Algoritmi Genetici per la decriptazione di testi

Giovanni Ottaviano
Matricola: 962079

Capitolo 1

Introduzione

1.1 Crittografia

La crittografia (dal greco, "scrittura segreta") si occupa di studiare metodi che permettono di nascondere le informazioni tramite l'utilizzo di cifrari, in modo che siano comprensibili sono agli individui a cui sono destinate. In particolare, anche se un individuo esterno venisse a conoscenza della sequenza criptata non riuscirebbe ad ottenerne nessuna informazione.
L'idea della crittografia classica è quella di applicare un algoritmo C (*cifrario*) che utilizza una *chiave* (k) per codificare il messaggio; la conoscenza della chiave e dell'algoritmo permette di riottenere il messaggio originale, semplicemente applicando la trasformazione inversa C^{-1}.
Per esempio, il più antico sistema di cifratura è il *cifrario di Cesare*: si tratta di un cifrario a sostituzione monoalfabetica in cui ciascuna lettera del testo in chiaro viene sostituita da un'altra lettera, che si trova k posizioni più avanti.

$$C_k(x) = (x + k) \pmod N$$
$$C_k^{-1}(x) = (x - k) \pmod N$$

dove N è il numero di lettere dell'alfabeto e si è usata l'aritmetica modulare.
Tuttavia questo cifrario è molto debole, perché il numero di possibili soluzioni è pari a N, che nell'alfabeto inglese è 26, e questo permette di decrittare il messaggio anche solo con un metodo a forza bruta, cioè provando tutte le possibili chiavi in poco tempo.

Un caso più complicato del cifrario precedente è quello che si potrebbe chiamare *cifrario a permutazione*: in questo caso l'alfabeto cifrante si ricava da una permutazione dell'alfabeto in chiaro e si cripta il testo sostituendo ciascuna lettera con la sua corrispondente nella permutazione. In questo caso la chiave è la permutazione degli N caratteri (σ) che ha creato il nuovo alfabeto e la sua inversa permette di decrittare il testo.

$$C_\sigma(x) = \sigma(x)$$
$$C_\sigma^{-1}(x) = \sigma^{-1}(x)$$

Questo cifrario è migliore del precedente in quanto ammette $N!$ possibilità (più di 10^{26}), ma è ancora debole perché può comunque essere forzato da un'analisi di frequenza dei caratteri e di altri gruppi di lettere comuni in una data lingua.

Per ottenere dei cifrari migliori è necessario passare a una codifica polialfabetica, cioè in cui il testo è codificato utilizzando più di un alfabeto. L'esempio più semplice è il *cifrario di Vigenère*, una sorta di generalizzazione a più alfabeti del cifrario di Cesare: in questo caso il testo è criptato partendo da una parola (chiave) di lunghezza L che viene ripetuta fino ad ottenere la stessa lunghezza del testo, a questo punto ciascuna lettera viene scalata, applicando il metodo di Cesare, di un numero di posti pari alla posizione della lettera della chiave associata:

$$C_i(x_i) = (x_i + k_i) \pmod N$$
$$C_i^{-1}(x_i) = (x_i - k_i + 26) \pmod N$$

in cui k_i è il numero (intero in $[0, 25]$) associato al i-esima lettera della chiave ripetuta. Questo cifrario è molto più sicuro dei precedenti, non tanto per il numero di possibili combinazioni che sono comunque 26^L, ma per il fatto che tecniche come lo studio della frequenza non sono più efficaci, visto che una stessa lettera può essere criptata usando caratteri diversi. Nonostante ciò, il metodo è indebolito dalla natura ripetitiva della chiave e scoprendone la lunghezza è possibile rompere i singoli cifrari di Cesare individualmente, visto che periodicamente si ha sempre lo stesso alfabeto scalato. Uno dei metodi più semplici (e antichi) per scoprire la dimensione della chiave è il test di Kasiski, che sfrutta sequenze di caratteri identici ripetute ad una certa distanza nel testo criptato.

