Università degli Studi di Salerno

Dipartimento di Informatica

Corso di Strumenti Formali per la Bioinformatica

Relazione di progetto

Utilizzo di Reinforcement Learning per un Genome Assembler de novo

Napoli Riccardo

Matricola: 0552501560

Anno Accademico 2023/2024

Indice

Abstract

Questo lavoro si concentra sull'utilizzo di Reinforcement Learning (RL) per l'assemblamento di un genoma de novo. L'obiettivo ultimo è quello di costruire un agente che, avendo a disposizione una serie di frammenti più piccoli chiamati reads genomiche, riesca autonomamente a ricomporre il genoma completo. A partire da un modello che combina RL e un algoritmo genetico, si propone un approccio RL end-to-end, dove viene utilizzato non solo per la composizione di un pool di cromosomi per un approccio evolutionary-based, ma anche nelle fasi di mutazione e crossing-over. Analizzando l'approccio su un genoma d'esempio, i risultati mostrano un'alta precisione nell'assemblaggio, riducendo la dipendenza da algoritmi iterativi probabilistici e dall'intervento di esperti umani. Tuttavia, questo approccio comporta un aumento significativo del tempo di calcolo, dovuto alla complessità aggiuntiva delle chiamate multiple al Q-Learning. Infine, vengono evidenziati possibili miglioramenti futuri per migliorare ulteriormente le performance dell'agente.

Capitolo 1

Introduzione

1.1 Introduzione

L'assemblaggio di un genoma è uno dei compiti più importanti e computazionalmente complessi in bioinformatica, che consiste nel ricostruire un genoma a partire da porzioni più piccole chiamate reads, che corrispondono rispettivamente a sequenze di nucleotidi (A, C, T, G) per DNA ed RNA ed a sequenze di amminoacidi nel caso delle proteine.
Ricostruire un genoma è estremamente utile: le attuali tecnologie utilizzate gli attuali strumenti utilizzati in tale ambito, chiamati sequenziatori, non ci consentono di "leggere" un genoma interamente per via della loro lunghezza, anche nel caso di microrganismi. Le reads utilizzate per tale scopo vengono prodotte tramite la tecnica nota come shotgun sequencing [1]: grazie ai sequenziatori di DNA il genoma viene diviso randomicamente in segmenti più piccoli sottoforma di sequenze di testo. Compito dell'assemblatore sarà ricongiungere i frammenti nel corretto ordine analizzando la loro sovrapposizione o overlap, come visibile in figura 1.1. Inoltre, nel processo di sequenziamento possono essere frequenti errori come l'aggiunta, la rimozione o una lettura sbagliata di un nucleotide, per cui un assemblatore ideale deve essere in grado di tenere in considerazione la qualità delle reads sequenziate, in particolare per reads molto corte.

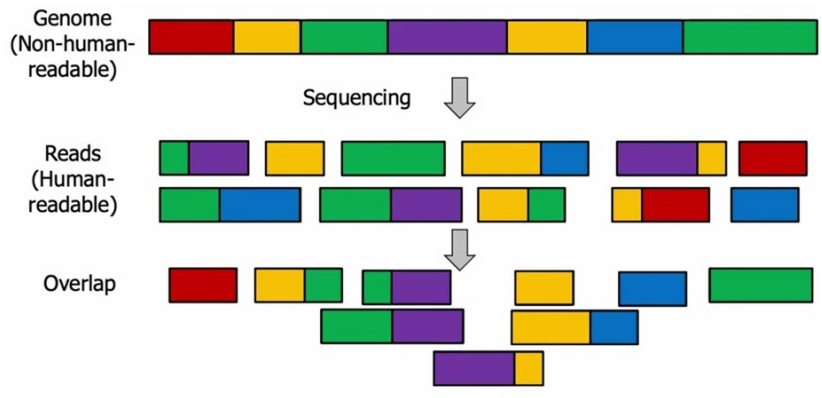

Figura 1.1: Illustrazione di un processo di assemblaggio di un genoma.

