Università degli Studi di Salerno

Dipartimento di Informatica

Corso di Laurea Triennale in Informatica

Sviluppo di un tool per la comparazione qualitativa di algoritmi di confronto fra stringhe basati sulle Minimal Absent Words

Relatrice
Prof.ssa Rosalba Zizza

Candidato
Antonio Gravino
Matricola: 05121 07161

Anno Accademico 2021/2022

Alla mia famiglia,
faro in un oceano di incertezze.

Ai miei colleghi universitari,
esempi di vigore e costanza.

A me,
per aver perseverato nonostante le avversità.

E infine al popolo ucraino,
che davanti ad un nemico apparentemente invincibile,
ha dato dimostrazione di grande forza e risolutezza,
rinnovando profondamente il mio spirito europeista.

Indice

Abstract

I confronti fra stringhe sono un campo applicativo critico nell'informatica e, in particolare, nella bioinformatica. In letteratura, nel corso dei decenni sono state proposte numerose soluzioni nella forma di algoritmi più o meno efficienti in grado di misurare in maniera precisa il grado di similitudine fra un insieme di stringhe. Tuttavia, ci si pone il problema di elaborare un algoritmo di confronto fra stringhe che, invece di computare in funzione dell'informazione positiva - ottenuta analizzando la composizione propria e reale delle stringhe in input - sfrutti piuttosto l'informazione negativa intrinseca delle stesse nella forma delle loro *minimal absent words*. Successivamente, si analizza lo stato dell'arte dei tool visuali adibiti alla generazione di grafici illustrativi riguardanti i medesimi confronti, e si propone un'alternativa adatta alle speciali esigenze in materia di studio, sviluppando un applicativo web in grado di accogliere, integrare e computare algoritmi basati sull'informazione negativa in virtù di un'opportuna metrica, e dunque capace di elaborare uno spettro di rappresentazioni grafiche che consentano di effettuare valutazioni qualitative degli algoritmi impiegati.

Capitolo 1

Introduzione

Il confronto fra stringhe è, in generale, un campo di ricerca di largo interesse da parte della comunità scientifica internazionale. Le stringhe risultano una codifica comune per un vasto spettro di oggetti che possono vantare proprietà di confrontabilità. Fra questi, il genoma rappresenta uno dei soggetti principali dei confronti fra stringhe, essendo banalmente codificabile su un alfabeto ben definito, quale l'alfabeto del genoma. L'analisi del genoma è legata strettamente ai confronti con altre sequenze genomiche, motivo per cui la genomica computazionale ne fa una delle sue principali materie di studio. In generale, la codifica in stringa del genoma permette di individuare caratteristiche qualitative e di rilevare similiarità con filamenti differenti, risultando dunque centrale in ambiti di ricerca internazionali e non.

In particolare, in questa tesi viene affrontanto il problema degli algoritmi di confronto fra stringhe su alfabeti del genoma con implementazioni *non-allineate* (o *alignment-free*), e cioè che non manipolano la composizione propria della stringa genomica, al fine di ottenere una risultato quanto più efficiente possibile. D'altro canto, il lavoro di tesi riguarda per lo più l'ambito della comparazione qualitativa fra questi algoritmi di confronto fra stringhe; vengono dunque esplorati tool, più o meno recenti, che permettono di effettuare considerazioni di questa natura. Tuttavia, l'assenza di tool significativamente compatibili con gli algoritmi in esame porterà

alla realizzazione di un applicativo ad hoc per le necessità evidenziate.

Nel Capitolo 2 si introducono nozioni matematiche e cenni teorici fondamentali alla successiva comprensione degli argomenti trattati. Fra le nozioni considerate, le più importanti sono banalmente le *minimal absent words* e la metrica LW, entrambe centrali sia nell'implementazione dell'algoritmo in esame che del tool oggetto di sviluppo; vengono altresì esplorate strutture dati efficienti impiegate in numerose applicazioni nel campo della bioinformatica, come i *suffix arrays*.

Nel Capitolo 3 si analizza con precisione il funzionamento e parte dell'implementazione dell'algoritmo `scMAW`, recente risultato originale di un gruppo di studio che propone una soluzione efficiente circa i confronti fra stringhe impiegando le *minimal absent words*. Si forniscono dettagli sulla costruzione delle strutture matematiche impiegate e sulle modalità di immissione dell'input e restituzione dell'output.

Nel Capitolo 4 si analizza il tool SMART, ideato per ospitare algoritmi di confronto fra stringhe e, in particolare, per individuare il numero di occorrenze di specifici pattern all'interno di testi più o meno voluminosi. Si osserveranno, in particolare, le cause per le quali il tool SMART risulta incompatibile e inoperabile con la specificità degli algoritmi in esame.

Nel Capitolo 5, verrà delineata la realizzazione e lo sviluppo del tool LW Index, che rappresenta il risultato originale di questa tesi di laurea. Verranno fornite specifiche circa i requisiti funzionali dell'applicativo, le scelte progettuali e architettoriali, la selezione delle tecnologie; verrà, inoltre, fornita una panoramica completa sui più importanti componenti dell'implementazione del tool, arricchendo l'esposizione con porzioni di codice ed esempi illustrativi.

Nel Capitolo 6, infine, si effettuano diverse considerazioni sul lavoro svolto e si delineano alcuni possibili sviluppi futuri circa il tool realizzato.

Capitolo 2

Cenni teorici e nozionistici

In questa tesi si assumeranno per scontate le nozioni basilari di sequenza, fattore e fattorizzazione, parola, stringa, alfabeto. Inoltre, si assumerà per scontata l'operazione di differenza simmetrica fra due stringhe.

2.1 Absent Words, Minimal Absent Words

Una *absent word* o *parola assente* (detta talvolta *forbidden word* o *parola proibita*) di una certa sequenza S è tale se essa non occorre nella sequenza S stessa.

Le *absent words* rappresentano, in effetti, la tipologia più genuina di informazione negativa: costituiscono, infatti, un dato che non si presenta in nessuna forma o composizione nella sequenza in esame. In generale, se la sequenza S ha lunghezza n, il numero di *absent words* è al più esponenziale in n.

E' altresì possibile individuare specifiche classi di *absent words*, quali le *minimal absent words*, il cui numero è lineare in n. Le *minimal absent words* sono *absent words* di una sequenza S che non occorrono propriamente nella sequenza, ma di cui - invece - tutte le fattorizzazioni proprie occorrono nella sequenza. Si è dunque davanti ad un contrasto interessante: da un lato informazione negativa cruda e grezza; dall'altro, informazione positiva derivata dalla fattorizzazione di informazione negativa.

$M_L = \{aub | a, b \in \Sigma*, aub \notin L, au, ub \in L\}$

Lo studio di queste fattorizzazioni e la loro applicazione pratica nell'implementazione di scMAW ci permetterà di stabilire un metodo efficiente per confrontare stringhe impiegando, alla base, informazione genuinamente negativa.

Risulta dunque immediato osservare che, impiegando le *minimal absent words* di sequenze S_1 e S_2, è possibile determinarne la similiarità in tempo proporzionale rispetto alla loro lunghezza n_1 e n_2, a patto che le due sequenze appartengano al medesimo alfabeto. In particolare, studieremo l'applicazione delle *minimal absent words* nella cornice dell'alfabeto del genoma $\Sigma_{DNA} = \{A, C, G, T\}$.

Risultati basati sulle Minimal Absent Words

In letteratura, un moderato numero di risultati sono stati prodotti a partire dalla nozione di *minimal absent words*. Si tratta di soluzioni di complessità, generalmente, $O(n)$, sia in termini di spazio che di tempo, posto e fissato un alfabeto preciso, di lunghezza ben definita. Tuttavia, è interressante osservare la moltitudine di strutture dati impiegate nell'implementazione di questi algoritmi: suffix arrays, suffix trees, suffix automata [**CROCHEMORE1998111**], per citarne alcuni. Alcune soluzioni risultano meno performanti di altre, presentando una complessità (nel caso peggiore) pari a $O(n^2)$ [1].

In questa tesi, presenteremo e analizzeremo - nel Capitolo 3 - l'algoritmo scMAW, uno dei più recenti risultati nell'ambito di confronto fra stringhe impiegando le *minimal absent words* e di soddisfacente complessità, sia in termini di tempo che di spazio.

2.2 Suffix Arrays

Come osservato, è possibile introdurre il concetto di minimal absent words impiegando le strutture dati più disparate, come gli automi e i suffix arrays.

Questi ultimi sono centrali nell'implementazione dell'algoritmo in esame, quale `scMAW`.

I suffix arrays, risultato originale del 1990 di Manber-Myers [2], sono un array ordinato di tutti i suffissi di una stringa S. E' una struttura dati ampiamente utilizzata nell'indicizzazione di testi, negli algoritmi di compressione e nel campo della bibliometrica. La loro complessità è, sia in media che nei casi peggiori, $O(n)$, risultando dunque una struttura dati efficiente e di semplice costruzione.

In generale, posta $S = S[1]S[2]...S[n]$ una stringa di lunghezza n denotata dai suoi fattori monocarattere, il suffix array A di S è definito come array di interi costituito dalle posizioni iniziali dei suffissi S in ordine lessicografico. $A[i]$ conterrà, in effetti, la posizione iniziale dell'i-esimo suffisso più piccolo in S. Ogni suffisso compare esattamente una volta. Didatticamente, viene generalmente utilizzata la stringa *banana* per fornire un esempio.

Si ponga $S = banana$. Risulta che $S[1] = b, S[2] = a, S[3] = n, ..., S[6] = a, S[7] = blank$. Il *blank* è generalmente indicato con un trattino basso.

Si considerino i suffissi di S.

Suffisso	**i**
*banana_*	**1**
*anana_*	**2**
*nana_*	**3**
*ana_*	**4**
*na_*	**5**
*a_*	**6**
_	**7**

Ordiniamo i suffissi in ordine lessicografico mantenendo gli indici attuali.

Suffisso	**i**
_	**7**
*a_*	**6**
*ana_*	**4**
*anana_*	**2**
*banana_*	**1**
*na_*	**5**
*nana_*	**3**

Il suffix array A associato conterrà, in maniera associativa, gli indici i soprastanti. Risulterà, dunque, $A[1] = 7, A[2] = 6, A[3] = 4, ..., A[7] = 3$.

E' altrettanto possibile costruire un suffix tree in funzione dei suffix arrays. La costruzione dell'albero è, tuttavia, soggetto alla specificità dell'algoritmo che lo sfrutta.

2.3 Range Minimum Queries

Una range minimum query [3] permettere di trovare il valore minimo in una sottostringa di un array di oggetti confrontabili (nel nostro caso, numeri interi). Sono generalmente utilizzati lateralmente agli LCP (Longest Common Prefix) arrays, quando necessario.

Dato un array $A[1...n]$ di n oggetti confrontabili e ordinati, il range minimum query $RMQ_A(l, r)$ coincide con $arg_{min}A[k]$, con $1 \leq l \leq k \leq r \leq n$. La funzione restituisce la posizione dell'elemento più piccolo nel sotto-array determinato da $A[l...r]$.

Ad esempio, per l'array $A = 0, 5, 2, 5, 4, 3, 1, 6, 3$, risulterà che il range minimum query di $A[3...8] = 2, 5, 4, 3, 1, 6$ è 7, poiché $A[7] = 1$ ne é il valore più piccolo, dunque il minimo, ottenuto tramite confronti più o meno lineari.

La nozione di range minimum queries è, analogamente a quella di suffix arrays, centrale alla costruzione dell'algoritmo `scMAW`.

2.4 Metrica LW di Crochemore

Chairungsee e Crochemore del King's College di Londra hanno presentato, nel 2012, una misura di similiarità fra due stringhe x e y impiegando le *minimal absent words* [4] di ciascuna delle due, rispettivamente M_x^l e M_y^l, dove l è la lunghezza massima fra le due stringhe.

LW Index sta per *Length-Weighted Index*.

La metrica, che restituisce un float, è calcolata come sommatoria di $\frac{1}{|w|^2}$ dove ogni w è una parola ottenuta dalla differenza simmetrica fra le minima absent words di x e di y.

$$LW_l(x, y) = \sum_{w \in M_x^l \Delta M_y^l} \frac{1}{|w|^2}$$

Ricordiamo, inoltre, che la *differenza simmetrica* fra due insiemi X e Y è definita come unione fra la differenza tra il primo insieme e il secondo insieme, e la differenza fra il secondo insieme e il primo insieme. Si osservi che il motivo per cui la metrica fa uso dell'inverso delle stringhe è proprio a causa della differenza simmetrica stessa.

In generale, nell'implementare l'algoritmo `scMAW` considereremo una più generica metrica LW che non tiene conto della lunghezza l.

$$LW(x, y) = \sum_{w \in M_x \Delta M_y} \frac{1}{|w|^2}$$

Ad esempio, posto $x = abaab$ e $y = aabbbaa$, risulterà che $M_x = aaa, aaba, bab, bb$ (osserviamo che tutte le fattorizzazioni di M_x sono presenti in x) e $M_y = aaa, bbbb, aba, abba, bab, baab$.

Risulterà:

$M_x \Delta M_y = aaba, aba, abba, baab, bb, bbbb$

Di consequenza, il soprastante risultato verrà dato in pasto alla som-

matoria produttrice della metrica LW. Ogni lunghezza (inversa, ed elevata al quadrato) degli elementi dell'insieme (*aaba*, *aba*, ...) sarà sommata, al fine di produrre la metrica.

