Università degli Studi di Milano - Bicocca
Scuola di Scienze
Dipartimento di Informatica, Sistemistica e Comunicazione
Corso di laurea in Informatica

Inferenza di alberi tumorali tramite Particle Swarm Optimization

Relatore: Prof. Della Vedova Gianluca
Correlatore: Dott. Ciccolella Simone

Relazione della prova finale di:
Castro Tenemaya Adrian David
Matricola 816015

Anno Accademico 2016-2019

Indice

Premessa e ringraziamenti

Il presente lavoro è frutto del lavoro svolto come tirocinio all'interno dell'Università di Milano-Bicocca, e viene anche utilizzato come tesi finale ai fini del conseguimento della laurea in Informatica. È però necessario chiarire che il progetto in questione non sarà abbandonato nè una volta terminata la stesura di questa relazione, nè dopo il conseguimento della laurea. È mia intenzione contribuire al meglio delle mie possibilità in quello che ritengo essere uno dei campi con il quale mi sento più legato, sia a livello di interesse professionale, che a livello strettamente personale: la ricerca sul cancro. Secondo il *National Cancer Institute*, nel 2012 sono stati riportati 14.1 *milioni* di nuovi casi, e di questi, 8.2 *milioni* hanno portato alla morte [1]. I dati mostrano anche quelli che può sembrare all'apparenza una realtà discordante: il numero totale di morti per cancro è in crescita, ma il rapporto delle morti per individuo sta calando [10]. Nel 1990, 161 persone su 100.000 nel mondo sono morte a causa del cancro. Nel 2016, questo numero è calato a 134 su 100.000. Questo miglioramento è dovuto indubbiamente ad un numero molto elevato di fattori, tra cui l'aumento della qualità di vita ed un migliore sistema sanitario, ma è anche grazie alla crescita incessante della ricerca sul cancro, ed ai campi sui quali essa si appoggia. Lo sviluppo di algoritmi sempre più efficienti e performanti, e l'utilizzo di calcolatori super-veloci, ha permesso a questo settore di ricerca di ottenere dei considerevoli risultati.

Con questo progetto spero, quindi, di aver dato un contributo in questo settore, anche se in una percentuale minuscola.

Vorrei ringraziare mia mamma **Laura**, mio padre **José**, mia sorella **Valeria**, i miei fantastici nonni, le bellissime e meravigliose persone che hanno contribuito, in maniera diretta ed indiretta, a farmi appassionare all'informatica e, in questo caso, alla bioinformatica. Un importante ringraziamento va ad Erica e Valentina per avermi aiutato nella correzione del primo capitolo. Uno speciale ringraziamento va ad Iris.

Prefazione

Il presente lavoro è stato svolto sotto la guida ed il supporto di AlgoLab, laboratorio presso il dipartimento di informatica dell'Università di Milano-Bicocca, che ha lo scopo di progettare, studiare, analizzare ed implementare algoritmi efficienti per problemi computazionali. Il tirocinio è cominciato il 22 Marzo 2019, ed è stato condotto per la maggior parte in maniera autonoma, da remoto. Il problema affrontato è l'*inferenza di progressioni tumorali* su dati single-cell, al fine di determinare l'ordine e la frequenza con cui le variazioni somatiche vengono acquisite durante una progressione tumorale. Spesso ciò è basato sulla "Infinite Sites Assumption", dove le mutazioni possono solo essere acquisite, e mai perse. Lo stage si colloca nella ricerca del superamento di tale assunzione, utilizzando il modello della *filogenesi persistente*, dove ogni mutazione può essere persa al massimo una volta nell'intero albero. Più precisamente, si è investigata la tecnica *Particle Swarm Optimization*, un algoritmo di ottimizzazione di tipo euristico, ispirato al movimento degli sciami. I dati single-cell sono caratterizzati da un elevato tasso di errore e di valori mancanti: ciò rende inutilizzabili gli approcci noti in letteratura per i dati di *bulk sequencing*. In particolare, sono state analizzate quali strutture dati utilizzare per rendere l'algoritmo efficiente ed efficace, e quali operazioni considerare per inferire predizioni accurate.

Abstract

Al fine di ricostruire gli alberi filogenetici tumorali, negli ultimi anni si è fatto uso del modello *infinite-sites*, ipotizzando le progressioni tumorali come accumulazioni di mutazioni. Recenti studi che sfruttano la sequenziazione *single-cell* mostrano, evidenziando la presenza di perdite di mutazioni, come questa assunzione non si riveli sempre vera. La presenza di strumenti che possano fare inferenze sulle filogenesi di alberi genetici con perdite di mutazioni è però limitata.

In questo lavoro viene illustrato ed analizzato un nuovo strumento di analisi per l'inferenza di progressioni tumorali tramite *particle swarm optimization*.

Capitolo 1

Introduzione

1.1 Descrizione

Il cancro è la seconda causa più comune di morte [10], arrivando nel 2017 a contare il 17.08% delle morti nel mondo, per un totale di 8.93 *milioni* di decessi.

1.2 Storia

Era il 1869 quando venne isolato per la prima volta nella storia dell'umanità l'*Acido Desossiribonucleico*, anche conosciuto come *DNA*. Il pioniere di questa scoperta è Friedrich Miescher (Figura 1.1), medico e ricercatore nato in Svizzera nel 1844. Durante il processo di scoperta, Miescher aveva realizzato che nonostante avesse proprietà simili alle proteine, la nuova sostanza – il DNA – non lo era. Prima di isolare le cellule dal pus presente nelle bende chirurgiche dell'ospedale in cui lavorava, Miescher fu molto attento ad assicurarsi che il materiale che stava utilizzando fosse fresco e non contaminato. Fu solo più tardi, nel 1871, che il ricercatore iniziò a lavorare sullo sperma di salmone, una specie di pesce che affluiva numerosa durante il periodo autunnale nella città di Basel.

Figura 1.1: Friedrich Miescher

1.3 Nozioni di biologia

Al fine di poter comprendere appieno il lavoro svolto, in questa sezione verranno trattate nozioni base di biologia, partendo dalla cellula fino alla rappresentazione in modello del DNA in essa contenuta.

1.3.1 La cellula

Le cellule costituiscono le fondamenta di tutti gli organismi viventi. Il corpo umano è composto da trilioni di cellule. Esse danno forma al corpo, estraggono le sostanze nutritive

dal cibo, convertono quelle sostanze nutritive in energia, ed hanno delle funzioni specifiche. Le cellule contengono anche il materiale ereditario del corpo, e possono fare copie di loro stesse. Esse sono a loro volta costituite da diverse parti, tra le quali analizzeremo il nucleo e ciò che esso contiene, il DNA.

1.3.2 Il DNA

Il *DNA*, o *acido desossiribonucleico*, è il materiale ereditario degli organismi viventi presente in ogni cellula. La maggior parte del DNA è situato all'interno del nucleo della cellula (dove è chiamato *DNA cellulare*), ma può trovarsi anche all'interno dei mitocondri, organelli addetti alla respirazione cellulare. Le informazioni nel DNA sono conservate come un codice composto da quattro basi azotate: **adenina** (A) (Figura 1.3a), **guanina** (G) (Figura 1.3b), **citosina** (C) (Figura 1.3c), e **timina** (T) (Figura 1.3d). L'ordine, o la sequenza, di queste basi determina le informazioni disponibili per costruire e mantenere operativo un organismo.

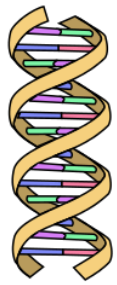
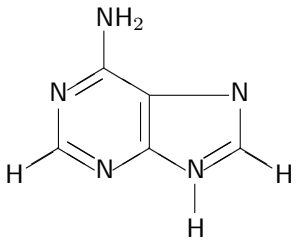
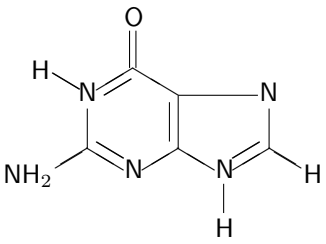


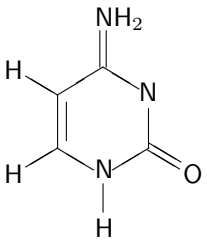
Figura 1.2: Il DNA



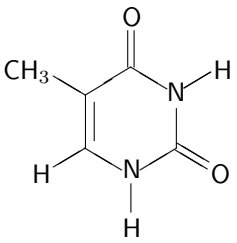
(a) Adenina (A)



(b) Guanina (G)



(c) Citosina (C)



(d) Timina (T)

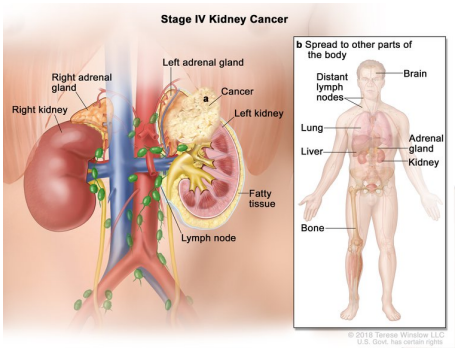
Figura 1.3: Basi azotate

Tali basi si combinano tra di loro, A con T e C con G, in maniera tale da formare una coppia. Assieme ad uno zucchero (*desossiribosio*) ed una molecola di fosfato, le basi costituiscono quello che è definito un *nucleotide*. I nucleotidi si organizzano in uno scheletro zucchero-fosfato che si dispone in modo tale da formare una struttura a *doppia elica*.

