per la generazione delle combinazioni. In [Knu98] sono trattati i principali approcci. Per quanto ci riguarda, l'algoritmo che proponiamo adesso è diverso da ogni altro algoritmo. Come già accennato nel capitolo precedente, il metodo alternativo costruisce incrementalmente le combinazioni che fanno parte della base obbiettivo. La generazione non è di tipo lessicografica come nella soluzione naive, è invece, basata su un metodo che sviluppa la generazione vista nel Lemma 11, secondo cui ogni combinazione di tipo A di un livello i ha una combinazione di tipo B del livello $i-1$ che la genera aggiungendo un componente. Il processo non genera le combinazioni di tipo A di livello i se prima non ha generato le opportune combinazioni del livello $i-1$.

Entriamo nel dettaglio della soluzione mostrando l'Algoritmo 3 di generazione alternativa delle combinazioni.

L'Algoritmo 3 usa una coda per raccoglie le combinazioni man mano esaminate. Ad ogni iterazione del ciclo while, si estrae una combinazione c dalla coda, se la lunghezza di c è maggiore della profondità attuale della generazione, allora significa che si è appena iniziata la generazione per un nuovo livello dell'albero binomiale. Le nuove combinazioni c' generate da c sono esattamente tutte le combinazioni di tipo B in relazione $\mathcal{R}_d$ con c, per dei valori d calcolati opportunamente $(c_t - (L - k - 1 - |c|))$, con un nuovo componente $c'_L = L - 1$. Ogni nuova combinazione c' generata viene inserita nella coda e si ripete il ciclo fino a che la profondità attuale non diviene superiore all'altezza della base obbiettivo.

Algorithm 3 Algoritmo per la generazione alternativa delle combinazioni

```
1: procedure GenALLRidCombination(L, k)
2:     t ← L
3:     c_t ← L − 1
4:     Stampa c
5:     coda ← push(c)                          ▷ coda è una normale coda FIFO
6:     pa ← 0
7:     while coda ≠ Null do
8:         c ← pop(coda)    ▷ push e pop sono le operazioni di inserzione e
   estrazione dalla coda
9:         if |c| > pa then
10:            pa ← |c|
11:            t ← t − 1
12:            r ← L − k − 1 − |c|
13:        end if
14:        if |c| = pa then
15:            N ← c_t − r    ▷ N conta il numero di combinazioni di tipo B
   generabili da questa combinazione
16:        end if
17:        for d ← N, 1 do
18:            for j ← L, |c| do
19:                c'_{j−1} ← c_j − d
20:            end for
21:            c'_L ← L − 1
22:            if |c'| = L − k then
23:                Stampa c'
24:            else if |c'| < L − k then
25:                Stampa c'
26:                coda ← push(c')
27:            end if
28:        end for
29:    end while
30: end procedure
```

Il numero di combinazioni generate ed esaminate dall'Algoritmo 3 è il valore binomiale $\binom{L}{k+1}$.

L'ordine di generazione delle combinazioni è mostrato in dettaglio nell'Esempio 22, che mostra la sequenza di nodi combinazione dell'albero binomiale B_5 visitati con l'Algoritmo 3. Ogni figura mostra anche i nodi già generati al momento della generazione della combinazione obbiettivo a cui si riferisce la figura stessa.

L'algoritmo di inferenza sostituisce la stampa delle combinazioni nell'Algoritmo 3 con la chiamata alla procedura che calcola l'indice ATW corrispondente alla combinazione esaminata. Ad ogni chiamata il metodo con raffinamento divide (se può) ogni insieme di occorrenze OCC dell'albero ATW passato come parametro. Voglio una precisazione algoritmica. Ai fini dell'inferenza dei motivi gli indici ATW come alberi non sono così importanti, perchè le informazioni sulle ripetizioni dei motivi sono mantenute negli insiemi di occorrenze di suffissi associati alle foglie, per questo motivo l'algoritmo di costruzione degli indici ATW considera solo gli insiemi associati alle foglie.

Gli effetti possibili del raffinamento su un insieme di occorrenze dell'indice ATW di partenza sono due: o l'insieme esaminato non viene raffinato, in tal caso rimane invariato, oppure viene raffinato e si divide in sottoinsiemi. Adesso vediamo l'algoritmo. L'Algoritmo 4 ha come input l'albero ATW di partenza, il quorum q, l'insieme combinazione c' per il nuovo indice ATW in costruzione, il valore d visto per l'Algoritmo 2, e la sequenza s. Il quorum ha una funzione molto importante perchè quando la cardinalità di un insieme di indici di suffissi diventa inferiore al valore del quorum, allora l'insieme viene eliminato. Così facendo, gli insiemi che non rappresentano ripetizioni interessanti, vengono esclusi immediatamente. Il processo di raffinamento ed esclusione degli insiemi di indici di suffissi rappresenta il fulcro

dell'Algoritmo 4. Assumiamo per semplicità che l'alfabeto di s sia Σ_{DNA}.

Ad ogni iterazione l'Algoritmo 4 considera un insieme di occorrenze dell'albero ATWp, legge tutti gli indici di suffissi dell'insieme inserendo man mano l'indice letto in un insieme, associato ad una particolare lettera dell'alfabeto della sequenza s, a seconda del carattere in s corrispondente a l'indice esaminato. Questo passaggio rappresenta il processo di divisione degli indici e quindi il raffinamento. Una volta letti tutti gli indici di suffissi di un insieme si inseriscono i nuovi insiemi in un indice ATW'. Alla fine della procedura si ritorna l'indice ATW' con tutti gli insiemi raffinati.

