
SICILIAE STVDIVM GENERALE
1434


ING
NF
cogito ergo sum

EVE


evolution, reimplemented

EVE

Programma evolutivo per modelli ODE a generazioni non separate, con supporto al modello a isole a fitness locale e strategie del disastro e della migrazione.

CARATTERISTICHE SALIENTI

- Supporto per il **modello a isole**
- Supporto alla **strategia del disastro**
- Supporto alla **strategia della migrazione**
- Generazioni non separate: **popolazione unica**
- Possibilità di impostare la **tipologia di fitness indipendentemente per ogni isola**
- Selezione dell'individuo gestita da **selettore gaussiano con parametri auto-adattanti** alle statistiche della popolazione
- presenza nell'individuo di **parti temporaneamente inerti** (fino all'intervento di certi operatori di mutazione), **soggette ad evoluzione a lungo termine**
- **17 operatori di mutazione** con probabilità di scelta personalizzabile; predisposizione per la creazione di operatori di mutazione
- **6 metodi di calcolo della fitness inclusi di cui 5 canonici e uno misto**; predisposizione alla creazione di metodi di fitness personalizzati
- Implementazione **pienamente multithreaded**
- **Struttura a Framework**: la libreria di classi fornita consente di creare configurazioni estremamente personalizzate
- **Interfaccia grafica avanzata**, capace di controllare fino a 4 isole evolutive; **grafici avanzati** per controllare l'andamento dell'evoluzione

Inoltre Eve supporta la pre-elaborazione dei profili di espressione genica degli esperimenti, con le seguenti possibilità elaborative

- **Selezione per varianza minima** (eliminazione dei profili di espressione genica piatti)
- Clustering tramite algoritmo non supervisionato **QTClustering** (libreria esterna) per la unificazione dei profili di espressione genica con andamento simile
    - **5 tipologie di distanza** tra modelli ODE impostabili
- Interpolazione **SP-Line** (libreria esterna)

Popolazione a generazioni comunicanti

La popolazione dell'isola evolutiva non si evolve per generazioni separate, piuttosto **la generazione è unica e le successive prelevano gli individui genitori da quella attuale con i nuovi individui che non vengono inseriti in una generazione a parte, bensì nella stessa degli individui genitori**, concorrendo con gli stessi in termini di possibilità riproduttive, avendo quelli con fitness migliore maggiore probabilità di riproduzione

Rispetto ad un modello a generazioni stagne, quello adottato in Eve ha un'aderenza maggiore alla realtà evolutiva delle specie sulla Terra, dove gli individui della stessa specie si incrociano prescindendo dalla generazione di appartenenza; in effetti **il susseguirsi delle generazioni sulla Terra non è un processo discreto, ma un processo continuo. La continuità generazionale è per un certo grado simulata grazie alla reciproca asincronia tra isole evolutive.**

Questo modello comporta che **gli individui vengano eliminati solo qualora la loro fitness non gli consenta di rientrare nella popolazione massima dell'isola evolutiva**; i nuovi individui con fitness eccessivamente scarsa vengono quindi eliminati alla prima successiva fase di rientro nella popolazione massima ammessa per la singola isola evolutiva.

In generale per la migliore fitness dell'isola è possibile dire che essa non può che migliorare o al più rimanere costante nel susseguirsi delle generazioni.