1.2 Algoritmi genetici

A partire dalla seconda metà del secolo scorso, gli scienziati hanno iniziato ad interessarsi sempre di più alla *computazione naturale* (natural computing), che consiste nell'utilizzare dei metodi per la risoluzione di problemi (spesso di ottimizzazione) che prendono spunto dalla natura; tra questi ci sono gli *algoritmi genetici* (GA), che si basano sulla teoria evoluzionistica di Darwin e sul principio di selezione e riproduzione del più adatto.
In particolare, questo tipo di algoritmi sfruttano 3 operazioni ispirate alla biologia: la *selezione*, con cui vengono scelti gli individui migliori per la riproduzione, il *crossover* (o ricombinazione) che serve a creare dei figli incrociando i genitori, e la *mutazione*, che modifica le sequenze di geni per aumentare la diversità.
Due ingredienti fondamentali per gli algoritmi genetici sono: la *rappresentazione in geni* degli individui della popolazione, in cui ciascun individuo è una sequenza (cromosoma) composta da un numero fissato di geni, che deve poterne racchiudere tutte le caratteristiche, e la funzione di *fitness*, che permette di stimare quanto una soluzione sia buona e vicina all'ottimo.

Una volta che sono state decise le implementazioni delle operazioni descritte sopra, è possibile schematizzare il funzionamento di un generico algoritmo genetico:

1. Si genera una nuova popolazione di individui (solitamente scelti random)
2. Si ripete iterativamente la generazione di una nuova popolazione:
   2.1. si calcola la funzione di fitness per tutti gli individui
   2.2. si selezionano i migliori e si fanno riprodurre
   2.3. si applicano il crossover e le mutazioni (con una certa probabilità)
3. Al termine delle generazioni, si sceglie l'individuo migliore che sarà la soluzione (ottima o sub-ottima) del problema.

É importante fare alcune considerazioni: in primo luogo bisogna scegliere bene la funzione di fitness in modo che la soluzione del problema di ottimizzazione sia effettivamente quella per cui la funzione ha valore massimo (o minimo), altrimenti l'algoritmo evolverà verso una soluzione sbagliata. Successivamente è necessario scegliere con cura le probabilità con cui vengono eseguiti il crossover e la mutazione, perché nel caso fossero troppo basse c'è il rischio di non avere abbastanza diversità nella popolazione, mentre se fossero troppo alte si rischierebbe di eliminare gli individui migliori, impedendo all'algoritmo di raggiungere la soluzione.

Capitolo 2

Applicazione degli algoritmi genetici alla crittografia

Si vuole applicare la metodologia degli algoritmi genetici alla decriptazione di testi in lingua inglese, con una lunghezza di circa mezza pagina (indicativamente 2000 caratteri). Lo studio si basa sui due cifrari esposti nella Sezione 1.1 (monoalfabetico a permutazione e polialfabetico) e affronta alcune metodologie specifiche per migliorare i risultati.
Nella Sezione 2.1 si riporta la metodologia utilizzata per la ricerca della chiave e la decodifica di un testo criptato utilizzando il cifrario di Vigenere; la prima versione dell'algoritmo è fatta supponendo di conoscere la lunghezza della chiave, mentre la seconda assume di conoscere solo un range all'interno di cui è presente la lunghezza di chiave giusta.
Nella Sezione 2.2 si usa un algoritmo genetico per attaccare un cifrario a permutazione, valutando le differenze con il caso precedente e le strategie per migliorare l'algoritmo.
I programmi che utilizzano questi metodi sono stati scritti in Python3 (testati sulla versione 3.5.3).

2.1 Algoritmo Genetico: cifrario polialfabetico

Implementazione

Per poter definire un buon algoritmo genetico è necessario trovare una buona codifica per gli individui; in questo caso conviene utilizzare degli array di lunghezza fissata, pari a quella della chiave k, in cui ciascun gene è una lettera dell'alfabeto che si sta utilizzando. In questo caso sia l'alfabeto in chiaro che quello criptante sono composti dalle lettere inglesi minuscole ($N = 26$).
Un secondo elemento importante è la funzione di fitness; per costruirla si è scelto di utilizzare i digrammi (combinazioni di 2 lettere) e trigrammi (combinazioni di 3 lettere) più frequenti nella lingua inglese [1] e di assegnare un punteggio all'apparizione di ciascuno. In questo modo la fitness di un individuo è calcolata come la somma tra tutti i punteggi dati dai digrammi e trigrammi che appaiono nel testo che viene decriptato utilizzando come chiave l'individuo stesso.

