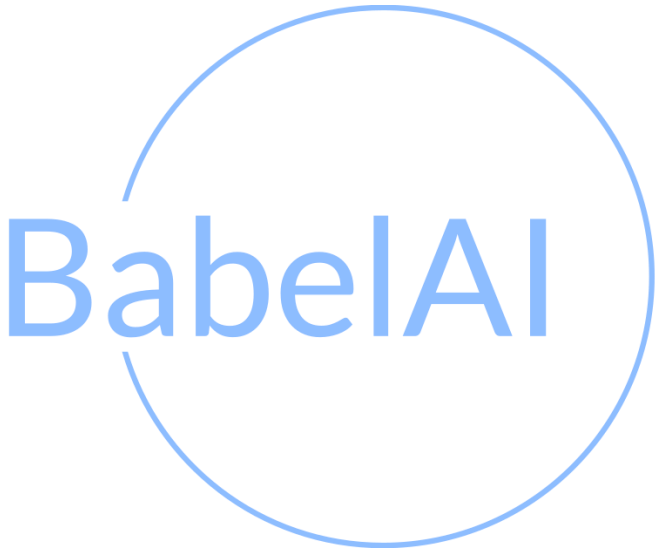
UNIVERSITÀ DEGLI STUDI
DI SALERNO

Corso di Laurea in Informatica

Fondamenti di Intelligenza Artificiale

BabelAI - Documento



Anno Accademico: 2022/23

Docente:

Prof. Fabio Palomba

Studenti:

Mirko Danilo Pacelli 0512112321

Indice

1. Definizione del problema

1.1 Introduzione

La lingua inglese ha un vocabolario molto ampio, con stime che identificano più di 250.000 diverse parole. Nonostante ciò, letterati ed insegnanti inglesi affermano che basta la conoscenza delle 3000 parole inglesi più comuni per avere una comprensione del 90-95% di conversazioni, libri, giornali o film.

BabelAI è un software basato su un algoritmo genetico che genera una parola inglese reale, confrontata con un dataset contenente le 3000 parole inglesi più comuni, dato in input il numero di lettere della parola da generare. Lo scopo del progetto è definire tale algoritmo genetico e ottimizzarlo per migliorarne le prestazioni.

1.2 Specifica dell'ambiente

Di seguito è riportata la specifica PEAS del problema:

- **Performance**: la misura di prestazione principale adottata prevede la minimizzazione del tempo di esecuzione medio dell'algoritmo;

- **Environment**: l'ambiente in cui l'agente opera è formato da tutte le possibili combinazioni di lettere che può assumere una parola, la cui lunghezza è decisa all'inizio dall'utente. L'ambiente è:
  - **Completamente osservabile**: in ogni momento l'agente conosce tutte le reali parole inglesi presenti nel database;
  - **Stocastico**: lo stato successivo non è determinabile dato lo stato corrente a causa della presenza di comportamenti casuali.
  - **Discreto**: l'ambiente fornisce un numero limitato di azioni distinte;
  - **Statico**: non è presente l'aggiunta di nuovi dati oltre quelli definiti inizialmente;
  - **Singolo**: è presente un solo agente;
  - **Episodico**.

- **Actuators**: l'agente mostra a video informazioni come la lista di generazioni con popolazione e la parola inglese trovata, specificando anche la generazione a cui appartiene e il tempo di esecuzione;

- **Sensors**: l'agente usa il dataset contenente le 3000 parole inglesi più comuni e prende in input dall'utente il numero di lettere di cui si vuole generare una parola (da 2 a 14 lettere possibili).