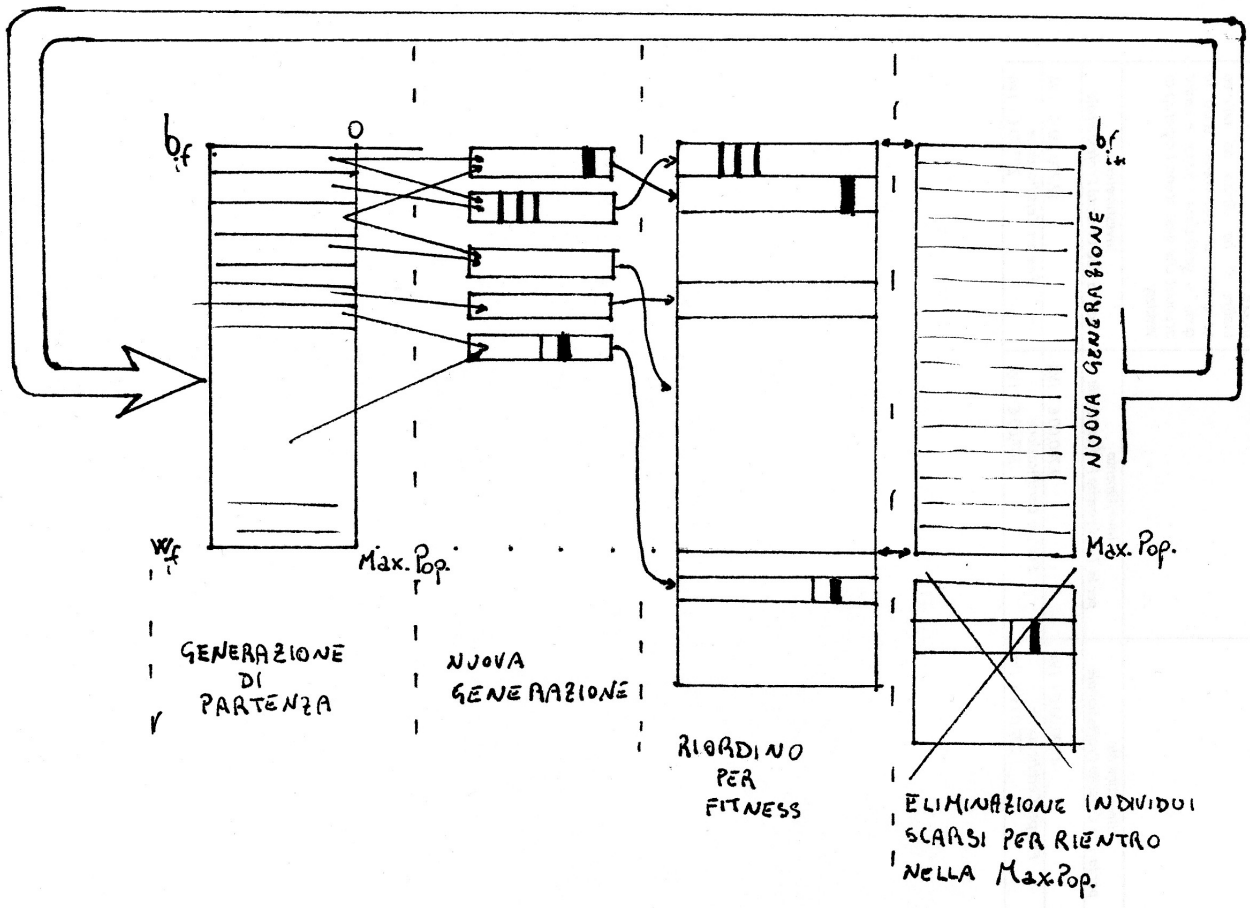

Illustrazione 1: Schema del ciclo generazionale

Selezione dell'individuo gestita da selettore gaussiano con parametri auto-adattanti alle statistiche della popolazione

Il modello di gestione della popolazione dell'isola evolutiva basa il suo funzionamento sull'ordinamento degli individui in base alla loro fitness. Nello specifico **l'implementazione di Eve prevede che una fitness numericamente minore sia migliore di una fitness numericamente maggiore**. **L'ordinamento** non è legato all'ordine in un certo array, piuttosto **è modellato come un posizionamento nella retta dei numeri reali maggiori o uguali a zero**, cioè gli individui con migliore fitness sono quelli più vicini allo zero.
Anziché usare un sistema di pescaggio casuale basato su probabilità linearmente decrescente col peggiorare della fitness, si è preferito usare la **distribuzione gaussiana**. La distribuzione gaussiana **modella correttamente molte statistiche nella natura del mondo terrestre e Eve prende spunto** proprio da questo fatto.

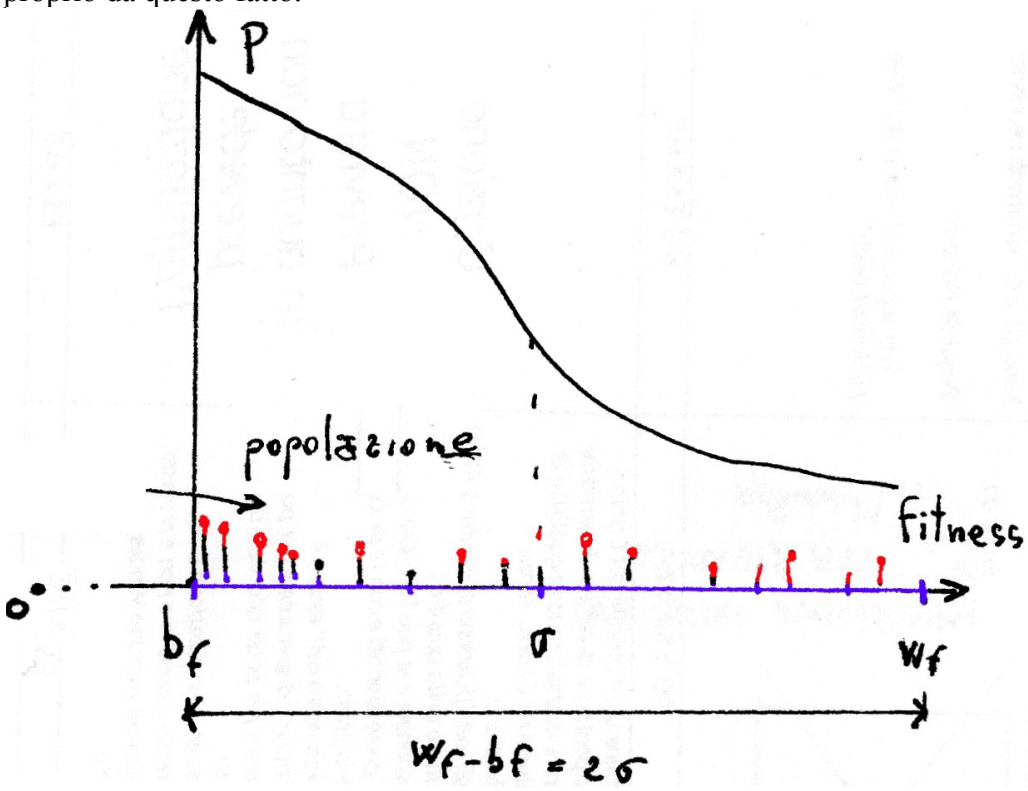

Illustrazione 2: Grafico della probabilità di selezione in funzione della fitness