$$f(x) = 4 \sum_{i \in MCB} C(i, x) + 9 \sum_{j \in MCT} C(j, x) \tag{2.1}$$

in cui MCB e MCT sono, rispettivamente, gli insiemi con i digrammi e trigrammi più comuni e $C(i, x)$ è il numero di occorrenze del N-gramma i nel testo decriptato usando come chiave l'individuo x.
L'operatore di *selezione* è stato implementato utilizzando un campionamento pesato basato

sulla fitness, cioè tale per cui gli individui più adatti avessero più possibilità di riprodursi.[1] É da notare che si è scelto di applicare una variante nel processo di selezione, detta *elitismo*: questa consiste nel salvare immutati gli individui migliori e solo successivamente applicare a tutta la popolazione l'operatore di selezione. Questo metodo garantisce che gli individui più adatti siano sempre preservati.
Per l'operatore di *crossover* si è scelto di utilizzare un crossover standard, detto *1-point Crossover*: questo sceglie casualmente una posizione in cui tagliare i due genitori e genera i figli scambiando le due code. Ad esempio:

Genitore 1:	a b c | d e f
Genitore 2:	a d k | n l w
Figlio 1:	a b c | n l w
Figlio 2:	a d k | d e f

Per l'implementazione della *mutazione* si è scelto di utilizzare un solo tipo di mutazione, cioè il cambio di lettera, che consiste nel cambiare casualmente una delle lettere della chiave con un'altra diversa, anch'essa scelta a caso.
Infine la popolazione iniziale è stata generata con una grandezza fissata e con individui casuali, ma sempre e solo con elementi di lunghezza pari alla lunghezza della chiave.

Nel caso in cui la lunghezza della chiave non fosse nota, si è scelto di dividere la popolazione in tante sotto-popolazioni, all'interno di un range in cui si stima sia compresa la dimensione giusta, in modo da poter far girare l'algoritmo su ciascuna di esse. Inoltre, ad ogni nuova generazione la dimensione di ciascuna sotto-popolazione viene scelta in modo proporzionale al massimo della funzione di fitness, mantenendo costante la dimensione totale della popolazione;[2] in questo modo, le sotto-popolazioni con una lunghezza di chiave più probabile avranno sempre più elementi, facilitando la ricerca della soluzione, mentre le altre ne avranno meno.

Risultati

I valori tipici dei parametri che sono stati usati per lo studio sono riportanti nella Tabella 2.1 e sono stati ottenuti dopo varie run di test. É da precisare che tali valori permettono di ottenere dei buoni risultati per lunghezze della chiave intorno a 10; se la lunghezza fosse più grande sarebbe opportuno aumentare la popolazione e il numero di generazioni, in modo da poter esplorare meglio lo spazio delle possibilità. Invece gli altri parametri possono essere lasciati invariati.

Parametro	Valore
Popolazione	400
Generazioni	50
Elitismo	10%
P. Crossover	75%
P. Mutazione	5%

Tabella 2.1: Valori tipici dei parametri per GA (polialfabetico)

Si osserva che, fissata la lunghezza di chiave, l'aumentare della popolazione non sempre garantisce dei vantaggi all'algoritmo genetico. Infatti, come riportato in Figura 2.1, una

[1] Una possibilità alternativa sarebbe stata quella di ordinare tutti gli individui nella popolazione con fitness decrescente ed implementare una roulette truccata. Ad esempio $\text{INT}(N_{pop} \cdot r^p)$ con r numero casuale nell'intervallo $[0, 1)$ e $p > 1$

[2] In alternativa si potrebbe scegliere di utilizzare la media di tutte le fitness in ciascuna sotto-popolazione.

popolazione piccola non permette di raggiungere la fitness ottima, cioè non si trova la chiave del cifrario, mentre una molto grande permette di arrivare alla soluzione cercata, ma potrebbe richiedere più generazioni di una popolazione con grandezza media.[3]