L'assemblaggio di un genoma è perciò un problema non ancora risolto, i cui risultati possono variare dipendentemente dall'assemblatore che si utilizza e che richiede ancora il supporto di esperti del settore per la configurazione. Sono nati così approcci alternativi al problema, in particolare tramite l'utilizzo del Reinforcement Learning (RL). Nel contesto del RL, un agente effettua osservazioni in un ambiente e prende decisioni ricevendo ricompense o reward positive o negative. L'obiettivo del RL è quello di imparare a scegliere le azioni da intraprendere massimizzando il punteggio di reward. Un interessante approccio per combinare il RL all'assemblaggio di un genoma consiste nell'utilizzare l'agente per riconoscere il corretto ordine delle reads osservando il reward prodotto dalle successive azioni, eliminando il bisogno di biologi esperti a supervisionare l'approccio.
Il presente studio mira a migliorare il modello di RL combinato ad un algoritmo genetico per l'assemblaggio genomico di Padovani et Al. [2]. L'obiettivo è quello di integrare il RL non solo nella parte di composizione del pool di cromosomi che l'algoritmo generico riceve in input, ma includere il RL anche nella restante parte dell'algoritmo, in particolare nelle fasi di mutation e crossing-over dei cromosomi presenti nel pool, per poi valutarne i risultati. Il paper è strutturato come segue: il background teorico è descritto nel capitolo 2; L'algoritmo oggetto di studio è presentato nel capitolo 3; Le modifiche proposte ed i risultati ottenuti vengono esposti e discussi nel capitolo 4; Le conclusioni ed eventuali sviluppi futuri sono infine fornite nel capitolo 5.

Capitolo 2

Background

In questa sezione verrà presentato il background teorico necessario per comprendere l'approccio utilizzato per questo lavoro. Nello specifico, si fornisce una panoramica sull'assembling de novo, reinforcement learning e q-learning.

2.1 Assembling de Novo

Una particolare strategia per l'assemblaggio di un genoma è quella dell'*assembling de novo*. L'assembling de novo consiste nella ricostruzione di un genoma a partire dalle reads senza il genoma originale di riferimento. Assemblare un genoma senza una base di partenza è un approccio di particolare importanza, poichè solo un numero esiguo tra gli interi genomi presenti in natura è conosciuto interamente. Tuttavia, assemblare reads senza riferimenti rientra nella classe dei problemi computazionali più complessi, i problemi NP-difficili.
Le più comuni strategie adottate per l'assembling de novo sono basate su euristiche e grafi: Greedy, Overlap-Layout-Consensus (OLC) e Grafo di De Bruijn. In particolare, utilizzando la tecnica su cui si basa l'algoritmo proposto da questo studio, l'OLC, ogni read viene rappresentata come un nodo in un grafo di overlap, mentre gli archi rappresentano il punteggio di overlap tra le reads. L'overlap tra due stringhe *s* e *s'*, come mostrato in figura 2.1 coincide con la lunghezza della più lunga sottostringa presente alla fine di s e all'inizio di s' o viceversa.

Figura 2.1: Overlap tra due stringhe.

Il genoma originale rappresenta quindi il cammino che collega tutte le read con lo score di overlap più alto, che corrisponde all'algoritmo NP-difficile del cammino Hamiltoniano. L'OLC può anche essere utilizzato per individuare regioni differenti di frammenti di genomi senza assemblarne tutta la sequenza.

2.2 Reinforcement Learning

Il reinforcement learning (RL) è un noto campo del Machine Learning. Esso costituisce un approccio differente dai problemi di training supervisionato e non supervisionato: non vi è bisogno nè di dati con relative label nè di un training set. Il reinforcement learning consiste nell'allenare il modello a compiere delle scelte in base ad un ambiente in cui si trova attraverso un meccanismo di ricompense e penalità, con l'intento di raggiungere un obiettivo. Il RL permette quindi di simulare il processo di decisione di un essere umano. Senza ricevere input di sorta, l'agente intraprenderà determinate azioni nell'ambiente e, imparando dai propri errori iterativamente, sarà in grado autonomamente di fornire una soluzione al problema specificato. Questo particolare approccio di addestramento trova spiccato utilizzo nell'ambito del gaming, ove il meccanismo di premi e penalità è presente intrinsecamente, e nel campo delle auto a guida autonoma. Tuttavia, in particolari contesti la tecnica potrebbe essere utilizzata anche nel caso di problemi di difficile o ignota soluzione mediante algoritmi standard, proprio come nel caso del genome assembling. Il processo di assemblaggio di un genoma con RL avviene come in figura 2.2: a partire dall'environment contenente le reads distribuite in ordine sparso, l'agente, scegliendo la read da appendere alla sequenza, compie delle azioni che porteranno nello stato successivo, con il rispettivo punteggio di reward. Ripetendo questo processo iterativamente, sarà in grado di trovare autonomamente la permutazione di reads originale.