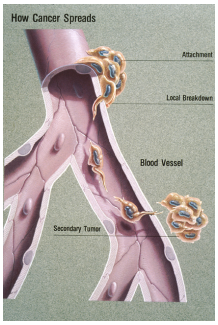
Un'importante proprietà del DNA è che si può replicare, ovvero fare copie di se stesso. Uno qualunque dei due filamenti può essere utilizzato nel processo di duplicazione per ottenere una copia identica del DNA di partenza. Questa è una fase cruciale nella divisione

di una cellula, poiché la nuova copia di essa deve avere lo stesso identico DNA della cellula di origine.

1.3.3 Cancro e tumore



(a) Cancro al rene



(b) Diffusione del cancro

Figura 1.4: Un cancro al rene ed un esempio di diffusione del cancro tramite vasi sanguini

Si stima che durante la replicazione solo una base su 10^9 [4] sia errata. Vari fattori possono influenzare questa delicata fase, come l'esposizione ad agenti chimici ed irradiazione. Questi errori molto spesso sono corretti in vari modi, ma quando questo non basta, possono essere la causa scatenante che porta una cellula a diventare *cancerogena*. In generale, una cellula è cancerogena quando inizia a moltiplicarsi senza controllo. Quando questo processo avviene in un tessuto solido come un organo (Figura 1.4a), muscolo od ossa, prende il nome di *tumore*. Ci sono due tipi di tumore: *maligno* (cancerogeno) e *benigno* (non cancerogeno). I primi possono invadere i tessuti circostanti del corpo e, mentre crescono, alcune cellule possono viaggiare nel sangue (Figura 1.4b) o altri mezzi a formare delle *metastasi*, dei tumori secondari. I tumori benigni d'altro canto conservano le caratteristiche del tessuto di origine e non hanno la tendenza ad invadere gli organi circostanti. Un tumore benigno non è quindi un cancro, ma una massa che può raggiungere dimensioni considerevoli, senza diffondersi in altre parti del corpo.

1.3.4 Eterogeneità Intra-Tumorale

Durante la divisione di una cellula, sia essa sana o tumorale, questa può acquisire delle mutazioni. Nel caso di cellule tumorali o cancerogene, può capitare che le mutazioni avvengano in posizioni diverse nel DNA sia tra tumori dello stesso tipo, *eterogeneità inter-tumorale*, che tra cellule appartenenti allo stesso tumore, *eterogeneità intra-tumorale*. L'acquisizione di mutazioni è randomica e derivante dalla crescente instabilità genomica[1] di ogni nuova generazione. Questo rappresenta un grande problema dal punto di vista della diagnosi e della terapia di cura per il cancro [11].

1.4 Modelli di sostituzione

In filogenetica il DNA può essere rappresentato come una sequenza di simboli, utilizzando le basi (sottosezione 1.3.2) corrispondenti alle posizioni degli allineamenti come caratteri.

[1] Elevata frequenza di mutazioni nel genoma di una discendenza cellulare

AGTCCAGGACAT GGCATTCAATCA

Figura 1.5: Esempi di sequenze di DNA

La Figura 1.5 rappresenta un esempio di *modello di sostituzione*, un modello che in biologia descrive il processo per cui una sequenza di simboli cambia in un'altra, modificandone i tratti che rappresenta. In cladistica[2] viene utilizzato per rappresentare delle caratteristiche presenti, utilizzando il carattere "1", o assenti, utilizzando il carattere "0", in una specie.

10011 01110

Figura 1.6: Esempio di modello di sostituzione in cladistica

L'esempio in Figura 1.6 può ipoteticamente rappresentare due specie: la prima può digerire i latticini, non depone uova, è una creatura a sangue freddo, vola e sa nuotare; la seconda non può digerire i latticini, può deporre uova, è una creatura a sangue caldo, vola e non sa nuotare. Lo stesso ragionamento può essere utilizzato per rappresentare le *mutazioni* presenti all'interno di una cellula nel caso di cellule tumorali (Figura 1.7).

BBS4	CAMSAP1	DOCK3	EPHA10	EYA4	HIPK4	HIST1H2AG	INTS8	MAL2	MYOM3	OAZ3	PPIG	PTPRQ	RGS11	RYR3	SERPINF2	SMOC1	TTN	TUFT1	ZNF540
1	1	1	0	0	0	1	0	1	1	1	1	1	1	1	0	0	1	0	0
1	1	1	1	1	0	1	0	1	0	0	0	0	1	1	0	1	0	1	0
0	0	0	0	0	1	0	1	1	1	1	1	1	0	1	1	0	1	0	1
0	0	0	0	0	1	0	1	1	1	1	1	1	0	1	1	0	1	0	1
0	0	0	0	0	1	0	1	1	1	1	1	1	0	1	1	0	1	0	1
1	1	1	1	1	0	1	0	1	0	0	0	0	1	1	0	1	0	1	0
0	0	0	0	0	1	0	1	1	1	1	1	1	0	1	1	0	1	0	1
1	1	1	1	1	0	1	0	1	0	0	0	0	1	1	0	1	0	1	0
0	0	0	0	0	1	0	1	1	1	1	1	1	0	1	1	0	1	0	1
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
0	0	0	0	0	1	0	1	1	1	1	1	1	0	1	1	0	1	0	1
1	1	1	1	1	0	1	0	1	0	0	0	0	1	1	0	1	0	1	0

Figura 1.7: Dataset di cellule tumorali e delle relative mutazioni

1.4.0.1 Tipi di modello di sostituzione

In biologia, al fine di poter spiegare le mutazioni che avvengono nel corso della vita di una cellula, si possono utilizzare due tipi di modello di sostituzione:

- modello a *finite-sites*
  esiste un numero finito di posizioni dove può avvenire una mutazione, ergo se un carattere è 0 ad un tempo t, può darsi che non è avvenuta nessuna mutazione,

[2]Metodo di classificazione degli esseri viventi che si basa sul grado di parentela, ovvero sulla distanza nel tempo dell'ultimo progenitore comune

oppure che c'è stata una mutazione da 1 a 0, oppure che c'è stata una mutazione da 0, poi 1 ed infine a 0, e così via

- modello a *infinite-sites*
  esiste un numero infinito di posizioni dove può avvenire una mutazione, di conseguenza ogni mutazione deve avvenire *per forza* in una nuova posizione rispetto alle precedenti

1.4.1 Single-Cell Sequencing

La *single-cell sequencing* (SCS) è una tecnica che esamina le informazioni sulle sequenze di singole cellule con tecnologie di sequenziamento di nuova generazione (*NGS*). Il processo dell'amplificazione del genoma[3], processo essenziale per la SCS, introduce una serie di tipi di rumore, che risulta in inferenze sbagliate sui genotipi. Gli errori comprendono: errori di perdita allelica (ADO), errori di falsi positivi (FPs) e regioni di scarsa copertura. ADO in particolare è un errore frequente nei dati SCS e contribuisce ad un numero considerevole di falsi negativi (FNs) nei dati. Gli algoritmi che verranno introdotti nel Capitolo 2 ed il lavoro di questa relazione introdotto nel Capitolo 3 cercano di mitigare questi problemi assumendo che questi errori avvengano uniformemente su tutti i valori della matrice in input, che rappresenta un modello di sostituzione analogo alla Figura 1.7.

1.4.2 Clade

In filogenetica un clade è un gruppo di organismi che includono un antenato e tutti i discendenti di quell'antenato.

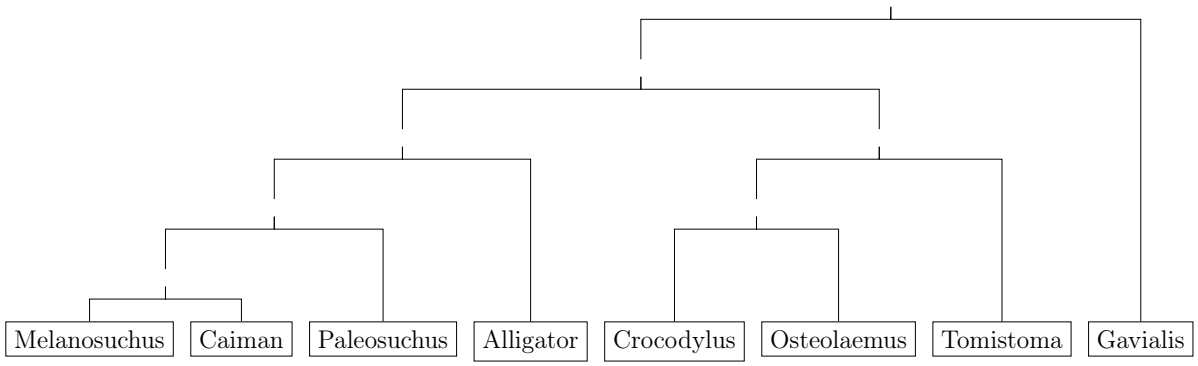


Figura 1.8: Esempio di clade

1.5 Richiami di ottimizzazione matematica

In questa sezione verranno trattate inizialmente nozioni base di ottimizzazione matematica, in particolare sulla ricerca locale dell'ottimo, per poi introdurre due tecniche di ottimizzazione che verranno utilizzate nel Capitolo 2.