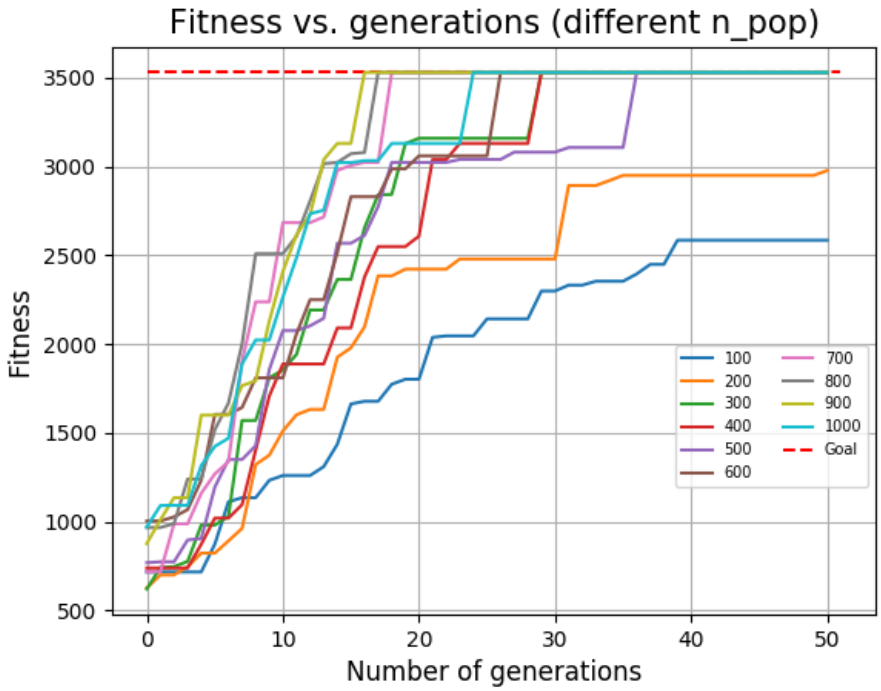

Figura 2.1: Andamento della fitness massima con il numero di generazioni per diverse dimensioni della popolazione. La chiave usata per criptare è *"carnevalesco"*.

La seconda versione di questo programma è stata concepita con l'idea di non conoscere l'esatta lunghezza della chiave usata per il cifrario polialfabetico. Per questo si è scelto di far ripetere l'algoritmo precedente per diverse lunghezze della chiave, fino ad identificare quella giusta e di fare variare la dimensione delle sotto-popolazioni, come descritto nella Sezione 2.1. Dopo un'attenta analisi dei risultati e dei tempi di esecuzione, si è scelto di utilizzare la media della funzione di fitness e non il massimo per l'aggiornamento del numero di individui delle sotto-popolazioni; questo perché la media causa una variazione apprezzabile ma inferiore in modo da non essere troppo condizionata dalla crescita repentina dei massimi per lunghezze di chiave piccole e non arrivare ad avere sotto-popolazioni troppo grandi, che comportano un aumento dei tempi di esecuzione ma non migliorano i risultati.
La Figura 2.2 riporta l'andamento per il massimo della funzione, in ciascuna sotto-popolazione. Questo permette di osservare che tutti i valori dei massimi aumentano, ma quello della sotto-popolazione associata alla giusta dimensione della chiave ha un valore decisamente superiore.

Infine è da notare che questo metodo di crittoanalisi non funziona per qualsiasi lunghezza del testo; in particolare si è osservato che se il rapporto tra la lunghezza del testo e la lunghezza della chiave non è almeno superiore a 15, allora l'individuo con la fitness massima non corrisponde alla chiave con cui è stato criptato il testo. Questo potrebbe essere dovuto al fatto che per ogni carattere della chiave è necessario avere un numero minimo di confronti

[3]Nelle run di test si è osservata una relazione lineare tra la dimensione della popolazione e il tempo di esecuzione del programma. Inoltre, nonostante si sia fatto un uso massiccio del calcolo parallelo, i tempi di un linguaggio non compilato come Python possono estendersi notevolmente.

