1 Simulate Annealing

1.1 Convergenza

La convergenza descrive le condizioni sotto le quali una metaeuristica trova l'ottimo globale (e non solamente locale).

Per la ricotta possiamo fare una analisi matematica precisa di questa condizione.

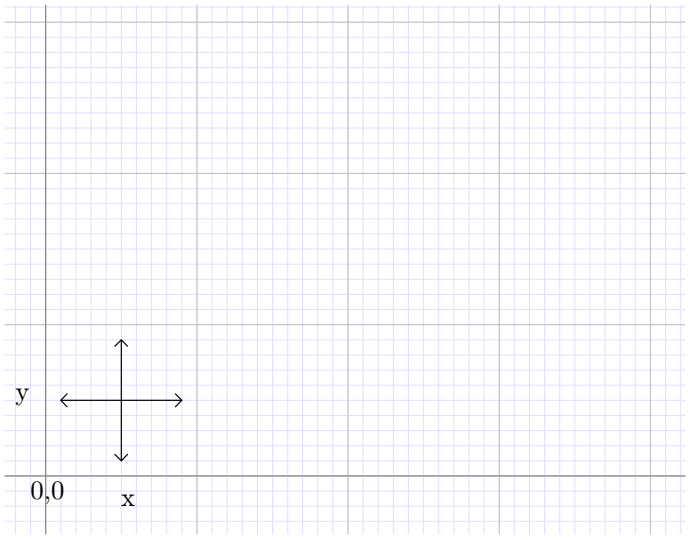
Si può calcolare la probabilità di convergenza. La soluzione trovata sarà albitrariamente vicina all'ottimo. con una probabilità ... vicina di 1.

Questo e' vero se:

- Le trasformazioni o movimenti sono reversibili se $s \in V(r)$ e $r \in V(s)$
- Tutti i punti di S sono raggiungibili da qualsiasi soluzione con un numero finito di trasformazioni
- La temperatura iniziale deve essere elevata
- La temperatura deve scendere lentamente: $T(t) \sim \frac{C}{\log(t)}$ dove t e' l'indice dell'iterazione e C e' una costante che dipende dal landscape dell'energia (l'ampiezza delle variazioni di E tra i vicini)

  In pratica C è sconosciuta, e la diminuzione di temperatura è troppo lenta per essere accettabile. Per le applicazioni pratiche ci si accontenterà di un ottimo accettabile con tempi molto più rapidi.

Illustriamo ora le ipotesi del teorema di convergenza tramite un esempio, che ci permetterà di comprendere le nozioni di probabilità di trovare una soluzione.



$|S| = n_x \cdot n_y$

$S \subset \mathbb{Z}^2$

$\text{mvt} = \{N, S, W, E\}$

Supponiamo di darci una funzione di energia $E(x, y)$

Possiamo assegnare ad ogni punto in S una indice $i = n_y(y-1)+x$

Nell'esempio y a 100 valori possibili per i.

Ci piacerebbe calcolare la probabiltà di $P(t,i)$, ovvero la probabilità che alla t-sima iterazione la ricotta sia al punto $i \in S$ (con i l'ottimo globale)

(per $t \to \infty$ ci piacerebbe avere $P(t,i) = 1$)

Possiamo calcolare $P(t,i)$ come:

$P(t+1,j) = \sum P(t,i) W_{\mathrm{ij}}(t)$

Dato che la ricotta è un processo Markoviano.

Qui, $W_{\mathrm{ij}}(t)$ è la probabilità che all'iterazione t si passa da i a j.

W_{ij} è una matrice di taglia $n_x n_y \times n_x n_y$. (quindi infattibili per spazi grandi). E si calcola:

$$W_{\mathrm{ij}}(t) = \begin{cases} 0 & \text{se } j \text{ non è un vicino} \\ \dfrac{1}{|V|} \cdot P\left(E_i, E_j, T(t)\right) & \text{se } j \text{ è vicino} \end{cases}$$

$W_{\mathrm{ij}} = 1 - \sum W_{\mathrm{ij}}(t)$ (probabilità di rigetto)

Vogliamo ora calcolare $P(t,k)$ in funzione di $P(0,l)$ che è la probabiltà di scegliere l come punto iniziale della ricotta.

$$\begin{aligned} P(t,k) &= \sum_j P(t-1,j) W_{\mathrm{jk}}(t-1) \\ &= \sum_j \left[\sum_i P(t-2,i) W_{\mathrm{ij}}(t-2) \right] W_{\mathrm{jk}}(t-1) \\ &= \sum_i P(t-2,i) \sum_j W_{\mathrm{ij}}(t-2) W_{\mathrm{jk}}(t-1) \qquad \text{Prodotto matriciale} \\ &= \sum_i P(t-2,i) [W(t-2) \times W(t-1)]_{\mathrm{ik}} \end{aligned}$$

Iterando questa relazione si trova che:

$$P(t,k) = \sum_l P(0,l) \cdot W(0,t-1)_{\mathrm{lk}}$$

Dove $W(0,t-1)$ è definita come il prodotto delle matrici $W(t')$ per $t' \in \{0 \ldots t-1\}$.

Un caso particolare se $W(0,t-1)$ tutte le linee sono uguali:

$$W_{\mathrm{lk}}(0,t-1) = W_{1k}(0,t-1) = W_{2k}(0,t-1) \ldots$$

In questo caso allora:

$$P(t,k) = W_{1k}(0,t-1) \sum_l P(0,l) = W_{1k}(0,t-1)$$

Questo significa che il punto di partenza è irrilevante.