$LW(x, y) = \frac{1}{4^2} + \frac{1}{3^2} + \frac{1}{4^2} + \frac{1}{4^2} + \frac{1}{2^2} + \frac{1}{4^2}$

Questa misura permette di quantificare la distanza fra due parole, definendone la similitudine. Risulta, infatti, $LW(x, x) = 0$.

Capitolo 3

Analisi dell'algoritmo scMAW

Nel 2018, un gruppo di studio internazionale composto da studenti del King's College di Londra, dell'Università di Palermo e dell'Universitità di Loughborough (Regno Unito), propose una soluzione efficiente ed originale in merito al problema del confronto fra stringhe impiegando le *minimal absent words*, la cui nozione è descritta nel Capitolo 2.

Al tempo della stesura del paper relativo all'algoritmo in questione [5], gli algoritmi più efficienti, generalmente di complessità $O(n)$ in media e $O(n^2)$ nei casi peggiori, impiegavano strutture dati variegate, alcune più o meno adatte a trattare l'informazione negativa rappresentata dalle *minimal absent words*.

Analogamente, il gruppo di studio decise di sfruttare una metrica originale, precedentemente introdotta da Chairungsee e Crochemore [6], chiamata Metrica LW, o Indice LW.

L'indice permette di definire la similiarità, o similitudine, fra due stringhe - generalmente su un alfabeto ben definito, come quello genomico o proteico - in funzione delle loro *minimal absent words*, ed in particolare sulla loro differenza simmetrica, generalmente calcolabile con complessità $O(m + n)$ in media, con m e n le lunghezze delle stringhe in input.

La prima sfida posta dall'algoritmo giaceva nella ricerca delle *minimal absent words* delle stringhe oggetto di confronto. Il gruppo di studio decise

di utilizzare soluzioni ben note, delineate in [6].

Questo risultato accademico dimostra che è possibile computare, date due stringhe x e y, le corrispettive *minimal absent words* M_x e M_y in tempo lineare relativo alle lunghezze delle stringhe, rispettivamente n e m.

La strategia fondamentale dell'algoritmo, d'ora in poi denominato in questo Capitolo come `scMAW`, è di effettaure una *merge* dei due insiemi M_x e M_y, unendoli dopo che i due insiemi sono stati ordinati tramite l'ausilio dei suffix arrays.

A questo fine, verrà costruito il suffix array associato alla parola $w = xy$. Questa struttura può essere costruita con complessità di tempo e spazio $O(m + n)$. Attraverso questa struttura dati, è immediato ordinare in maniera lessicografica (e cioè tramite i suffissi delle parole associate) i due insiemi di *minimal absent words*. Questo significa che gli insiemi sono considerati associati se, posti x_1 e x_2 tuple di uno degli insiemi, $x_1 < x_2$ se la prima lettera dopo il *longest common prefix* di x_1 è lessicograficamente più piccola di quella di x_2.

In generale, il *longest common prefix* è una struttura prodotta a partire dal preprocessamento di un suffix array. E' intuitivo individuarne la relazione: i suffix array lavorano sui suffissi, che in maniera laterale possono condurre al calcolo dei prefissi.

3.1 Computazione dell'insieme $M_{x,y}$ e di $LW(x, y)$

Una volta ordinati M_x e M_y, è possibile procedere con la *merge*. Si tratta di un'operazione elementare. L'obiettivo è costruire l'insieme $M_{x,y} = M_x U M_y$ utilizzando due indici incrementali i e j, tali che l'insieme non contenga ridondanze e sia, al termine della computazione, ordinato. L'insieme è dunque definito come segue.

$$\begin{cases} |M_{x,y}U\{x_i\}| \text{ e incrementa } i, \text{ se } x_i < y_j \\ |M_{x,y}U\{y_j\}| \text{ e incrementa } j, \text{ se } x_i > y_j \\ |M_{x,y}U\{x_i = y_j\}| \text{ e incrementa } i, j, \text{ se } x_i = y_j \end{cases}$$

Contemporaneamente alla costruzione dell'insieme $M_{x,y}$ è possibile costruire l'indice delle similitudini fra le stringhe calcolando $LW(x, y)$ e seguendo l'incremento degli indici i e j. La costruzione di tale valore è l'obiettivo finale dell'algoritmo, nonché l'output della sua implementazione. E' banale osservare che il calcolo produrrà un *float*.

$$\begin{cases} \left| LW(x,y) + \dfrac{1}{|x_j|^2} \right| \text{ e incrementa } i, \text{ se } x_i < y_j \\ \left| LW(x,y) + \dfrac{1}{|x_j|^2} \right| \text{ e incrementa } j, \text{ se } x_i > y_j \\ |LW(x,y)| \text{ e incrementa } i, j, \text{ se } x_i = y_j \end{cases}$$

E' interessante osservare che quando $x_i = x_y$, l'indice $LW(x, y)$ non incrementa. In alternativa, quando $x_i \neq x_y$, l'indice incrementa in funzione della composizione propria della parola, delineata in base al quadrato della sua cardinalità.

I calcoli elementari nell'algoritmo, come le somme, hanno complessità costante; vengono, altresì, effettuate per ogni tupla in M_x e M_y: è dunque possibile concludere che l'intera operazione richieda $O(m + n)$

E' importante sottolineare che questo algoritmo è disponibile in una controparte dedicata alle sequenze circolari; tuttavia, questa tesi si limiterà ad analizzare unicamente il risultato ottenuto per le sequenze lineari.

3.2 Implementazione e versioning

Le implementazioni di `scMAW` utilizzano, per la ricerca delle *minimal absent words* e il conseguente popolamento degli insiemi M_x e M_y, algoritmi già noti, a cui ci si riferirà come `MAW`.

`scMAW` è stato implementato nel linguaggio C.

Ammette, come input, un file in formato FASTA (o MultiFASTA) con una formattazione ben precisa, definita nel Capitolo 5, ed un parametro che indica se valutare le sequenze in input come lineari o circolari.

Restituisce, come output, un file (generalmente FASTA.OUT, oppure PHYLIP) formattato come altrettanto descritto nel Capitolo 5.

In generale, il suddetto file d'output presenterà una matrice, dove ogni cella $[x, y]$ denoterà il corrispettivo indice $LW(x, y)$. Il file risulta di lettura più o meno immediata, consentendone dunque il processamento da parte di un software come il tool LW Index, la cui realizzazione originale è ampiamente descritta nel Capitolo 6.

Il codice sorgente di `scMAW` è disponibile su GitHub [7].

La root della repository contiene, per lo più, file di installazione e compilazione (Makefile, pre-install.sh, INSTALL), informativi e/o di copyright (LICENSE, README), e collezioni di strutture dati e funzioni generiche impiegate negli algoritmi, fra cui `MAW` (stack, maw). La root contiene anche diverse subdirectory, fra cui una cartella denominata `scMAW`, che contiene l'implementazione dell'omonimo algoritmo in esame.

La directory `~root/scMAW` contiene due subfolders, `data` ed `exp`. La prima cartella contiene alcuni file FASTA di esempio, popolate con stringhe casuali sull'alfabeto del genoma. La seconda, invece, contiene i corrispettivi file di output generati a partire dai file in input nella prima cartella.

La cartella principale, invece, è popolata con numerosi file, per lo più di installazione. Gli unici file relativi al risultato originale sono `functions` e `mawdefs`. Il secondo file è fondamentalmente uno snapshot a basso livello delle funzioni impiegate, elencando esclusivamente le firme delle funzioni stesse.

Il contenuto del primo file, d'altro canto, tratta di funzioni come `getMaws(..)`,

per computare e popolare gli insiemi M_x e M_y, e `maw_seq_comp(..)`, per l'effettivo computo degli indici $LW(x, y)$. La maggior parte delle funzioni sono per lo più ausiliari e di scarso interesse. Le restanti, come la creazione del file di output e la stampa dei risultati, hanno generalmente lo scopo di coadiuvare il computo (es. predisporre le strutture dati necessarie) e/o occuparsi del pre-computo o post-computo (es. leggere il file di input o generare il file di output).

La codifica degli algoritmi e il corrispettivo in C si assesta sull'ordine delle migliaia di righe e non risulta di conseguenza opportuno includerne porzioni decontestualizzate in questo documento; si invita, dunque, a consultare, in caso di interesse, la repository che ne deposita il codice sorgente.

Capitolo 4

Analisi del tool SMART

Lo String Matching Algorithm Research Tool, da qui in poi SMART [8], è un tool dedicato ai ricercatori e a gli studenti di algoritmi di confronto fra stringhe. Il tool offre un ambiente versatile, scalabile e facilmente integrabile con nuovi algoritmi per effettuare considerazioni qualitative nella cornice dei problemi derivati dalla più ampia materia del *pattern matching*. In generale, SMART propone un framework di lavoro adatto a testare, progettare, valutare e comprendere le soluzioni del problema del confronto fra stringhe.

Il tool è stato sviluppato da un gruppo accademico internazionale [9], il quale ha successivamente rilasciato il codice sorgente dell'applicativo su GitHub in una repository pubblica. Il tool ha successivamente ricevuto una moderata documentazione al fine di coadiuvare i futuri sviluppatori che intendessero estendere le funzionalità del tool o, più banalmente, integrare nuovi algoritmi di confronto fra stringhe [10].

E' possibile utilizzare SMART in ambiente Windows, MAC OS X e Linux. Per semplicità, si è deciso di utilizzare una macchina virtualizzata montata con Ubuntu 64bit. L'ambiente di lavoro *barebones* e Unix-based ha permesso al tool di lavorare al massimo dell'efficienza e senza alcun disguido di natura tecnica, che avrebbero potuto aver luogo in contesti non-Unix, come Windows e MAC OS X, soprattutto in termini di compatibilità con librerie e dipendenze di cui SMART fa ampiamente uso.

4.1 Funzionamento di SMART

Figura 4.1: Schermata di benvenuto del tool SMART.

Ad alto livello, il tool SMART:

- Permette di selezionare al più N algoritmi $A+$, dove:
  - $A+$ è un algoritmo che sia testato, funzionante e *valido*. Un algoritmo A non testato, funzionante e valido non può essere selezionato. Una volta testato tramite gli appropriati strumenti offerti dal tool, un simbolo + viene utilizzato per indicare la traslazione di stato da algoritmo non selezionabile A ad algoritmo selezionabile $A+$.
  - N è il numero di algoritmi disponibili. In particolare, N non coincide necessariamente con i sorgenti degli algoritmi presenti nel filesystem del tool, bensì con il numero di algoritmi presenti nella cartella `~root/sources/algos` al momento della compilazione e del bundling tramite `Makefile`. Il processo è descritto nel dettaglio in seguito.
- Permette di selezionare uno o più testi T su cui lanciare la computazione. I testi, incapsulati in file TXT, sono fra i più variegati:

  - Testi accademici o letterari scritti in linguaggio naturale e in lingue diverse. Di default, SMART offre testi in lingua italiana, inglese, francese e cinese. Si tratta di testi narrativi più o meno lunghi, oppure di articoli accademici di dominio pubblico. La lunghezza di questi testi si aggira sulle centinaia di migliaia di parole.
  - Descrizioni del genoma. Insiemi di stringhe su alfabeti DNA o PROT.

- Permette di selezionare l'upper bound e il lower bound dei pattern da riscontrare nelle sequenze. In generale, SMART confronta le stringhe in input in funzione dei loro pattern costituenti: è possibile, dunque, specificare una lunghezza minima e massima dei suddetti pattern; all'esecuzione, il tool incrementerà la lunghezza dei pattern in maniera esponenziale (ad esempio, potrebbe iniziare confrontando le stringhe con pattern di lunghezza 2, poi 4, 8, 16, 32 e così via). E' anche possibile determinare specifiche regioni di testo sulla quale ricercare i pattern.

- Una volta selezionato almeno un algoritmo $A+$ e almeno un testo T, ed eventualmente aver definito parametri limitativi sui pattern da rilevare nelle stringhe accolte, sarà possibile lanciare l'applicativo. Il terminale verrà popolato da barre di caricamento sufficientemente descrittive che segnaleranno l'avanzamento della computazione. Una volta conclusa, SMART indicizzerà l'esperimento con l'attuale *timestamp* seguendo una nomenclatura precisa (es. $EXP20220203$).

  Il tool creerà una cartella in `/results` col nome dell'esperimento e inserirà in quest'ultima i risultati ottenuti sotto forma di alcune pagine HTML elaborate con CSS e JavaScript. La pagina conterrà, essenzialmente, dei grafici informativi sulla computazione appena effettuata, dando spazio a considerazioni di natura qualitativa e comparativa degli algoritmi $A+$ selezionati in funzione dei testi T impiegati.