Il meccanismo di selezione casuale ha lo scopo di **favorire gli individui migliori senza però penalizzare eccessivamente gli individui peggiori**. La volontà di non volere penalizzare fortemente gli individui peggiori ma dare anche a loro una discreta probabilità riproduttiva è basata sulla necessità di **evitare l'incidente** della popolazione dell'isola che si sofferma eccessivamente all'interno di un minimo locale del Pareto Front; questa eventualità porterebbe **la popolazione ad approfondire solo un minimo del fronte di Pareto**, portandola a **sterilizzazione per eccessiva similarità tra individui**, ovvero i nuovi individui avrebbero basse probabilità di esplorare altri minimi del fronte.
Per servire a questo scopo, la distribuzione gaussiana che pilota l'estrazione dei genitori della prossima generazione è stata adattata, o meglio, resa **auto-adattante alle statistiche della fitness della popolazione** dell'isola. Nello specifico, alla variabile aleatoria gaussiana è stato applicato il **valore assoluto**, la sua **media resa uguale alla fitness del migliore individuo** e la **varianza è stata resa uguale a metà dell'intervallo tra la fitness migliore e la fitness peggiore** (parametro comunque configurabile).

17 operatori di mutazione con probabilità di scelta personalizzabile e predisposizione per la creazione di operatori di mutazione personalizzati

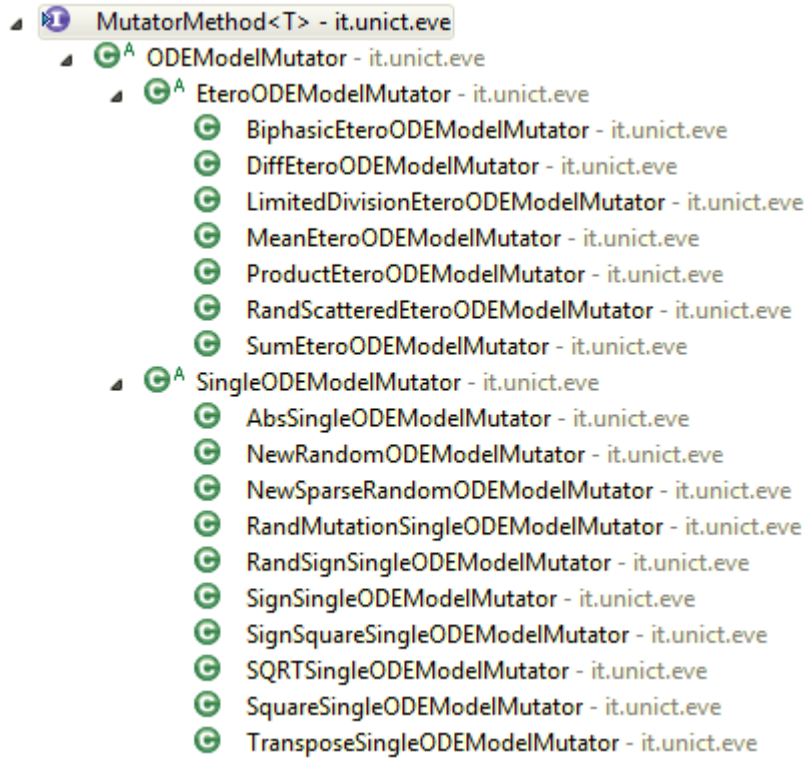

Illustrazione 3: Schema della gerarchia delle classi

Eve include, tra le altre, **17 classi ognuna delle quali implementa un operatore di mutazione per gli oggetti `ODEModel`** (che non sono altro che delle matrici quadrate, che rappresentano un modello ODE).

Gli operatori di mutazione si dividono in **due gerarchie di classi** principali: `SingleODEModelMutator` e `EteroODEModelMutator`.

`SingleODEModelMutator` è la classe capostipite delle classi implementanti operatori di mutazione di tipo mutazione, ovvero che attingono ad un **singolo genitore e da questo generano un nuovo individuo mutando i geni** del genitore secondo certi criteri.

`AbsSingleODEModelMutator`
: applica il valore assoluto ad ogni elemento dell'individuo genitore.

`NewRandomSingleODEModelMutator`
`NewSparseSingleODEModelMutator`
: genera un **nuovo individuo in maniera casuale**, sfruttando l'individuo genitore unicamente per estrarre le informazioni relative alle proprietà dell'`ODEModel`.
Rispettivamente, gli elementi del nuovo individuo sono generati in maniera casuale, oppure sono quelli di una matrice sparsa, generata comunque in maniera casuale.
Questi operatori **aiutano ad ampliare lo spazio di ricerca**, contribuendo a **limitare il problema della stagnazione della popolazione dentro un minimo locale** dello spazio delle soluzioni.

RandMutationSingleODEModelMutator
: il nuovo individuo è frutto della mutazione in maniera casuale di un numero casuale di elementi, selezionati in maniera casuale da quelli del genitore.

SignSingleODEModelMutator
RandSignSingleODEModelMutator
: il nuovo individuo ha elementi che hanno lo stesso valore assoluto di quelli del padre, ma, rispettivamente, hanno segno opposto di tutti gli elementi o di alcuni, scelti in maniera casuale.

SignSquareSingleODEModelMutator
SQRTSingleODEModelMutator
: il nuovo individuo ha elementi che hanno, rispettivamente, il quadrato o la radice quadrata di ogni elemento dell'individuo genitore, pur mantenendo uguale il segno

SquareSingleODEModelMutator
: il nuovo individuo ha elementi che sono il quadrato degli elementi dell'individuo genitore

TransposeSingleODEModelMutator
: il nuovo individuo ha elementi che sono la trasposta degli elementi dell'individuo genitore

EteroODEModelMutator, invece, è la classe capostipite delle classi implementanti **operatori di mutazione di tipo cross-over**, ovvero che attingono ad coppia di genitori e da questi generano un nuovo individuo **incrociando secondo certi criteri i geni dei genitori**.