1.5.1 Hill climbing

In matematica, *hill climbing* è un algoritmo di ricerca dell'ottimo, migliorando la soluzione ripetutamente fino a quando non si raggiunge un criterio di ottimalità. L'idea è quella

[3]WGA: whole-genome amplification

di partire da una soluzione sub-ottimale, che per analogia viene paragonato al partire alla base della collina, per poi migliorare la soluzione ottimale, che viene comparato allo scalare la collina, fino al raggiuntimento di una condizione, cioè raggiungere la cima della collina. In maniera generale, si può modellare nella forma descritta in Algoritmo 1.

Algoritmo 1: Hill Climbing

```
1  inizializzazione
2  while non raggiunta condizione di ottimalità do
3  |  seleziona e applica nuova operazione
4  |  if nuovo stato è ottimo then
5  |  |  termina
6  |  end
7  |  if nuovo stato è migliore del precedente then
8  |  |  stato = nuovo stato
9  |  end
10 end
```

Esistono numerose variazioni di questo algoritmo, ma le più conosciute sono *simple hill climbing*, *steepest hill climbing* e *stocastic hill climbing*, applicate a seconda delle proprietà del problema in questione. Un esempio di funzione da ottimizzare, in questo caso massimizzare, può essere come quella rappresentata in Figura 1.9a, dove esiste un solo ottimo locale ($f(x, y) = 0$) cioè la funzione è monomodale[4]. In questo caso, il *simple hill climbing* e lo *steepest hill climbing* ottengono sempre il risultato migliore.

D'altro canto, funzioni plurimodali[5] come "Eggcrate" rappresentato in Figura 1.9b, dove le tecniche citate precedentemente falliscono miseramente. Entrano quindi in gioco algoritmi di ricerca locale che ammettono, con un certo grado di libertà, di accettare un risultato peggiore di quello attuale, nella speranza di non rimanere intrappolati in un ottimo locale.

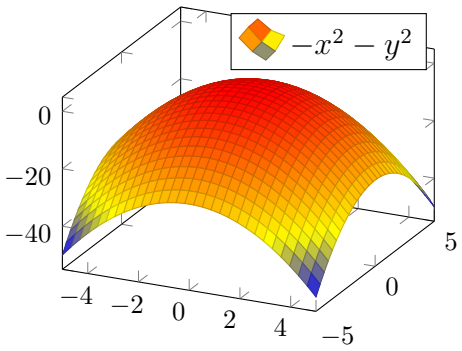
1.5.2 Simulated Annealing

Per cambiare e migliorare delle caratteristiche di un solido, in metallurgia viene utilizzata la tecnica della *ricottura* (in inglese "anneal"), dove i solidi come l'acciaio, bronzo o alluminio, vengono portati ad altissime temperature, per essere poi raffreddati ad una certa velocità chiamata *cooling rate*, che determina le caratteristiche finali del metallo. Alle alte temperature, gli atomi si muovono molto velocemente e rompono le strutture cristalline che avevano formato precedentemente. Mano a mano che la temperatura cala, gli atomi rallentano, e si recristallizzano.

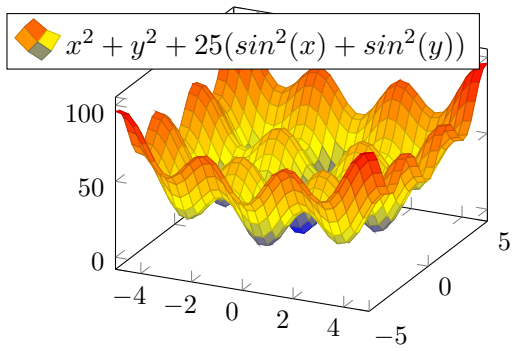
Analogamente, la tecnica matematica del *simulated annealing* è un algoritmo di ricerca che utilizza la temperatura per riuscire a scappare da eventuali ottimi locali. Quando la temperatura è alta, l'algoritmo è meno propenso ad accettare nuovi soluzioni, anche se migliori. Mano a mano che la temperatura diminuisce, la probabilità che si accetti una soluzione migliore aumenta, fino al raggiuntimento della condizione di ottimalità, cioè quando la temperatura non può più scendere, ed il risultato non può che essere un ottimo locale.

[4]Una funzione monomodale è una funzione con un solo ottimo locale, che corrisponde anche all'ottimo globale della funzione stessa

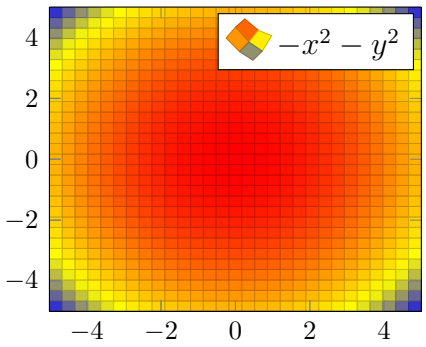
[5]Una funzione plurimodale è una funzione con più di un ottimo locale. Non è detto che esista un unico ottimo globale



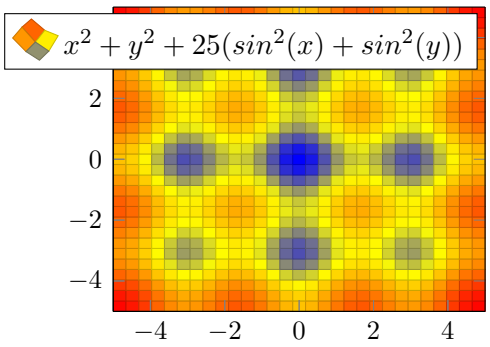
(a) Paraboloide, monomodale



(b) "Eggcrate", plurimodale



(c) Paraboloide, monomodale



(d) "Eggcrate", plurimodale

Figura 1.9: Esempi di funzione monomodale e plurimodale

Questa funzione però molto spesso fallisce, specie se si cerca di ottimizzare una funzione plurimodale come in Figura 1.9b.

1.5.3 Particle Swarm Optimization

Un'altra tecnica di ricerca dell'ottimo è quella del *particle swarm optimization*. Questo algoritmo iterativo nasce inizialmente come simulazione del comportamente sociale di stormi di uccelli che si sincronizzano in volo, o un branco di pesci alla ricerca di cibo [7]: membri individuali del branco possono trarre vantaggio dalle scoperte ed esperienze passate di tutti gli altri membri durante la ricerca del cibo, un vantaggio che può rivelarsi decisivo per superare la competizione. Questa è un'ipotesi fondamentale per poter definire il *particle swarm optimization*, ed in generale di tutti gli algoritmi dello stesso tipo, nella letteratura denominati *algoritmi genetici*.

Nell'algoritmo, il branco di pesci viene sfruttato come analogia per lo *swarm* (rappresentato in Figura 1.10), che indica l'insieme degli elementi appartenenti ad una popolazione, questi indicati come *particelle* dello swarm.

Ad ogni iterazione, la posizione di ogni particella viene aggiornata per ogni dimensione del problema, basandosi sulla migliore posizione della particella p_i e sulla migliore posizione globale dello swarm g (in Figura 1.11 si vede un esempio di questa combinazione, dove la posizione finale del vettore è data dalla combinazione di due valori). Questa combinazione dei due valori migliori risolve il problema di rimanere intrappolati in un ottimo locale che presentano gli algoritmi visti nella sottosezione 1.5.1. Con il proseguire dell'algoritmo sarà possibile osservare che le singole particelle mano a mano convergono verso un ottimo locale, o di più, se la funzione da ottimizzare è multimodale (come in Figura 1.9b). È possibile inoltre introdurre la *velocità* come parte dell'algoritmo, al fine

Algoritmo 2: Simulated Annealing

```
1  best ← random()
2  T ← 1.0
3  T_min ← 0.0001
4  cooling_rate ← 0.9
5  while T > T_min do
6      new_best = neighbour(best)
7      ap ← acceptance_probability(best, new_best, T)
8      if ap > random() then
9          best ← new_best
10     T = T * cooling_rate
```

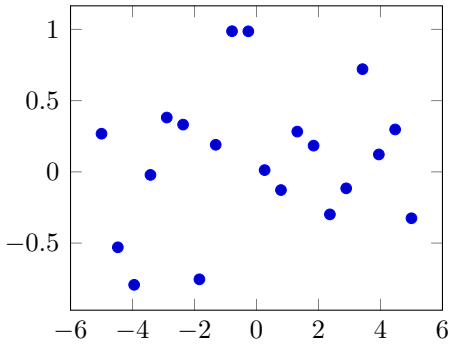


Figura 1.10: Esempio di swarm

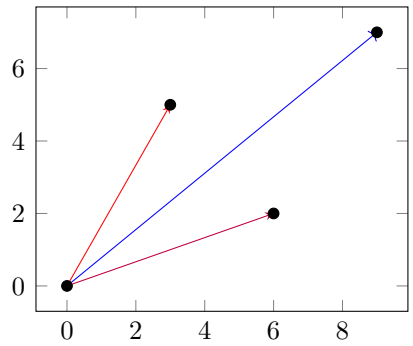


Figura 1.11: Somma vettoriale tra p_i e g

di influire su quanto velocemente una particella si muove verso la soluzione, che in algebra vettoriale si traduce nella modifica del modulo del vettore di spostamento.