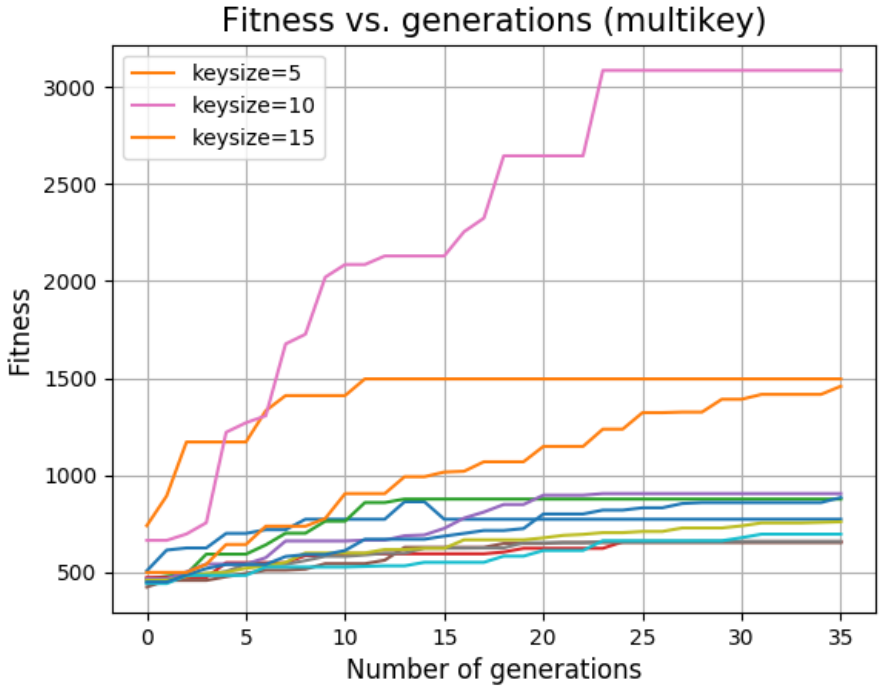

Figura 2.2: Andamento della fitness massima con il numero di generazioni per sotto-popolazioni. La chiave usata per criptare è *"alfabetico"*.

all'interno del testo, basati su digrammi e trigrammi, per poter identificare correttamente la lettera che fa parte della chiave. Nel caso ce ne fossero di meno, è possibile che lettere diverse diano un valore della funzione di fitness maggiore.

2.2 Algoritmo Genetico: cifrario monoalfabetico

Implementazione

Per poter implementare l'algoritmo genetico nel caso del cifrario a permutazione è stato necessario fare diverse modifiche rispetto alla versione riportata in Sezione 2.1. Innanzi tutto, in questo caso gli individui sono codificati come degli array di interi compresi nell'intervallo $[0, 25]$, in cui ciascun valore si riferisce al numero della lettera nell'alfabeto. Infatti la chiave in questo tipo di cifrario è una permutazione dell'alfabeto in chiaro. É stato anche necessario modificare la funzione di fitness[4]e si è scelto di utilizzare una funzione basata sulla differenza tra la frequenza assoluta dei digrammi, nella lingua inglese, e la frequenza riscontrata nel testo decriptato utilizzando un individuo come chiave. [2][3] [5]

$$f(x) = 1 - \alpha \sum_{i,j \in A} |F_{ij}(x) - E_{ij}| \tag{2.2}$$

[4]Si è osservato che in questo caso la fitness del testo decriptato, data dalla (2.1), non era il massimo assoluto e inoltre non c'era una buona correlazione tra l'aumento della funzione e il numero di cifre esatte nell'individuo.

[5]Un metodo alternativo sarebbe quello di usare direttamente la frequenza delle lettere, che richiede un tempo computazionale minore; sfortunatamente questo metodo non dà buoni risultati per testi corti, come in questo caso. Si potrebbe rendere la funzione più precisa sfruttando anche i trigrammi, ma questo è computazionalmente molto dispendioso. [4]

dove α è un coefficiente positivo, $F_{ij}(x)$ è la frequenza del digramma formato dalle lettere ij nel testo decriptato usando come chiave l'individuo x, E_{ij} è la frequenza "assoluta" del digramma e gli indici della sommatoria corrono su tutte le lettere dell'alfabeto.