BiphasicEteroODEModelMutator
: l'individuo figlio ha la prima parte dei geni uguali a quelli del padre e la seconda parte uguale ai geni della madre; la quota delle due porzioni è casuale.

DiffEteroODEModelMutator
ProductEteroODEModelMutator
MeanEteroODEModelMutator
SumEteroODEModelMutator
: il nuovo individuo ha gli elementi che sono frutto, rispettivamente, della differenza, del prodotto, della media o della somma dei corrispondenti elementi degli individui genitori.

LimitedDivisionEteroODEModelMutator
: il nuovo individuo ha gli elementi che sono frutto del rapporto dei corrispondenti elementi degli individui genitori; tuttavia per evitare che la divisione per numeri decimali molto piccoli in valore assoluto generi numeri molto grandi, è applicato un limite; inoltre l'operazione è protetta contro divisioni per lo zero.

RandScatteredEteroODEModelMutator
: il nuovo individuo ha elementi che sono del padre e della madre, intercalati tra loro in maniera casuale

A capo della gerarchia delle classi implementati operatori di mutazione per modelli ODE è posta la classe ODEModelMutator; questa classe è astratta e consente, attraverso la sua **derivazione, di creare nuovi operatori evolutivi a piacere**.

Presenza nell'individuo di parti inerti soggette ad evoluzione a lungo termine

L'`ODEModel` non è altro che una matrice quadrata; tuttavia al momento della simulazione, la parte della matrice considerata è solo quella triangolare superiore e la diagonale; questo comporta che in effetti la parte triangolare inferiore non rientri nel calcolo del valore di fitness.
Questo fenomeno comporta la creazione di una **riserva di variabilità all'interno della popolazione, data da parti dell'individuo che sono dormienti**; queste parti sono **libere di variare durante le generazioni in quanto non subiscono l'influenza della selezione** (non contribuendo al calcolo della fitness).
Nel momento in cui l'operatore evolutivo `TransposeSingleODEModelMutator` interviene, la triangolare inferiore diventa la triangolare superiore, generando un **individuo nuovo**, slegato dagli altri, ovvero **frutto di continue mutazioni e incroci non supervisionati dal controllo della fitness**. Ancora una volta, vi è un **contributo nell'ampliare lo spazio di ricerca delle soluzioni** ed evitare la stagnazione nei minimi locali.

Questa condizione ricorda quella **parte del DNA umano apparentemente priva di funzione poiché non soggetta a trascrizione in RNA**, ma che alcuni studiosi considerano semplicemente DNA non più funzionale (**junk DNA**), accumulato nei millenni ma che ha perso la sua funzione nel corso dell'evoluzione. Nel momento in cui intervengono particolari condizioni, ad esempio una mutazione, parte di questo codice genetico potrebbe tornare ad essere funzionale, contribuendo alla ricchezza del corredo genetico di una specie.

6 metodi di calcolo della fitness inclusi di cui 5 canonici e uno misto; predisposizione alla creazione di metodi di fitness personalizzati

L'interfaccia `MatrixDistanceCalculator` definisce il metodo per calcolare la distanza tra due matrici della classe `Matrix`; **in Eve, la fitness** di un `ODEMoel` è **definita dalla distanza che intercorre tra la matrice dei campioni reali** (l'esperimento, istanza della classe `Experiment`) **e la matrice dei campioni calcolati dall'individuo** `ODEModel`.
Pertanto tutte le classi che calcolano la distanza tra due `Matrix` sono adatte a ricoprire il ruolo del calcolatore di fitness dell'individuo; inoltre, essendo **l'interfaccia `MatrixDistanceCalculator` pubblica, è possibile creare nuove classi che calcolano una versione personalizzata della fitness** a partire da due matrici `Matrix`.

Le 5 classi di base per il calcolo della distanza tra due Matrix (e quindi adatte a ricoprire il ruolo di calcolatore di fitness) sono:

`TotalErrorMatrixDistanceCalculator`
: calcola l'errore totale tra gli elementi corrispondenti delle due matrici

`MeanErrorMatrixDistanceCalculator`
: calcola l'errore medio tra gli elementi corrispondenti delle due matrici

`MaximumErrorMatrixDistanceCalculator`
: calcola l'errore massimo tra gli elementi corrispondenti delle due matrici

`MeanSquareErrorMatrixDistanceCalculator`
: calcola il mean square error tra gli elementi corrispondenti delle due matrici

`RootMeanSquareErrorMatrixDistanceCalculator`
: calcola il root mean square error tra gli elementi corrispondenti delle due matrici

Eve contiene anche la classe `MixedErrorMatrixDistanceCalculator`, atta al calcolo di una sorta di **errore che è sintesi di tre tipi diversi di errore**: l'errore medio, l'errore totale e l'RMSE.
Tale tipologia di fitness misti possono servire qualora non si voglia ottimizzare la soluzione per uno specifico tipo di errore, ma piuttosto si **cerchino individui che presentino un comportamento ibrido, in grado di ottimizzare ogni forma di errore considerato**.
Questo tipo di **fitness ibrido è supportato anche dal modello a isole**, ove **in ogni isola ogni individuo si ottimizza per un certo tipo di fitness e, grazie alla strategia della migrazione, può cambiare isola** portando con se preziosi geni da incrociare coi nativi dell'isola di destinazione.
In natura questo comportamento è stato osservato in quei fenomeni per cui certe specie, cambiando ambiente, portano con se geni da potere ibridare, a volte con successo, con la popolazione locale. Della stessa tipologia sono quei fenomeni che vedono una specie impiantarsi con successo in un ambiente ed espandersi così velocemente da penalizzare la popolazione locale.