1.6 Nozioni Extra

In questa sezione verranno introdotti dei concetti aggiuntivi che non rientrano nelle sezioni sopracitate, e le cui basi si assumono essere già presenti per la comprensione degli stessi. Queste nozioni verranno utilizzate nei capitoli successivi per complementare la spiegazione di modelli, termini o algoritmi.

1.6.1 Catena di Markov

In teoria probabilistica, una *Catena di Markov* è un modello stocastico che descrive una sequenza di possibili eventi la cui probabilità dei singoli eventi dipende solo ed unicamente dallo stato dell'evento che lo precede. Questa definizione si traduce nell'assunzione che gli stati passati e futuri di ogni evento sono indipendenti rispetto alla storia di tutti gli eventi già avvenuti all'interno della catena.

Il modello è strutturato come una *macchina a stati finiti*, dove i rami che collegano uno stato ad un altro hanno un valore numero associato, chiamato la *probabilità di transizione*, che indica quanto è probabile passare allo stato di destinazione.

Una catena di Markov può essere vista anche come una sequenza di variabili aleatorie $X_0, X_1, X_2, \ldots$ che soddisfano la proprietà di indipendenza. Questa soddisfacibilità è definita come *Proprietà di Markov*:

Algoritmo 3: Particle Swarm Optimization

```
1  n = numero particelle
2  m = numero dimensioni dello spazio di ricerca
3  for i ← 1 to n do
4      x_i ~ U(b_low, b_up) ▷ Inizializzo ogni particella con un valore random
         nel mio spazio di ricerca, delimitato da un lower bound b_low ed
         un b_up
5      p_i ← x_i ▷ Inizializzo la posizione migliore della particella alla
         sua posizione iniziale
6      if f(p_i) > f(g) then
7          g ← p_i ▷ Aggiorno la posizione migliore globale
8      v_i ~ U(−|b_up − b_low|, |b_up − b_low|) ▷ Inizializzo la velocità iniziale
         della particella
9  while criterio di terminazione non soddisfatto do
10     for i ← 1 to n do
11         for d ← 1 to m do
12             r_p, r_g ~ U(0, 1) ▷ Parametri di casualità
13             v_{i,d} ← ωv_{i,d} + φ_p r_p(p_{i,d} − x_{i,d}) + φ_g r_g(g_d − x_{i,d}) ▷ Aggiorno la
                 velocità della particella
14             x_i ← x_i + v_i ▷ Aggiorno la posizione della particella
15             if f(x_i) > f(p_i) then
16                 p_i ← x_i ▷ Aggiorno la posizione migliore della particella
17                 if f(p_i) > f(g) then
18                     g ← p_i ▷ Aggiorno la posizione migliore dello swarm
```

$$P(X_n = i_n | X_{n-1} = i_{n-1}) = P(X_n = i_n | X_0 = i_0, X_1 = i_1, \ldots, X_{n-1} = i_{n-1}) \tag{1.1}$$

Le probabilità di transizione in un dato tempo t per ogni stato della catena possono essere rappresentate come una matrice P_t, denominata *matrice di transizione* e definita come segue:

$$(P_t)_{i,j} = \mathbb{P}(X_{t+1} = j | X_t = i) \tag{1.2}$$

Questo significa che ogni riga della matrice è un *vettore di probabilità*, e la somma dei suoi elementi è sempre 1.

Una matrice di transizione $P_t^(k) = P_t \cdot P_{t+1} \cdots P_{t+k-1}$ soddisfa quindi:

$$P_t^{(k)} = \begin{pmatrix} \mathbb{P}(X_{t+k} = 1 \mid X_t = 1) & \mathbb{P}(X_{t+k} = 2 \mid X_t = 1) & \ldots & \mathbb{P}(X_{t+k} = n \mid X_t = 1) \\ \mathbb{P}(X_{t+k} = 1 \mid X_t = 2) & \mathbb{P}(X_{t+k} = 2 \mid X_t = 2) & \ldots & \mathbb{P}(X_{t+k} = n \mid X_t = 2) \\ \vdots & \vdots & \ddots & \vdots \\ \mathbb{P}(X_{t+k} = 1 \mid X_t = n) & \mathbb{P}(x_{t+k} = 2 \mid X_t = n) & \ldots & \mathbb{P}(X_{t+k} = n \mid X_t = n) \end{pmatrix}. \tag{1.3}$$

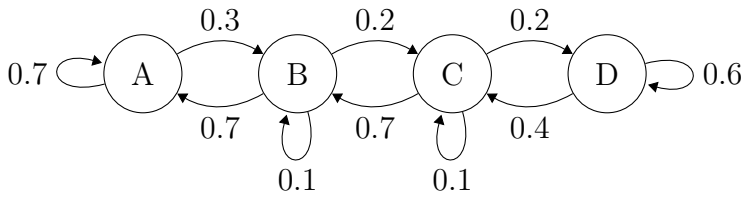


Figura 1.12: Esempio di catena di Markov

Nell'esempio in Figura 1.12 la matrice di transizione a $k = 0$ è come segue:

$$P_t^0 = \begin{pmatrix} 0.7 & 0.3 & 0 & 0 \\ 0.7 & 0.1 & 0.2 & 0 \\ 0 & 0.7 & 0.1 & 0.2 \\ 0 & 0 & 0.4 & 0.6 \end{pmatrix} \tag{1.4}$$

1.6.1.1 Catena di Markov: Variante Monte Carlo (MCMC)

Il metodo *Monte Carlo* in statistica si basa sul campionamento casuale per ottenere dei risultati. Un esempio del loro utilizzo è dato dal calcolo dell'area di una funzione complessa: vengono generati n punti casuali, e si calcola quanti di questi rientrano all'interno della funzione della quale vogliamo calcolare l'area. Il numero di questi punti che ricade in questa categoria ci fornisce un grado di approssimazione proporzionale al numero di punti che generiamo per ottenere il risultato finale: più punti generiamo, più è accurata la stima dell'area, o più in generale, di qualunque cosa stiamo cercando di approssimare.

Volendo ad esempio stimare una distribuzione a posteriori[6] è possibile utilizzare una variante della Catena di Markov sfruttando le proprietà aleatorie del metodo Monte Carlo: la tecnica così ottenuta è definita una *Monte Carlo Markov Chain* (MCMC). Ogni valore generato è aleatorio, ma le scelte fatte per generare i valori successivi sono limitate dallo stato corrente e dalla distribuzione dei valori attuali. Una MCMC quindi può essere paragonata ad un cammino aleatorio[7] che con l'aumentare delle iterazioni converge alla distribuzione (o modello matematico) che si vuole stimare.

1.6.2 Likelihood

Uno dei problemi della genomica (ma anche di altri campi, come la linguistica [5]) è quello di ricostruire un albero filogenetico partendo da dati raccolti di caratteri appartenenti una popolazione. I dati in questione seguono il modello di sostituzione (sezione 1.4) ed i tratti possono essere qualunque aspetto della popolazione, come altezza, peso, capacità di volare o di respirare sott'acqua. Nel presente lavoro, la popolazione presa in analisi sono cellule, ed i tratti considerati sono le mutazioni acquisite, ed il problema si può interpretare come la ricerca del migliore albero filogenetico che meglio rappresenta il modello di sostituzione. Per stabilire come un albero è *migliore* di un altro, viene definita una funzione di *likelihood*, che esprime appunto quanto è probabile un particolare set di parametri stastistici (nel nostro caso la configurazione dei nodi dell'albero inferito) data il modello di sostituzione iniziale. Supponendo di aver generato un albero T con n nodi, un nodo per ogni mutazione, e ad ogni foglia può essere associata una cellula. Le cellule

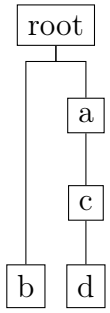
[6] Distribuzione di una variabile aleatoria condizionata a delle informazioni rilevanti su di essa

[7] Modello matematico che descrive un processo stocastico come una successione di step aleatori all'interno di uno spazio matematico (e.g. $\mathbb{Z}$)

così assegnate avranno una serie di caratteri, che vengono acquisiti o persi una volta per ogni nodo padre.

a	b	c	d
1	0	0	0
0	1	0	0
1	0	1	0
1	0	1	1

(a) Modello di sostituzione



(b) Esempio di albero inferito

Figura 1.13

Il calcolo della likelihood $\mathcal{L}$ si basa sulla probabilità condizionata della configurazione attuale dell'albero T dato il modello di sostituzione in input M, con n righe (una per ogni cellula) e m colonne (una per ogni mutazione), che si traduce in termini generali nella formula che segue:

$$\mathcal{L}(T) = \mathcal{L}(1) \cdot \mathcal{L}(2) \cdots \mathcal{L}(n) = P(M|T) = \prod_{j=1}^{m} P(M_j|T) \tag{1.5}$$

Spesso è utile riportare la somma logaritmica delle likelihood ad ogni posizione, poiché le probabilità condizionate di ogni nodo si traducono in numeri molto piccoli. La formula per il calcolo della *logaritmic likelihood* si traduce quindi come segue:

$$\ln \mathcal{L} = \ln \mathcal{L}(1) + \ln \mathcal{L}(2) + \ln \mathcal{L}(n) = \sum_{j=1}^{m} \ln \mathcal{L}(j) \tag{1.6}$$

1.6.3 Matching

Nella teoria dei grafi un *matching* M è definito come un sottoinsieme di coppie di nodi di un grafo $G = (V, E)$ tale per cui ogni coppia non è adiacente ad un altra, e nessuna coppia è un loop; ciò significa che nessuna coppia condivide un nodo.