1.2 Guida pratica alla Ricotta

Ci resta di precisare:

- Come definire la temperatura iniziale
- Quando decidere di abbassare la temperatura (in pratica, non abbasseremo la temperatura ad ogni iterazione, ma al contrario faremo come illustrato precedentemente)
- Come abbassare la temperatura (di quanto?)
- Quale è la condizione di stop

Non c'è una risposta unica a questi problemi, ma ci sono delle raccomandazioni generali.

Per specificare un problema di ricotta, bisogna:

- Codificare lo spazio di ricerca S
- Definire le trasformazioni (movimenti) possibili
- Scegliere una soluzione s_0 iniziale

La temperatura iniziale T_0 si determina impiricamente attraverso la seguente procedura (partire con temperature troppo alte genererà un tempo di ricerca troppo lungo):

1. A partire dalla soluzione iniziale s_0, si fanno 100 trasformazioni casuali.
2. Si calcolano la variazioni di energia ΔE di questi 100 campioni casuali
3. Si sceglie un tasso di accettazione Γ_0 secondo la qualità ipotetica della soluzione iniziale:

   Γ_0=0.5 se la soluzione iniziale è considerata mediocre

   Γ_0=0.2 se invece è considerata buona
4. Si risolve dunque che la temperatura è:

$$
\begin{aligned}
e^{-\frac{\Delta E}{\Gamma_0}} &= \Gamma_0 \\
\Rightarrow \frac{\Delta E}{\Gamma_0} &= \ln(\Gamma_0) \\
\Rightarrow T_0 &= \frac{\Delta E}{\ln(\Gamma_0)}
\end{aligned}
$$

Intuitivamente questo vuole dire che possiamo poter fare dei salti di dimensione ΔE con una buona probabilità.

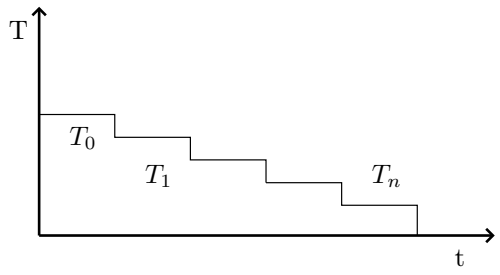
I cambiamenti di temperatura

In particolare si utilizza la stessa temperatura per diverse iterazioni.

Si cambia la temperatura nelle seguenti condizioni:

- abbiamo accettato $12n$ candidati
- abbiamo analizzato $100n$ candidati

Dove n è il numero di gradi di libertà del sistema.



La **progressione** della temperatura è: $T_{k+1} = 0.9T_k$

La condizione di stop

Se dopo almeno 3 gradini di temperatura successivi, la soluzione trovata non si trova un miglioramento della soluzione, ci si ferma.

Validazione del risultato

Si ripete più volte l'esperienza con condizioni iniziali diverse, se ogni volta si trova lo stesso **valore ottimale** (di energia, **nb** ci possono essere più soluzioni ottimali con la stessa energia) allora probabilmente si tratterà di una buona soluzione.

1.3 Temperamento parallelo

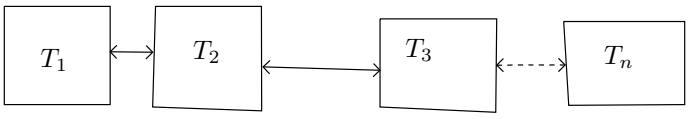
E' una variante della ricotta che permette la parellizzazione, originato da **Geyes** nel 1991 per le simulazioni Monte-Carlo sulle proteine, dove più ricotte simultanee interagiscono.

L'idea principale è quella di fare passare le soluzioni possibili tra le ricotte di temperature vicine (in modo da esplorare meglio lo spazio).

Temperaggio (tempering) significa un ciclo caldo-freddo in un processo (per esempio nel processo di fabbricazione del cioccolato).

Nelle ricotte tradizionali si può solo scendere di temperatura, invece in questo caso no.

Si può rappresentare l'algoritmo del "temperamento parallelo" come:



Abbiamo dunque n ricotte, ciascuna con la sua propria temperatura T_i, che non cambia, a parte per aggiustamenti spiegati in seguito.

Si scelgono delle temperature tali che:

$$T_1 < T_2 < T_3 ... < T_n$$

L'evoluzione nel tempo de la temperatura della ricotta standard è qui rimpiazzata per una variazione attraverso le n repliche.

La replica m-esima avrà una alta temperatura e esplorerà lo spazio, mentre la replica 1 permetterà la ricerca di un risultato ottimizzato.

In ciascuna dell n repliche abbiamo una configurazione che evolve secondo le regole di una ricotta a temperatura fissa. Ma andiamo anche ad aggiungere la possibilità di cambiare le soluzioni tra le ricotte vicine.

Le configurazioni C_i e C_j sono scambiate se $i = j \pm 1$ con una probabilità

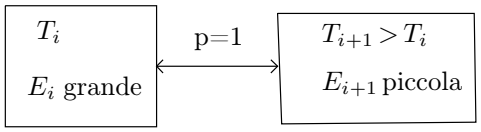
$$p = \min(1, e^{-\Delta_{ij}})$$

Dove Δ_{ij} è calcolato cosi:

$$\Delta_{ij} = (E_i - E_j)\left(\frac{1}{T_j} - \frac{1}{T_i}\right)$$

Dove E_i e E_j sono le fitness della configurazione C_i e C_j, mentre T_i e T_j sono le temperature delle repliche i e j. (**nb: configurazione=soluzione**)

Questa regola di scambia indica che una configurazione a bassa energia e alta temperatura sarà sempre scambiata con una configurazione ad alta energia e bassa temperatura.