Disaster Strategy

Eve cerca di **fare in modo che la popolazione delle isole non si vada ad intrappolare**, nel susseguirsi delle generazioni, **in un qualche minimo locale** dello spazio delle soluzioni. Questo obiettivo è ottenuto **simulando un disastro nell'isola, in seguito al quale la popolazione viene ridotta ad una frazione di quella di partenza**. **La generazione successiva avrà ampio spazio per evolversi** senza l'assillo dei limiti di popolazione massima ammessa nell'isola, inoltre il gestore della popolazione varierà i parametri della variabile aleatoria gaussiana per la selezione casuale, **aumentando ampiamente la larghezza della campana della curva gaussiana**, per consentire anche agli individui con fitness non buona di potersi riprodurre. I **sopravvissuti** al disastro rimangono nella popolazione, **garantendo che i risultati precedenti al disastro non siano comunque persi**.
Questa strategia, quando attiva, **ha mostrato nei test di garantire una più ampia copertura dello spazio delle soluzioni**, riducendo di un certo grado il pericolo di stagnazione della popolazione in un'area ristretta dello spazio delle soluzioni.
Quando la disaster strategy è attiva, l'evento disastro ha **cadenza periodica** rispetto al numero di nuove generazioni create (ad esempio un evento disastro ogni 1000 generazioni).
Nel mondo reale l'uso di un antibiotico eccessivamente blando per combattere un'infezione batterica può portare allo sviluppo di batteri resistenti a quel tipo di antibiotico o comunque in generale l'antibiotico elimina solo gli esemplari batterici più deboli, consentendo agli esemplari migliori di disporre di maggiori risorse e quindi di maggiori possibilità riproduttive.
Sebbene la disaster strategy adottata da Eve crei un fenomeno diverso da quello di cui sopra, le analogie sono pure presenti.

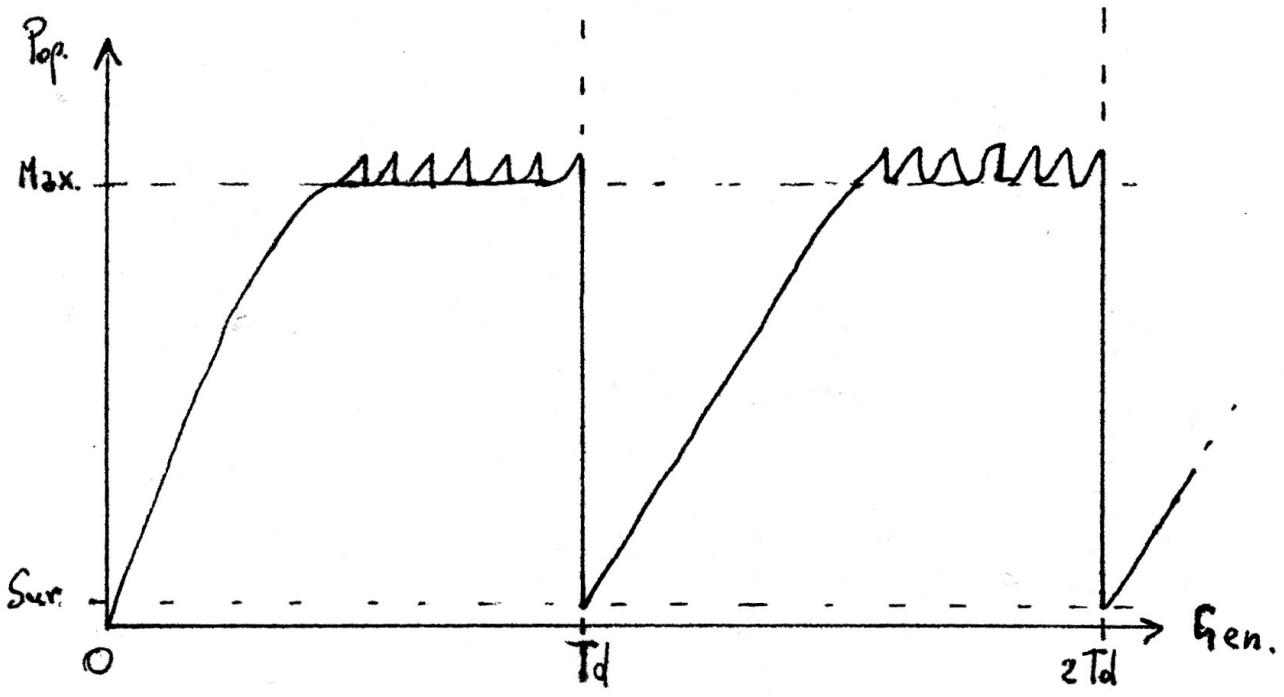

Illustrazione 4: Andamento della popolazione totale lungo il susseguirsi delle generazioni quando la disaster strategy è attiva