La complessità al caso pessimo per l'Algoritmo 4 è $O(n)$ a causa della lettura di tutti gli indici dei suffissi. Se l'operazione di eliminazioni degli insiemi non è mai avvenuta, allora la costruzione di questo ATW ha un costo paragonabile a quello che è stato necessario per costruire i suoi predecessori. Se per effetto del raffinamento l'eliminazione di uno o più insiemi invece viene effettuata, allora tali insiemi e tutti i loro indici non saranno considerati nelle prossime costruzioni che saranno generati a partire dal sopra menzionato ATW. Il motivo di questo è, se un insieme non ha soddisfatto il quorum q, allora tale insieme di occorrenze viene diviso in sottoinsiemi di cardinalità minore, quindi il quorum non sarà soddisfatto neanche per questi insiemi. Eliminare un insieme di occorrenze che non soddisfa il quorum è necessario.

La complessità dell'algoritmo di inferenza di motivi approssimati è costituito dal costo per la generazione delle combinazioni, che come abbiamo visto è $\binom{L}{k+1}$ e dal costo per costruire un albero ATW per ogni combinazione che è n. Quindi la complessità al caso pessimo è $O(n * \binom{L}{k+1})$.

Algorithm 4 Algoritmo di raffinamento

```
1: procedure ATWBuilder(ATWp, q, c', d, s)
2:     OCC(ATW') ← {}
3:     A ← {}
4:     C ← {}
5:     T ← {}
6:     G ← {}
7:     for all I ∈ OCC(ATWp) do
8:         for all j ∈ I do
9:             if s(j − d) = a then
10:                A ← Put(j-d)
11:            else if s(j − d) = c then
12:                C ← Put(j-d)
13:            else if s(j − d) = t then
14:                T ← Put(j-d)
15:            else if s(j − d) = g then
16:                G ← Put(j-d)
17:            end if
18:        end for
19:        if |A| < q then
20:            remove(A)
21:        else
22:            ATW' ← Put(A)
23:        end if
24:        if |C| < q then
25:            remove(c)
26:        else
27:            OCC(ATW') ← Put(C)
28:        end if
29:        if |T| < q then
30:            remove(T)
31:        else
32:            OCC(ATW') ← Put(T)
33:        end if
34:        if |G| < q then
35:            remove(G)
36:        else
37:            OCC(ATW') ← Put(G)
38:        end if
39:        A ← {}
40:        C ← {}
41:        T ← {}
42:        G ← {}
43:    end for
44:    returnATW'
45: end procedure
```

5.4 Analisi degli algoritmi proposti e confronto

Il caso pessimo per il raffinamento si ha quando ogni indice ATW corrispondente ad una combinazione della base obbiettivo ha ogni insieme di occorrenze nelle foglie con una cardinalità maggiore o uguale al quorum q. Il numero di foglie per ogni indice ATW, in questo caso, non supera mai $n_q = \frac{n}{q}$. Il valore $n_q < n$ per ogni $q > 1$ e $n_q = n$ se il quorum $q = 1$. Noi ci aspettiamo che, in casi reali, all'aumentare di q, il valore n_q decresca significativamente.

Volendo avere, però, una valutazione delle prestazioni al caso pessimo, confrontiamo i risultati dei tre algoritmi visti in questo capitolo. L'Algoritmo 1 ha una complessità dell'ordine di $O(2^{L-k-1} + n_q * \binom{L}{k})$, l'Algoritmo 2 ha una complessità dell'ordine di $O((\binom{L}{k+1} - \binom{L-1}{k-1}) + n_q * \binom{L-1}{k-1})$ e l'Algoritmo 3 ha una complessità dell'ordine di $O(n_q * \binom{L}{k+1})$. Dal punto di vista della discussione tecnica, il coefficiente binomiale $\binom{L}{k+1}$ si può riscrivere come:

$\binom{L}{k+1} = \frac{L!}{(k+1)!*(L-k-1)!} = \frac{(L-k)*L}{(k+1)*k} * \binom{L-1}{k-1}$

Analizziamo i casi sul valore di k. Confrontiamo prima i due risultati principali rappresentati dagli Algoritmi 2 e 3. Se $k \leq \frac{L}{2}$ é preferibile usare l'Algoritmo 2 perchè il fattore $\frac{(L-k)*L}{(k+1)*k}$ è maggiore di 1 quindi il coefficiente binomiale $\binom{L}{k+1}$ dell'Algoritmo 3 è maggiore del coefficiente $\binom{L-1}{k-1}$ dell'Algoritmo 2. Se $k > \frac{L}{2}$ è preferibile usare, invece, l'Algoritmo 3 perchè $\frac{(L-k)*L}{(k+1)*k}$ è minore di 1 quindi, il coefficiente binomiale $\binom{L}{k+1}$ dell'Algoritmo 3 è minore del coefficiente $\binom{L-1}{k-1}$ dell'Algoritmo 2.

Passiamo all'altro confronto tra i due risultati principali e l'Algoritmo naive. Nel caso $k \leq \frac{L}{2}$, il valore del coefficiente binomiale $\binom{L}{k}$ è sempre maggiore del coefficiente $\binom{L-1}{k-1}$ dell'Algoritmo 2, quindi è preferibile l'Algoritmo 2. Nel caso in cui $k > \frac{L}{2}$, il valore del coefficiente binomiale $\binom{L}{k}$ è maggiore del coefficiente $\binom{L}{k+1}$ perchè il valore $\frac{L-k}{k+1}$ è minore di 1 per qualsiasi valore di $k > \frac{L-1}{2}$ quindi anche di $\frac{L}{2}$. È anche in questo caso preferibile

il risultato dell'Algoritmo 3.