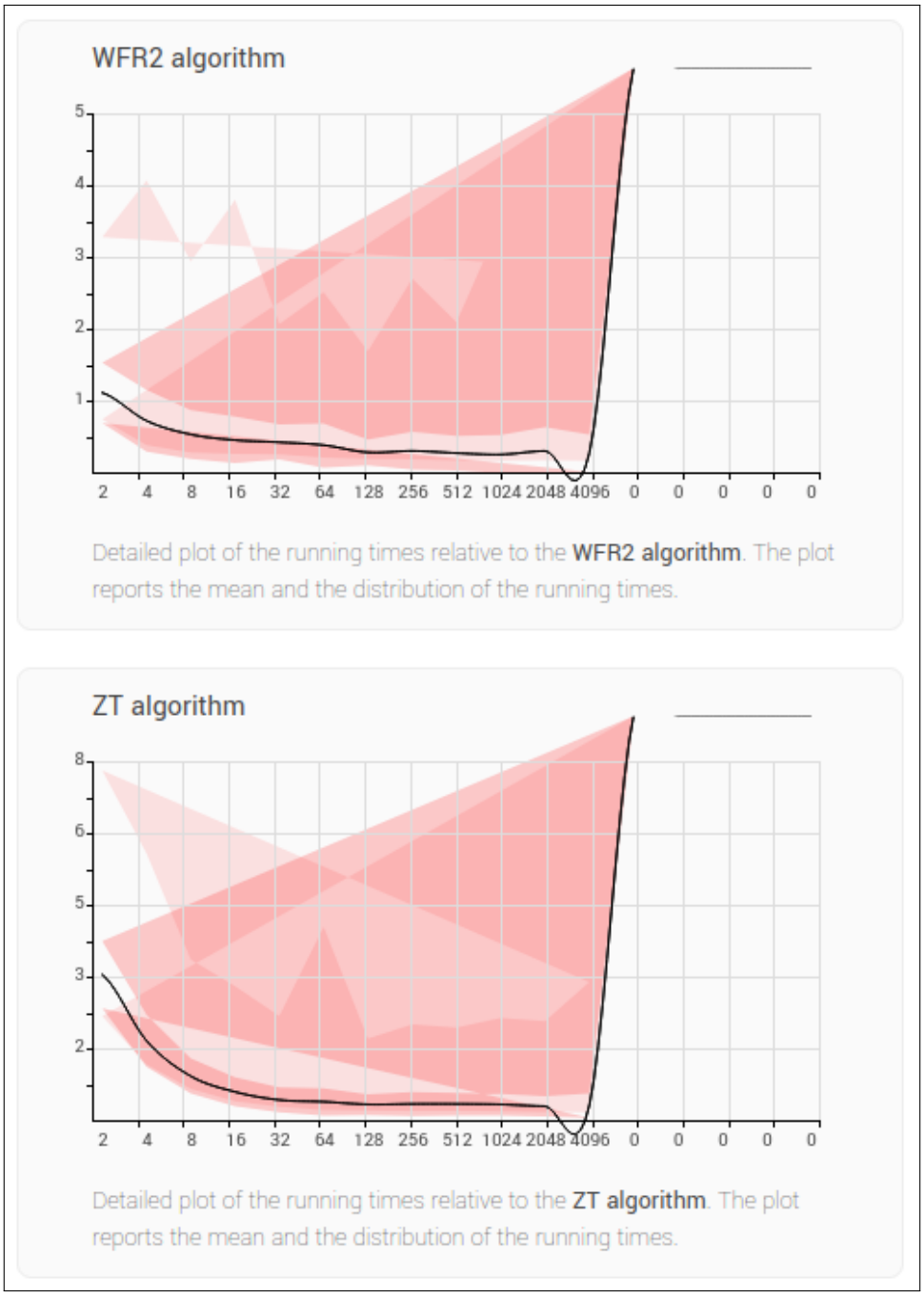

Figura 4.2: Dettaglio di un grafico sulla *Distribution* generato da SMART relativo a due algoritmi: WFR2 e ZT.

4.1.1 Installazione e compilazione

Analizzando il codice sorgente di SMART su GitHub [11] è possibile delineare il funzionamento sia ad alto livello che a basso livello del tool. In generale, gli sviluppatori hanno previsto un processo di installazione immediato: è sufficiente infatti clonare la repository tramite GitHub ed avviare

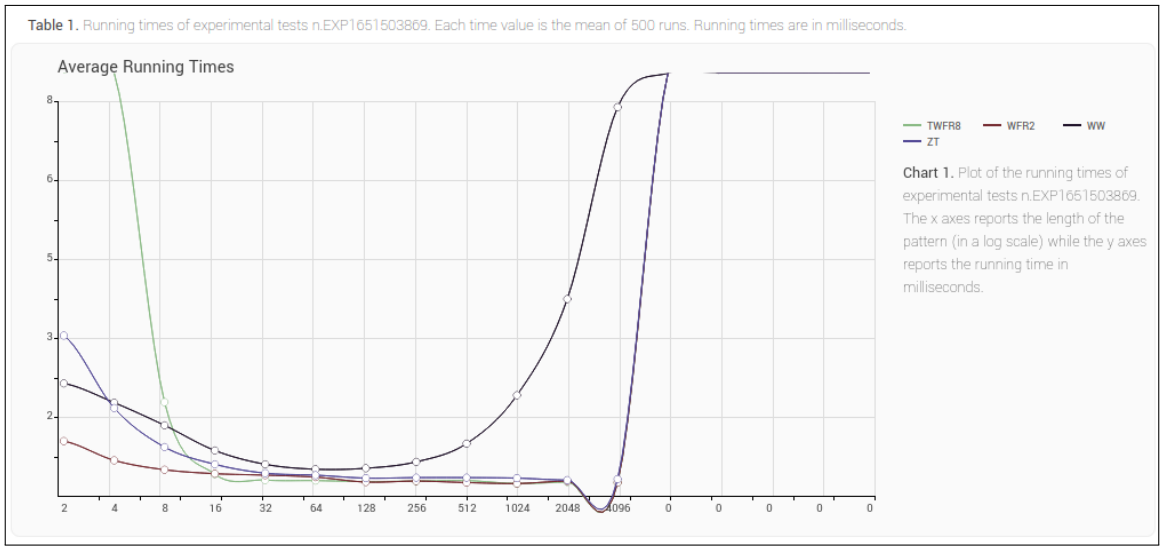

Figura 4.3: Dettaglio di un grafico sull'*Average Running Time* generato da SMART.

il processo di compilazione e bundling tramite `Makefile`:

```
https://github.com/smart-tool/smart.git
cd smart-tool
./makefile
```

E' importante osservare che è necessario ricompilare tramite `Makefile` ogni qual volta s'intende integrare un nuovo algoritmo. Infatti, il processo di bundling è esteso alle sotto-directory della `~root`, inclusa la folder `/bin`, che conterrà i file binari di output degli algoritmi. In generale, il filesystem di SMART è così strutturato:

4.1.2 Struttura del filesystem

- `~root`: Contiene una serie di eseguibili extension-less lanciabili tramite terminale, fra cui `./SMART` e `./SELECT`. Un ulteriore eseguibile è disponibile per verificare la correttezza dei nuovi algoritmi da integrare nel tool.
  - `./SMART`: E' il core dell'applicativo. Da linea di comando, ammette un insieme di opzioni che permettono di descrivere l'input del programma, l'alfabeto utilizzato, la lunghezza massima dei pat-

tern da ricercare nelle stringhe, nonché ulteriori flag opzionali per confronti più specializzati.

– `./SELECT`: analogamente all'eseguibile SMART, ammette una serie di opzioni che hanno l'obiettivo di determinare gli algoritmi da utilizzare quando si lancerà il prossimo confronto con SMART. I comandi ammessi, fra gli altri, sono `-show` (stampa un elenco esaustivo degli algoritmi selezionabili), `-none` (deseleziona tutti gli algoritmi), `-all` (seleziona tutti gli algoritmi) e `[ALGO_NAME]` (seleziona uno specifico algoritmo, a patto che sia presente nella cartella `/algos`). Il comando `-which` permette di identificare gli algoritmi attualmente selezionati e in procinto di essere computati con la prossima invocazione del comando `./SMART`. Infine, il comando `-add` permette di integrare nuovi algoritmi nel tool: è un comando centrale nel funzionamento ad alto livello di SMART, poiché permette a sviluppatori esterni di estenderne le prospettive computative.

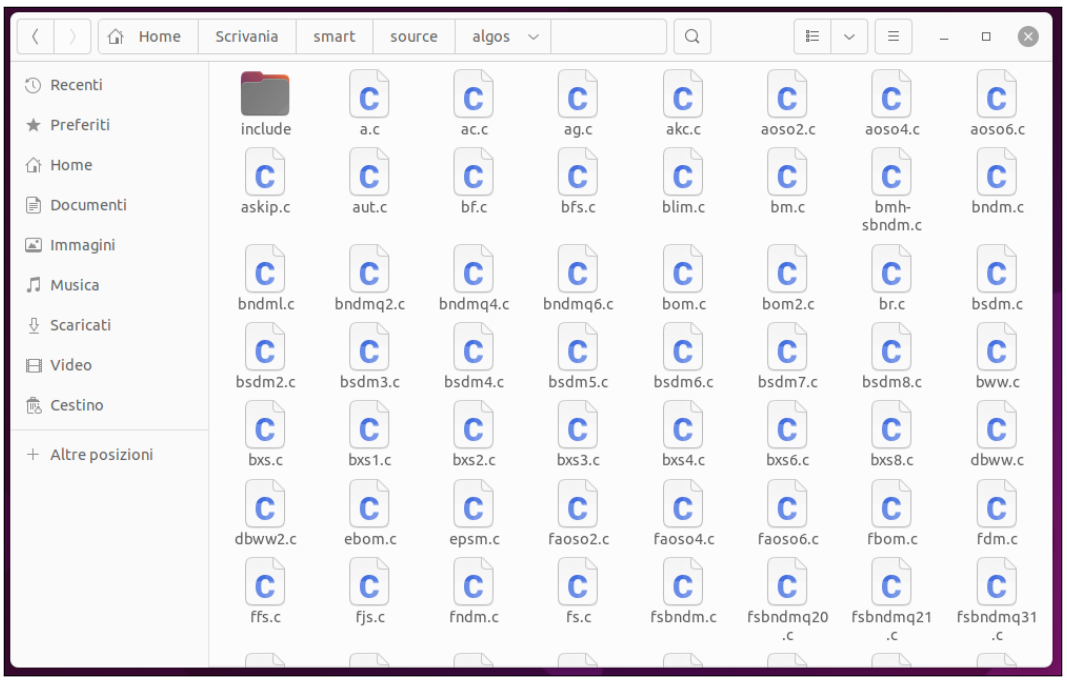

Figura 4.4: Contenuto della cartella ALGOS. L'anteprima non mostra la presenza di una sottocartella BIN, motivo per cui è immediato assumere che l'attuale istanza di SMART non sia stata ancora compilata tramite MAKEFILE.

La root del filesystem contiene anche file di copyright e guide all'installazione del software.

- `/data`: contiene una serie di sottocartelle descrittive dei possibili testi da fornire in input al programma. Ad esempio, la cartella `/data/englishTexts` contiene `bible.text` e `index.txt`. Il primo conterrà il testo grezzo in UTF-8; il secondo conterrà alcune informazioni di indicizzazione del testo per permettere all'applicativo di utilizzarlo come input. Di default, infatti, il testo in inglese di riferimento è la trascrizione della Bibbia; invece, i testi in italiano di riferimento sono, fra gli altri, l'Orlando furioso, la Divina Commedia e le Ultime lettere di Jacopo Ortis. Il file `data/italianTexts/index.txt` sarà così strutturato:

1: data/italianTexts/index.txt

```
La divina commedia
Dante Aligheri (1307)
[url della fonte]
#la_divin.txt#
551,9 kB

Orlando furioso
Ludovico Ariosto (1532)
[url della fonte]
#orlando_.txt#
1,5 MB

Ultime lettere di Jacopo Ortis
Ugo Foscolo (1817)
[url della fonte]
#ultime_l.txt#
281,2 kB
```

- `/results`: contiene tante sottocartelle quanti gli esperimenti (eg. *computazioni*) elaborati tramite SMART. I nomi delle cartelle seguono una nomenclatura univoca, come descritto in precedenza, e associano ad ogni esperimento un preciso identificativo, determinato dal timestamp.

- `/source`: contiene i file sorgenti C del core del software (ad esempio `./smart.c` per l'eseguibile `./SMART/`) e degli algoritmi di confronto fra stringhe disponibile. Circa 80 algoritmi sono disponibili di default, e nuovi sono integrabili. In particolare:

  – La cartella `/algos` contiene i file sorgenti C degli algoritmi, strut-

turati in modo specifico al fine di poter essere compatibili col tool. Il `Makefile` di SMART compilerà questi file sorgente C in binari di output.

– La cartella `/bin` contiene i file binari risultato della compilazione dei sorgenti in C di `/algos`. In effetti, questi file binari sono esattamente quelli utilizzati durante i processi computazionali del tool.

4.2 Integrazione di nuovi algoritmi

Per aggiungere un nuovo algoritmo al tool SMART, è necessario creare un file sorgente in C nella cartella `~root/source/algos` e notificare il tool della sua presenza tramite `./select -add [ALGO_NAME]` dopo averlo opportunamente testato e validato tramite le strumentazioni automatiche del tool.

Il contenuto del file C dev'essere compatibile con il modo di computare i confronti fra stringhe e le ricerche di pattern di SMART. L'algoritmo può utilizzare *include* esterni, come librerie e altre forme di dipendenze programmatiche. Il file sorgente può includere una serie di funzioni ausiliare per favorire il riuso e la leggibilità, oppure per motivi di iterazione o ricorsione. Tuttavia, SMART prevede che almeno una funzione (definita come tale) sia presente nel sorgente C, e che questa funzione abbia una firma ben definita.

```
28
29 int search(unsigned char *x, int m, unsigned char *y, int n) {
30    unsigned int D, B[SIGMA], M, s;
31    int i,j,q, count, first;
```

Figura 4.5: Firma di un algoritmo predefinito di SMART. Il contenuto della funzione è irrilevante, ma la firma è cruciale.