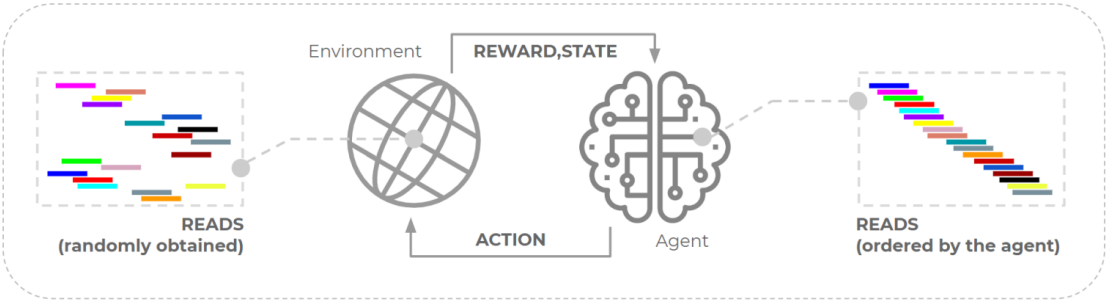

Figura 2.2: Genome assembling con Reinforcement Learning.

In un qualsiasi problema di RL è necessario definire la policy di riferimento, i diversi stati in cui entrare e le azioni da intraprendere. La policy rappresenta una serie di regole volte ad attuare una strategia in base all'ambiente in cui si trova l'agente, mentre le azioni sono le decisioni che portano da uno stato all'altro dell'apprendimento, ovvero una vista dell'ambiente di training.
Il RL può essere implementato utilizzando diversi algoritmi: uno di essi, utilizzato in questo studio per l'assemblaggio di un genoma, è il Q-Learning.

2.2.1 Q-Learning

Il Q-Learning è uno degli algoritmi più utilizzati per implementare il Reinforcement Learning. La particolarità del q-learning è quella di essere un algoritmo di RL "model-free": non viene

definita specificatamente una policy da attuare ma, soltando definendo l'ambiente e come le coppie stato-azione vengono visitate e aggiornate, l'agente stesso apprende iterativamente la strategia che produce il reward ottimo, di fatto rappresentando il puro apprendimento per tentativi [3]. Nella sua forma più semplice, il q-learning è definito tramite l'equazione seguente:

$$\underbrace{\text{New}Q(s,a)}_{\text{New Q-Value}} = \underbrace{Q(s,a)}_{\text{Current Q-Value}} + \underbrace{\alpha}_{\text{Learning Rate}} [\underbrace{R(s,a)}_{\text{Reward}} + \underbrace{\gamma}_{\text{Discount rate}} \overbrace{\max Q'(s',a')}^{\text{Maximum predicted reward, given new state and all possible actions}} - Q(s,a)]$$

Ad ogni passo dell'algoritmo di q-learning una particolare struttura, la Q-table, viene aggiornata prendendo in considerazione il q-value dell'iterazione precedente, un tasso di apprendimento arbitrario e il reward massimo prodotto dalla prossima azione per tutti i possibili stati futuri. Pre-computando il punteggio di reward futuro, l'algoritmo riesce in maniera più efficace a predire la futura azione da intraprendere e, di conseguenza, la sequenza di azioni dalla reward ottimale. Nel caso dell'assemblaggio di un genoma, i possibili stati futuri sono le successive read da aggiungere per continuare la ricostruzione: data un'azione intrapresa, la q-table viene aggiornata analizzando l'overlap massimo che le successive read fornirebbero.

Capitolo 3

Metodologia

3.1 Algoritmo Iniziale

Il progetto proposto da Padovani et Al. [4] si pone come obiettivo l'implementazione di un assembler de novo con RL. Gli autori citati forniscono anche un ambiente dove è possibile addestrare il modello su 23 genomi di dimensioni, lunghezza e numero delle reads incrementali. L'algoritmo utilizzato [4], riassunto in figura 3.1, si divide in tre fasi:

- Scelta di azioni con ϵ-greedy policy;
- Terminazione di un episodio con aggiunta del cromosoma nel pool;
- Esecuzione dell'algoritmo genetico.