Le buone soluzioni dunque vanno verso le basse temperature (intensificazione) e quelle cattive verso le temperature più alte (diversificazione).

Ma c'è sempre una possibilità di fare il contrario.

Questa possibilità diminuisce più la differenza di temperatura e di energia sono grandi.

Parametri di configurazione

Questo metodo richiede di specificare più elementi:

- Il numero m di repliche, spesso $m = \sqrt{n}$, con n uguale al numero di gradi di libertà
- le condizioni di scambio, ex quando due ricotte arrivano all'equilibrio
- la distribuzione delle temperature delle repliche: la replica di temperatura T_{max} deve avere una temperatura uguale a quella iniziale di una ricotta standard T_0, mentre la replica con temperatura T_{min} dovrà avere una temperatura abbastanza piccola per cercare i minimi di energia

  Possiamo decidere di ridistribuire le temperature se il tasso di scambio è troppo grande o piccolo.

  Se questo tasso è inferiore al 0.5% allora diminuiremo le temperature di ΔT:

  $$\Delta T = 0.1(T_{i+1} - T_i)$$

  Altrimenti se il tasso è maggiore del 2% si aumenteranno la temperature di ΔT

2 Ant System

In questo caso l'intelligenza collettiva e la sua percezione è molto migliore di quella del singolo individuo.

I problemi di questi tipo sono chiamati dai fisici problemi complessi, dove il tutto (il problema) è di più della somma delle sue parti.

Il risultato delle interazioni tra individui dona luogo a delle strutture organizzate nel tempo e nello spazio. Questa si chiama auto-organizzazione

In etologia si parla di eterarchia che si oppone alla gerarchia. Non ci sono livelli di conoscenza o di intelligenza diversi tra gli individui e nessuno fra di loro ha una conoscenza globale del problema. Dunque non c'è bisogno di capi.

I sistemi auto-organizzati sono interessanti per la semplicità degli individui presenti. In più questi sistemi sono robusti ai disfunzionamenti o agli errori (addirittura a volta benefici, per esplorare delle soluzioni migliori).

- Adattamento ai cambiamenti di condizione/configurazione

Inoltre questi algoritmi dal punto di vista informatico sono molto interessanti in quanto si paralizzano facilmente.

La storia

Nel 1992 Colani, Dorigo e Marizzo propongono l'algoritmo **Ant-System** (AS) che permette di risolvere problemi di tipo combinatorio (tipo TSP, QAP...)

L'ingrediente principale dell'AS è l'esistenza del ferormone, una sostanza chimica che le formiche lasciano sulla loro strada per guidare le altre.

Dopo questo primo algoritmo, altri ricercatori hanno studiato e replicato il comportamento animale per creare nuovi metaeuristiche, per esempio (si veda capitolo successivo):

- Bee colony
- PSO
- Lucciole

2.1 La pista dei ferormoni: una ottimizzazione naturale

Inizieremo descrivendo il ruolo dei ferormoni delle formiche per mostrare come questo meccanismo può dare luogo a una ottimizzazione.

Per effettuare un lavoro collettivo, le formiche devono comunicare. Per farlo depositano dei ferormoni, una sostanza chimica odorante, che attira le altre formiche.

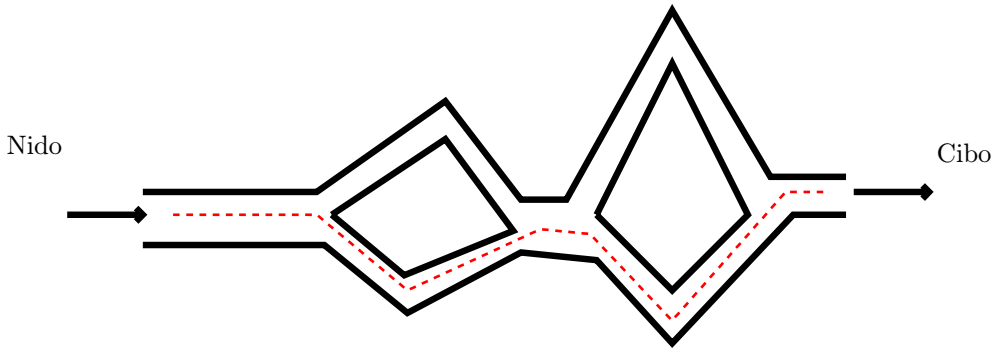
Dunque, le formiche possono marcare un punto nello spazio che ritengono interessante.

Tra i biologi, questo meccanismo si chiama Stignergia o chemotaxia: orientare gli spostamenti in funzione di una sostanza chimica depositata nello spazio.

La durata dei ferormoni è limitata nel tempo. Questi evaporano, e se non ci sono altre formiche a mantenere la pista, questa si cancella.