Nonostante il processo di selezione sia stato lasciato invariato, sono state necessarie delle modifiche anche a crossover e mutazione, infatti rispetto al caso precedente, qui vengono usate permutazioni di una sequenza di elementi diversi e quindi operatori come quelli definiti precedentemente finirebbero per creare degli individui non corretti. Per questo motivo si è scelto di applicare un crossover definito appositamente per problemi con permutazioni, detto *Cycle Crossover*: quest'operatore sceglie random uno dei genitori e fissa il primo elemento, a quel punto cerca la posizione in cui si trova tale elemento nell'altro genitore e continua a fissare elementi come nel primo finché non torna ad una posizione già occupata. A questo punto si è formato un ciclo e il processo ricomincia dalla prima posizione libera, sempre scegliendo un genitore casualmente. Ad esempio:

$$\begin{aligned} \text{Genitore 1}: &\quad 5\,2\,6\,0\,3\,4\,1 \\ \text{Genitore 2}: &\quad 6\,5\,1\,2\,4\,3\,0 \\ \text{Figlio 1}: &\quad 5\,2\,6\,0\,4\,3\,1 \\ \text{Figlio 2}: &\quad 6\,5\,1\,2\,3\,4\,0 \end{aligned}$$

Inoltre sono state implementate due mutazioni diverse: la prima è lo scambio tra due posizioni scelte casualmente, mentre la seconda è l'inversione dell'ordine in cui appaiono tutti gli elementi in un certo range, sempre scelto a caso.
Anche in questo caso la popolazione iniziale è scelta random con individui di lunghezza $N = 26$, permutazioni dell'array di interi $[0, \ldots, 25]$.

Risultati

I valori tipici dei parametri che sono stati usati per lo studio sono riportanti nella Tabella 2.2. In questo caso è necessario aumentare il numero di individui nella popolazione rispetto al caso polialfabetico, dato che lo spazio delle configurazioni è molto più grande ($N! \approx 4 \cdot 10^{26}$ e $N^{15} \approx 2 \cdot 10^{21}$, per una chiave di lunghezza 15). Inoltre l'aggiunta di una mutazione in più e l'aumento della probabilità permette di avere una maggiore diversificazione all'interno della popolazione.

Parametro	Valore
Popolazione	800
Generazioni	150
Elitismo	10%
P. Crossover	75%
P. Mutazione	8%

Tabella 2.2: Valori tipici dei parametri per GA (monoalfabetica)

Nella Figura 2.3 è riportato il grafico con l'andamento della fitness, in funzione del numero di generazioni (rispetto a (2.2) è moltiplicata per 100), e il numero di lettere corrette nell'alfabeto criptante; si nota che è presente una buona correlazione tra l'aumento della funzione di fitness e il numero di lettere esatte nell'individuo, cosa che invece non si realizzava utilizzando la fitness definita in (2.1).
Inoltre questo grafico riporta un problema che si è incontrato spesso: la fitness massima corrisponde ad avere corrette solo 24 lettere, a differenza delle 26 cercate. Questo è dovuto al fatto che le 3 lettere che appaiono meno nell'alfabeto inglese (e nei bigrammi), cioè la

terna "j q z", non riescono ad essere correttamente identificate dalla funzione e quindi sono spesso scambiate. Nonostante ciò, il testo decriptato usando un individuo in cui 23 o 24 lettere sono corrette è comprensibile e quindi queste possono essere modificate a posteriori.