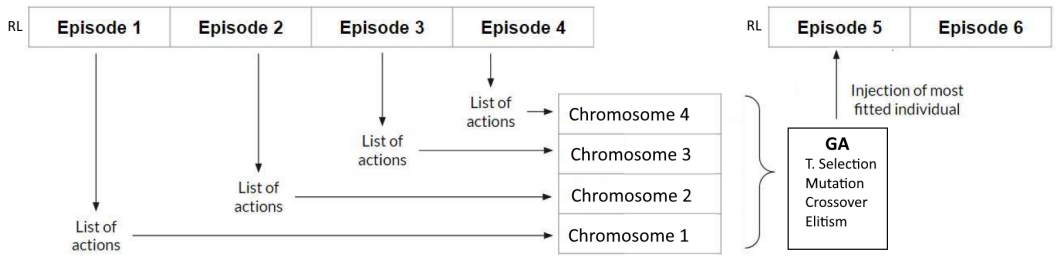

Figura 3.1: Algoritmo Gymnome-assembly [4] che combina RL e algoritmo genetico (GA).

3.1.1 Composizione del Pool con Q-Learning

Il reinforcement learning, mediante l'algoritmo di Q-Learning, viene utilizzato solo nella prima fase dell'algoritmo, per costruire il pool di cromosomi da passare all'algoritmo genetico e far evolvere la popolazione. Una volta partito il primo episodio e inizializzati i parametri necessari, viene selezionata un'azione con la ϵ-greedy policy. Il fattore ϵ indica la casualità nell'esplorazione: ad ogni step dell'episodio, che corrisponde ad una read presa per l'assemblaggio, l'agente esplora l'insieme delle read in maniera aleatoria con probabilità ϵ, o greedy,

scegliendo l'azione con il q-value più alto, con probabilità 1-ϵ. Inizialmente il valore di ϵ è alto per poi diminuire gradualmente [5]. Di conseguenza, per i primi episodi l'esplorazione è completamente casuale, mentre col passare degli episodi si tenderà sempre più ad utilizzare la tecnica greedy, servendosi dei path delle azioni prese con i relativi score di overlap memorizzati. Una volta scelta un'azione da intraprendere, i q-values vengono aggiornati secondo l'equazione seguente:

$$r(s,a,s') = \begin{cases} ol_{norm}(s,s') + 1.0 & \text{se s' è uno stato finale} \\ ol_{norm}(s,s') & \text{altrimenti} \end{cases}$$

dove $ol_{norm}(s,s')$ è il punteggio di overlap normalizzato in range [0,1] tra la read che ha consentito di arrivare allo stato s e quella che porta allo stato s'.
La fase di esplorazione viene reiterata fino a quando non si entra in uno stato assorbente. Gli stati che richiedono esattamente N reads per essere raggiunti sono assorbenti. Tuttavia, si entra in uno stato assorbente anche quando la read presa ha un punteggio di overlap pari a 0 con la precedente, e nel caso in cui ciò avvenga, vengono aggiunte casualmente le read rimanenti per formare il cromosoma. Una volta raggiunto uno stato assorbente l'episodio termina e il cromosoma risultante viene aggiunto in un pool sul quale verrà eseguito l'algoritmo genetico, per poi far ripartire la fase di esplorazione per il prossimo episodio. L'algoritmo genetico verrà eseguito solo una volta che il pool sarà completo, raggiunta la dimensione di N=60 elementi.

3.1.2 Algoritmo Genetico

L'algoritmo genetico dell'esperimento originale si divide nelle seguenti quattro fasi, ripetute per 300 generazioni:

- Selezione da torneo;
- Mutazione;
- Crossing-over;
- Elitismo.

Nella selezione da torneo, mostrata in figura 3.2, si estraggono randomicamente n=3 cromosomi dal pool, si calcola il punteggio di overlap complessivo di tutte le read che compongono i cromosomi e si inserisce in un pool di vincitori di stessa dimensione il cromosoma con il punteggio totale maggiore. Il pool di vincitori verrà utilizzato per il resto delle operazioni.