Modello a isole

Eve è strutturato come un **mondo** (`EvolutiveWorld`) **fatto di oceani e di isole** (EvolutiveIsland).
Ogni isola è separata dalle altre dall'oceano, ovvero **gli individui di un'isola non possono trasferirsi in un'altra isola** (a meno di un esplicito evento migrazione).
L'isola ha una popolazione (`GaussianProbabilisticObjectSource<ODEModel>`), un ambiente e un set di operatori di mutazione (`UniformProbabilisticObjectSource<ODEModelMutator>`). Ognuna di queste componenti può essere resa diversa tra un'isola e l'altra; in effetti **le isole sono completamente slegate tra loro**, tant'è che eseguono su parallelamente su thread diversi.
Tipicamente, l'Experiment (ovvero i dati reali su cui basare la fitness) è lo stesso per tutte le isole e il gestore della popolazione è gaussiano, ma può essere sostituito con un altro selettore. Invece **la funzione di fitness può essere uguale se si vuole che tutte le isole ottimizzino i loro individui per uno stesso tipo di fitness, oppure la si può impostare diversa affinché ogni isola ottimizzi i propri individui per il suo tipo di funzione di fitness**.
Ancora, le isole mostrano una certa **predisposizione per il GRID computing**, ove ogni computer della griglia può fare da isola o da mondo separato; in effetti **questo modello esprime pienamente al pieno le sue potenzialità quando la migration strategy è attiva**.

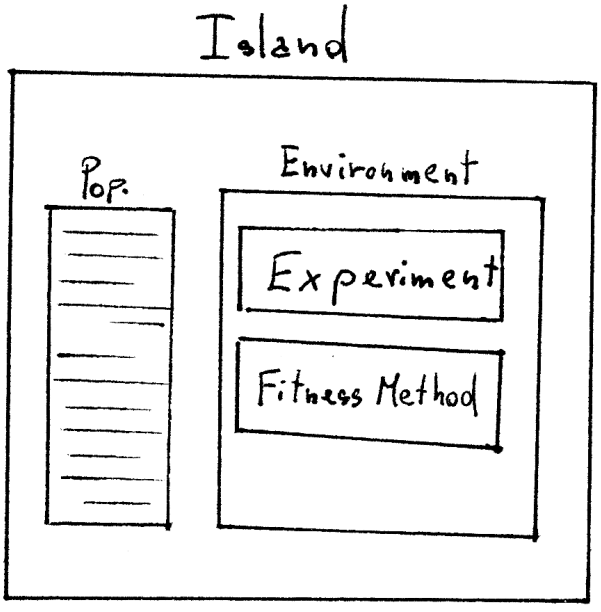

Illustrazione 5: Composizione di un'isola evolutiva

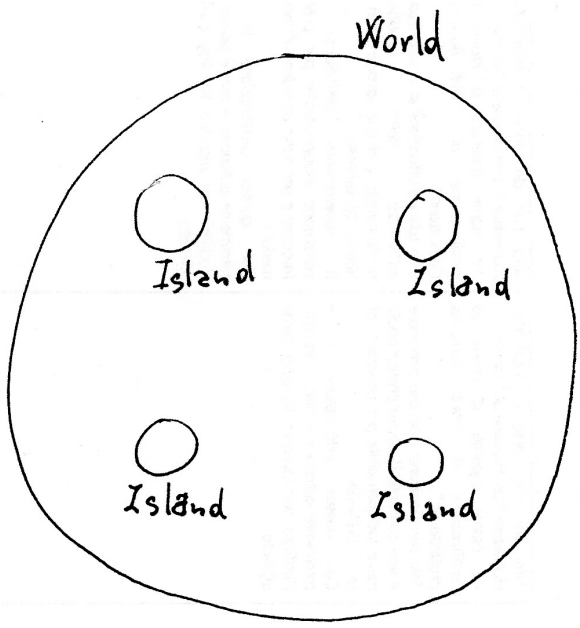

Illustrazione 6: Struttura del mondo evolutivo (EvolutiveWorld) di Eve

Migration Strategy

Una delle strategie più importanti adoperate in Eve è quella della migrazione.
Durante un evento migrazione, **una parte della popolazione di un'isola viene copiata e trasferita verso le altre isole del mondo**; gli individui immigranti vengono inseriti nella popolazione locale secondo la fitness dell'isola e l'accoglienza prevede che il gestore della popolazione non sia eccessivamente severo nei confronti dei nuovi arrivati con fitness scarsa; infatti la **campana gaussiana viene allargata per favorire l'incrocio tra gli individui nativi dell'isola e gli immigrati dalle altre isole**; la successiva generazione ripristinerà le statistiche della campana gaussiana.
Questa strategia è utile in due occasioni.
Quando tutte le isole valutano gli individui secondo lo **stesso tipo di fitness, lo scambio degli individui migliori permette di dividere il carico della popolazione su più isole; questa caratteristica rende Eve adatto al GRID computing** (sebbene questa funzionalità necessiti ancora di essere implementata), consentendo, attraverso un insieme di computer, di potere valutare una quantità relativamente elevata di individui.
Qualora le isole lavorino sugli stessi dati ma valutino gli stessi secondo **fitness diverse, lo scambio dei migliori individui e il loro incrocio con la popolazione locale consente di creare individui ibridi che hanno buone chance di avere un buon comportamento per più tipi di fitness**. Questo ha senso, poiché spesso un individuo che ha un buon comportamento con un tipo di fitness, ottiene allo stesso tempo un comportamento non ottimale con un altro tipo di fitness; **la migration strategy cerca di ottenere individui ibridi che si comportino bene con più tipi di fitness**. Anche in questo caso il costo per passare ad una implementazione GRID di Eve è basso, proprio grazie al modello a isole.
Senza la strategia di migrazione, le isole sarebbero effettivamente isolate le une dalle altre, probabilmente perdendo buona parte del loro senso d'esistere. Infatti, **nella letteratura sulla programmazione genetica, il modello a isole è strettamente legato al trasferimento di individui tra le varie isole.**
Quando sono attivi sia la migration strategy che la disaster strategy, **all'accadere dell'evento disastro tutta la popolazione superstite viene copiata e trasferita verso le altre isole, provocando a tutti gli effetti un evento migrazione**.