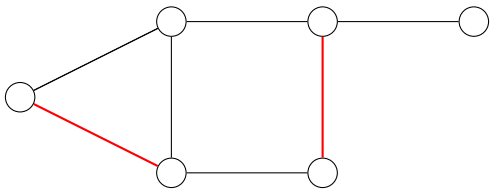


Figura 1.14: Esempio di matching

1.6.3.1 Maximum Weight Matching

Dato un grafo pesato $G = (V, E, w)$, con $w : E \to \mathbb{R}$, il *maximum weight matching* è un matching per cui la somma:

$$\sum_{e}^{E} w(e) \tag{1.7}$$

È massima.

Capitolo 2

Stato dell'arte

2.1 Introduzione

Con l'avvento delle tecnologie per il sequenziamento del DNA partendo da singole cellule (SCS), iniziano ad essere disponibili dati di alta qualità. Queste tecnologie forniscono il sequenziamento di dati da singole cellule, permettendo quindi di ricostruire l'albero filogenetico di una cellula. È però da tenere in considerazione l'alto tasso di errore associato a questo tipo di dati, innalzando di conseguenza il grado di difficoltà del processo di ricostruzione della filogenesi. In questo capitolo, verranno esaminate le tecnologie già presenti che hanno affrontato questa sfida, focalizzando l'analisi sul modello di ricerca dell'ottimo, del calcolo della *likelihood* dell'albero inferito e della complessità dell'algoritmo utilizzato.

2.2 SciTe [6]

SCITE (*Single-Cell Inference of Tumor Evolution*) usa un approccio basato sulla likelihood dell'albero inferito per fare una ricerca stocastica dell'albero migliore rispetto ai dati in input. Ricordando l'assunzione che le mutazioni siano indipendenti l'una dall'altra, dati i tassi di errore $\theta = (\alpha, \beta)$, la matrice M di partenza di dimensioni $n \times m$, dove n è il numero di cellule e m il numero di mutazioni, possiamo calcolare la likelihood come segue:

$$P(M|T, \sigma, \theta) = \prod_{i=1}^{n} \prod_{j=1}^{m} P(M_{i,j}|D_{i,j}) \tag{2.1}$$

Lo strumento è costruito attorno al funzionamento di una catena di Markov Monte Carlo (sottosottosezione 1.6.1.1) che permette di trovare l'albero migliore basandosi sulla likelihood calcolata, oppure basandosi sulla distribuzione a posteriori. Uno dei principali vantaggi indicati è quello della scalabilità lineare, proporzionale con il numero dei campioni.

2.2.1 Complessità

Ad ogni step, la MCMC impiega $O(mn)$ per calcolare la likelihood dell'albero, impiegando un tempos stimato di $O(mn^3ln(n))$ per una convergenza all'albero migliore. Il tempo è linearmente dipendente con il numero dei campioni e di mutazioni.

2.3 SiFit: inferring tumor trees from single-cell sequencing data under finite-sites models [12]

Il progetto *SiFit* affronta il problema presupponendo un modello a posizioni finite (*finite sites model*), quindi permettendo delle back-mutation[1]. Lo strumento accetta sia matrici binarie, con un set di valori possibili $\{0, 1, X\}$, che matrici ternarie, per le quali accetta il set di valori $\{0, 1, 2, X\}$, dove 0 denota un genotipo *reference* omozigota, 1 e 2 denotano un genotipo eterozigota ed omozigota *non-reference*, rispettivamente, e X denota la mancanza di informazioni.

Vengono principalmente usati due tipi di mosse: mosse di *prune and regraft* e mosse di *swap*. Le mosse di *prune and regraft* prevedono il cambiamento randomico della topologia dell'albero attraverso il riposizionamento di un sottoalbero all'interno (rSPR, *random Subtree Prune and Regraft*) e la lunghezza dei rami dell'albero (eSPR, *extending Subtree Pruning and Regrafting*). Le mosse di *swap* prevedono lo scambio di nodi interni all'albero (stNNI, *stochastic nearest-neighbour interchange*) e dei rami (rSTS, *random Sub-Tree Swapping*) [12, 8].

2.3.1 Complessità

Ad ogni passo dell'algoritmo proposto, calcolare la *likelihood* dell'albero è il processo più dispendioso. Per n singole cellule e m mutazioni, il calcolo della likelihood impiega $\mathcal{O}(nk^2m)$, dove k è il numero massimo di stati per mutazione, quindi $k = 3$ e $k = 2$ per una matrice ternaria e una matrice binaria in input, rispettivamente.

Il numero di iterazioni i è definito dall'utente, ottenendo quindi come complessità generale dell'algoritmo $\mathcal{O}(nk^2mi)$.

2.4 SASC - Inferring Cancer Progression from Single-Cell Sequencing while Allowing Mutation Losses [2]

In SASC (*Simulated Annealing Single-Cell* inference) viene utilizzato il modello *finite-sites* insieme al modello di Dollo-k. Il modello della Parsimonia di Dollo prevede che ogni mutazione può essere acquisita una sola volta, ma una perdita di una mutazione può accadere un numero indefinito di volte. La versione ristretta di Dollo-k assume che ogni mutazione possa essere acquisita una sola volta e persa al massimo k volte. Per trovare l'albero migliore il problema viene modellato sull'algoritmo euristico *simulated annealing* (sottosezione 1.5.2). Le mosse utili alla scalata della collina sono: *aggiunta di back-mutation*, *rimuzione di back-mutation*, *node switch* e *prune-and-regraft*.

Ad ogni riduzione della temperatura viene cercato un albero topologicamente vicino a quello attuale applicando una delle quattro operazioni precedenti, finché non si raggiunge la temperatura minima. La bontà dell'albero è definita dalla *likelihood logaritmica*, e viene calcolata ad ogni operazione effettuata sull'albero inferito nella seguente maniera:

$$max \sum_{i}^{n} \sum_{j}^{m} \log(P(M_{i,j}|D_{i,j})) \tag{2.2}$$

[1] Mutazioni all'indietro, una mutazione viene persa durante la vita di una cellula

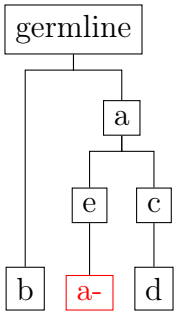


Figura 2.1: Esempio di back-mutation

2.4.1 Complessità

L'algoritmo Simulated Annealing impiega $O(\log t)$ step fino a quando non raggiunge la temperatura minima, con t ad indicare la temperatura di partenza, e per ogni step impiega $O(nm)$ per calcolare la likelihood logaritmica. In generale, SASC ha una complessità di $O(nm \log t)$.

Capitolo 3

Inferenza di Alberi Tumorali tramite Particle Swarm Optimization

In questa sezione verrà descritto lo strumento di inferenza di alberi tumorali tramite Particle Swarm Optimization (d'ora in poi **PSO**), le ipotesi effettuate, i risultati ottenuti ed eventuali conclusioni. Viene utilizzata la tecnica euristica Particle Swarm Optimization, descritta precedentemente nel sottosezione 1.5.3.

3.1 Strumenti Utilizzati

Lo strumento è stato sviluppato in Python. Inizialmente si era cercato di utilizzare Cython[1], poi abbandonato poiché lo studio di questo linguaggio di programmazione avrebbe ridotto il tempo utile alla vera e propria implementazione dello strumento. Si pensa però in futuro di riscrivere il codice utilizzando *C++*.

L'ambiente di sviluppo scelto è *Visual Studio Code*, con l'ausilio delle estensioni di debugging per Python. In generale sono sempre stati utilizzati strumenti che seguono la filosofia *Open Source*, di conseguenza questa tesi ed il codice sviluppato saranno entrambi disponibili liberamente sul profilo GitHub [3] del sottoscritto sotto licenza GPL3 e MIT rispettivamente.