Capitolo 6

Conclusioni e lavoro futuro

Il problema dell'inferenza di motivi di lunghezza fissa con wild card trova applicazione nella ricerca, in sequenze biologiche, dei binding site e dei fattori di trascrizione dei geni. Date le dimensioni dei dati di ingresso in problemi di questo tipo, è apparso molto promettente un approccio basato sulla costruzione efficiente degli indici. Infatti, con la risoluzione proposta dal raffinamento, all'aumentare del quorum q il costo della costruzione di ogni singolo indice è diminuito significativamente. Il quorum ha rappresentato in ogni caso un parametro determinante per la risoluzione efficiente dell'inferenza di motivi con wildcard. Il principale sviluppo futuro di questa tesi riguarderà senza dubbio la ricerca di una variante efficiente per l'inferenza di motivi di lunghezza variabile con errori, che rimane tutt'oggi una delle principali sfide scientifiche in campo biotecnologico, insieme all'inferenza di motivi con wildcard.

Appendice A

Gli alberi ATW(s, L, l) mostrati di seguito rappresentano gli alberi relativi a tutte le possibili combinazioni di errore per l'albero binomiale in Figura 3.2.

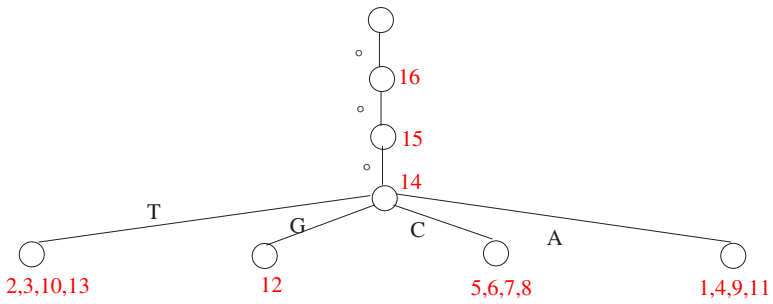


Figura A.1: ATW$(s, 4, \{0\})$

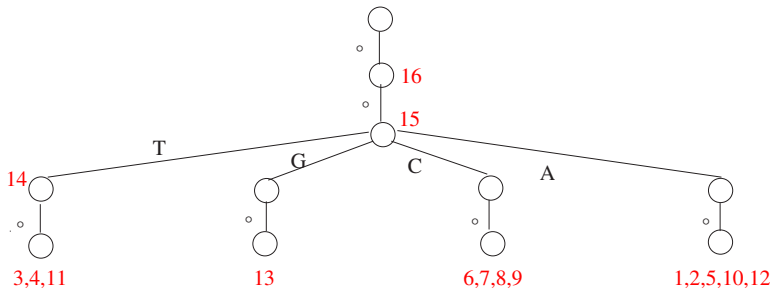


Figura A.2: ATW$(s, 4, \{1\})$

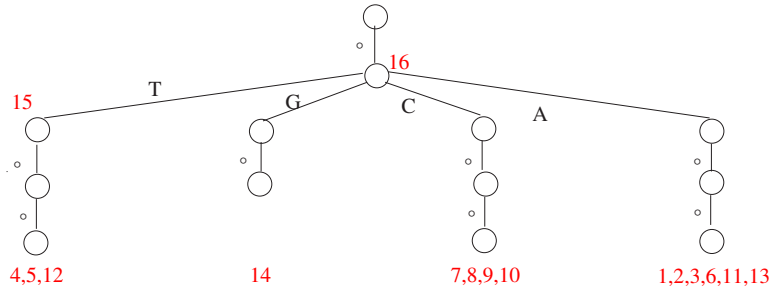


Figura A.3: ATW$(s, 4, \{2\})$

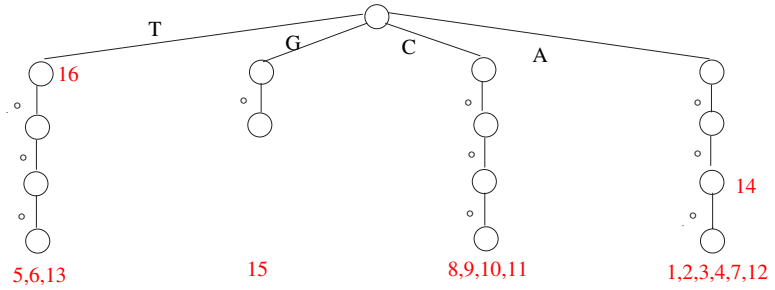


Figura A.4: ATW$(s, 4, \{3\})$

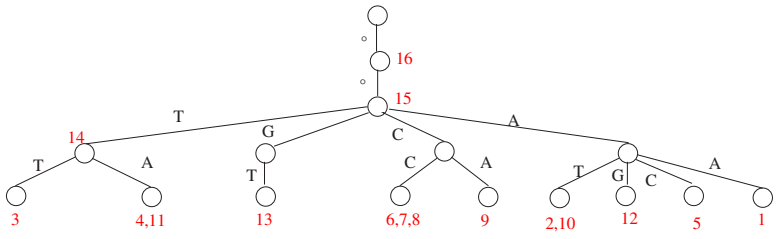


Figura A.5: ATW$(s, 4, \{1, 0\})$

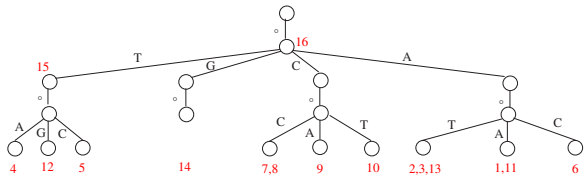


Figura A.6: ATW($s, 4, \{2, 0\}$)