In particolare, la funzione in questione, denominata *search*, deve restituire un intero e accogliere esattamente tre parametri nel seguente ordine:

- `unsigned char *x`, pattern da ricercare all'interno del testo $*y$

- `int *m`, lunghezza del pattern `*x`
- `unsigned char *y`, testo sul quale ricercare il pattern `*x`
- `int *n`, lunghezza del testo `*n`

Se l'algoritmo non trova alcuna occorrenza del pattern, l'algoritmo dovrà restituire -1. In alternativa, dovrà **restituire *esattamente* il numero di occorrenze del pattern** $*x$ nel testo $*y$.

Il contenuto della funzione è irrilevante, a patto che rispetti i requisiti soprastanti. Questo significa che è possibile usare qualsiasi struttura dati, tecnica algoritmica e metodo di computazione per arrivare all'intero richiesto. La complessità dell'algoritmo è altresì irrilevante e non vincolante all'implementazione o all'integrazione con SMART; sarà difatti compito del tool effettuare le dovete comparazioni degli algoritmi.

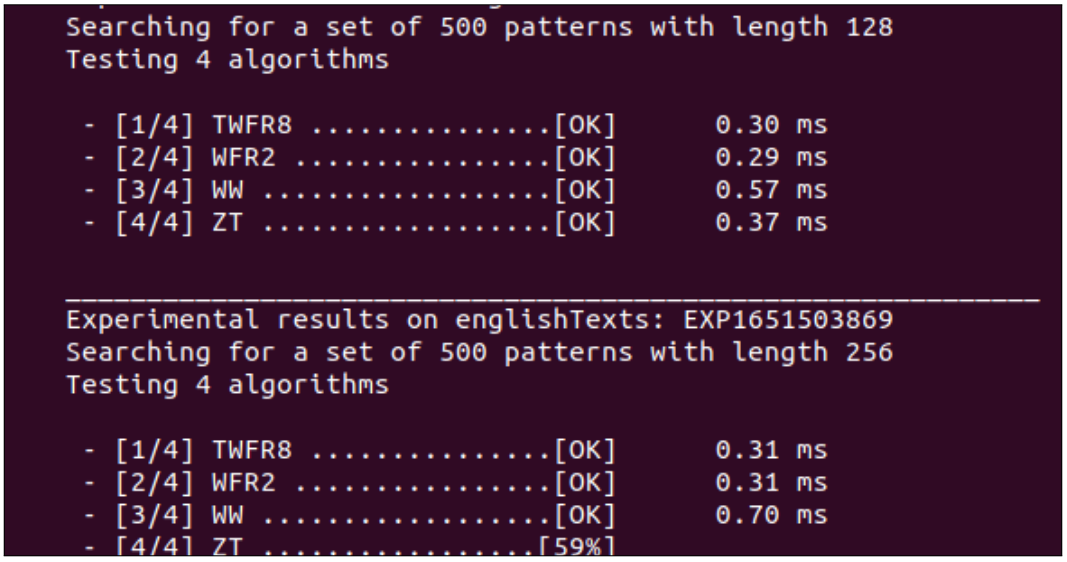

Figura 4.6: Porzione di una fase del processo di computazione di SMART. In questa anteprima, SMART sta computando quattro algoritmi: TWFR8, WFR2, WW e ZT. In particolare, sta ricercando un preciso numero di pattern (500) di lunghezza esponenziale (128, 256). Ogni riga presenta il tempo richiesto in millisecondi.

E' possibile validare l'algoritmo attraverso il comando `./TEST [ALGO_NAME]`. Il tool tenterà di eseguire una computazione su parametri predefiniti, se il numero di occorrenze restituito dall'algoritmo testato coinciderà con l'intero che il tool si aspetta, allora l'algoritmo sarà considerato valido, testa-

to e pienamente funzionante; sarà possibile, a questo punto, integrarlo e selezionarlo tramite `./SELECT`.

4.3 Accenni al versioning parallelo di SMART-GUI

Gli autori del tool SMART, per venire incontro ad esigenze di usabilità, hanno deciso di sviluppare una versione parallela che, invece di poter essere utilizzata esclusivamente da linea di comando, sia invece adoperabile tramite un'interfaccia grafica. Questa versione è stata denominata SMART-GUI ed è disponibile su GitHub [12].

Il funzionamento è del tutto analogo alla sua controparte da terminale. L'unica sostanziale differenza è, appunto, l'introduzione di un'interfaccia grafica, che sostituisce i comandi principali dell'applicativo da CLI, come `./SMART` e `./SELECT`, adesso integrati in un'unica finestra sotto forma di riquadri di input, tabs e bottoni interattivi.

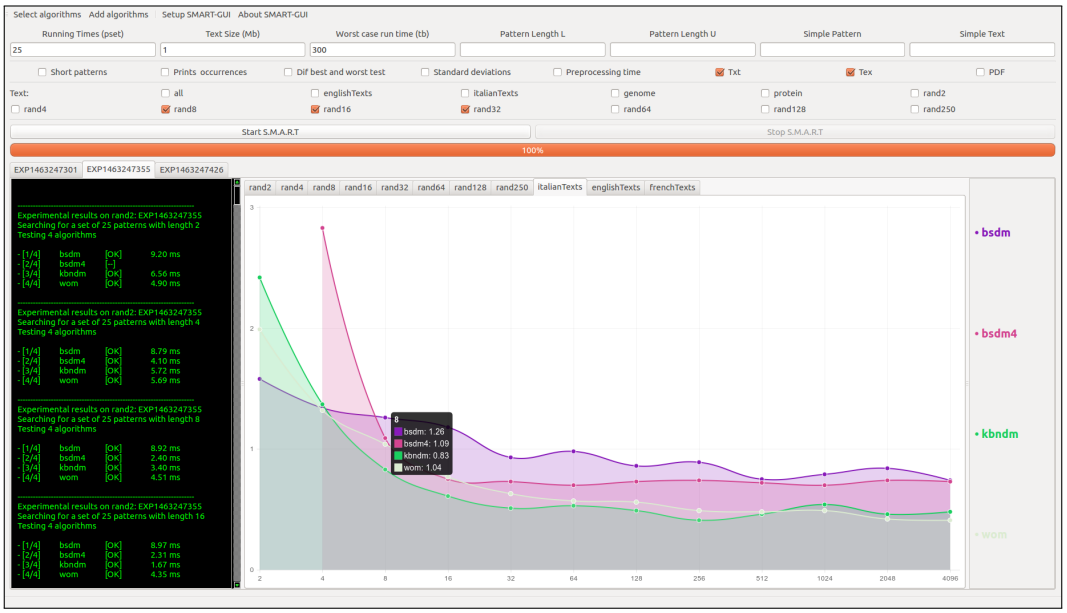

Figura 4.7: Anteprima di SMART-GUI.

La schermata centrale propone i grafici prodotti dal tool a serguito di un processo computativo. Il lato sinistro, invece, simula le fasi computative

proprio come la controparte da terminale del tool. Ulteriori campi di input permettono di specificare i parametri più variegati.

Tuttavia, l'autore di questa tesi ha individuato empiricamente alcuni problemi di compatibilità relativi alle librerie impiegate per renderizzare l'interfaccia grafica, fra cui `QtLibrary` e `libqtwebkit4`. Potrebbe essere estremamente difficoltoso, o comunque lento e poco efficiente, installare questa versione SMART dotata di interfaccia grafica, per l'utente medio. Numerosi problemi relativi a versioni non più supportate o obsolete delle dipendenze in uso rendono, dunque, la versione da terminale di SMART quella più mantenibile, robusta e funzionante, che risulta altresì immediata e semplice da installare ed utilizzare.

4.4 Conclusioni e criticità del software

Inizialmente, il tool SMART è stato analizzato e studiato come possibile candidato ad ospitare algoritmi come `scMAW`. In generale, si parla di algoritmi che sfruttano le *minimal absent words* e la metrica LW per indicare la similitudine fra le stringhe oggetto di confronto. Dopo un'attenta analisi e a seguito delle dovute considerazioni, risulta che SMART non sia adeguato né per `scMAW`, né per tutti gli algoritmi che, analogamente a quest'ultimo, forniscono matrici di metriche LW come output.

E' banale osservare come `scMAW` e SMART, in relazione a quanto analizzato in questo Capitolo e nei Capitolo 2 e 3, non possa essere integrabile nel tool secondo i metodi previsti dagli autori. Difatto, come delineato nel Capitolo 4.2, gli algoritmi in via di integrazione devono seguire schemi ben precisi e non posso, almeno nella forma dei parametri accolti e dei tipi restituiti, divergere dalle condizioni vincolanti della piattaforma. Nello specifico, SMART impone la restituzione del numero di occorrenze di un certo pattern in un determinato corpo di testo. Invece, `scMAW` prevede la generazione di un risultato di più ampio respiro, quale un file FASTA.OUT

(descritto nel Capitolo 5), contenente indici LW per insiemi ben definiti di stringhe.

Risulta fondamentale osservare come SMART agisca in funzione del numero di occorrenze dei pattern risoluti, mentre gli algoritmi in esame - proprio perché sfruttano l'informazione negativa (nella forma di *absent words* e simili) piuttosto che quella positiva - sono esclusivamente in grado di fornire un grado di similitudine fra gli elementi input, restringendoli fra l'altro ad un alfabeto specifico, quale quello del genoma.

Risulta evidente che SMART non sia stato pensato per accogliere algoritmi di questa tipologia. L'implementazione di `scMAW` non può essere adeguata alla forma richiesta dal tool, né sarebbe opportuno tentare di convertire output così distanti e sconnessi fra loro per tentare di assecondare i vincoli del software.

Per questi motivi, oltre che per motivi di natura didattica, è stato deciso di realizzare un software ad hoc per integrare algoritmi come quello in esame.

In particolare, il tool in sviluppo dovrà essere in grado di elaborare algoritmi che restituiscono un file FASTA.OUT, prendendone in input uno di tipo FASTA e ammettendo esclusivamente l'alfabeto del genoma. Il tool dovrà, similmente a SMART, elaborare dei grafici illustrativi per permettere non solo considerazioni di tipo comparativo sulla qualità degli algoritmi selezionati, ma anche sulla loro correttezza e robustezza in termini degli indici LW restituiti a seguito dell'elaborazione.

Capitolo 5

Realizzazione del tool LW Index

In virtù delle mancanze del tool SMART delineate nel Capitolo 5, si è deciso di procedere alla realizzazione di un'applicativo web ad hoc per sopperire alle lacune del suddetto tool.

Il tool in sviluppo, denominato LW Index, in riferimento all'omonima metrica presentata nel Capitolo 2, è definito come SMART-like, o simil-SMART. Il tool SMART permetteva di confrontare una serie di algoritmi di confronto fra stringhe, a prescindere dalla loro natura, e generare conseguentemente dei grafici predisposti a valutazioni di natura qualitativa da parte dell'utente.

Benché sia disponibile in una versione parallela adoperante un'interfaccia grafica, SMART nasce come programma da terminale; nella fattispecie, come si è osservati nel Capitolo 4, è in grado di produrre - a partire da una collezione di stringhe (o più banalmente un testo) e uno spettro di algoritmi selezionati - una pagina web costruita tramite elementi di HTML, CSS e JavaScript.

D'altro canto, l'interazione dell'utente nasce e muore nel contesto del terminale: difatto, la pagina web autogenerata da SMART è di mera consultazione e non prevede ulteriore interazione con l'utente.

L'obiettivo è dunque di creare un tool che riesca ad accogliere algoritmi basati sulla metrica LW, incompatibili coi metodi di elaborazione e computazione di SMART; si noti, in ogni caso, che la metrica LW e le *minimal*

absent words sono strettamente legate, motivo per cui il tool è stato pensato specificamente per algoritmi che, oltre a misurare la similitudine con la metrica LW, facciano uso di *minimal absent words* e derivati nella loro composizione funzionale.

5.1 Progettazione del tool

Si osservi che, d'ora in poi, l'espressione "tool in sviluppo" sarà utilizzata in maniera intercambiabile col suo proprio nome, quale LW Index.

5.1.1 Caratteristiche e funzionalità ad alto livello

In generale, il tool in sviluppo permetterà di creare grafici (al più 4) in maniera del tutto simile al tool SMART; svuotare la tavola di lavoro (cioè distruggere i grafici–tutti o individualmente) e analizzare i grafici. Creare un grafico significa, ad alto livello, generare un'istanza di lavoro vuota, popolabile con dati e stringhe durante la fase di analisi.

Analizzare i grafici significa porre in evidenza uno dei grafici fra quelli creati sulla tavola di lavoro ed effettuare una serie di azioni, fra cui visionare, in maniera grafica o testuale, il processo computazionale svolto per arrivare a quei dati, e modificare i grafici aggiungendo, rimuovendo o alterando stringhe.

E' importante sottolineare che modificare il grafico significa necessariamente creare una nuova richiesta computativa.

Sarà possibile cambiare algoritmo. Il tool sarà sufficientemente versatile da accogliere un insieme diversificato di algoritmi che hanno in comune l'impiego della stessa metrica di similitudine fra stringhe, quale l'indice LW.