2. Soluzione del problema

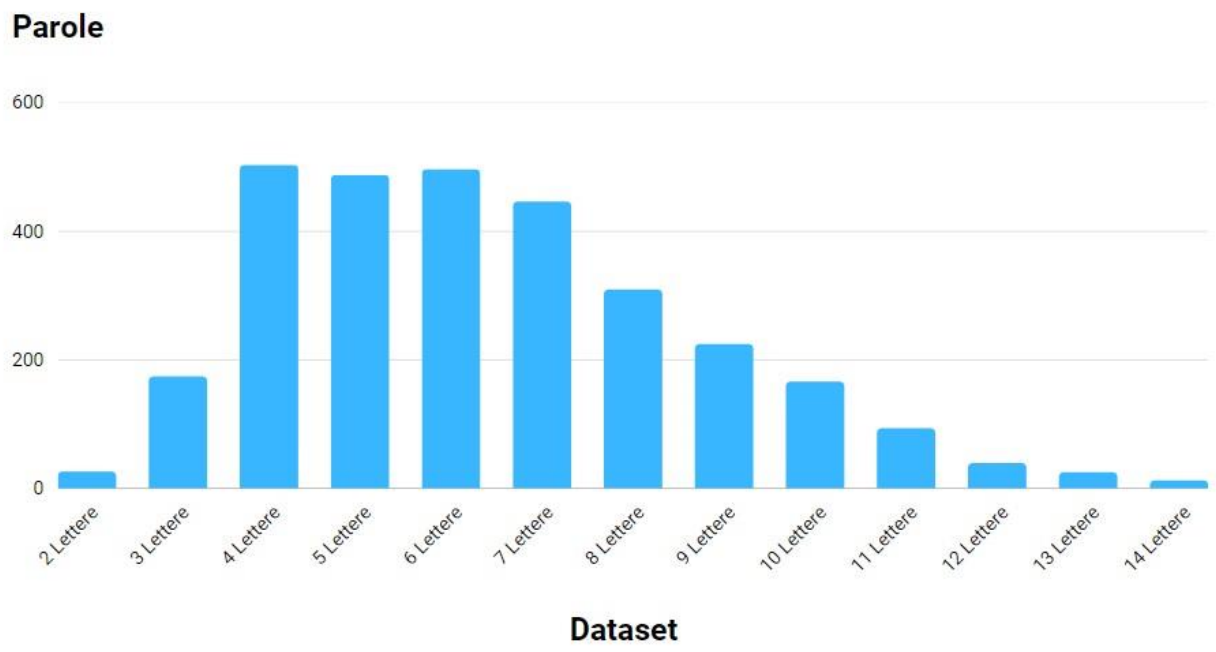
La seguente sezione del documento descrive in dettaglio le specifiche dell'algoritmo genetico usato per il problema, mentre le versioni successive, oltre ad analisi delle prestazioni e comparazioni, verranno descritte nella sezione successiva del documento.

2.1 Tecnologie utilizzate

Il linguaggio di programmazione scelto è Java 17, senza l'uso di librerie esterne.

2.2 Dataset utilizzato

Il dataset (FullDictionary.txt) contiene l'elenco delle 3000 parole inglesi più comuni, che vanno da un minimo di 2 lettere ad un massimo di 14. L'utente dovrà, all'inizio dell'esecuzione del software, inserire il numero di lettere della parola da generare. È stato quindi deciso di separare l'originale dataset in 13 dataset più piccoli contenenti soltanto parole della medesima lunghezza. Nella fase iniziale, il software caricherà quindi il dataset corretto una volta ottenuto l'input dall'utente e ciò migliora nettamente le prestazioni senza drawback (visto che, per esempio, se si vuole generare parole lunghe 5 lettere non è necessario avere anche parole di 4 o 6 lettere, che verrebbero scartate immediatamente). I dataset originati da questa decisione sono descritti nel seguente grafico:



Per le analisi delle prestazioni verranno considerati tre dataset diversi (4, 8 e 11 lettere), e verrà usato in alcuni casi il dataset più piccolo di 14 lettere come misura di affidabilità dell'algoritmo.

2.3 Rappresentazione degli individui della popolazione

Gli individui della popolazione sono rappresentati da una stringa di x lettere (definite dall'utente all'inizio dell'esecuzione del software), ed ogni lettera ha come valori definiti le 26 lettere dell'alfabeto inglese.

Nella prima versione dell'algoritmo, la size della popolazione è di 10 individui per generazione, e non c'è alcun controllo nella formazione degli individui della popolazione iniziale.

2.4 Funzione di fitness

La funzione di fitness usata per determinare quali individui siano più forti di altri si basa sulla distanza di Levenshtein (o distanza di edit), una misura per calcolare la distanza fra due stringhe A e B, ovvero il numero minimo di modifiche elementari che consentono di trasformare A in B.

Nel caso dell'algoritmo genetico definito, in cui la lunghezza delle parole è fissata, sono ignorate le modifiche elementari di cancellazione e inserimento di un carattere, e si considera solo la sostituzione di un carattere con un altro.

Da queste considerazioni, si è definita una funzione di fitness che analizza il numero di match tra due stringhe, ovvero il conteggio delle lettere corrette nella posizione corretta. Il confronto viene eseguito tra ogni individuo della generazione e ogni parola del dataset usato, e ogni individuo avrà il valore di fitness più alto trovato.

Quando il valore di fitness è pari alla lunghezza della parola, allora è stata trovata una parola inglese reale, che verrà stampata a schermo con informazioni sulla generazione corrente e il tempo di esecuzione in ms.