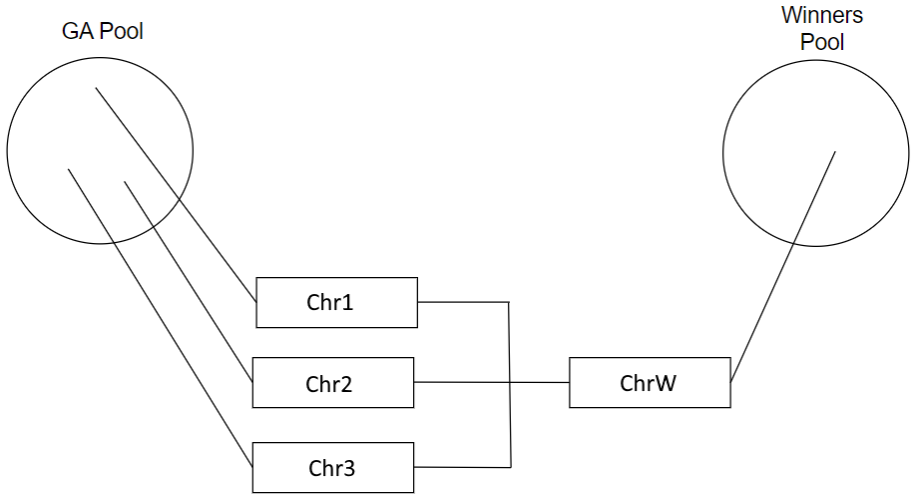

Figura 3.2: Tournament Selection nell'algoritmo genetico

Nella fase di mutazione, con una certa probabilità (20%) due read nei cromosomi del pool di vincitori vengono scambiate e sostituite ai cromosomi del pool originale. Lo stesso accade per il crossing-over: dati due cromosomi (i, j) del pool dei vincitori, il 70% delle volte vengono combinati e sostituiti ai cromosomi (i, j) della popolazione originale.
La ricombinazione dei cromosomi mediante il processo di crossing-over avviene come in figura 3.3: data una coppia di cromosomi, viene selezionato un intervallo di read casuale X. Da un cromosoma vengono aggiunte, in coda ed in ordine, le read appartenenti all'altro meno quelle che sono già presenti nell'intervallo scelto, per escludere read doppie. Il procedimento viene effettuato per entrambi i cromosomi.

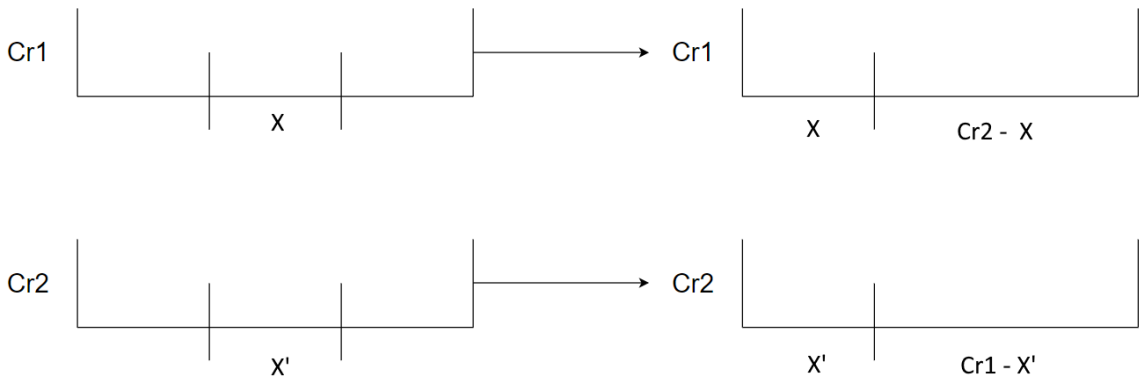

Figura 3.3: Crossimg-over nell'algoritmo genetico.

Nell'ultima fase, di elitismo, il punteggio di overlap complessivo di ogni cromosoma del pool viene ricalcolato. Se il cromosoma di punteggio massimo ha uno score di overlap maggiore dei precedenti aggiorno il valore massimo, altrimenti quest'ultimo viene iniettato nel primo slot della popolazione originale.
Una volta concluso l'algoritmo genetico ed ottenuto il cromosoma con punteggio complessivo

massimo tra le generazioni, è possibile evolvere la popolazione. Il processo di esplorazione riparte da quel cromosoma, che viene scelto interamente dall'algoritmo.