I grafici rappresenteranno sulle ascisse le stringhe (su alfabeti ben definiti) e sulle ordinate il corrispettivo indice LW. Questo significa che, per

ogni stringa, il grafico produrrà una curva distinta a cui assocerà un indice LW per ogni altra stringa di quel grafico.

Come accennato, il tool potrà ospitare al più 4 grafici per volta, ed ogni grafico sarà analizzabile e modificabile indipendentemente dagli altri.

5.1.2 Architettura dell'applicazione

Il tool LW Index adotterà un'architettura web di tipo client-server. L'utente si interfaccerà con l'applicazione attraverso un browser web come Google Chrome o Mozilla Firefox. Questa scelta risulta funzionale perché il tool dovrà offrire un maggior numero di opzioni d'interazione rispetto a SMART, e preservare i dati fra un'esecuzione e l'altra.

Gli algoritmi di confronto dovranno essere ospitati sul server in una specifica directory.

Nel dettaglio, l'utente potrà avviare l'applicativo tramite terminale. Una volta lanciato uno specifico comando, un'istanza del client (ossia una pagina web) si aprirà e contemporaneamente il webserver si avvierà.

Fra le caratteristiche proposte al punto 5.1.1, l'unica funzionalità che richiede l'intervento del server è la modifica di un grafico. Si noti che, pure se un grafico è vuoto (e cioè non è stato ancora popolato da stringhe), si sta affrontando un'operazione strettamente di modifica. Ogni volta che si richiede di aggiungere, eliminare o modificare stringhe, e si avvia il processo computativo alla pressione di un tasto, il webserver dovrà essere contattato e informato sulla natura della richiesta, per poi trattare i dati in put ed eventualmente fornire in output i dati necessari alla generazione del grafico.

5.1.3 Scelta delle tecnologie impiegate nell'implementazione

L'applicativo sarà implementato con tecnologie moderne e convenienti, al fine di semplificare la fase di implementazione e ridurre al minimo i casi di errore da gestire. Si è dunque deciso di implementare il webserver e il

backend dell'applicazione con Node, mentre il frontend del client con React. Il techstack costituito da React e Node è un'alternativa popolare all'utilizzo grezzo della tecnologia fondamentale dalla quale entrambi derivano, quale JavaScript.

Per ciò che concerne il frontend, le applicazioni web implementate in React possono sfruttare JavaScript o TypeScript; quest'ultimo, notoriamente, è un'alternativa al primo ed è caratterizzato da una forte tipizzazione, del tutto mancante invece in JavaScript.

E' stato deciso di implementare il tool in sviluppo con la variante TypeScript di React.

Per ciò che concerne il backend, Express, popolare framework di Node, è stato selezionato per l'implementazione del webserver. Express permette di avere immediatamente disponibile un ambiente di sviluppo facilmente configurabile, al quale è possibile agganciare nuove API attraverso sistemi di routing personalizzabili.

Questa scelta risulta immediata poiché il server dovrà ammettere un singolo processo elaborativo, quale la trasmissione dei dati fra il client e l'eseguibile dell'algoritmo selezionato per la computazione, sintetizzabile in un'unica route del webserver.

Ulteriori informazioni su React [13] e Node [14] sono reperibili in italiano sui rispettivi siti web.

5.1.4 Struttura del filesystem dell'applicazione

A basso livello, la struttura del filesystem di LW Index si presenta come segue. La directory `/api` conterrà il sorgente del backend, mentre la directory `/client` il sorgente del frontend. Entrambe le directory sono contenute in un macrospazio di lavoro denominato `codebase`.

```
~/{root}
    /api
```

```
/client
```

Il contenuto predefinito di ambo le cartelle è costituito dai file predefiniti di React (variante TypeScript) e Node (variante Express). Una volta installati rispettivamente React e Node, infatti, le due cartelle si popoleranno con un insieme di file critici per il funzionamento dell'applicazione. Dalla struttura base fornita dal processo di installazione, è possibile iniziare ad implementare le funzionalità dell'applicativo.

Gli algoritmi di confronto saranno collezionati in una sub-directory di `/api` che prenderà il nome di `/algo`. La cartella `/algo` presenterà, dunque, un insieme di sottocartelle, ognuna contenente il codice sorgente del proprio algoritmo.

```
/api
    /algo
        /Algoritmo_1
        /Algoritmo_2
        ...
        /Algoritmo_N
```

Il contenuto della i-esima cartella `/Algoritmo_i` è del tutto indipendente rispetto al resto dell'applicativo. Questo significa che gli algoritmi possono essere strutturati in qualsiasi modo (ad esempio utilizzare certe strutture dati piuttosto che altre), a patto che siano implementati in C o C++ e che risultino pre-compilati ed eseguibili una volta inseriti nella propria directory. Non è, inoltre, possibile avere casi di omonimia fra directory ospitanti i sorgenti degli algoritmi.

5.2 Implementazione del tool

5.2.1 Backend e Webserver

La macrocartella `/api`, come accennato nel paragrafo 5.1.2 e 5.1.4, conterrà l'implementazione del backend dell'applicativo, incluse le routine logiche necessarie al webserver per accogliere, processare e modellare le richieste in arrivo dal webclient in modo corretto.

Una volta installato Express seguendo le istruzioni presente sul sito del framework, un vasto numero di file si autogenererà nella cartella in cui si è lanciato il comando d'installazione. Di questi file, sarà sufficiente modificare esclusivamente `app.js` e `routes/computeAPI.js`. Si noti che quest'ultimo file dev'essere creato manualmente.

Backbone: App.js

Il file `app.js` avrà la seguente struttura:

- In cima, una serie di require() per imporre l'utilizzo di specifiche dipendenze di routing, parsing e pathing, oltre che di gestione di errori.
- Immediatamente dopo il listing delle dipendenze, comandare l'utilizzo di Express tramite `app = express()`
- Successivamente, indicare le routes (punti di contatto delle API) da rendere disponibili all'esterno, e in particolare al webclient, tramite `app.use('/api/computeAPI')`.
- Infine, stabilire le modalità di gestione degli errori HTTP più comuni, come il 404.
- Come ultimo passaggio, esportare il nucleo del backend come modulo tramite `module.exports = app`.

2: app.js

```
var createError = require('http-errors');
var express = require('express');
var path = require('path');
var cors = require('cors');
var bodyParser = require('body-parser');

[..]

var computeRouter = require('./routes/computeAPI');

var app = express();

[..]

app.use('/api/computeAPI', computeRouter);
app.use(function (req, res, next) {
        next(createError(404));
});

[..]

module.exports = app;
```

Per quanto sia possibile arricchire ulteriormente `app.js`, non risulta necessario nel nostro caso. Infatti, questo file - che rappresenta il backbone, o spina dorsale, del backend - ha semplicemente due funzioni:

- Segnalare casi di errore, ed eventualmente ridirezionare la richiesta
- Accogliere la richiesta di computazione del confronto, di conseguen-

za accedere ed invocare la route `/api/computeAPI`. Quando il client spedirà una richiesta verso `/computeAPI`, il server sarà in grado di applicare un'interfaccia logica fra l'API `/computeAPI` e il contenuto del file `routes/computeAPI.js`. In effetti, il contenuto di quest'ultimo file è il cuore del backend, e ha il compito di gestire le richieste in entrare e restituire dei dati come risposta.

Router: ComputeAPI.js

Il webclient, al momento della spedizione di una nuova richiesta computativa, dovrà inserire come endpoint della richiesta la route `/computeAPI`, indicando l'indirizzo di rete del webserver come origine. L'unica richiesta che questa route è in grado di accogliere è di tipo *POST*.

Inoltre, la route ammette esclusivamente richieste sicure, che rispettino il protocollo CORS e che prevedano JSON come formato comune di elaborazione.

3: computeAPI.js > Header

```
router.post('/', function (req, res, next) \{
        res.set({
                'Content-Type': 'application/json',
                'Access-Control-Allow-Origin': '*',
        });

    [..]
\}
```

A basso livello, l'obiettivo del webserver è generare un file FASTA a partire da un insieme di stringhe in input; successivamente, il webserver dovrà fornire il file FASTA appena creato in pasto all'algoritmo selezionato. A questo punto, il webserver dovrà attendere l'esecuzione dell'algoritmo e,

in particolare, che quest'ultimo stampi i risultati sul file di output. Infine, il webserver dovrà interpretare il file di output e convertirne i dani estrapolati come JSON. A questo punto, il webserver sarà in grado di rispondere al webclient con i dati necessari per generare il grafico.

Il primo step risulta, dunque, convertire il JSON pervenuto dal webclient in formato FASTA. Il fulcro del lavoro di conversione giace nell'interpretare le stringhe contenute nel corpo della richiesta ed inserirle con le opportune tabulazioni e spaziature in un file di testo vergine; similmente, ogni stringa dovrà essere associata ad un indice incrementale con la quale potrà essere, in seguito, identificata visivamente sul grafico prodotto.

4: computeAPI.js > Conversione del JSON in FASTA

```
const strings = req.body.strings;

strings.forEach((string) => {
        FASTA += '>' + index + '\n';
        FASTA += string.toUpperCase() + '\n';
        stringNames.push(string.toUpperCase());
        ++index;
});
```

Di seguito si propone il template di un file FASTA e esempio popolato da quattro stringhe.

5: computeAPI.js > Template di un file FASTA

```
>[index]
STRING
>[index 2]
STRING 2
...
>[index N]
STRING N
```

6: computeAPI.js > Esempio di un file FASTA

```
>1
GGTAAC
>2
AAAAAACCCTGTGT
>3
TTTTATATACC
>4
CCGTTTAAACC
```

Risulta altresì fondamentale elaborare un protocollo di denominazione che generi nomenclature univoche sia per i file di input, che per i file di output. Si decide di utilizzare il timestamp tramite `Date.now()` per creare associazioni uno-a-uno fra i due file. E' importante osservare che i file saranno gestiti in UTF-8.

I file di input saranno costruiti come segue.

Costante	Contenuto
__IN_FILE_NAME	algo + '-request-' + Date.now()
__IN_EXTENSION	'.fasta';
__IN_DIRECTORY	'./algo/' + algo + '/lw-index/';

Analogamente, i file di output saranno costruiti come segue.

Costante	Contenuto
__OUT_FILE_NAME	Date.now();
__OUT_EXTENSION	'.fasta.out';
__OUT_DIRECTORY	'./algo/' + algo + '/lw-index/result/';

Poiché gli algoritmi sono lanciati da eseguibili ottenuti come risultato di bundle precompilati generalmente adatti ad un ambiente Linux, e riservandoci l'abilità di poter sfruttare il tool in sviluppo anche su ambiente Windows, si decide di utilizzare Windows Subsystem for Linux (WSL) per lanciare gli eseguibili degli algoritmi. E' sufficiente *spawnare* (ossia istanziare) un terminale WSL e utilizzare STDIN, STDOUT e STDERR come un qualsiasi ambiente dotato di stream di input e output.

In questo modo, è possibile lanciare l'eseguibile dell'algoritmo selezionato simulando l'immissione di un comando personalizzato. Immediatamente dopo, per motivi di integrità dell'input, chiudiamo il flusso STDIN tramite `end()`.

7: computeAPI.js > Avvio dell'esecuzione dell'algoritmo

```
wsl.stdin.write(
        './algo/scMAW/sc-maw␣-a␣DNA␣-i␣' +
        __IN_FILE +
        '␣-o␣' +
        __OUT_FILE +
        '[opzioni]'
);

wsl.stdin.end();
```

Si osservi che, nelle fasi di integrazione di nuovi algoritmi, basterà sostituire `scMAW`, l'unico algoritmo integrato in maniera predefinita nel tool, con una variabile di tipo stringa.

A questo punto, l'algoritmo eseguirà in autonomia i propri sottoprocessi. Il webserver rimarrà in attesa di un riscontro sullo standard output STDOUT. La chiusura del processo di WSL fungerà da indicatore al router di iniziare ad estrapolare ed interpretare i dati dal file di output.

Il file di output sarà di tipo FASTA.OUT ed è strutturato come segue.

8: computeAPI.js > Template di un file FASTA.OUT

```
[# Numero di stringhe]
[index 1]        Lw(1, 1)   LW(1, 2)    LW(1, n)
[index 2]        LW(2, 1)   LW(2, 2)    LW(2, n)
...
[index N]       LW(n, 1)    LW(n, 2)    LW(n, n)
```

9: computeAPI.js > Esempio di un file FASTA.OUT

```
3
1       0.000000        0.861111        2.751123
2       0.861111        0.000000        1.989003
3       2.751123        1.989003        0.000000
```

La struttura prevede una riga per ogni stringa in input. Ogni stringa sarà dotata di una colonna per ogni stringa in input, più uno. La prima colonna di ogni riga rappresenta l'indice della stringa (da 1 a N), mentre la prima riga del file contiene il numero di stringhe.

Tralascinado la prima colonna, le colonne rimanenti di ogni riga eccetto la prima conterranno il valore della matrica LW fra la stringa rappresentata dall'indice di quella riga, e ogni altra stringa nell'insieme in input. Questa struttura permette di avere una visione complessiva delle similitudini fra tutte le stringhe in esame.

$LW(i, j)$ restituirà la similitudine (in float) fra la stringhe i e la stringa j. Banalmente, $LW(i, i) = 0$.

E' possibile consultare il Capitolo 2 per ulteriori informazioni circa il calcolo grezzo dell'indice LW.