2.5 Selezione degli individui

La selezione degli individui è effettuata tramite un algoritmo *K-way Tournament* con K=2 (20% della popolazione) nella prima versione, e tra i due concorrenti viene scelto l'individuo con valore di fitness maggiore. L'algoritmo viene eseguito *N* (size della popolazione, nella prima versione 10) volte, e gli individui scelti faranno parte della prossima generazione dopo le operazioni di crossover e mutazione.

2.6 Crossover

L'operazione di crossover usa un algoritmo *single-point*, ovvero viene scelto un indice casuale ed il figlio che le parole genitori generano avrà una prima sequenza di caratteri ereditata dal primo genitore, e una seconda sequenza ereditata dal secondo genitore. La probabilità di crossover definita è pari a 1, quindi viene eseguita con una probabilità del 100%.

2.7 Mutazione

L'operazione di mutazione usa un algoritmo *swap* che consiste nello scambio di un carattere con un altro scelto casualmente dall'alfabeto inglese.

Nella prima versione dell'algoritmo la probabilità di mutazione è pari a 0.05 (5%), ed è indipendente per ogni carattere della parola. Questo meccanismo è essenziale nelle ultime generazioni, formate da individui con valore di fitness pari alla lunghezza della parola – 1, perché permette di trovare più velocemente la lettera mancante a completare la reale parola inglese.

2.8 Condizione di stop

L'esecuzione del software termina con successo se viene trovata una parola inglese reale, e viene stampata a video con informazioni sul tempo di esecuzione e generazione di cui fa parte, oppure con insuccesso dopo 500 generazioni.

2.9 Passi dell'algoritmo

Il primo passo dell'algoritmo è la richiesta di un input da parte dell'utente per fissare la size dell'individuo.

Il secondo passo è il caricamento in una lista delle parole del dataset corrispondente al numero di lettere scelto.

Il terzo passo è la generazione della prima popolazione, senza controlli sulla validità delle parole ottenute (modificato nelle successive versioni).

Il quarto passo è il calcolo del valore di fitness di ogni individuo della popolazione.

Il quinto passo è la formazione della nuova generazione, passando per le operazioni di selezione, crossover e mutazione.

Il quarto ed il quinto passo vengono eseguiti ciclicamente fino al raggiungimento della condizione di stop.

2.10 Analisi performance (prima versione)

Le prestazioni della prima versione dell'algoritmo sono testate e documentate sul dataset di 4, 8 e 11 lettere, mentre per i restanti dataset verrà usato il metodo empirico.

Sono tenuti in considerazione il tempo di esecuzione medio ed il numero medio di generazioni.

Dopo 10 test:

Dataset	Tempo di esecuzione medio	N° di generazioni medio
4 Lettere	37,8 ms	46
8 Lettere	78 ms	237,28
11 Lettere	-	-

La prima versione dell'algoritmo ha buone prestazioni temporali con dataset di 4 e 8 lettere, sebbene per quest'ultimo ci sia un alto numero di generazioni create, mentre invece non riesce a trovare una parola corretta in inglese dal dataset di 11 lettere, arrivando alla stopping condition delle 500 generazioni.

3. Versioni successive

3.1 BabelGA_V2

La seconda versione dell'algoritmo modifica la generazione della prima popolazione controllando che tutti gli individui siano parole inglesi con un formato corretto, ovvero con sequenze di massimo 2 vocali consecutive e massimo 3 consonanti consecutive.

Dopo 10 test:

Dataset	Tempo di esecuzione medio	N° di generazioni medio
4 Lettere	45,4 ms	48,6
8 Lettere	80,5 ms	218
11 Lettere	-	-

Il tempo di esecuzione medio ed il numero di generazioni medio rimane costante; quindi, la funzione di controllo per una popolazione così bassa non dà gli effetti sperati. Per il dataset di 11 lettere c'è un success rate del 7%, e per questo non sono stati riportati valori nella tabella, nonostante comunque un netto miglioramento rispetto alla prima versione con un success rate nullo.

3.2 BabelGA_V3

La terza versione dell'algoritmo modifica la funzione di fitness, non più basata sulla distanza di Levenshtein (o distanza di edit) ma sulla Longest Common Subsequence (LCS), ovvero gli individui possiedono valore di fitness pari alla lunghezza della più lunga sottosequenza di caratteri in comune con le parole nel dataset.

Dopo 10 test:

Dataset	Tempo di esecuzione medio	N° di generazioni medio
4 Lettere	562,8 ms	152,9
8 Lettere	-	-
11 Lettere	-	-

La terza versione mostra un enorme peggioramento in ogni valore, e si ha un basso success rate anche con il dataset di 8 lettere. Risulta quindi inefficace l'implementazione di una funzione di fitness basata sulla LCS, mentre invece la funzione descritta all'origine, dopo altri test non documentati, risulta essere la migliore per questa tipologia di problema. Questa terza versione è stata scartata, e la quarta è stata modellata sulla base della seconda versione.