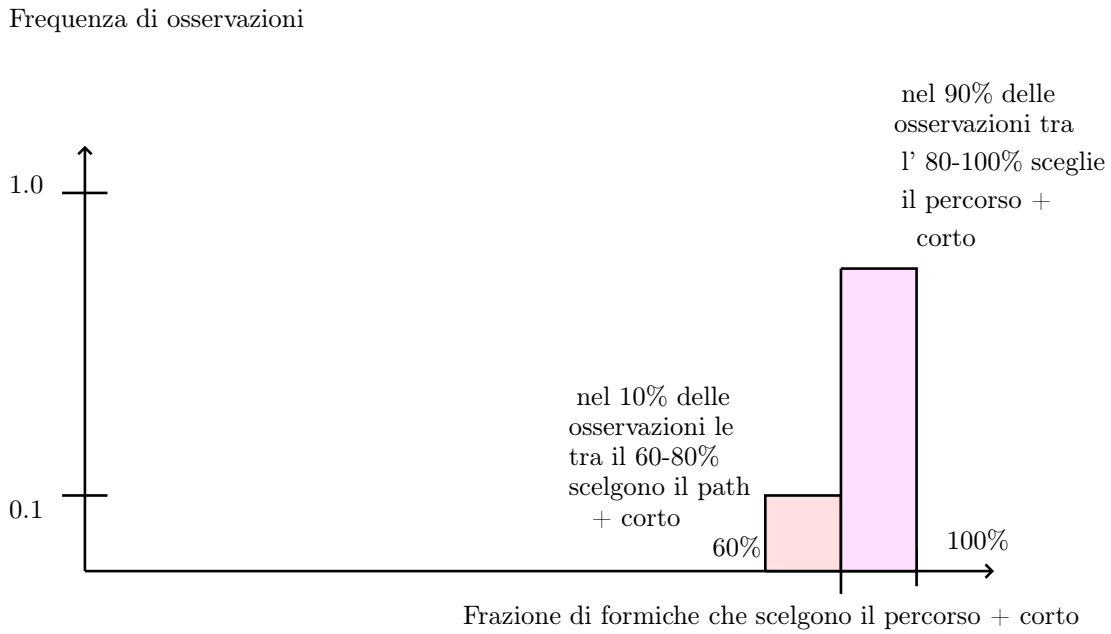
L'evaporazione permette di eliminare delle soluzioni sotto-ottimali o sbagliate.

2.1.1 Esperienza di Goss (1989)



Disponendo di un sistemi di tubi con percorsi di diversa lunghezza che portano dal nido di formiche al cibo, Goss ha osservato che le formiche, che inizialmente percorrevano tutti i percorsi in modo equivalente, con il tempo utilizzarono solo quello corto (con qualche eccezione statistica).

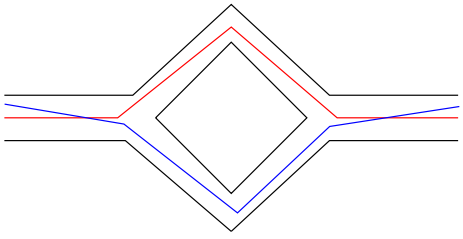
Il risultato e' stato analizzato in modo statistico nel seguente modo:



Questo si spiega per la seguente ipotesi: la formica che casualmente sceglie il miglior percorso sarà la prima a ritornare e quindi rimarcare il percorso.

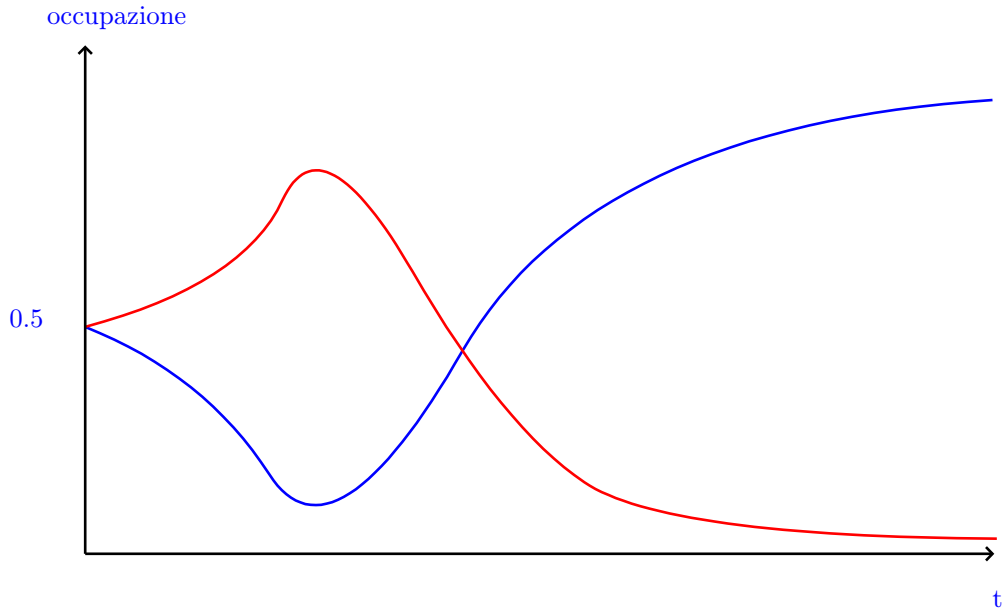
Le formiche successive, rientrando da cammini più lunghi riprenderanno il cammino corto trattandosi del più marcato.

2.1.2 Esperienza di Goss e Dereborg (1990)



Questa volta la configurazione è simmetrica.

Si osserva dunque le seguenti varianti



Si osserva che all'inizio i due cammini sono occupati in modo equivalente (50/50). Dopo una certa fase di oscillazione (+/- lunghi) uno dei due cammini prende il "potere" e l'occupazione diventa (0-100).

2.1.3 Modello matematico

Grass e Deneborg hanno proposto questa definizione:

Definizione 1.