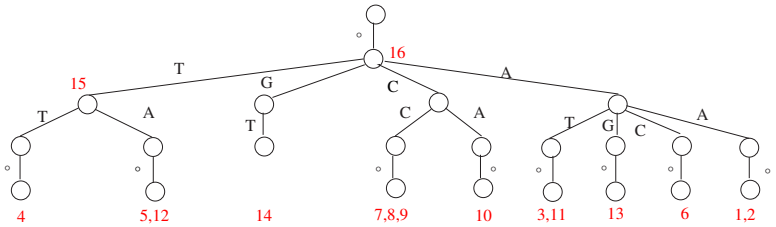


Figura A.7: ATW($s, 4, \{2, 1\}$)

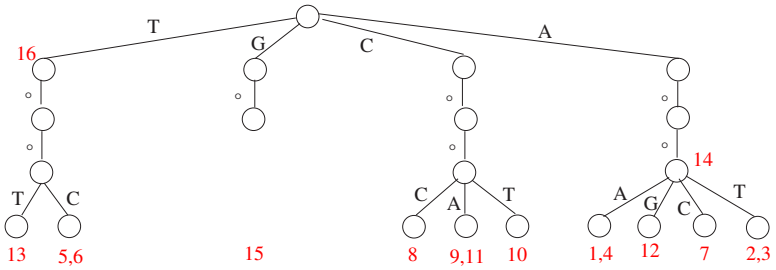


Figura A.8: ATW($s, 4, \{3, 0\}$)

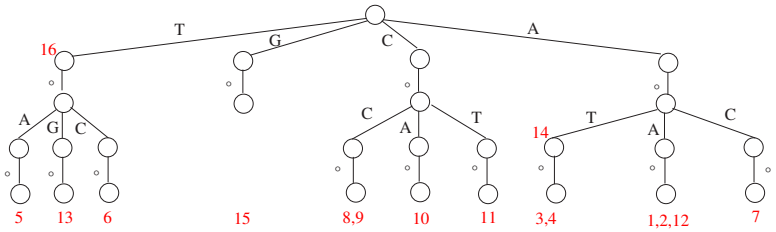


Figura A.9: ATW($s, 4, \{3, 1\}$)

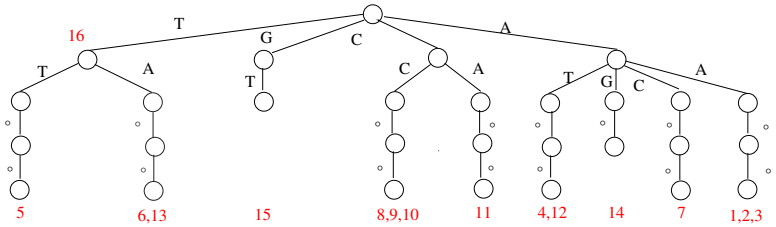


Figura A.10: ATW(s,4,$\{3, 2\}$)