Capitolo 4

Esperimenti e risultati

4.1 Modifiche Proposte

Dalla descrizione dell'algoritmo oggetto di studio è possibile notare che, una volta costruito il pool di cromosomi tramite il RL, l'algoritmo genetico per l'evoluzione del pool di cromosomi è puramente iterativo e probabilistico. Le modifiche proposte volgono verso l'implementazione di un assembler de novo con RL end-to-end, includendo il RL non solo nella parte di composizione del pool da evolvere, ma anche nelle operazioni di mutation e crossing-over dell'approccio evoluzionario. Ciò ha permesso di valutare se un sistema di questo tipo potesse migliorare la precisione o la velocità dell'assemblaggio.
Gli episodi di q-learning nelle fasi di mutation e crossing-over sono stati ridotti da 60 a 40, mentre le generazioni di evoluzione della popolazione sono state ridotte da 300 a 15.
Durante gli esperimenti e l'implementazione è diventato evidente che, per genomi di complessità più alta, includere molti richiami al q-learning originale richiedesse potenza di calcolo e scalabilità maggiori, per cui la discussione dei risultati si concentrerà solo sul primo degli esperimenti.
L'operazione di mutation presente nell'algoritmo originale prevedeva, non su tutti i cromosomi ma solo in maniera aleatoria (20%), di invertire due reads presenti nei cromosomi scelte randomicamente.
L'operazione di mutation proposta prevede la divisione del cromosoma in tre parti e il riutilizzo del q-learning sulla parte centrale del cromosoma, com'è possibile vedere in figura 4.1. In questo modo è possibile ricombinare con l'utilizzo del RL la parte centrale del genoma cercando composizioni con maggiore score di overlap senza al contempo intaccare la geolocalità delle reads.

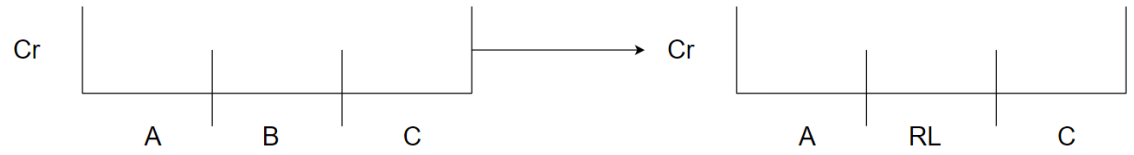

Figura 4.1: Mutation con Reinforcement Learning.

Di seguito il codice proposto:

```
def _RLmutation(self, reads, cromossome):
    if type(cromossome) == list:
        cromossome = np.array(cromossome)

    #divide cromosome in three chunks
    lenght = len(cromossome)
    part_size = lenght // 3
    remainder = lenght % 3

    part1 = cromossome[:part_size]
    part2 = cromossome[part_size:2 * part_size + (1 if remainder == 1 else
       2 if remainder == 2 else 0)]
    part3 = cromossome[2 * part_size + (1 if remainder == 1 else 2 if
       remainder == 2 else 0):]

    reads1 = []
    picked_index = []

    #picks training reads
    for i in part2:
        reads1.append(reads[i])
        picked_index.append(i)

    rl = np.array(qlearning(reads1, 40, dm_rm_metrics=False))

    new = []
    #maps rl reads to cromosome indexes
    seen_elements = set()
    for k in rl:
        for l in range(len(reads)):
           if reads1[int(rl[k])] == reads[l]:
               if picked_index[int(rl[k])] == l:
                   if l in seen_elements:
                       continue
                   new.append(l)
                   seen_elements.add(l)
                   break
    cromossome = np.concatenate((part1, np.array(new), part3))
    return cromossome
```

Listing 4.1: Mutation con Q-learning.

La nuova operazione di crossing-over con RL viene eseguita come in figura 4.2. In questo caso, dati i cromosomi Cr1 e Cr2, il RL per la parte centrale di Cr1 viene eseguito con le reads che compongono la parte centrale di Cr2 e viceversa, per cui è possibile effettuare il crossing-over solo nel caso in cui lo scambio non formi cromosomi dalle reads ripetute.

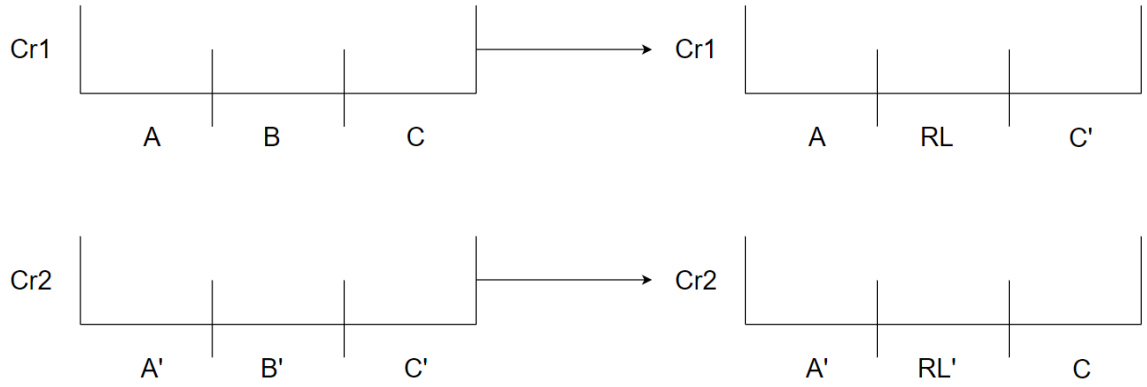

Figura 4.2: Crossing-over con Reinforcement Learning.