Il file risulta di semplice lettura. Infatti, è sufficiente troncare le singole righe, e successivamente gli spazi tabellari, per individuare la matrice di indici LW desiderata. Questa matrice rappresenta il risultato atteso dall'algoritmo, e sarà incapsulato nella proprietà `graphData: result` di `responseData`. Allo stesso tempo, l'algoritmo terrà conto del tempo impiegato (in millisecondi) per eseguire l'algoritmo in funzione delle stringhe in input, e depositerà questo dato nella proprietù `quality.time`, rendendo disponibile future considerazioni qualitative da parte dell'utente.

10: computeAPI.js > Conversione FASTA.OUT in JSON

```
fs.readFile(__OUT_FILE, ENCODING, (err, data) => {
        if (err) throw err;
        var result = [], j = 0;

        const rows = data.split(/\r?\n/);
        rows.forEach((row) => {
                const splitData = row.split('\t');
                let string = stringNames[j++]
                let lw = [];

                splitData.shift();
                splitData.pop();
                splitData.forEach(
                    (data) => lw.push(data)
                );

                result.push({
                        string: string,
                        lw: lw,
                });
        });

        const responseData = {
                quality: {
                        time: time,
                },
                graphData: result,
        };

        res.json(responseData);
});
```

```
LW-Index > Esecuzione di scMAW termin
ata
LW-Index > Conversione OUT-JSON
LW-Index > Conversione riga 1   0.000
000     0.277778        1.645833    2
.083333 1.333333
LW-Index > Conversione riga 2   0.277
778     0.000000        1.423611    2
.361111 1.611111
LW-Index > Conversione riga 3   1.645
833     1.423611        0.000000    1
.506944 2.979167
LW-Index > Conversione riga 4   2.083
333     2.361111        1.506944    0
.000000 2.194444
LW-Index > Conversione riga 5   1.333
333     1.611111        2.979167    2
.194444 0.000000
{
  quality: { time: '0.014279' },
  graphData: [
    { string: 'GGTA', lw: [Array] },
    { string: 'GGTATATA', lw: [Array] },
    { string: 'TTTAT', lw: [Array] },
  ]
}
POST /api/compute 200 125.734 ms - 454
```

Figura 5.1: Anteprima di una porzione del processo di computazione svolto dal server.

L'istruzione conclusiva `res.json(responseData)` rappresenta la chiusura della richiesta e l'invio della risposta al client. A questo punto, il webserver rimarrà attivo e in attesa di nuove richieste da parte del client. Non eseguirà, in ogni caso, altre azioni in autonomia.

5.2.2 Frontend e Webclient

L'implementazione del frontend di una piattaforma web in React è dissimile rispetto ai metodi tradizioni di costruzione di siti web interattivi con tecnologie primitive come JavaScript. React, per definizione, è un framework per JavaScript che adotta una filosofia a componenti. Ogni elemento sulla pagina viene interpretato logicamente come un componente funzionale

della piattaforma, dipendente o meno rispetto ad un componente padre, e contenente o meno uno o più componenti figli.

I componenti React sono estremamente versatili e poliedrici, in grado di assumere connotazioni particolarizzanti in funzione degli obiettivi dello sviluppatore.

Oltre a React, la marcatura della piattaforma sarà costruita tramite HTML e CSS.

Per quanto si vorrebbe dare una spiegazione ampia e dettagliata di ogni singolo aspetto dell'implementazione del frontned, non è possibile data l'enorme mole di codice, che si attesta sull'ordine delle migliaia di righe al netto dei file autogenerati dal processo di installazione. Per questo motivo, il codice sorgente è liberamente e gratuitamente consultabile su GitHub. [15]

Il sistema dovrà essere fornito di una dashboard interattiva e un menù laterale che permetta di effettuare operazioni. Le operazioni, essenzialmente, avranno il compito di alterare il contenuto della dashboard, e di conseguenza modificare lo stato dell'applicazione.

Trasmissione globale delle informazioni

L'applicativo è strutturato a componenti funzionali, più o meno indipendenti l'uni dagli altri.

Tuttavia, tutti fanno riferimento ad un singolo componente per la trasmissione globale di dati e informazioni; altresì, questo stesso componente è responsabile della generazione di nuovi componenti e, in generale, del corretto funzionamento dell'applicazione.

L'intero applicativo è, ad un certo istante, un'istanza delle possibili configurazioni che il suddetto componente può assumere. Il componente in questione è chiamato `App`. Fra le altre cose, il componente è responsabile della creazione dei grafici e della corretta renderizzazione del markup sulla pagina.

Se un qualsiasi componente ha necessità di spedire un dato, oppure propagare un'informazione, al resto dell'applicativo, si limiterà piuttosto a delegare al componente `App` tale compito, specificando la natura della trasmissione e il contenuto del messaggio.

Di seguito viene riportata una porzione del componente `App`. In generale, il componente è declinato da una moltitudine di funzioni, variabili e constanti in grado di alterare l'applicazione nel suo complessivo.

11: App

```
const [alertProps, setAlertProps] = ..
const [alertStatus, setAlertStatus] = ..
const [graphs, setGraphs] = ..
const [ability, setAbility] = ..
const [selectedAlgo, setSelectedAlgo] = ..

[..]
```

Dashboard e gestione dello stato

Il menù laterale prevede tre macrosezioni: **Operazioni**, **Algoritmo** e **Debug**.

- La sezione **Operazioni** sarà l'area principale di interazione con l'utente. L'utente potrà creare nuove istanze di lavoro, analizzarle, eliminarle e svuota l'intera dashboard. Si osservi che 'tavolo di lavoro' e dashboard sono intercambiabili.
- La sezione **Algoritmo** fornisce informazioni critiche sull'algoritmo selezionato, permettendo inoltre di selezionarne di differenti.
- La sezione **Debug** permette di generare una serie di grafici esemplificativi, riempiendo il tavolo di lavoro con quattro istanze popolate, e di accedere al codice sorgente dell'applicazione, che è open source.

Il componente `Nav` governa il comportamento del menù laterale, distiguendo le operazioni esegubili in funzione dello **stato del sistema**. Il tool, infatti, ad un certo istante può avere solo uno dei due seguenti stati.

- **SingleGraph**: stato di analisi. Per accedere a questo stato, è necessario che almeno un grafico (vuoto o meno) sia presente sul tavolo di lavoro, e che uno di questi grafici sia stato selezionato tramite "Analizza istanza" sul menù laterale, nella sezione Operazioni.
- **MultiGraph**: stato di *default*. Ammette la creazione, rimozione e analisi dei grafici; la selezione di nuovi algoritmi; la generazione di esemplificazioni.

Le configurazioni dello stato dell'applicativo sono elaborate in un dato speciale definito `AppStatusSet`. I componenti dell'applicativo possono richiedere cambiamenti di stato trasmettendo al componente `App` un'opportuna richiesta. Inoltre, `AppStatusSet` contiene informazioni sui grafici attivi: in questo modo, i cambiamenti di stato non produrranno modifiche accidentali (e potenzialmente distruttive) al tavolo di lavoro.

12: AppStatusSet

```
const [appStatusSet, setAppStatusSet] =
  useState<AppStatusSet>({
    appStatus: AppStatus.__MultiGraph,
    graph: {
      ...BlankGraph,
          algo: selectedAlgo,
    },
  });
```

L'operazione di creazione di nuove istanze di lavoro genererà dei quadranti svuoti, identificati da un indice incrementale. Le istanze sono analizzabili o eliminabili. Come accennato in precedenza, creare o distruggere grafici è direttamente legato allo stato dell'applicativo; motivo per cui, alla richiesta di creazione o eliminazione di grafici, il componente `Nav` trasmetterà la richiesta al componente `App`, che eventualmente procederà ad alterare lo stato dell'applicazione.

In particolare, la richiesta di creazione ed eliminazione di grafici, assieme alla richiesta di analisi di grafici, sono fra le uniche che richiedono al componente `App` di ridisegnare l'interfaccia utente, e cioè modificare direttamente l'aspetto esteriore dell'applicazione.

In alternativa, il tavolo di lavoro è svuotabile di tutte le sue istanze di lavoro. Questa operazione non riporta a zero l'indice identificativo dei grafici, al fine di tener traccia delle attività dell'utente.

Fase di analisi delle istanze di lavoro

E' possibile analizzare le istanze di lavoro a prescindere dal loro stato; di conseguenza, un'istanza è analisi anche se non ancora popolata da stringhe. Analizzare un'istanza trasformerà il tavolo di lavoro generale in un tavolo

di lavoro specifico per l'istanza in analizi. Da questa schermata, è possibile effettuare la maggior parte delle operazioni previste dalle funzionalità ad alto livello indicate in precedenza.

Se l'istanza in analisi è neonata o non popolata, il semispazio superiore della schermata sarà occupato da due assi esenti da curve. In alternativa, se l'istanza è stata precedentemente popolata o si richiede una nuova computazione in essere, il semispazio sarà occupato da un grafico ricco di tante curve quante le stringhe immesse nella parte sinistra del semispazio inferiore della schermata di analisi.

D'altro canto, il semispazio inferiore consta - nel lato sinistro - di una sezione di immissione, rimozione e modifica delle stringhe in input. E' possibile aggiungere o rimuovere form di input tramite il pulsante "Inserisci". Accanto ad ogni form di input, sarà presente - oltre ad un pulsante per verificare la validità, e dunque aggiungere, la stringa digitata - un tasto per rimuovere il form dalla schermata, e un quadrante che indicherà la validità della stringa immessa.

Difatti, algoritmi diversi ammettono stringhe composte a partire da alfabeti distinti. Nel caso dell'algoritmo predefinito scMAW, l'unico alfabeto consentito è DNA. In caso di immissione di una stringa non consentita, il quadrante si popolerà con una X rossa; in alternativa, in caso di validità, il quadrante si popolerà con un OK verde. Fino alla pressione del pulsante "Aggiungi", il quadrante sarà occupato da un ?.

Il lato destro del semispazio inferiore proporrà una serie di informazioni circa la natura e la composizione dell'insieme di stringhe S da dare in pasto all'algoritmo. D'altro canto, un grafico relativo alla qualità (in termini di secondi) della computazione

Selezione e integrazione degli algoritmi

La sezione Algoritmi prevede, fra le altre cose, una voce per selezionare l'algoritmo da usare. Alla pressione, un alert personalizzato si presenterà all'utente, oscurando il restante contenuto della pagina e portando l'attenzione dell'utilizzare al selezione. L'alert conterrà una lista di algoritmi, ognuno presentato come bottone interattivo. L'interazione con uno qualsiasi dei bottoni presenti condurrà ad un cambio di stato dell'applicazione, per il quale il sistema sarà notificato di impiegare l'algoritmo selezionato.

Una volta selezionato l'algoritmo, la piattaforma sarà in grado di riformulare le modalità di computazione alla prossima richiesta. E' dunque possibile tornare (o avviare) una fase di analisi e procedere ad una nuova computazione. Quest'ultima verrà eseguita impiegando l'ultimo algoritmo selezionato.

La selezione di algoritmi è governata dal componente `AlgoSelector` il quale predispone un array di elementi (quali i pulsanti di selezione degli algoritmi), ognuno dei quali interagibili al *click*. Ogni click produce un cambiato di stato: in particolare, viene lanciata la funzione `setSelectedAlgo()` per comandare all'hub `App` di trasmettere all'intero sistema le proprietà del nuovo algoritmo.

E' interessante osservare come l'array di algoritmi viene generato.

13: AlgoSelector

```
ALGOS.forEach((algo) => {
  temp.push(
    <div
      onClick={() => {
                [..] props.setSelectedAlgo(algo);
          }}
          className='selector'
        >
          {algo.name}
    </div>
  );
});
```

Nel componente in esame, così come nel resto del sistema, `ALGOS` rappresenta logicamente gli algoritmi disponibili. Se uno svluppatore volesse integrare il proprio algoritmo nel tool, sarebbe sufficiente aggiungere una nuova entry nel seguente array, indicando le proprietà `name` (nome dell'algoritmo), `info` (informazioni riguardanti il funzionamento dell'algoritmo – ad esempio, le strutture dati utilizzate) e `abbr` (un nome alternativo sufficientemente corto).

14: ALGOS: Algo[ ]

```
const ALGOS: Algo[] = [
  {
    name: 'Sequence␣Comparison␣by␣Absent␣Words',
        info: "[..]",
        abbr: 'scMAW',
  },
  {
    name: 'Test␣Algo',
        info: 'Informazioni␣Test␣Algo',
        abbr: 'test',
  },
];
```

Posto che il server sia popolato dai file necessari (rif. Capitolo 5.1) per eseguire l'algoritmo, il sistema sarà in grado di riconoscere immediatamente la presenza di un nuovo algoritmo fra le sue file, e permettere dunque nuovi confronti fra stringhe impiegandolo.

5.2.3 Interfaccia utente

L'interfaccia di LW Index è spartana e alterna colorazioni scure a tonalità più chiare, al fine di mettere in risalto gli elementi interagibili della UI.

Generazione automatica di grafici esemplificativi

La voce *Genera esemplificazioni* del menù laterale è particolarmente utile per avere un'anteprima del funzionamento dell'applicativo. Alla pressione, infatti, il tool autogenererà quattro grafici con dati casuali, simulando il processo di elaborazione di cui si occupa il server. Proprio perché i dati sono casuali, il riavvio dell'applicazione resetterà i parametri che generano i grafici, creandone di nuovi ogni volta.