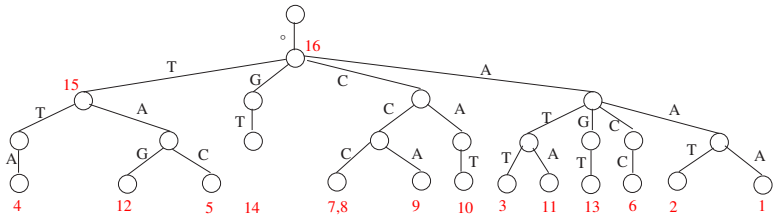


Figura A.11: ATW$(s, 4, \{2, 1, 0\})$

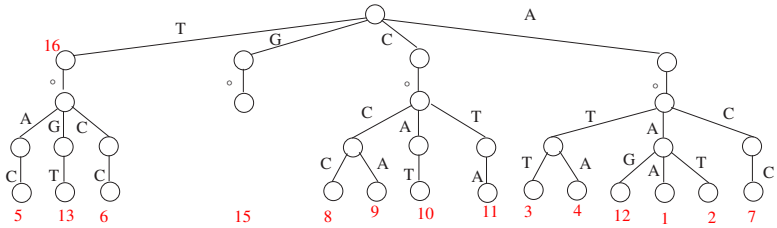


Figura A.12: ATW$(s, 4, \{3, 1, 0\})$

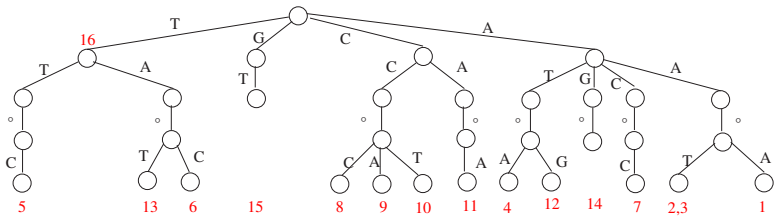


Figura A.13: ATW$(s, 4, \{3, 2, 0\})$

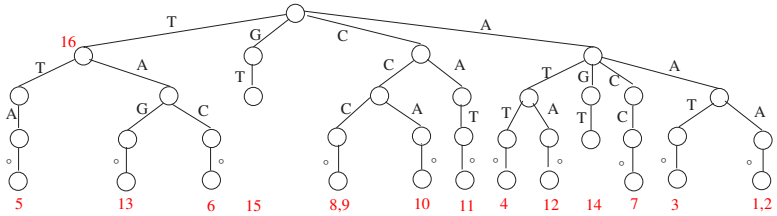


Figura A.14: ATW$(s, 4, \{3, 2, 1\})$

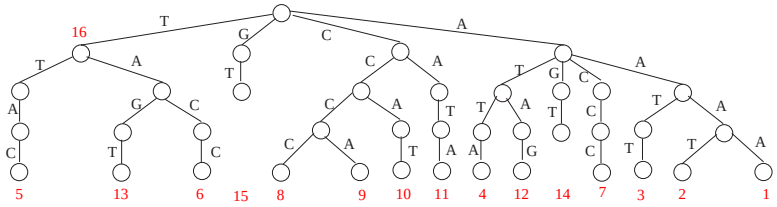


Figura A.15: ATW$(s, 4, \{3, 2, 1, 0\})$

Appendice B

Gli alberi ATW(s, L, l) mostrano la sequenza di combinazioni generate con l'Algoritmo 2.

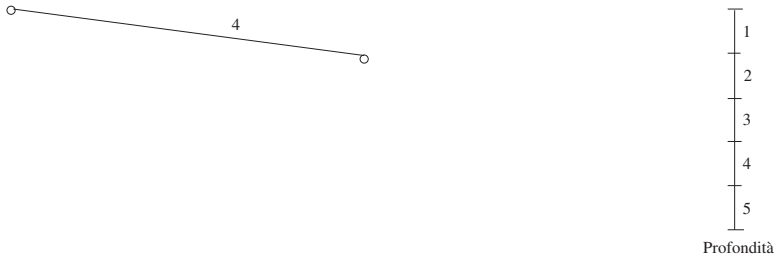


Figura B.1: Generazione del nodo combinazione 4

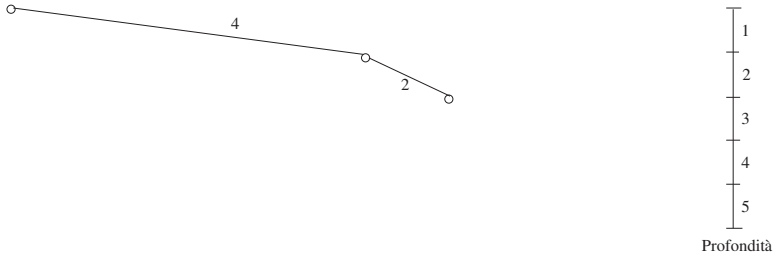


Figura B.2: Generazione del nodo combinazione 42

Esempio 21.