Viene per questo implementato il seguente metodo checkCross che controlla le condizioni per il crossing-over, assieme a un contatore del numero di crossing-over effettuati.

```
#checks if there are no repeated reads
def _checkCross(self, part_size, A, C, A1, C1):
    for i in range(part_size):
        for j in range(part_size):
            if A[i] == C1[j]:
                return 0
            if A1[i] == C[j]:
                return 0
    return 1
```

Listing 4.2: Metodo checkCross per controllare eventuali reads ripetute nel crossing-over.

Di seguito il codice proposto per il crossing-over.

```
def _RLcrossover(self, reads, cromossome1, cromossome2, counter):
    if type(cromossome1) == list:
        cromossome1 = np.array(cromossome1)
    if type(cromossome2) == list:
        cromossome2 = np.array(cromossome2)

    #divide cromosome in three chunks
    lenght = len(cromossome1)
    part_size = lenght // 3
    remainder = lenght % 3
    part1 = cromossome1[:part_size]
    part3 = cromossome1[2 * part_size + (1 if remainder == 1 else 2 if
                                    remainder == 2 else 0):]
    part4 = cromossome2[:part_size]
    part6 = cromossome2[2 * part_size + (1 if remainder == 1 else 2 if
                                    remainder == 2 else 0):]
```

```
    reads1 = []
    reads2 = []

    if self._checkCross(part_size, part1, part3, part4, part6) == 1:
        #picks training reads
        for i in cromossome1:
            if i not in part1:
                if i not in part6:
                    reads1.append(reads[i])
                    picked_index1.append(i)
                if i not in part4:
                    if i not in part3:
                        reads2.append(reads[i])
                        picked_index2.append(i)

        rl1 = np.array(qlearning(reads1, 2, test_each_episode = False,
        dm_rm_metrics=False))
        rl2 = np.array(qlearning(reads2, 2, test_each_episode = False,
        dm_rm_metrics=False))

        #maps rl reads to cromosome indexes
        seen_elements1 = set()
        seen_elements2 = set()
        new2 = []
        new5 = []
        for k in rl1:
            for l in range(len(reads)):
                if reads1[int(rl1[k])] == reads[l]:
                    if picked_index1[int(rl1[k])] == l:
                        if l in seen_elements1:
                            continue
                        new2.append(l)
                        seen_elements1.add(l)
                        break
        for k in rl2:
            for l in range(len(reads)):
                if reads2[int(rl2[k])] == reads[l]:
                    if picked_index2[int(rl2[k])] == l:
                        if l in seen_elements2:
                            continue
                        new5.append(l)
                        seen_elements2.add(l)
                        break

        child1 = np.concatenate((part1, np.array(new2), part6))
        child2 = np.concatenate((part4, np.array(new5), part3))
        counter +=1
        return child1, child2, counter
    return cromossome1, cromossome2, counter
```

Listing 4.3: Crossing-Over con Q-learning.

4.2 Risultati Sperimentali e Discussione

Per la valutazione dei risultati si è considerato inizialmente un test sul primo genoma esclusivamente, data la struttura più semplice, per poi effettuare un test su tutti i genomi presenti nel database. La metrica su cui i due approcci sono stati valutati è la Reward Measure (RM): un episodio si considera avvenuto con successo se il valore della total reward ottenuto è maggiore o uguale di quello prodotto dal caso in cui l'agente esplori la permutazione ottimale delle reads. Calcolando la media della RM per tutti gli episodi, è possibile valutare il grado di efficienza dell'agente nell'esplorare le reads. Le caratteristiche del sistema ove il primo test è stato effettuato, assieme ai risultati ottenuti, sono evidenziati in tabella 4.1. Inoltre, il test effettuato su tutti i genomi è presente in tabella 4.2.