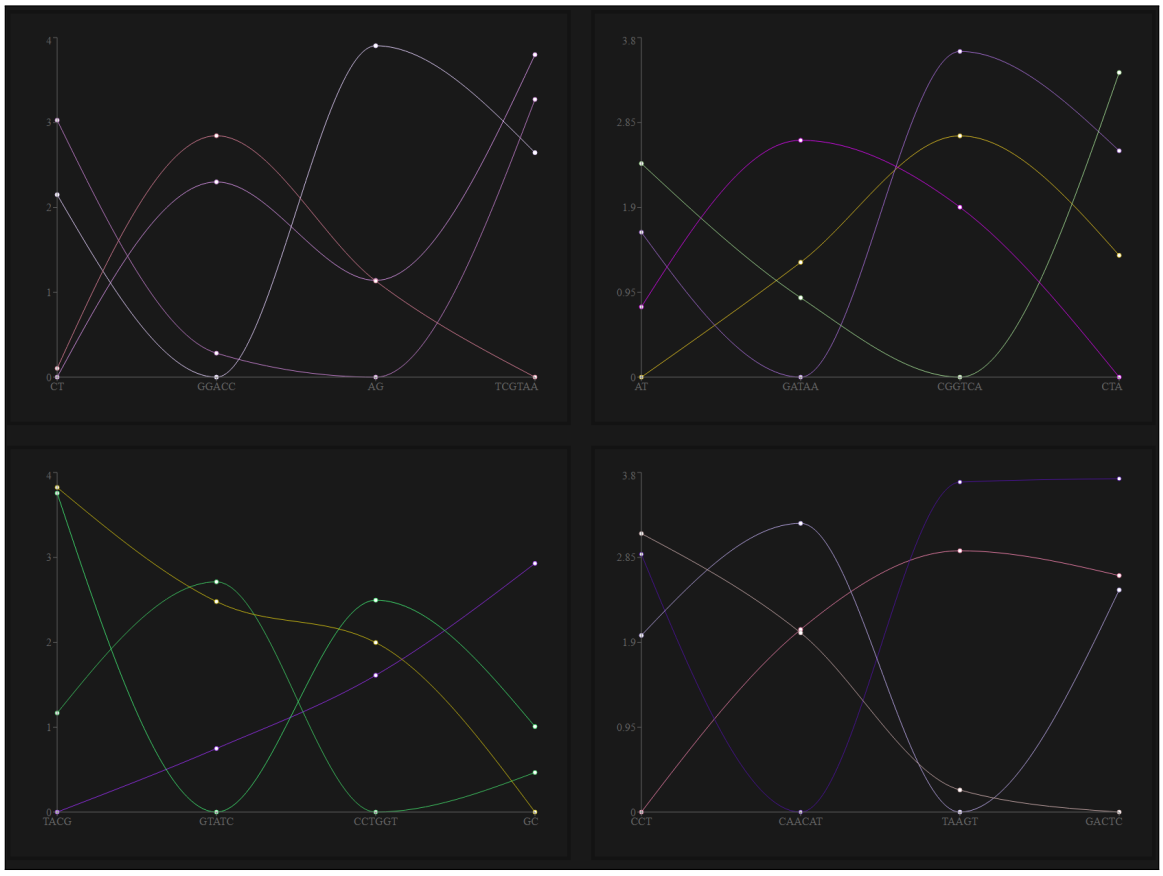

Figura 5.2: L'operazione di debug "Genera esemplificazioni" riempie il tavolo di lavoro con grafici casuali come in figura.

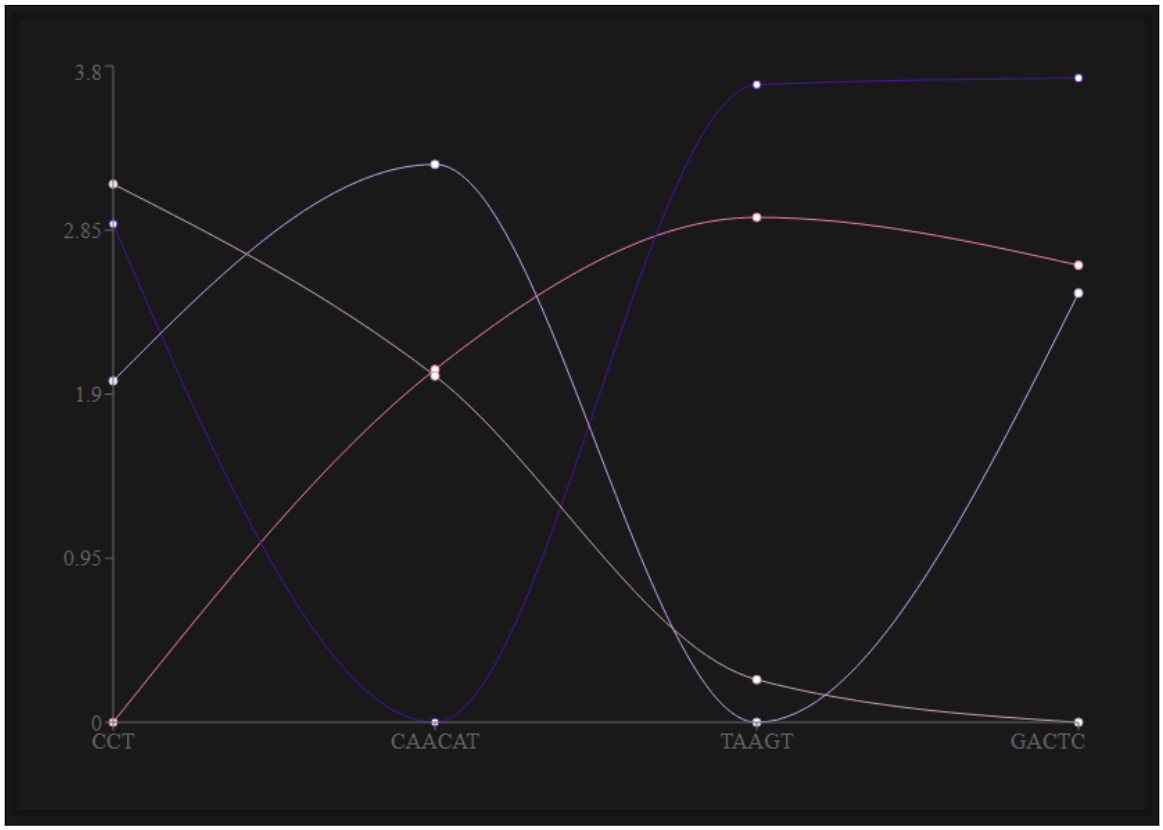

Figura 5.3: Dettaglio di un singolo grafico esemplificativo autogenerato tramite operazioni di debug.

Selezione degli algoritmi

L'interfaccia di selezione di algoritmi è costituita da un semplice alert composto da bottoni interattivi. Ogni bottone rappresenta uno degli algoritmi selezionabili, ordinati lessicograficamente. La pressione di un bottone produrrà un significativo cambio di stato nell'applicativo.

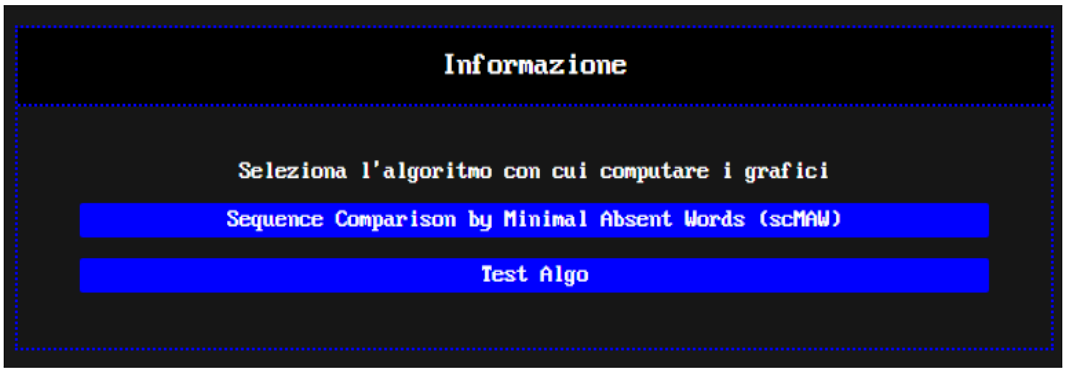

Figura 5.4: Gli algoritmi sono elencati per nome. Nell'esempio in figura, sono disponibili solo due algoritmi.

Schermata di analisi

La schermata di analisi è del tutto bianca, per indicare all'utente che si è in una fase dell'utilizzo dell'applicazione genuinamente interattiva. La maggior parte delle componenti sulla schermata sono interagibili e produrranno significativi cambi di stato o l'invocazione del webserver per l'esecuzione di operazioni nel backend.

Figura 5.5: La fase di analisi prevede una schermata bianca divisa in due semispazi, come in figura.

Stringhe in input

Due schermate in particolare permettono di tener traccia e di manipolare le stringhe in input. E' fondamentale che l'utente sia in grado di discernere con rapidità le stringhe ammesse, non ammesse o non ancora verificate; d'altro canto, è importante che salti all'occhio la possibilità di modificare, aggiungere o alterare l'insieme di stringhe in input, motivo per cui sono state utilizzate tonalità uniche, disponibile ersclusivamente nella schermata di analisi.

Figura 5.6: La successiva computazione che l'utente avvierà avrà come input l'insieme $S = GGTA, GGTATATA, TTTAT, TTACC, GGTACCA$ delle stringhe. L'algoritmo selezionato è indicato col suo nome completo e un suo eventuale acronimo.

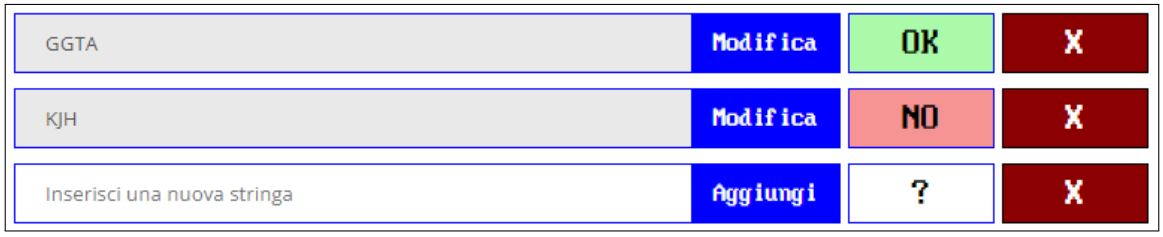

Figura 5.7: Nell'esempio in figura, l'utente ha generato tre form di input, digitando $GGTA$ nel primo, KJH nel secondo e lasciando vuoto il terzo. Ha, inoltre, premuto "Aggiungi" (sostituito adesso da "Modifica") nei primi due form. Il primo risulta valido, e cioè composto da lettere amesse dall'alfabeto; il secondo, invece, no. Poiché solo una stringa valida è stata immessa, l'utente non potrà computare alcun confronto da stringhe.

Grafici illustrativi

I grafici offerti sono fondamentalmente due: il *grafico di validità* degli indici LW, che permette di assicurare la robustezza dell'algoritmo selezionato; e il *grafo di qualità* dell'algoritmo, che permette di confrontare, sfruttando il tempo di computazione come parametro, gli algoritmi impiegati.

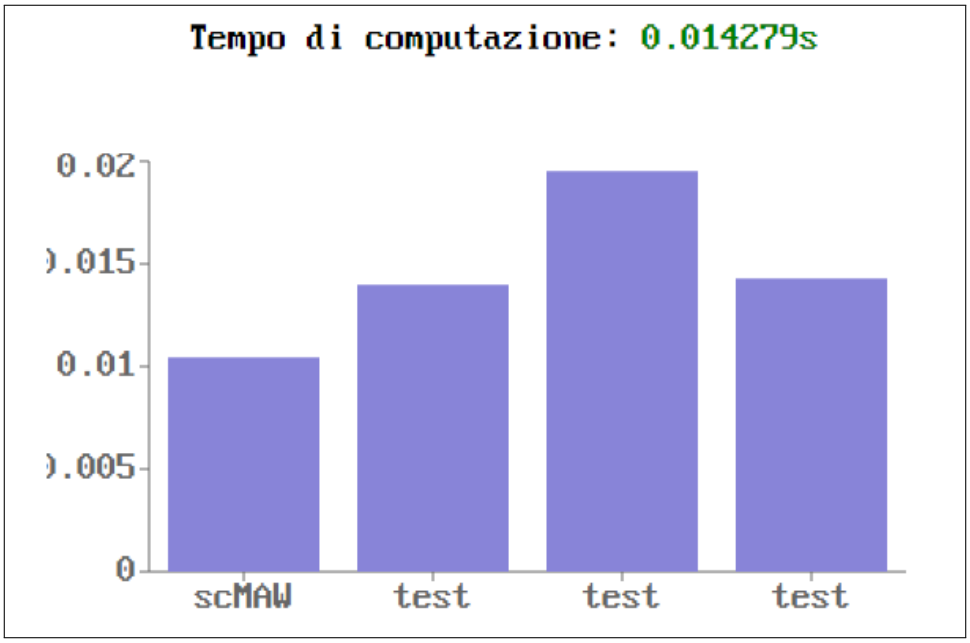

Figura 5.8: L'utente ha effettuato 4 computazioni. L'ultima computazione ha impiegato 0.014279 secondi. Le prime tre computazioni sono state effettuate con l'algoritmo *scMAW*, mentre le restanti tre con l'algoritmo *test*. E' possibile determinare la qualità degli algoritmi proposti in maniera visiva; d'altro canto, insiemi di stringhe in input più vasti sarebbero in grado di produrre dati più significativi.