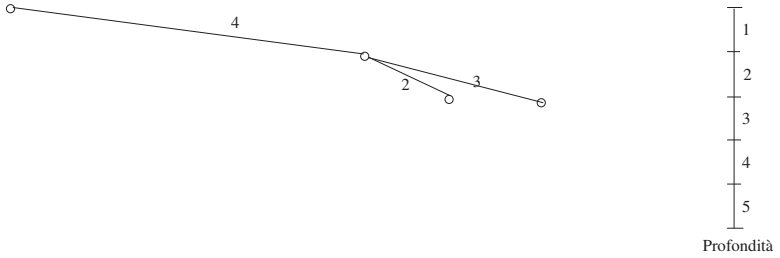


Figura B.3: Generazione del nodo combinazione 43

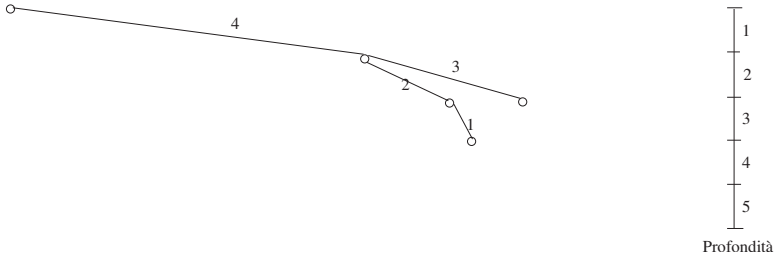


Figura B.4: Generazione del nodo combinazione 421

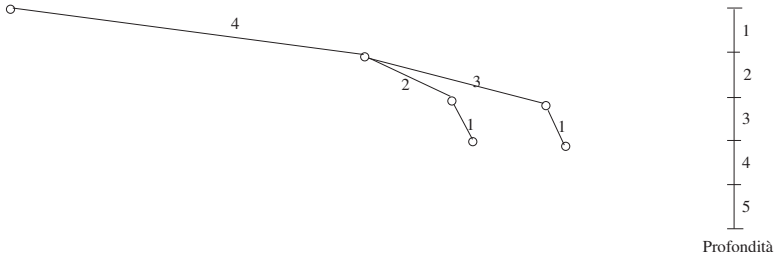


Figura B.5: Generazione del nodo combinazione 431

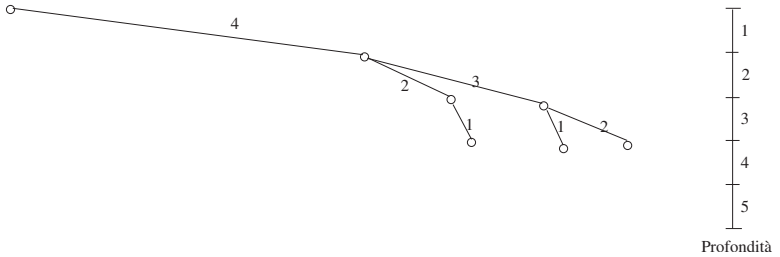


Figura B.6: Generazione del nodo combinazione 432

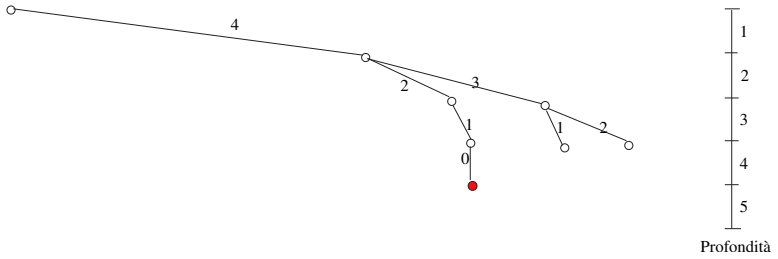


Figura B.7: Generazione del nodo combinazione 4210

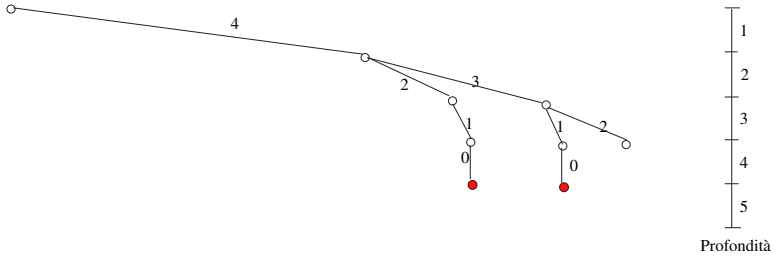


Figura B.8: Generazione del nodo combinazione 4310

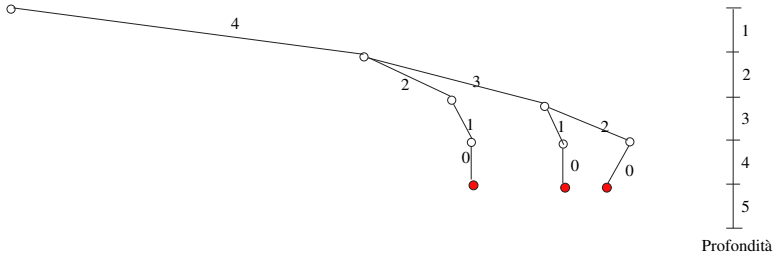


Figura B.9: Generazione del nodo combinazione 4320

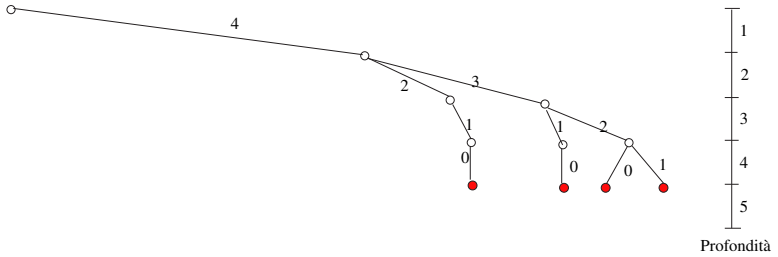


Figura B.10: Generazione del nodo combinazione 4321

Appendice C

Gli alberi ATW(s, L, l) mostrano la sequenza di combinazioni generate con l'Algoritmo 3.

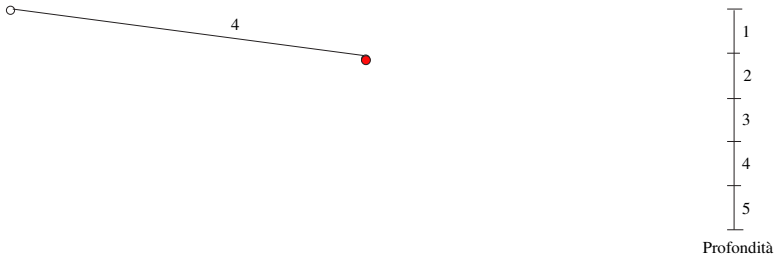


Figura C.1: Generazione del nodo combinazione 4

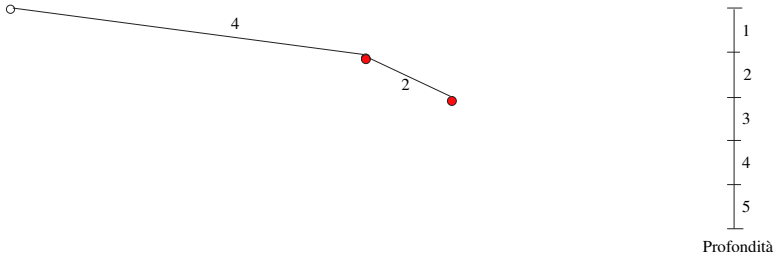


Figura C.2: Generazione del nodo combinazione 42

Esempio 22.