Caratteristiche del sistema: **Modello: Victus by HP Laptop 15-fa1xxx** **Processore: 13th Gen Intel(R) Core(TM) i7-13700H 2.40 GHz** **RAM installata: 8,0 GB** **Sistema operativo a 64 bit, processore basato su x64**					
Algoritmo Originale			**Algoritmo Proposto**		
Episodi	**Tempo (min)**	**RM Media**	**Episodi**	**Tempo (min)**	**RM Media**
1000	0.5	98.70%	1000	1	97.90%
10000	3	99.85%	10000	15	88.05%
20000	5	99.95%	20000	28	99.08%
30000	7	99.96%	30000	50	82.93%

Tabella 4.1: Statistiche di tempo e Reward Measure (RM) media su una crescente quantità di episodi per l'algoritmo originale e proposto.

Algoritmo Originale		**Algoritmo Proposto**	
Tempo Totale	**RM Media**	**Tempo Totale**	**RM Media**
17h 03m	80.87%	174h 24m	67.48%

Tabella 4.2: Statistiche di tempo e Reward Measure (RM) media su tutti i genomi presenti per l'algoritmo originale e proposto.

Dai risultati è possibile evincere che le prestazioni dell'algoritmo proposto sembrano reggere il confronto con l'algoritmo originale, tuttavia l'algoritmo proposto risulta sensibilmente più lento in termini di tempo. Le quattro chiamate al q-learning, effettuate ad ogni run dell'algoritmo genetico per ogni elemento del pool e non solo con una certa probabilità come nell'algoritmo originale, assieme ai controlli necessari nell'algoritmo proposto, rallentano in maniera sensibile la computazione.

Capitolo 5

Conclusioni e possibili sviluppi

Avendo introdotto il problema dell'assembling de novo con Reinforcement Learning, con tutte le tecniche utilizzate atte alla risoluzione del problema, è stato valutato il potenziale della combinazione delle due metodologie per il problema, assieme ad un primo passo per un assembler con RL end-to-end. Partendo dal lavoro di Padovani et Al. [2], si è evidenziato un possibile metodo di sviluppo dell'algoritmo originale. Analizzandone i risultati, l'algoritmo proposto riesce ad ottenere una precisione media comparabile, eliminando l'utilizzo di algoritmi iterativi probabilistici e di esperti del settore per configurarli, nonostante vi sia necessità di maggiore potenza computazionale, specialmente per genomi più complessi dalle reads più lunghe.
Si propongono i seguenti approcci come sviluppi futuri:

- Per aumentare il numero di crossing-over effettuati, è possibile implementare un crossing-over con RL con indice selezionato dinamicamente, utilizzando la stessa tecnica dell'algoritmo originale. Il reinforcement learning per il miglioramento della popolazione potrebbe avvenire sulla differenza rimanente del cromosoma. In tal caso, bisognerà mettere a punto il numero degli episodi di q-learning e di generazioni dell'algoritmo genetico;
- Ottimizzazione e parallelizzazione dell'algoritmo presente;
- Scelta variabile del numero di componenti del pool, di episodi di q-learning interno e di generazioni dell'algoritmo genetico in base alla dimensione del genoma su cui lavorare.

Una volta apportate le modifiche sopracitate, servendosi di più potenza di calcolo, sarà possibile effettuare di nuovo gli esperimenti anche su genomi più complessi, per valutare miglioramenti dell'algoritmo in termini di tempo e scalabilità.

Bibliografia

[1] R. Staden. "A strategy of DNA sequencing employing computer programs". In: *Nucleic Acids Research* 6 (1979). DOI: https://doi.org/10.1093/nar/6.7.2601.

[2] Kleber Padovani et al. "A step toward a reinforcement learning de novo genome assembler". In: (2024). DOI: https://doi.org/10.48550/arXiv.2102.02649.

[3] Richard Sutton e Andrew Barto. *Reinforcement learning: An Introduction*. MIT, 1998.

[4] *gymnome-assembly*. URL: https://github.com/kriowloo/gymnome-assembly.

[5] Aurèlien Gèron. *Hands-On Machine Learning with Scikit-Learn, Keras & Tensorflow (Third Edition)*. Jupyter, 2019. ISBN: 978-1-492-03264-9.

[6] *What is de novo assembly?* URL: https://thesequencingcenter.com/knowledge-base/de-novo-assembly/.

[7] Zhenyu Li et al. "Comparison of the two major classes of assembly algorithms: overlap–layout–consensus and de-bruijn-graph". In: *Briefings in Functional Genomics* 11 (2012). DOI: https://doi.org/10.1093/bfgp/elr035.

[8] *A brief introduction to reinforcement learning: Q-learning*. URL: https://www.qwak.com/post/a-brief-introduction-to-reinforcement-learning-q-learning.