I moduli e le librerie Python che sono state utilizzate sono elencate di seguito:

- random – modulo standard che implementa funzioni per la generazione di numeri pseudo-randomici
- multiprocessing – modulo standard utilizzato per implementare il supporto di parallelizzazione e gestione della concorrenza
- sys – modulo standard per accedere ad alcune variabili o funzioni di sistema
- time – modulo standard usato per accedere a funzioni relative al tempo, come l'accesso all'ora attuale, o per la misurazione dei tempi di esecuzione
- math – modulo standard che permette l'accesso a funzioni ottimizzate per il calcolo matematico
- copy – modulo standard utilizzato per copiare in profondità degli oggetti, con un nuovo puntatore all'oggetto

[1] Un linguaggio di programmazione simile a Python, ma con il vantaggio di avere velocità simili a quelle ottenute da C

- * graphviz – modulo esterno per la visualizzazione ed il salvataggio su file dei grafi da formato graphviz[2]
- * ete3 – modulo esterno utilizzato come base di appoggio per le funzioni essenziali relative ai nodi di un albero, come la ricerca dei nodi e l'aggiunta o rimozione di essi
- * networkx – modulo esterno sfruttato per il calcolo del *maximum weight matching*

* Moduli esterni

3.1.1 Dati utilizzati

Per poter testare l'algoritmo ed il codice sviluppato su di esso è essenziale avere dei dati sia veri che simulati. A questo proposito sono stati utilizzati i dati simulati dal laboratorio di appartenenza del progetto, AlgoLab. Nelle seguenti sezioni verranno analizzati i dati relativi al file simulato presente nella directory del progetto `data/simulatex/exp1/sim1_scs.txt`.

3.2 Adattamento del problema a Particle Swarm Optimization

Una delle principali sfide in questo progetto è stata di riuscire a trovare un'interpretazione valida ed efficacie adatta alla tecnica euristica scelta, appunto il Particle Swarm Optimization, il cui funzionamento ed algoritmo sono descritti dettagliatamente nella sottosezione 1.5.3. Il modello più immediato da adattare al problema è quello di associare una particella dello swarm ad un albero sul quale vengono effettuate delle operazioni. Queste operazioni saranno ciò che determina il "movimento" della particella nello spazio. Lo swarm non è quindi altro che l'insieme degli alberi. Possiamo dividere l'implementazione in quattro fasi:

1. Inizializzazione
2. Calcolo del movimento della particella
3. Aggiornamento della particella
4. Aggiornamento della particella migliore dello swarm e della soluzione migliore della particella

3.2.1 Inizializzazione

Possiamo considerare una particella dello swarm come un albero, la cui posizione è determinata dalla configurazione dei suoi nodi. L'inizializzazione viene effettuata randomizzando la topologia di un albero binario T con m nodi, uno per ogni mutazione, con una funzione randomica[3]. In questo modo, viene ottenuta una topologia simile a quella rappresentata nella Figura 3.1. Il metodo di generazione dell'albero binario è ripreso da quello implementato su SASC, ed è descritto nell'Algoritmo 4.

[2]Strumento che permette di disegnare rappresentare dei grafi utilizzando la notazione DOT, un linguaggio che descrive la struttura di grafi generici

[3]In realtà la funzione è del tipo pseudo-randomica, utilizzando il generatore *Mersenne Twister* [9], ma non avendo problemi dal punto di vista della sicurezza o della riproducibilità, è stato pervenuto che per i nostri scopi è più che valido e sufficiente

Algoritmo 4: RandomTreeInit

```
1  m = numero di mutazioni r = radice dell'albero
2  tree = [1, ..., m] ▷ Vettore con tutti i nodi dell'albero
3  random.shuffle(tree) ▷ Il vettore dei nodi viene randomizzato
4  nodes ← [r]
5  append_node ← 0
6  while i < m do
7      newNode ← new Node(name = mutation_name(m), parent =
         nodes[append_node], mutationid = tree[i])
8      nodes.append(newNode)
9      i ← i + 1
10     if i < m then
11         newNode ← new Node(name = mutation_name(m), parent =
             nodes[append_node], mutationid = tree[i])
12         nodes.append(newNode)
13     append_node ← append_node +1 i ← i + 1
14 return r
```

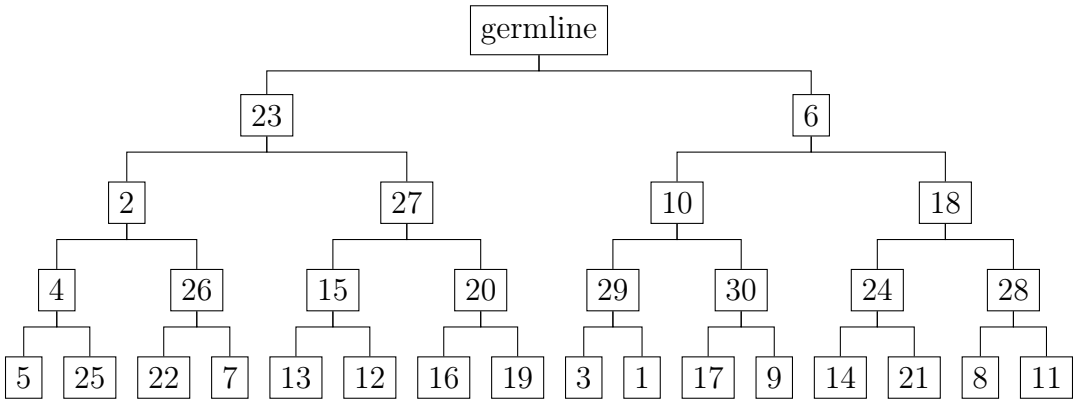


Figura 3.1: Esempio di topologia iniziale con $m = 30$

3.2.2 Calcolo del movimento di una particella

La seconda fase è il calcolo della velocità con la quale una particella si muove rispetto a p_i e g, cioè l'albero migliore che ha avuto la particella fino ad ora, e l'albero migliore in tutto lo swarm. Come parametro di paragone è definita la likelihood logaritmica descritta nell'Equazione 2.2.

Questo step è stato uno dei problemi principali da affrontare sin dal principio del progetto. Non è immediato trovare un modello diretto che rappresenti la velocità e la direzione di una particella verso gli alberi migliori p_i e g, data la natura non lineare del problema. Sono state, dunque, introdotte diverse metriche (o assenza di tali) per cercare di definire al meglio questo parametro in maniera tale che aderisse al meglio all'algoritmo scelto, quindi PSO. Nelle seguenti sottosezioni verranno descritti tali esperimenti, con risultati e considerazioni del caso.

3.2.2.1 Assenza del parametro di velocità

Nei momenti iniziali di questo progetto, come è stato descritto precedentemente, trovare una corrispondenza congeniale ed adeguata all'idea di *velocità* del PSO è stato impegnativo. Al fine comunque di poter dare un via al lavoro, una delle prime metriche adottate di velocità è stata la vera e propria assenza di tale metrica. I risultati descritti nella Figura 3.2 possono sembrare promettenti, dato che comunque provano la funzionalità del programma, e che la likelihood

aumenta. Quello che i dati non ci dicono è però qualcosa che possiamo intuire dall'intrinseco comportamento dell'assenza della velocità come metrica: siamo di fronte ad un semplice algoritmo di highest hill climbing, descritto nella sottosezione 1.5.1. Quello che succede quindi è che si cerca di migliorare l'algoritmo al primo miglioramento individuato, in questo caso corrisponde ad ogni volta che si trova una likelihood migliore. Questo può sembrare un ottimo risultato, ma è in realtà un problema. L'estrema e ripida scalata rischia di incastrare la ricerca in un ottimo locale, peggiorando considerevolmente le possibilità di trovare un albero filogenetico di **buona qualità**. È importante sottolineare la *bontà* del risultato, e non il basso valore della *likelihood*, in quanto questo è solo un valore indicativo della verosomiglianza dell'albero con i dati a nostra disposizione, ma non dell'effettiva qualità della loro rappresentazione. È molto meglio quindi cercare di trovare un albero filogenetico con una qualità migliore che quello con la likelihood più elevata.

L'algoritmo risultante per il calcolo della posizione della particella risulta quindi essere il seguente:

Algoritmo 5: ParticleIteration

```
1 r ← random()
2 if r < .33 then
3 |  tree ← p_i
4 else if r < .66 then
5 |  tree ← g
6 else
7 |  tree ← x_i
8 execute_random_operation(tree)
```

Particelle	Iterazioni	Likelihood Iniziale	Likelihood Migliore	CPU Time (s)
5	20	-8865.285307	-5807.871680	8.622138
10	20	-8865.285307	-3096.036336	17.478406
50	20	-8341.992690	-2258.473502	88.392956
85	20	-8341.992690	-1681.115794	153.431512
100	20	-8341.992690	-1650.249232	193.255694
150	20	-8163.910048	-1199.301909	275.721155
200	20	-7944.950318	-1173.538980	370.858453

Figura 3.2: Risultati ottenuti con assenza della velocità usando i parametri: $\alpha = 0.25, \beta = 1 * 10^{-5}, k = 3, seed = 1, n = 150, m = 30$

3.2.2.2 Prima metrica per la distanza tra filogenie

Non avere un parametro della velocità è quindi un problema abbastanza importante, e preclude il poter continuare a sviluppare il progetto sotto le condizioni dello sviluppo di un algoritmo genetico di tipo PSO. Ci si è quindi concentrati sullo studio di una metrica che permetta di trovare la distanza tra due alberi filogenetici. Il risultato di tale lavoro è stata la formulazione dell'Equazione 3.2. Per il *max_weight_matching* (descritto nella sottosottosezione 1.6.3.1) si assume che il grafo $G = (V, E)$ sia formato da tutti i nodi del primo albero, T_1 e del secondo albero, T_2, in maniera tale da creare un *grafo bipartito*[4], e la funzione peso $w : E \to \mathbb{N}$ indica