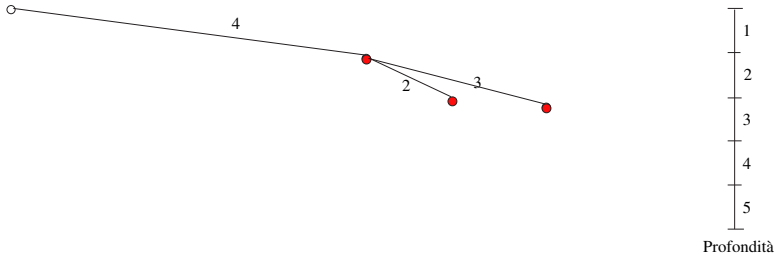


Figura C.3: Generazione del nodo combinazione 43

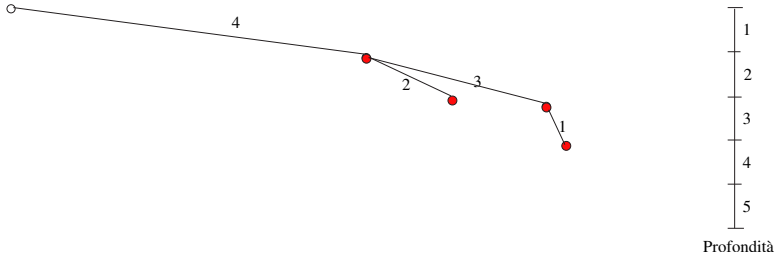


Figura C.4: Generazione del nodo combinazione 431

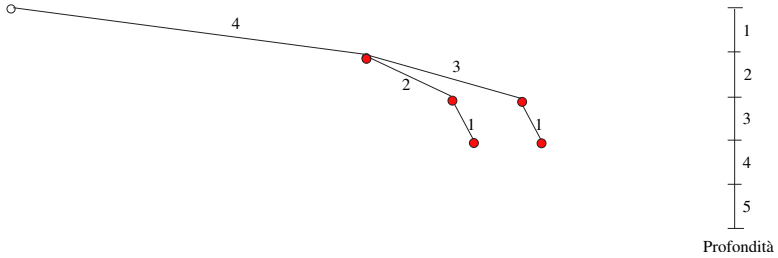


Figura C.5: Generazione del nodo combinazione 421

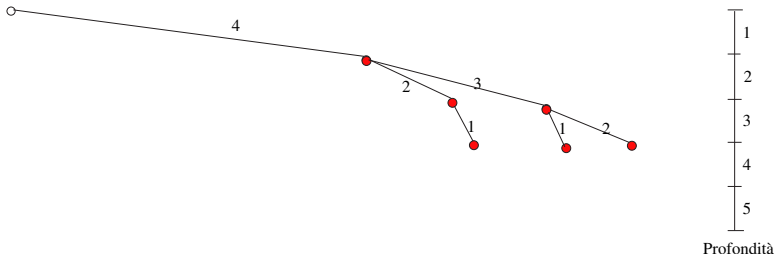


Figura C.6: Generazione del nodo combinazione 432

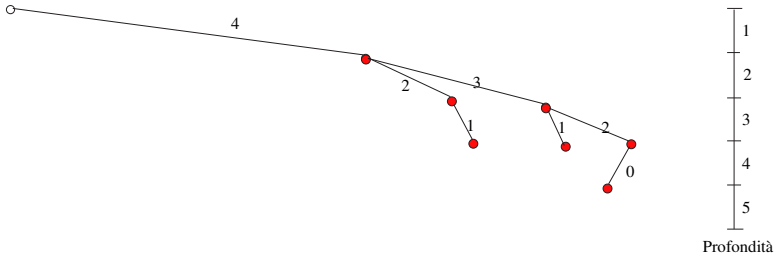


Figura C.7: Generazione del nodo combinazione 4320

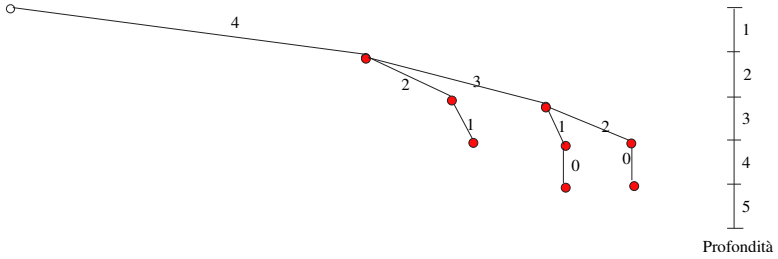


Figura C.8: Generazione del nodo combinazione 4310

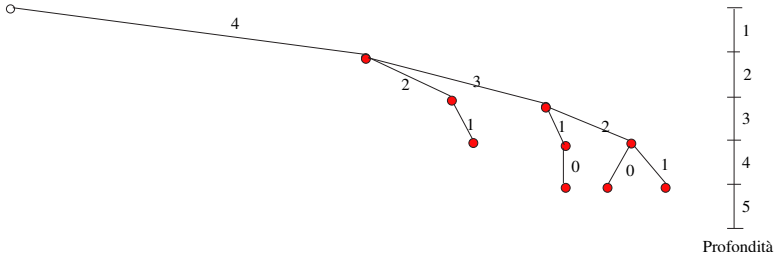


Figura C.9: Generazione del nodo combinazione 4321

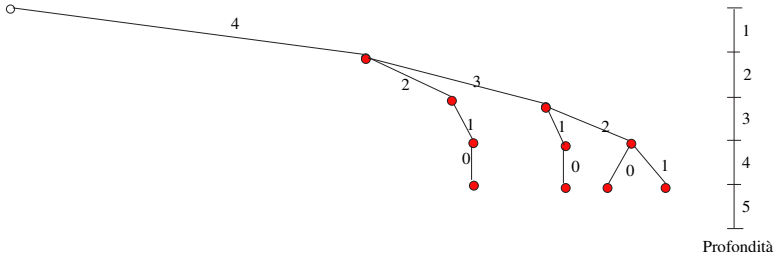


Figura C.10: Generazione del nodo combinazione 4210

Bibliografia

[ABG98] I. Eidhammer A. Brazma, I. Jonassen and D. Gilbert. Approaches to the automatic discovery of patterns in biosequences. *Journal of Computational Biology*, 5:278–305, 1998.