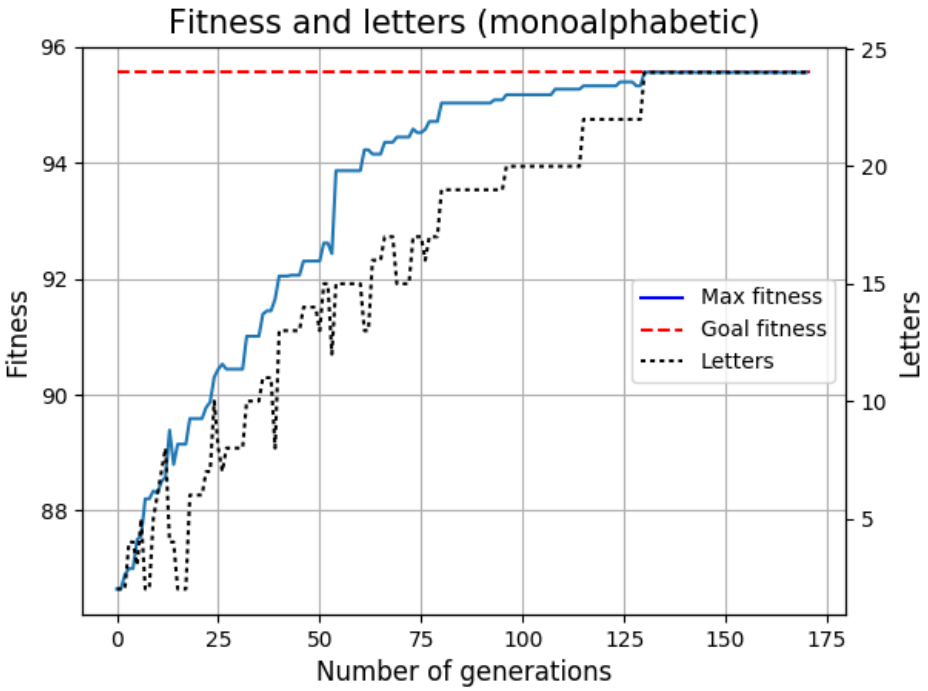

Figura 2.3: Andamento della fitness massima e numero di lettere corrette in funzione del numero di generazioni (GA monoalfabetico).

Un'altra prova effettuata durante le run di test è quella di fissare le grandezze riportate nella Tabella 2.2 e di cambiare il tipo di crossover, utilizzando altre due versioni apposite per i problemi con permutazioni di una sequenza. In particolare la scelta è ricaduta su *Order Based Crossover (OBX)* e *Modified Order Crossover (MOC)*. [5]
La Figura 2.4 mostra come l'utilizzo di diversi tipi di crossover possa influire notevolmente sull'andamento della funzione di fitness.

Un'ultima considerazione importante riguarda il comportamento dell'algoritmo al variare del seme del generatore di numeri casuali [6][6]: cambiando il seme del generatore i risultati arrivano comunque al valore massimo della fitness, ma possono richiedere un numero diverso di generazioni (anche di alcune decine).

Infine si è scelto di combinare le metodologie riportate in questa sezione e in quella precedente per poter creare un codice capace di decrittare un testo cifrato con una delle due metodologie precedentemente esposte. L'idea è semplicemente quella di eseguire simultaneamente i due algoritmi genetici, per due popolazioni separate (una con le sotto-popolazioni descritte in Sezione 2.1 e una con la popolazione qui descritta). Al termine del numero di generazioni fissate si studiano i massimi delle fitness raggiunte e si decide di quale cifrario si tratta. Questo passaggio viene fatto monitorando i valori delle fitness massime che le popolazioni assumono quando i testi sono criptati con metodi diversi: in particolare si è visto che (2.1) non supera mai il valore di 1500 quando il testo viene criptato con il cifrario monoalfabetico, mentre (2.2) (sempre moltiplicata per 100) non supera mai il valore di 89.5 per il criptaggio con il cifrario polialfabetico.

[6]In questo studio è stato utilizzato il modulo RANDOM, che a differenza della versione implementata in NUMPY, permette l'applicazione del calcolo parallelo.

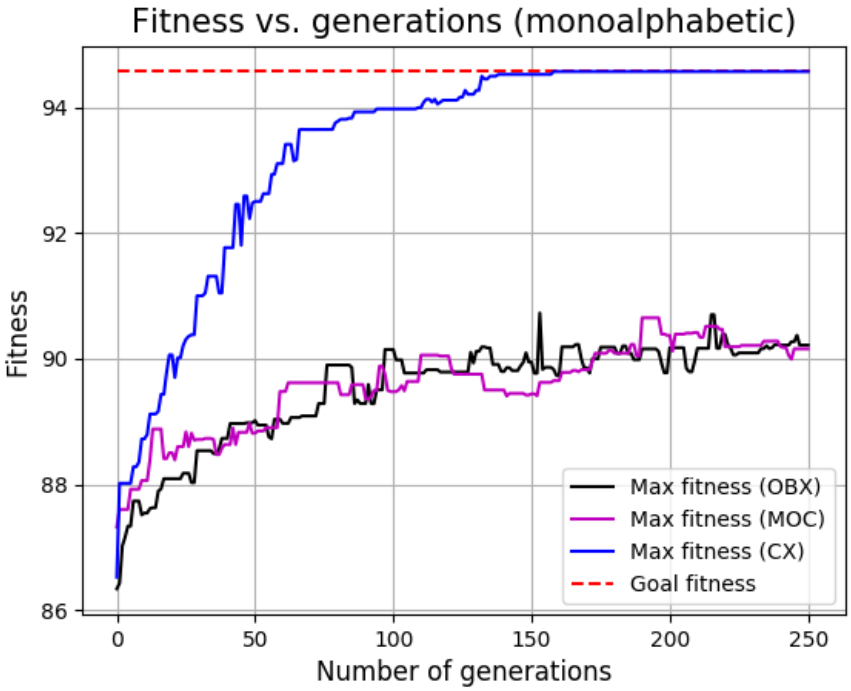

Figura 2.4: Andamento della fitness massima per diversi tipi di crossover.