$\boldsymbol{P_L(m+1)}$ *la probabiltà che la (m+1)sima formica scelga il cammino L (lower).*

$\boldsymbol{P_U(m+1)}$ *la probabiltà che la (m+1)sima formica scelga il cammino U (upper).*

Sia $\boldsymbol{U_m}$ *il numero di formiche tra le m formiche già passate che sono passate sul cammino U.*

$\boldsymbol{L_m}$ *ugualmente è il numero tra le m formiche di quelle che sono passate per il cammino L.*

Quindi: $m=U_m+L_m$ *e* $P_U(m+1)+P_L(m+1)=1$

A questo punto la probabilità è calcolata:

$$\boldsymbol{P_u(m+1)=\frac{(U_m+k)^h}{(U_m+k)^h+(L_m+k)^h}}$$

Dove h e k sono dei parametri.

k=20, h=2

k quantifica la differenza minima di formiche necessaria per influire nella scelta.

h quantifica il modo in cui le formiche percepiscono i ferormoni.

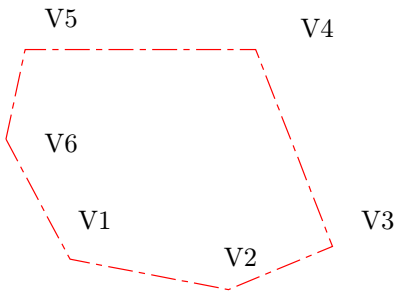
2.2 Algoritmo informatico di ottimizzazione

Dorigo nel 1992 introduce l'ant-system con l'obbiettivo di interpretare le piste di ferormone in un contesto di ottimizzazione combinatoria.

In seguito, delle variate sono state presentata, per esempio l'**Ant Colony System** (ACS). Più performanti in alcuni problemi.

Noi presenteremo la metaeuristica AS per analizzare il problma del commesso viaggiatore.

Ci si dona una lista di n città e una amtrice di distanze d_{ij} che rappresenta la distnza tra la città i e j.



Bisogna trovare il percorso più corto tra le n città, senza passare due volte per la stessa. Inoltre spesso si impone che il punto di partenza sia anche quello di arrivo (come nel caso di esempio), in questo modo il punto di partenza non conta.

Per l'algoritmo AS definiamo la visibilità η_{ij}:

$$\eta_{ij} = \frac{1}{d_{ij}}$$

Si individua cosi τ_{ij} come l'intenstità del percorso tra la città i e j.

Si definisce m il numero di formiche che esploreranno i percorsi possibili. Questi percorsi saranno influenzati dalla quantità di ormoni depostati tra le città (τ) e la prossimità tra queste due (η).

Bisognerà specificare inoltre il tasso di evaporazione dei ferormoni e la quantità di ferormoni depositati dalle formiche.

2.2.1 Algoritmo

A ogni iterazione t, lanciamo m formiche attraverso le n città, ciascuna attreverà in modo indipendente i percorsi del problema TSP secondo i valori di τ e η.

Durante una iterazione le m formiche determinano il loro percorso in modo indipendente l'una dall'altra. A l'iterazione t le m formiche sono influnzate solo dalle piste dei ferormoni lasciate dalle m formiche lasciate nell'iterazione $t-1$.

Nell'iterazione t possiamo definire P_{ij} la probabilita' che una formica vada dalla citta' i a quella j.

$$P_{ij} = \begin{cases} 0 & \text{se } j \notin J \\ \dfrac{[\tau_{ij}(t-1)]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{l \in J} \{[\tau_{il}(t-1)]^{\alpha}[\eta_{il}]^{\beta}\}} & \text{se } j \in J \end{cases}$$

Dove J è l'insieme di città ancora da visitare .

Il fattore $\sum_{l \in J} \{[\tau_{il}(t-1)]^{\alpha}[\eta_{il}]^{\beta}\}$ è un fattore di normalizzazione che assume che $\sum_{l \in J} P_{il} = 1$.

α e β sono parametri specificati.

In particolare i parametri α e β sono i parametri di guida della metaeursitica. Se α e' grande mettiamo più di importanza alla traccia di ferormoni (quindi intensificheremo il risultato) mentre se β è grande si darà più importanza alla distanza tra le città (favorendo quindi la diversificazione).

Una volta che tutte le m formiche hanno seguito il loro percorso si può mettere a giorno il tasso di ferormoni in ciascun percorso in questo modo:

$$\tau_{ij}(t) = (1-\rho)\tau_{ij}(t-1) + \Delta\tau_{ij}(t)$$

Dove ρ è il tasso di evaporazione e $\Delta\tau_{ij} = \sum_{k=1}^{m} \Delta\tau_{ij}^{m}$ e quindi

$$\Delta\tau_{ij}^{k} = \begin{cases} 0 & \text{se } (i,j) \notin T_k \\ \dfrac{Q}{L_k} & \text{se } (i,j) \in T_k \end{cases}$$

Dove Q e' un parametro e L_k è la lunghezza del tour T_k.

Concludendo ci sono 5 parametri da definire per l'algoritmo **AS**:

- m è il numero di formiche; in generale si sceglie $m = n$, dove n è il numero di città e ogni formica partirà da una città diversa
- In modo emprico si scelgono:
  - $\alpha = 1$
  - $\beta = 5$
  - $\rho = 0.5$
  - $Q = 1$

L'algoritmo **AS** ha avuto un grande successo su dei problemi di benchmarking, trovando soluzioni migliori di quelle mai trovate.

Invece non su tutti i problemi ha perfomato correttamente e per questo motivo una nuova variante dell'**AS** è stata proposta: **Ant Colony System (ACS)**.