3.3 BabelGA_V4

La quarta versione dell'algoritmo modifica la size della popolazione (da 10 a 70) e modifica la size del torneo (k passa da 2 a 4).

Dopo 10 test:

Dataset	Tempo di esecuzione medio	N° di generazioni medio
4 Lettere	32,7 ms	2.5
8 Lettere	61.9 ms	23.1
11 Lettere	72.6 ms	29.4

Il tempo di esecuzione per il dataset di 4 lettere è migliorato notevolmente, mentre per il dataset di 8 lettere c'è stato un miglioramento del 25% circa. Il numero di generazioni per entrambi i dataset risulta essere enormemente abbassato, dovuto ovviamente all'aumento della popolazione.

Il dataset di 11 lettere risulta quello più interessante, arrivando ad un success rate del 100% su 10 test, e con notevoli prestazioni sia al livello temporale che di generazioni medie.

Anche il dataset di 14 lettere, dopo alcuni test, restituisce una parola inglese reale con un success rate del 100% e con prestazioni simili al dataset di 11 lettere.

3.4 BabelGA_V5

La quinta versione dell'algoritmo implementa l'elitismo, per garantire che nelle successive generazioni venga mantenuto l'individuo più forte, e viene modificato il crossover, che ha adesso una probabilità di 0.85 (da 1).

Dopo 10 test:

Dataset	Tempo di esecuzione medio	N° di generazioni medio
4 Lettere	31 ms	2.3
8 Lettere	60.8 ms	25.6
11 Lettere	75.2 ms	32.5

Il tempo di esecuzione medio ed il numero di generazioni medio per entrambe le versioni risultano pressappoco invariati rispetto alla quarta versione, ma ha una varianza minore e quindi più consistenza della versione precedente.

3.5 BabelGA_V6

La sesta versione dell'algoritmo modifica l'algoritmo di selezione passando ad un k-way tournament con pressione di selezione, ovvero utilizzando per ogni torneo una roulette wheel pesata in base al valore di fitness di ogni partecipante.

Dopo 10 test:

Dataset	Tempo di esecuzione medio	N° di generazioni medio
4 Lettere	8.6 ms	1.4
8 Lettere	453,2 ms	94,5
11 Lettere	601,2 ms	167,4

Le prestazioni risultano, tranne per il dataset con 4 lettere, completamento peggiorate sia per il tempo di esecuzione che per il numero di generazioni medio. Questa sesta versione è stata perciò scartata e la prossima è modellata sulla base della quinta.

3.6 BabelGA_V7

La settima versione dell'algoritmo modifica la mutazione. Nella prima versione di BabelGA_V7 è stato utilizzato un algoritmo *scramble,* ovvero una permutazione casuale di una sottosequenza casuale di caratteri, scartata per le prestazioni tremendamente basse. Per la versione definitiva di BabelGA_V7 è stato utilizzato invece lo stesso tipo di mutazione, ovvero un *random resetting* con probabilità 0.2 (da 0.05). Tale probabilità però non viene applicata per ogni carattere dell'individuo, ma per ogni individuo stesso della popolazione. Quindi, in poche parole, c'è il 20% di possibilità che l'individuo subisca la mutazione di un carattere, rispetto al precedente 5% di probabilità che ogni carattere dell'individuo subisca mutazione.

Dopo 10 test:

Dataset	Tempo di esecuzione medio	N° di generazioni medio
4 Lettere	31 ms	3,9
8 Lettere	108,4 ms	34,6
11 Lettere	125,5 ms	72,6

La settima versione ha tempo di esecuzione medio e numero di generazioni medio di molto inferiori rispetto alla quinta versione, che risulta invece essere più varia. Il success rate di ogni dataset è del 100%.

4. Conclusioni

La quinta versione dell'algoritmo risulta la migliore, insieme alla quarta anche se più variabile, e dopo altre piccole modifiche non è stato possibile migliorare ulteriormente le prestazioni, comunque già notevolmente soddisfacenti. Sono state effettuate modifiche su ogni caratteristica descritta nella seconda sezione del documento, e sono stati identificati in BabelGA_V5 le migliori scelte. L'algoritmo può inoltre essere modificato ulteriormente con l'utilizzo di thread, ma non verrà provato o testato, ed è possibile utilizzarlo anche le lingue differenti dall'inglese, come l'italiano, che utilizzano lo stesso alfabeto, modificando semplicemente il dataset.