Conclusioni

In questo elaborato è stata studiata l'applicazione di algoritmi genetici per la decrittazione di testi in inglese, codificati con metodi classici.
Per farlo sono stati descritti due diversi metodi di criptaggio, uno monoalfabetico (cifrario a permutazione) e uno polialfabetico (cifrario di Vigenère), e ne sono state messe in luce le caratteristiche principali. Successivamente sono stati introdotti gli algoritmi genetici, i metodi e gli operatori che li caratterizzano e il metodo di scelta dei parametri per renderli performanti.
A quel punto si è parlato delle implementazioni dei due tipi di algoritmi genetici, riportando le diverse funzioni di fitness necessarie per ottenere dei risultati soddisfacenti e i valori tipici dei parametri utilizzati durante lo studio. Inoltre sono stati discussi i punti di forza, come la possibilità di ottenere sempre una soluzione sub-ottima, e di debolezza, come la difficoltà ad identificare le lettere meno frequenti, per le varie implementazioni tramite l'analisi delle funzioni di fitness.
In conclusione, questo lavoro ha permesso di mostrare come gli algoritmi genetici siano un potente metodo per risolvere problemi di decriptazione di testi, in quanto permettono di ottenere le chiavi di cifratura in tempi brevi (paragonati alle dimensioni degli spazi di soluzioni di questi problemi che sono super-esponenziali).

I possibili sviluppi futuri per questo lavoro sono di due tipi: l'ottimizzazione degli algoritmi e l'applicazione ad altri problemi. Il primo consiste nel migliorare la funzione di fitness, modificandone i parametri o trovando altre forme che permettano di raggiungere la soluzione più facilmente. Inoltre si possono modificare gli algoritmi in modo da ridurre i tempi di esecuzione, per esempio traducendo i codici in linguaggi compilati, come Cython. Il secondo, decisamente più interessante, è provare ad attaccare cifrari diversi, come i cifrari a trasposizione o i cifrari knapsack, utilizzando questa stessa metodologia.

Bibliografia

[1] Peter Norving. *Cryptanalysis Hints*. URL: https://www3.nd.edu/~busiforc/handouts/cryptography/cryptography%20hints.html (visitato il 22/06/2021).

[2] A. Clark. ≪Modern optimisation algorithms for cryptanalysis≫. In: (1994), pp. 258–262. DOI: 10.1109/ANZIIS.1994.396969.

[3] Barry Keating. *English Letter Frequency Counts: Mayzner Revisited*. URL: https://norvig.com/mayzner.html (visitato il 22/06/2021).

[4] Richard Spillman et al. ≪Use of a Genetic Algorithm in the Cryptanalysis of Simple Substitution Ciphers≫. In: *Cryptologia* 17.1 (1993), pp. 31–44. ISSN: 0161-1194. DOI: 10.1080/0161-119391867746.

[5] A.J. Umbarkar e P.D. Sheth. ≪Crossover Operators in Genetic Algorithms: a review≫. In: *ICTACT JOURNAL ON SOFT COMPUTING* (ott. 2015). ISSN: 2229-6956. DOI: 10.21917/ijsc.2015.0150.

[6] Python Documentation. *random — Generate pseudo-random numbers*. URL: https://docs.python.org/3/library/random.html (visitato il 22/06/2021).

[7] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. 1ª ed. Addison-Wesley Professional, 1989. ISBN: 0201157675.

[8] Melanie Mitchell. *Introduction to genetic algorithms*. Third Printing. Complex Adaptive Systems. The MIT Press, 1998. ISBN: 0262631857,0262133164.

[9] Simon Singh. *The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography*. First Edition. Fourth Estate e Doubleday, 1999. ISBN: 9781857028799.