2.3 Ant Colony System

Con la probabilità q la prossima città visitata è quella che massimizza: $\tau_{ij}(t-1)[\eta_{ij}]^{\beta}$ per tutte le $j \in J$. Qundi con la probabilità q scegliamo il *miglior* cammino. Con una probabilità di $1-q$, si estrare la città successiva a caso come prima:

$$p_{ij} = \frac{\tau_{ij}[\eta_{ij}]^{\beta}}{\sum_{l \in J} \tau_{il}[\eta_{il}]^{\beta}} \quad \text{per } j \in J$$

2.3.1 Calcolo di $\tau_{ij}(t+1)$ nel ACS

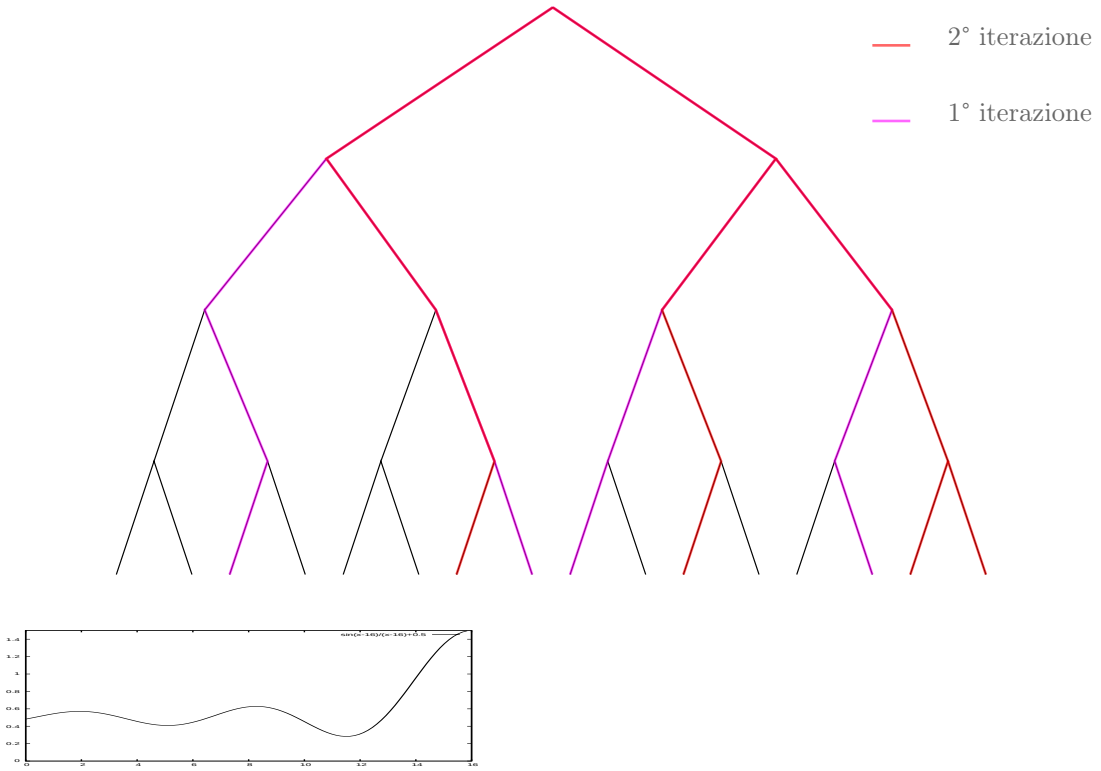
$$\tau_{ij}(t+1) = (1-\phi)\tau_{ij}(t) + \phi\tau_0$$

Dove $\phi\tau_0$ è una quantità costante di ferormoni aggiunta a ciascun percorso (i,j) percorsa da una formica.

ϕ è invece il fattore di evaporazione.

Sul miglior percorso trovato si agginge una quantità di ferormoni $\Delta\tau = \frac{1}{L_{\min}}$

2.3.2 Performance dell'ACS



Con uno sforzo computazionale di soli 4 formiche e 4 iterazione di dunque 36 iterazione, abbiamo un tasso di successo del 70% e con un errore del 3% dal massimo un successo del 90%, quando casualmente la probabilità di trovare il massimo sarebbe di solo 40%.

3 Particle Swarm Optimization

3.1 Introduzione

Il Particle Swarm Optimization (**PSO**) proposto nel 1995 da Eberhart e Kenredy, e ispirato dal metodo delle formiche **AS**.

In questo caso abbiamo un insieme di particelle che esplorano lo spazio di ricerca. Ed è un metodo adatto alla ottimiziazione di una funzione con uno spazio di ricerca $S \subset \mathbb{R}^n$, dove S è limitato e continuo.

Nella metafora del comportamento animale la **PSO** suppone che le particelle seguano il migliore individuo del gruppo (quello che conosce la migliore sorgente di cibo) ma allo stesso gli individui seguono anche il loro istinto e la loro esperienza personale passata.

Combinando le conoscenze personali e di gruppo (quindi quella del migliore individuo) si spera di trovare una soluzione ottimale (o quasi-ottimale) di S.

3.2 Algoritmo

Ogni particella è caratterizzata dalla sua posizione nello spazio di ricerca S, quindi si definisce

$$x_i(t) \in S$$

La posizione della particella i al tempo t. Cascusna delle particelle rappresenta una soluzione possibile del problema.

La **PSO** è una metaeuristica a popolazione, come quella delle formiche, ciò signifca che ad ogni iterazione avremo un numero di soluzioni uguale al numero di particelle, e non una sola: $x_i(t)$, $i = 1, ..., n$.

Il punto chiave di questa metaeuristica è che le particelle si muovono nello spazio S tra 2 iterazioni consecutive. Inoltre si spostano in funzione dalla loro velocità $V_i(t)$.