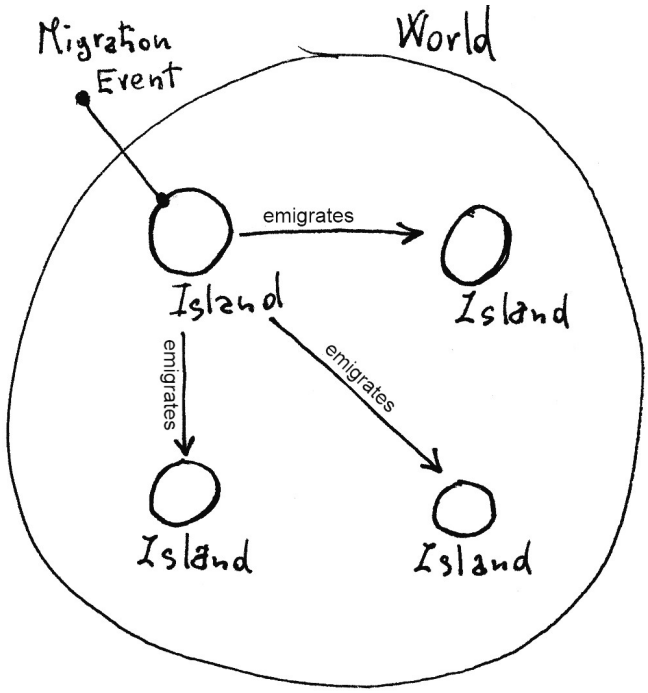

Illustrazione 7: Diffusione degli emigrati presso le altre isole del mondo evolutivo in seguito ad un evento migrazione

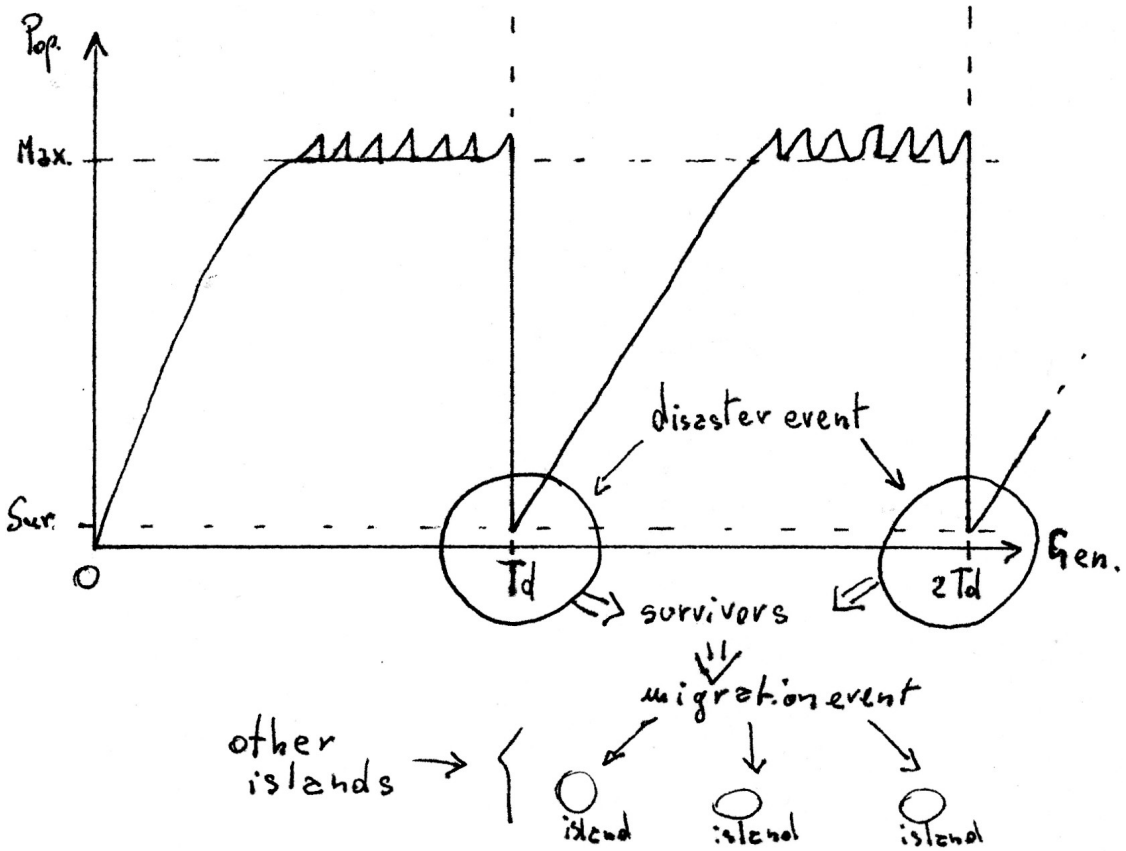

Illustrazione 8: Generazione di un evento migrazione da un evento disastro e diffusione dei sopravvissuti nelle altre isole del mondo evolutivo

Elaborazione dei dati sperimentali

Eve non si occupa direttamente della pre-elaborazione dei dati sperimentali, ma offre un programma di utilità che fornisce diverse funzionalità.

In Eve **i dati sperimentali sono raggruppati dagli oggetti della classe `Experiment`**; ogni `Experiment` è composto da più oggetti della classe `Profile`, ove ogni oggetto `Profile` rappresenta un profilo di espressione genica.