[AC01] M. Gromov A. Carbone. Mathematical slices of molecular biology. *La Gazette des Mathematiciens*, (88):11–80, 2001.

[BE94] Timothy L. Bailey and Charles Elkan. Fitting a mixture model by expectation maximization to discover motifs in biopolymers. In *Proceedings of the Second International Conference on Intelligent Systems for Molecular Biology*, pages 28–36. AAAI Press, 1994.

[Bet] Priscilla Bettini. http://www.dbag.unifi.it/didattica/sngenetica/mutazione.pdf. Internet site.

[Cor90] Leiserson C. E. Rivest Cormen, T. H. *Introduction to Algorithms*. 1990.

[DGW85] M. Eggert D.J. Galas and M.S. Waterman. Rigorous pattern recognition methods for dna sequences. *Journal of Molecular Biology*, (186):117–128, 1985.

[DH02] R. Dannenberg and N. Hu. Pattern discovery techniques for music audio, 2002.

[E.M76] E.M.McCreight. A Space-Economical Suffix Tree Construction algorithm. *Journal of the ACM*, 23(2):262–272, 1976.

[Esk02] Pevzner Pavel A. Eskin, Eleazar. Finding composite regulatory patterns in dna sequences. *Bioinformatics*, 18(10):354–363, 2002.

[Gie02] Robert Giegerich. Sequence, Similarity and Dynamic Programming. Lecture Notes Bioinformatics Summerschool, Bad Urach, 2002.

[GK97] Robert Giegerich and Stefan Kurtz. From Ukkonen to McCreight and Weiner: A unifying view of linear-time suffix tree construction. *Algorithmica*, 19(3):331–353, 1997.

[GLM02] Richard Groult, Martine Lonard, and Laurent Mouchard. Evolutive tandem repeats using hamming distance. In *MFCS '02: Proceedings of the 27th International Symposium on Mathematical Foundations of Computer Science*, pages 292–304, London, UK, 2002. Springer-Verlag.

[Gus97] D. Gusfield. *Algorithms on Strings, Trees and Sequences.* Cambridge University Press, 1997.

[Ham50] Richard W. Hamming. Error-detecting and error-correcting codes. *Bell System Technical Journal*, 29(2):147–160, 1950.

[JDW05] S P Bell A. Gann M. Levine R. Losick J. D. Watson, T.A. Baker. *Biologia molecolare del gene.* 2005.

[KMR72] Richard M. Karp, Raymond E. Miller, and Arnold L. Rosenberg. Rapid identification of repeated patterns in strings, trees and arrays. In *STOC '72: Proceedings of the fourth annual ACM symposium on Theory of computing*, pages 125–136, New York, NY, USA, 1972. ACM Press.

[Knu98] Donald E. Knuth. *The art of computer programming, volume 3: (2nd ed.) sorting and searching.* Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1998.

[Lev66] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. Technical Report 8, 1966.

[LW93] Altschul S.F. Bogouski M.S. Liu J.S. Neuwald A.F. Lawrence, C.E. and J.C. Wooten. Detecting subtle sequence signals: A gibbs sampling strategy for multiple alignment. *Science*, (262):208–214, 1993.

[MS00] L. Marsan and M.-F. Sagot. Algorithms for extracting structured motifs using a suffix tree with application to promoter and regulatory site consensus identification. *J. Comput. Bio.*, 7(3/4):345–360, 2000.

[MWG84] R. Arratia M. Waterman and D. Galas. Pattern recognition in several sequences: consensus and alignment. *Bulletin of mathematical biology*, 46:515–527, 1984.

[Pro] www.ceebt.embo.org/projects/project18/material/project18it.pdf. Internet site.

[PS00] P. A. Pevzner and S.-H. Sze. Combinatorial Approaches to Finding Subtle Signals in DNA sequences. In *ProceedingsInterna-*

tional Conference on Intelligent Systems for Molecular Biology, pages 269–278. AAAI Press, 2000.

[RKK03] Ghizlane Bana Roman Kolpakov and Gregory Kucherov. efficient and flexible detection of tandem repeats in dna. *Nucleic Acids Research*, 31(13):3672–3678, 2003.

[Sag98] Marie-France Sagot. Spelling approximate repeated or common motifs using a suffix tree. *Lecture Notes in Computer Science*, 1380:374, 1998.

[SM98] Marie-France Sagot and Eugene W. Myers. Identifying satellites and periodic repetitions in biological sequences. *Journal of Computational Biology*, 5(3):539–554, 1998.

[Sto00] Gary D. Stormo. DNA, Binding Sites: Representation and Discovery. *Bioinformatics*, 16(1):16–23, 2000.

[SVC95] Henri Soldano, Alain Viari, and Marc Champesme. Searching for flexible repeated patterns using a non-transitive similarity relation. *Pattern Recogn. Lett.*, 16(3):233–246, 1995.

[SW] Marie-France Sagot and Yoshiko Wakabayashi. Pattern inference under many guises.

[Tom99] Martin Tompa. An exact method for finding short motifs in sequences, with application to the ribosome binding site problem. In *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology*, pages 262–271. AAAI Press, 1999.

[Ukk95] Esko Ukkonen. On-Line Construction of Suffix Trees. *Algorithmica*, 14(3):249–260, 1995.

[WYKG04] Ydo Wexler, Zohar Yakhini, Yechezkel Kashi, and Dan Geiger. Finding approximate tandem repeats in genomic sequences. In *RECOMB '04: Proceedings of the eighth annual international conference on Resaerch in computational molecular biology*, pages 223–232, New York, NY, USA, 2004. ACM Press.

[ZL77] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. 23:337–343, 1977.