Per caclolare la velocità, bisogna fare capo ai principi della **PSO**: ovvero seguire il migliore individuo ma allo stesso tempo il proprio istinto e passate esperienze.

Si introduce $x_i^{\text{best}}(t)$ come il punto di migliore fitness osservata dalla particella i dall'inizio della sua tragettoria fino al tempo t.

$$x_i^{\text{best}}(t) = \begin{cases} x_i(t) & \text{se la } f(x_i(t)) \text{ è migliore di } f(x_i^{\text{best}}(t-1)) \\ x_i^{\text{best}}(t-1) & \text{altrimenti} \end{cases}$$

Questo x_i^{best} è anche chiamato *local-best*.

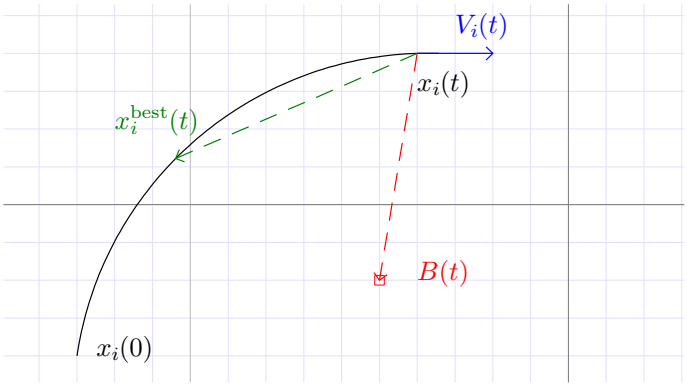
Abbiamo bisogno di introdurre anche il concetto di *global-best*, $B(t)$, ovvero la migliore soluzione trovata dal gruppo.

$B(t)$ è dunque il migiore dei *local-best*, per tutte le $i = 1, ..., n$, per un problema di massimizzazione:

$$B(t) = \arg\max_{x_i^{\text{best}}}(f(x_i^{\text{best}}(t))) \qquad i = 1, ..., n$$

In pratica questo implica che $B(t)$ può essere messo a giorno allo stesso tempo che si calcoa $x_i^{\text{best}}(t)$.

Si la tragettoria della particella i



La particella sarà quindi attratta da:

- Continuare per la direzione attuale secondo la velocità attuale $V_i(t)$
- Andare verso la migliore soluzione globale $B(t)$
- Andare verso la propria soluzione migliore $x_i^{\text{best}}(t)$

Dunque andremo a calcolare $V_i(t+1)$ ponderando questi fattori in questo modo

$$\begin{cases} V_i(t+1) = wV_i(t) + C_1r_1(t+1)[x_i^{\text{best}}(t) - x_i(t)] + C_2r_2(t+1)[B(t) - x_i(t)] \\ x_i(t+1) = x_i(t) + V_i(t+1) \end{cases}$$

Dove $w \in [0,1]$ è un coefficente di attenuzaione (una sorta di attrito), C_1 e C_2 sono dei coeffienti mentre r_1 e r_2 sono dei numeri estratti da una distribuzione uniforme in $[0,1]$.

Di fatto, x_i e V_i sono in generale dei vettori nello spazio $\mathbb{R}^d$ e si scelgono dei fattori r_1 e r_2 diversi per ogni componente del vettore V_i.

Inizialmente le particelle sono distribuite a caso dentro a S e la velocità iniziale è nulla.

Se una particella raggiunge un bordo di S rimbalza in dientro.

Inoltre si definisce una velocità limite $V_{\max}$ per avere delle velocità ragionevoli.

Il coefficente C_1 si chiama il coefficente cognitivo.

Il coefficente C_2 si chiama coefficente sociale.

Si suggerisce di prendere come valori $C_1 = C_2 = 2$, mentre per w si prende un valore leggermente inferiore a 1, per esempio $w = 0.9$.

3.3 Osservazioni

Si può dare ad ogni particella un orizzonte di visione finità in modo tale che il *global-best* sia determinato tra gli individui visibili (e non dall'insieme completo) dunque $B(t) \to B_i(t)$.

Questa conoscenza parziale può essere definita invece che per distanza per legami sociali tra gli individui. Quindi ogni indiviuo conosce un sotto insieme finito di particelle.

La **PSO** funziona bene in uno spazio $S \subset R^d$ ma non funziona molto bene in puno spazio discreto di permutazioni.

- Quale è la velocità in questo tipo di spazi?
- L'operatore $+$ non è definito in questo tipo di spazi: $x(t+1) = x_i(t) + V_i(t+1)$ come si calcola?

Esistono dunque delle varianti della **PSO** per questo tipo di spazi.

Per dei problemi di ottimizzazione continua la **PSO** converge rapidamente e la qualità del risultato è comparabile con gli algoritmi forniti o agli algoritmi genetici (che si presenteranno in seguito).

3.4 Firefly algorithm

Questo è un algoritmo ispirato dalla **PSO** e espresso nel caso delle lucciole che emettono della lucia per attirare le lore prede o dei loro simil.

Xin-Shen Yang ha introdotto questo algoritmo nel 2010 e conosce una popolarità crescente.

Anche questo algoritmo è adattatato ai probelmi continui ma esistono varianti per i problemi combinatori.

3.4.1 Algoritmo

Ciascuna lucciola i occupa una posizione nello spazio $x_i(t) \in S$ al tempo t. Si spera che le lucciole si muvano in direzione del ottimo globale di S.

Anche in questo caso $S \subset \mathbb{R}^d$.