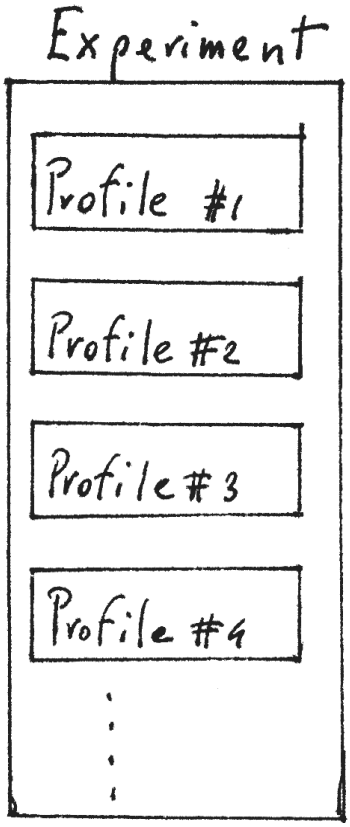

Illustrazione 9: Composizione di un Experiment

La classe `Experiment` fornisce diversi metodi per l'elaborazione dei `Profile` contenuti al suo interno.

La **selezione per varianza minima** consente di specificare la varianza sotto la quale un profilo (`Profile`) viene scartato dai dati sperimentali (cioè dall'`Experiment`); questo consente di **eliminare i profili di espressione genica piatti** che non apporterebbero informazioni utili ad Eve durante la sua esecuzione, ma anzi avrebbero un **effetto deleterio sulle prestazioni** a causa delle aumentate dimensioni dei dati.

Il metodo di clustering fornito dalla classe `Experiment` è il **QTClustering**, ovvero il Quality Threshold Clustering. Questo metodo richiede che venga specificato il raggio massimo di un cluster e, senza ulteriore supervisione, il metodo restituisce il conseguente numero di cluster di Profile che poi vengono mediati in un unico `Profile` dell'`Experiment`.
Il raggio specificato all'algoritmo di clustering è legato alla distanza tra i diversi Profile dell'Experiment. In particolare la **distanza è selezionabile da un elenco di distanze supportate**.
Il QTClustering è importato dalla libreria esterna edu.cornell.med.icb.clustering.

L'algoritmo **SP-Line** della classe `Experiment` consente di aumentare il numero di campioni dei `Profile` dell'`Experiment`.
L'interpolazione SP-Line è importata dalla libreria esterna flanagan.interpolation.

Interfaccia Grafica

Sebbene Eve fornisca una versione funzionante da linea di comando, è altresì disponibile una ricca interfaccia grafica che è in grado di controllare fino a 4 isole.
L'esperimento viene importato da file esterni (`Input Data`) e a questo è possibile applicare elaborazioni come selezione, clustering e interpolazione in qualsiasi ordine, con i parametri desiderati e con la possibilità di visionare i dati (`Look Experiment`).
Il clustering comporta che i nomi dei geni vengano accorpati come nome di un unico profilo; questo consente di tenere traccia di quali geni sia composto un cluster, ma ha l'effetto collaterale di generare nomi lunghi nei nomi dei profili. Il tasto `Clusterize Names` fa salvare la lista dei nomi dell'esperimto attuale e li sostituisce con una versione più semplice.
Il tasto `Lock Experiment` blocca la schermata di pre-elaborazione e abilita le funzionalità evolutive di Eve.
Nella scheda Eve è possibile impostare i parametri delle isole e abilitarne / disabilitarne secondo necessità. Il checkbox `Override setting from Island 1` consente di usare i parametri dell'isola 1 anche per le altre isole abilitate; l'opzione "`but Fitness Type`" indebolisce l'opzione precedente consentendo alle singole isole di specificare la loro specifica fitness.
Avviato il mondo evolutivo, nel tab `World Statistics` possiamo monitorare l'andamento delle isole, mentre nel tab `Convergence Graphics` sono graficate la fitness in scala logaritmica e i residui totali per istante di campionamento. `Best Individuals` mostra in tabelle i coefficienti dei migliori individui.
Qualora il computer dovesse risentire pesantemente dell'esecuzione di Eve, il tab `Emergency STOP` consente un arresto rapido.

Varie

Attualmente Eve è composto da più di 8.500 righe di codice strutturate in 35 classi.
La stesura del codice ha visto l'utilizzo di varie tecnologie (SVN, JUnit, jVisualVM) di due ambienti IDE (Eclipse e NetBeans) e di varie librerie (Flanagan, QTClustering, jFreeChart).
Tutte le classi (tranne quelle riguardanti l'interfaccia grafica) sono testate tramite JUnit.
Il profiling della memoria e del tempo CPU è stato eseguito sia col tool jVisualVM che col tool di profiling incluso in NetBeans.
Jdoc è stato usato per documentare l'uso delle classi Experiment e Profile.

Consigli

L'eseguibile è impostato per allocare un massimo di 1GB di ram.
Questa quantità consente di lavorare con quantità di geni non eccessivamente grandi e con popolazione sull'ordine del migliaio.
Se è necessaria più ram è consigliabile usare la riga di comando e far partire il .jar con parametro per la JVM -Xmx1500m ove 1500m è lo spazio massimo di 1,5GB allocabile attualmente da una JVM standard.
Esistono JVM che non soffrono di questa limitazione, ma il loro ambito è quello server.
Se dovesse essere necessaria una popolazione maggiore, allora si renderebbe necessario adattare Eve per supportare il GRID computing.