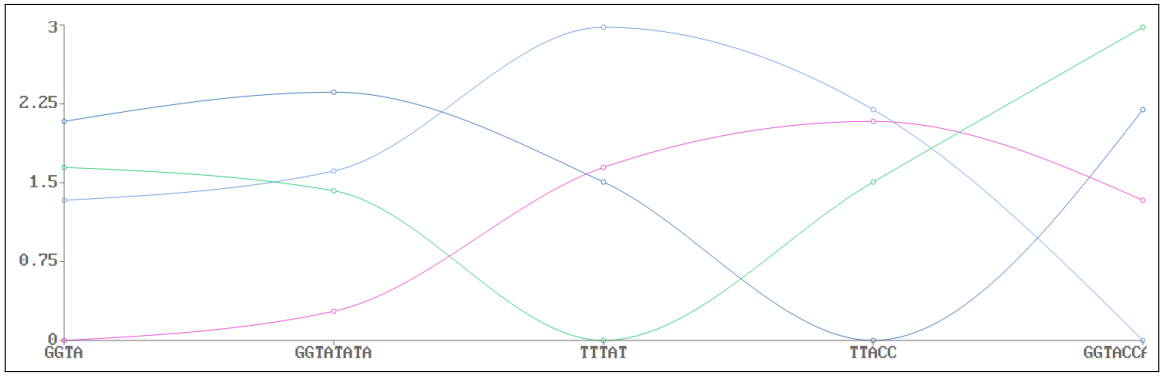

Figura 5.9: Esempio di grafico di validità generato dal tool.

Menù laterale

Il menù laterale della piattaforma è stato progettato per essere quanto più usabile e funzionale possibile. La pressione diretta di una voce del menù coincide con la pressione del testo presente sulla voce stessa; in generale, il menù laterale può essere visto come una tabella, composta da tante voci quante le operazioni possibili, più gli header (come ad esempio "Operazioni", o "Algoritmo") che non risultano interagibili. L'hover del mouse su una qualsiasi voce interagibile causerà un cambio di colore del testo di quest'ultima, passando da bianco ad azzurro.

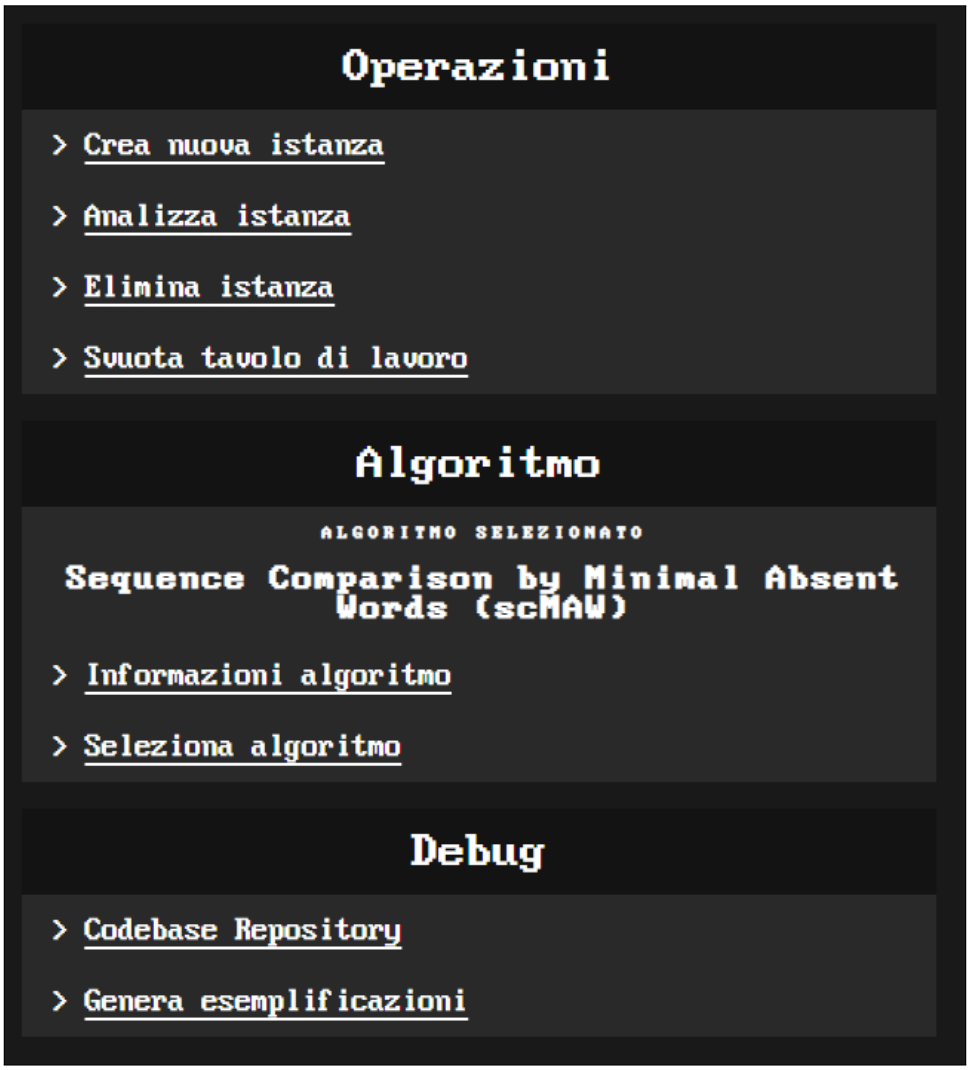

Figura 5.10: Menù laterale della piattaforma.

Generazione di nuove istanze

La creazione di nuove istanze di lavoro produrrà quadranti vuoti. Ogni quadrante sarà contrassegnato da un ID (detto *GID*, ossia *Graph ID*), il quale sarà stampato al centro del quadrante. Questi quadranti sono abilitati ad entrare in fase di analisi ed eliminazione.

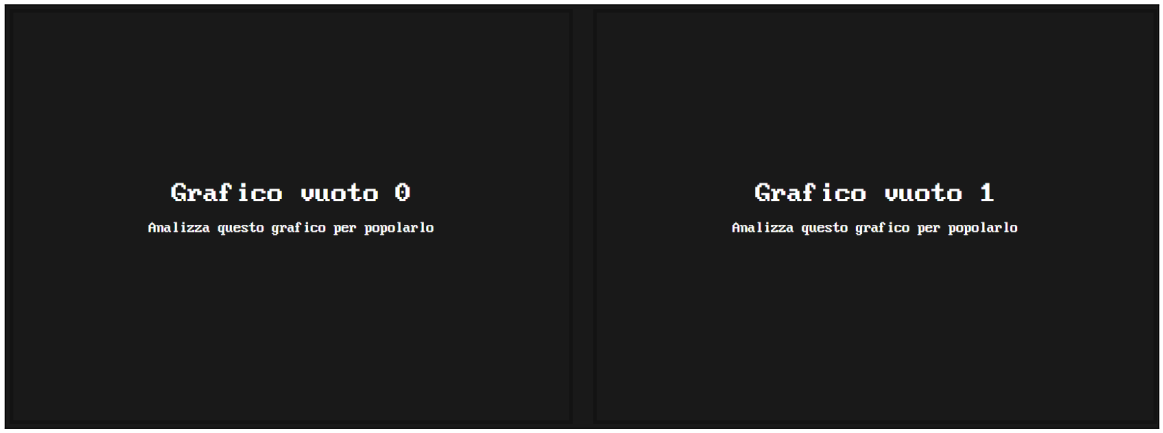

Figura 5.11: L'utente ha generato due grafici vuoti, che adesso popolano il tavolo di lavoro.

Capitolo 6

Conclusioni e possibili sviluppi futuri

Questo lavoro di tesi avevo lo scopo di esplorare, ad alto livello, le potenzialità dell'informazione negativa nell'ambito della bioinformatica, ed in particolare nella cornice del più ampio campo di lavori circa il confronto fra sequenze del genoma, benché sia applicabile - in generale - a stringhe su alfabeti ben definiti, come potrebbe essere quello proteico.

Successivamente, il lavoro di tesi ha imposto lo studio approfondito di un algoritmo che facesse dell'informazione negativa, nella forma delle *absent words* prima, e delle *minimal absent words* poi, la nozione matematica critica nella sua composizione algoritmica.

Lo studio ha dunque condotto al reverse engineering e alla comprensione generale del funzionamento del tool SMART che, dopo un'attenta revisione della natura dell'applicativo e delle modalità di elaborazione dei dati in input e ouput che adottasse, è risultato incompatibile con l'obiettivo generale di questo lavoro di tesi, il quale si prefissava - in origine - di integrare algoritmi come `scMAW` in un tool di comparazione qualitativa.

Poiché SMART non è risultato adatto ai nostri fini, i lavori di tesi sono stati ridirenzionati, mutando in parte. Dato che non esisteva, al meglio delle conoscenze dell'autore di questa tesi, e al momento della stesura di quest'ultima, un tool simil-SMART in grado di accogliere algoritmi che sfruttano l'informazione negativa e siano implementati nelle specifiche modalità indicate nei capitoli precedenti, si è deciso di realizzare un

tool ad hoc, del tutto originale, adottante un'architettura client-server e implementato con tecnologie moderne e all'avanguardia. Si tratta di una one-page web-application che soddisfa a pieno i requisiti funzionali previsti dal progetto originario della tesi di laurea.

Risulta evidente che il tool oggetto di sviluppo sia ampiamente arricchibile con nuove funzionalità, e che sia sufficientemente versatile da accogliere un importante numero di algoritmi di confronto fra stringhe. Risulta altresì evidente che il tool possa essere notevolmente migliorato, soprattutto sul fronte dell'usabilità; d'altro canto, un possibile sviluppo futuro potrebbe essere rappresentato, similmente a SMART, dalla realizzazione di una controparte desktop, e dunque non basata su web-browser, dell'applicativo. Poiché il software tratta un ambito di così ampio respiro come quello dei confronti fra stringhe, è banale evidenziare come le modifiche apportabili all'implementazione - funzionale e non - siano innumerevoli (ad esempio, si potrebbe permettere all'utente di immettere direttamente file FASTA precompilati).

Il lavoro di tesi ha, in conclusione, evidenziato le potenzialità dell'informazione negativa nel quadro della bioinformatica, e come la questione del calcolo della similitudine fra stringhe sia, ad oggi, un argomento ancora ricco di spunti di riflessione e innovazioni scientifiche.

Bibliografia

[1] Armando J Pinho et al. "On finding minimal absent words". In: *BMC bioinformatics* 10.1 (2009), pp. 1–11.

[2] Udi Manber e Gene Myers. "Suffix Arrays: A New Method for on-Line String Searches". In: *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '90. San Francisco, California, USA: Society for Industrial e Applied Mathematics, 1990, pp. 319–327. ISBN: 0898712513.

[3] Johannes Fischer. *Optimal Succinctness for Range Minimum Queries*. 2008. DOI: 10.48550/ARXIV.0812.2775. URL: https://arxiv.org/abs/0812.2775.

[4] Supaporn Chairungsee e Maxime Crochemore. "Using minimal absent words to build phylogeny". In: *Theoretical Computer Science* 450 (2012). Implementation and Application of Automata (CIAA 2011), pp. 109–116. ISSN: 0304-3975. DOI: https://doi.org/10.1016/j.tcs.2012.04.031. URL: https://www.sciencedirect.com/science/article/pii/S0304397512003866.

[5] Panagiotis Charalampopoulos et al. "Alignment-free sequence comparison using absent words". In: *Information and Computation* 262 (2018), pp. 57–68.

[6] M. Crochemore, F. Mignosi e A. Restivo. "Automata and forbidden words". In: *Information Processing Letters* 67.3 (1998), pp. 111–117. ISSN: 0020-0190. DOI: https://doi.org/10.1016/S0020-0190(98)00104-5. URL: https://www.sciencedirect.com/science/article/pii/S0020019098001045.

[7] *GitHub. maw: Minimal Absent Words. Solon P. Pissis*. https://github.com/solonas13/maw. Accesso: 2022-05-02.

[8] Simone Faro et al. "The String Matching Algorithms Research Tool". In: *Proceedings of the Prague Stringology Conference 2016*. A cura di Jan Holub e Jan Žďárek. Czech Technical University in Prague, Czech Republic, 2016, pp. 99–111. ISBN: 978-80-01-05996-8.

[9] *SMART | smart: GitHub Pages*. https://smart-tool.github.io/smart/. Accesso: 2022-05-02.

[10] *SMART. String Matching Algorithms Research Tool*. https://www.dmi.unict.it/faro/smart/howto.php. Accesso: 2022-05-02.

[11] *GitHub Repository: smart-tool/smart*. https://github.com/smart-tool/smart. Accesso: 2022-05-02.

[12] *GitHub Repository - smart-tool/smart-gui*. https://github.com/smart-tool/smart-gui. Accesso: 2022-05-02.

[13] *React – Una libreria JavaScript per creare interfacce utente*. https://it.reactjs.org/. Accesso: 2022-05-02.

[14] *Node.js*. https://nodejs.org/it/. Accesso: 2022-05-02.

[15] *GitHub Repository - lw-index: Tool simil-SMART*. https://github.com/v1enna/lw-index. Accesso: 2022-05-02.