[4]Un grafo bipartito è un grafo tale che l'insieme dei suoi vertici si può partizionare in due sottoinsiemi tali che ogni vertice di una di queste due parti è collegate solo a vertici dell'altra

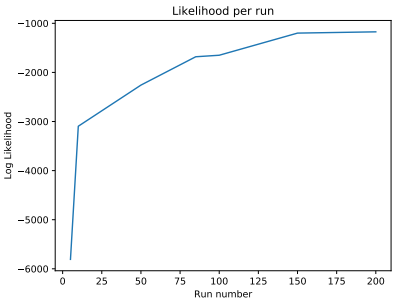


Figura 3.3: Grafico della likelihood sui vari parametri delle particelle con l'assenza della velocità

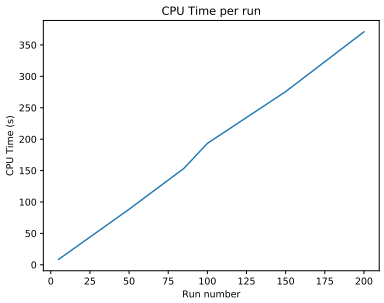


Figura 3.4: Grafico del tempo sui vari parametri delle particelle con l'assenza della velocità

il numero di mutazioni comuni tra due clade (sottosezione 1.4.2). Il max weight matching restituisce un arco il cui peso è il massimo rispetto a quello di tutti gli altri archi presenti nel grafo bipartito. Il risultato della formula dovrebbe risultare 0 quando T_1, T_2 rappresentano lo stesso albero, mentre > 0 quando differiscono.

$$dist(T_1, T_2) = mutations - max\_weight\_matching(T_1, T_2) \tag{3.1}$$

Un'analisi più approfondita della formula fa però emergere un problema. La somma delle mutazioni comuni calcolata dalla funzione w cresce all'aumentare della profondita in cui si trova il nodo nell'albero, poiché acquisisce più mutazioni. Un esempio è rappresentato nella Figura 3.7.

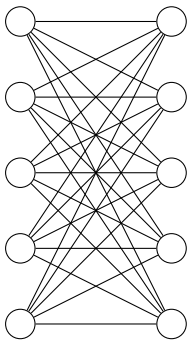


Figura 3.5: Esempio di grafo bipartito

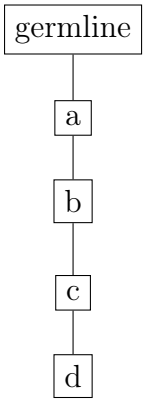


Figura 3.6: Somma delle mutazioni acquisite per $max(d) = 4$

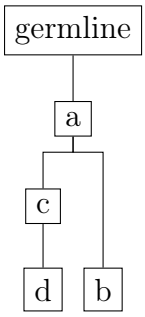


Figura 3.7: Somma delle mutazioni acquisite per $max(d) = 3$

Questo risulta in un calcolo della distanza che può risultare negativo, ed è quindi stato necessario adattare la metrica.

3.2.2.3 Seconda metrica per la distanza tra filogenie

L'analisi effettuata precedentemente ha portato alla formulazione di una nuova metrica (Equazione 3.2), stavolta rivelatasi corretta. D'ora in avanti verrà utilizzata questa metrica per la distanza tra due filogenie.

$$dist(T_1, T_2) = max\{\sum_{x \in T_1} m(x), \sum_{x \in T2} m(x)\} - max\_weight\_matching(T_1, T_2) \tag{3.2}$$

L'algoritmo utilizzato per il maximum weight matching è quello presente all'interno della libreria networkx, richiamabile con `networkx.algorithms.matching.max_weight_matching(G)`, ed ha come complessità generale $O(n^3)$. A questo si vanno a sommare i tempi per il calcolo delle mutazioni in comune, che risulta essere $O(n^2 + nm)$ ed il calcolo del numero di mutazioni di un albero $O(n \cdot m)$. In totale, la complessità dell'algoritmo per il calcolo della distanza risulta essere $O(n^3 + n^2 + nm)$.

3.2.2.4 Hill climbing con considerazione della distanza

Ora che è stata definita una metrica per la distanza, è possibile iniziare a ragionare in maniera più analoga sulla velocità rispetto a quanto non si faceva prima. Si è pensato quindi di utilizzare la nuova metrica introdotta per calcolare la distanza tra la particella attuale x_i e le due particelle migliori p_i e g, e scegliere il clade con meno mutazioni in comune tra l'albero meno distante tra i due. Il clade scelto randomicamente viene poi inserito randomicamente come figlio di un nodo dell'albero attuale, e le mutazioni duplicate rimosse. Nella teoria questo avrebbe dovuto portare ad un graduale avvicinamento della particella attuale verso l'ottimo, nella pratica l'implementazione seguita è risultata, ancora una volta, in un algoritmo di hill climbing. Eseguendo gli stessi test effettuati nella sottosottosezione 3.2.2.1 si ottengono i risultati rappresentati nella Figura 3.8.

Particelle	Iterazioni	Likelihood Iniziale	Likelihood Migliore	CPU Time (s)
5	20	-8865.28530	-8157.01522	17.90602278
10	20	-8865.28530	-7702.05544	41.00506496
50	20	-8341.99268	-3583.960402	239.6226656
85	20	-8341.99268	-2194.729592	410.934185
100	20	-8341.99268	-1639.93216	591.655797
150	20	-8163.91004	-1533.016705	789.562396
200	20	-7944.95031	-1540.622970	1127.387530

Figura 3.8: Risultati ottenuti con considerazione della distanza usando i parametri: $\alpha = 0.25, \beta = 1 \cdot 10^{-5}, k = 3, seed = 1, n = 150, m = 30$

Si può evidenziare dal test effettuato come i tempi siano quasi duplicati, quasi triplicati, relativamente al fatto che vengono effettuati dei test sulla distanza che incidono negativamente sulle prestazioni. È però da notare che questo aumento dei tempi non coincide con un miglioramento sostanziale della ricerca dell'ottimo da parte dell'algoritmo, ma anzi, sia in media peggiorata. Questo può essere causa di vari fattori, tra cui la possibilità che l'algoritmo si comporti allo stesso modo di quello presentato nella sottosottosezione 3.2.2.1, ma l'introduzione di un nuovo passaggio dell'hill climbing abbia rallentato e peggiorato la scalata.

3.2.2.5 Clade casuali tra p_i e g come avvicinamento all'ottimo

Fin'ora nel progetto sono stati implementati algoritmi che assomigliavano per la natura del loro comportamento più ad una scalata della collina (hill climbing) che ad un PSO. È stato quindi

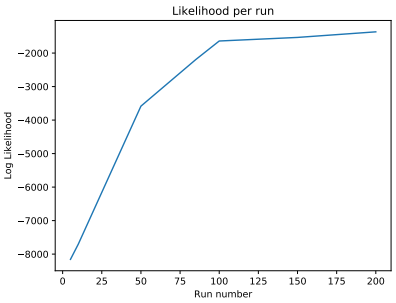


Figura 3.9: Grafico della likelihood sui vari parametri delle particelle con considerazione della distanza

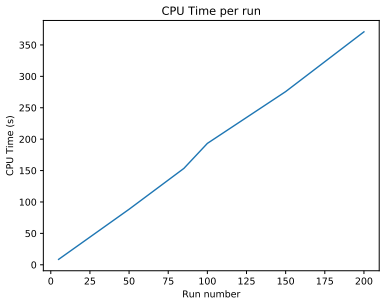


Figura 3.10: Grafico del tempo sui vari parametri delle particelle con considerazione della distanza

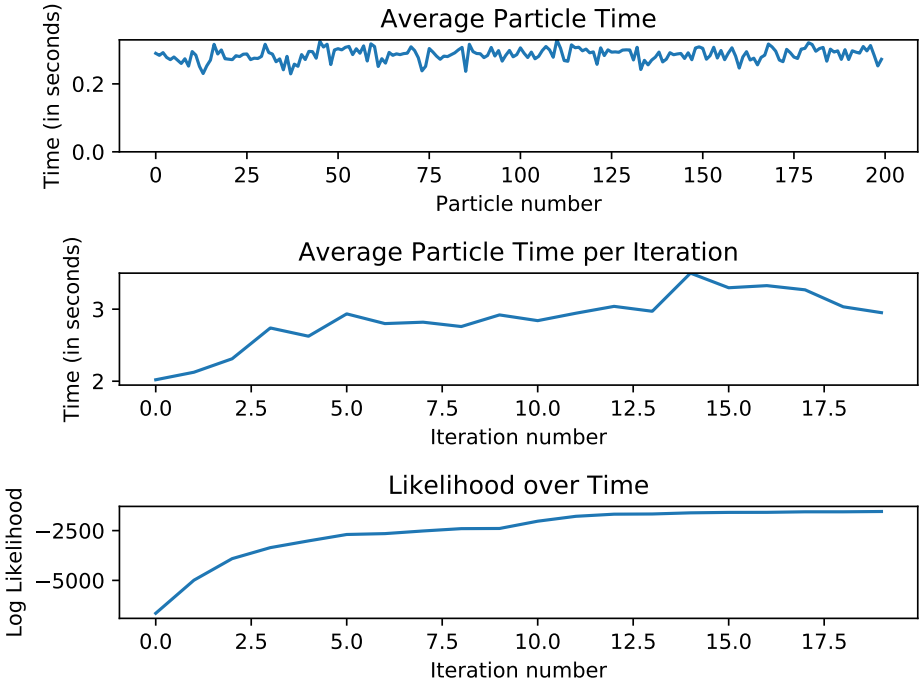


Figura 3.11: Grafici sui tempi ed il miglioramento della likelihood con il passare del tempo

un passo fondamentale quello di abbandonare totalmente la copia completa e completa delle particelle p_i e g, e di operare in maniera più sistematica.

La metrica della distanza rimane invariata, ma viene introdotta una primitiva logica di *velocità*. Ora l'algoritmo ad ogni iterazione calcola la distanza tra la particella corrente x_i, p_i e g. Utilizzando le distanze appena ottenute, vengono ricavate due liste di nodi, una per p_i ed una per g, i cui elementi hanno la caratteristica di avere un numero massimo di mutazioni minore alla loro distanza relativa alla particella corrente. Questo perché si è pensato che più la particella è distante da un ottimo, più nodi bisogna copiare da quest'ultima affinché ci si possa avvicinare. Più avanti verrà analizzato perché questo approccio non ha del tutto funzionato.

L'algoritmo è descritto di seguito:

3.3 Modello degli errori single-cell

I dati single-cell vengono rappresentati tramite un modello di sostituzione come quello illustrato nella sezione 1.4 tramite una matrice $M = n \times m$, con n cellule e m mutazioni. Viene poi

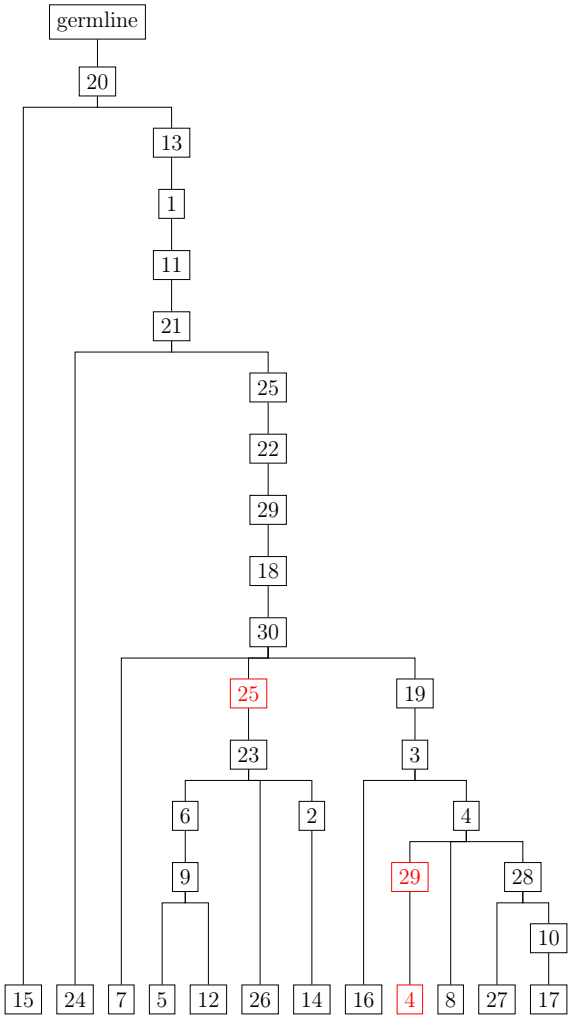


Figura 3.12: Albero inferito con numero di particelle = 200 e numero di iterazioni = 20

utilizzata una matrice $D = n \times m$ è ricavata dall'osservazione dell'albero inferito, ed è una versione imperfetta del vero genotipo della matrice M. Per mitigare i problemi causati dalle tecniche di WGA (sottosezione 1.4.1) viene utilizzato il parametro α per indicare la probabilità di incontrare un falso positivo, quindi osservare un 1 quando in realtà questo è uno 0, e falsi negativi, cioè di avere la probabilità β di osservare uno 0 quando in realtà è un 1:

$$\begin{aligned} P(M_{i,j} = 0 | D_{i,j} = 0) = 1 - \alpha, \qquad & P(M_{i,j} = 0 | D_{i,j} = 1) = \beta, \\ P(M_{i,j} = 1 | D_{i,j} = 0) = \alpha, \qquad & P(M_{i,j} = 1 | D_{i,j} = 1) = 1 - \beta \end{aligned} \tag{3.3}$$

3.4 Considerazioni aggiuntive

Al fine di poter eseguire delle analisi e considerazioni sui risultati ottenuti sono stati fatti diversi accorgimenti sul progetto, di seguito analizzati.

3.4.1 Riproducibilità

Durante lo sviluppo di un software può capitare di dover rappresentare una situazione, come un bug, una feature, o un risultato particolare. Questa necessità dà luogo al problema della *riproducibilità*, che implica la possibilità allo sviluppatore o a terzi di poter eseguire il software con parametri comuni al fine di ottenere gli stessi risultati e situazioni. Questo obiettivo è stato raggiunto con diversi accorgimenti, come l'elevata flessibilità di configurazione da linea di

Algoritmo 6: CasualClades

```
1  distance_particle ← dist(x_i, p_i)
2  distance_swarm ← dist(x_i, g)
3  particle_clades ← get_clades_max_nodes(distance_particle)
4  swarm_clades ← get_clades_max_nodes(distance_swarm)
5  max_clades ← mc
6  if distance_particle < max_clades and distance_swarm < max_clades or
     len(particle_clades) == 0 and len(swarm_clades) == 0 then
7      tree ← x_i
8  else
9      clades_attach ← ()
10     if distance_particle == 0 or len(particle_clades) == 0 then
           ▷ Same tree as the best in current particle
11         pick random max_clades from swarm_clades
12     else if distance_swarm == 0 or len(swarm_clades) == 0 then
           ▷ Same tree as the best in current particle
13         pick random max_clades from particle_clades
14     tree ← x_i
15     for clade in clades_attach do
16         clade_to_attach ← random node from tree
17         attach_clade_and_fix(clade_to_attach, clade)
18     fix_for_losses(tree)
```

comando (descritta in dettaglio nel sottosezione 3.4.2) e la possibilità di poter manipolare gli eventi aleatori fissando un *seed*[5].

3.4.2 Guida allo strumento nello stato attuale

3.4.3 Parallelizzazione

[5]Un numero, o vettore, utilizzato per inizializzare un generatore pseudo-casuale di numeri, come quello utilizzato dalla libreria *random* di Python

Capitolo 4

Risultati e conclusioni

4.1 Prospettive future

Bibliografia

[1] *Cancer Statistics*. URL: https://www.cancer.gov/about-cancer/understanding/statistics.

[2] Simone Ciccolella et al. «Inferring Cancer Progression from Single-cell Sequencing while Allowing Mutation Losses». In: *bioRxiv* (2018). DOI: 10.1101/268243.

[3] *GitHub Profile - IAL32*. URL: https://github.com/IAL32.

[4] Cooper GM. *The Cell: A Molecular Approach. 2nd edition - DNA Replication*. URL: https://www.ncbi.nlm.nih.gov/books/NBK9940/.

[5] Gerhard Jäger. «Global-scale phylogenetic linguistic inference from lexical resources». In: *Scientific Data* 5 (ott. 2018), 180189 EP -. DOI: 10.1038/sdata.2018.189.

[6] Katharina Jahn, Jack Kuipers e Niko Beerenwinkel. «Tree inference for single-cell data». In: *Genome Biology* 17.1 (mag. 2016), p. 86. ISSN: 1474-760X. DOI: 10.1186/s13059-016-0936-x.

[7] J. Kennedy e R. Eberhart. «Particle swarm optimization». In: vol. 4. Nov. 1995, 1942–1948 vol.4. DOI: 10.1109/ICNN.1995.488968.

[8] Clemens Lakner et al. «Efficiency of Markov Chain Monte Carlo Tree Proposals in Bayesian Phylogenetics». In: *Systematic Biology* 57.1 (feb. 2008), pp. 86–103. ISSN: 1063-5157. DOI: 10.1080/10635150801886156.

[9] Makoto Matsumoto e Takuji Nishimura. «Mersenne Twister: A 623-dimensionally Equidistributed Uniform Pseudo-random Number Generator». In: *ACM Trans. Model. Comput. Simul.* 8.1 (gen. 1998), pp. 3–30. ISSN: 1049-3301. DOI: 10.1145/272991.272995.

[10] Max Roser e Hannah Ritchie. *OurWorldInData - Cancer*. Lug. 2015. URL: https://ourworldindata.org/cancer.

[11] Giorgio Stanta e Serena Bonin. «Overview on Clinical Relevance of Intra-Tumor Heterogeneity». In: *Frontiers in Medicine* 5 (2018), p. 85. ISSN: 2296-858X. DOI: 10.3389/fmed.2018.00085.

[12] Hamim Zafar et al. «SiFit: inferring tumor trees from single-cell sequencing data under finite-sites models». In: *Genome biology* 18.1 (2017), p. 178. DOI: 10.1186/s13059-017-1311-2.