Ciascuna lucciola i emette una luce di intensità $I_i(t)$ proporzionale alla fitness della posizione della lucciola. Per un problema di massimizzazione si potrebbe scegliere:

$$I_t(t) = f(x_i(t))$$

Dove $f()$ è la fitness.

Ad ogni iterazione si considerano tutte le coppie di lucciole (i, j), con $i \neq j$. Tra le due lucciole i e j quella con la intensità più debole si sposterà in direzione di quella con l'intesità più forte.

La lungezza dello spostamento della lucciola i verso la lucciola j, dove $I_i(t) < I_j(t)$, dipenderà dalla attrattività β di j percepità da i.

Questa attrattività è definità come:

$$\beta = \beta_0 \cdot e^{-\left(\frac{r_{ij}}{\gamma}\right)^2}$$

Dove r_{ij} è la distanza tra la lucciola i e j mentre β_0 è la attrittività a distanza 0 ed è normalmente impostata a $\beta_0 = 1$. Invece γ è una costante che dipende dalla sceltà delle unità per misurare la distanza r_{ij}.

Allora si può calcolare lo spostamente della lucciola i verso la lucciola j $(I_i(t) < I_j(t))$:

$$x_i(t+1) = x_i(t) + \beta_0 \cdot e^{-\left(\frac{r_{ij}}{\gamma}\right)^2}(x_j(t) - x_i(t)) + \alpha(\text{rand}_d() - 0.5)$$

Dove $\text{rand}_d()$ è un vettore di d componenti aleatori di valore compreso tra [0,1[. Il coefficente α è scelto tra 0 e 1.

3.5 Pseudo Code

```
iteration = 0
//initialization of n firefly
xi = rand, i=1 to n, xi in S
//compute the initial intensity for each firefly
Ii=f(xi)
while(not end-condition)
{
  for i=1 to n
  {
    for j=1 to n
    {
       if Ii<Ij
       {
         //The firefly move in direction of xj
         //Compute new Ii
       }
    }
  }
}
```

4 Algoritimi evolutivi

4.1 Introduzione

E' un insieme di metaeuristiche ispirate alla evoluzione darwiniana:

"Survival of the fittest." Darwin

Sono delle metaeuristiche a popolazione che utilizzano dei processi di mutazione, incroci e di selezione per esplorare lo spazio di ricerca S.

Sono dei metodi apparsi negli anni 70, in particolare:

- Algoritmi genetici (Holland, John)
- Strategie d'evuluzione (Rechrnberg)
- Programmazione genetica (Fogel e Koza)

L'idea centrale di questi metodi e' di fare emergere dei buoni individui copiando quello che la natura fa per creare delle specie meglio adattate al loro ambiente.

4.2 Algoritmi genetici

Si avra' una popolazione di individui x_i, ciascuno rapressenta una soluzione possibile del problema, dunque una punto di $x_i \in S$.

Il codaggio degli individui dentro a S e' interpretato come del codice genetico (DNA) dell'individuo. Per questa analogia si parla spesso di cromosomi per descrivere il codaggio.

Questa specifica si chiama genotipo. Si definisce anche il phenotipo come il valore di fitness di questo individuo.

La fitness indica inoltre il grado di adattazione di ciascun individuo, dunque gli individui con buon fitness sopravviveranno e evolveranno per trovare fitness ancora migliori.

In seguito consideremo dei porblemi di massimizzazione della fitness. (e' sempre possibile considerare un problema di minimizzazione di f come un problema di massimizzazione di $-f$)

In biologia la fitness si misura come il numero di discendenti di ogni individuo.

Nel nostro contesto si parlera' dunque di generazioni invece di iterazioni.

Si avra' una popolazione al tempo t, $P(t)$, che evolvera' verso una nuova popolazione al tempo $t+1$, $P(t+1)$, la generazione seguente.

NB: In generale la dimensione della popolazione e' costante durante l'evoluzione.

4.3 Pseudo codice

```
generation = 0
//inizializzazione della popolazione
P(0)
while (not end-condtion)
{
  generation ++
  //compute individual fitness
  f(x[1..n]);
  //select individual
  x = select(x[1..n]);
  //crossover
  x = crossover(x);
  //mutation
  x = mutation(x)
```

```
}
return best_individual
```

In generale il migliore individuo della generazione t e' inserito anche nella generazione seguente $t+1$, spesso sostituendo l'individuo peggiore.

Infatti le operazioni di select e crossover non garantiscono che la generazione seguente sia $\geqslant$ di quella presente.

Una altro modo di rapresentare l'algoritmo genetico e' il seguente:

$$P(t) \xrightarrow[\text{select}]{} P^1(t) \xrightarrow[\text{crossover}]{} P^2(t) \xrightarrow[\text{mutation}]{} P^3(t) = P(t+1)$$

Dove $P(t), P'(t)...$ hanno tutti lo stesso numero di elementi.

In generale la dimensione della popolazione n e' scelta tra qualche decina a qualche centinaia.

La dimensione constatne della popolazione durante la evoluzione implica delle particolarita' per il select e crossover.

In generale la popolazione $P(0)$ iniziale e' aleatoria.

La condizione di arresto puo' essere un numero massimo di generazioni, o un tempo di stagnazione della fitness.

4.3.1 Selezione

L'obbiettivo della selezione e' di scegliere in $P(t)$ degli individui che seranno presenti nella fase di crossover.

Si costruice $P'(t)$ con un estrazione con rimessa (ovvero gli elementi vengo conservati nell'array originale) di n individui da $P(t)$.

Tra gli n elementi di $P'(t)$ sara' dunque possibile che ci siano piu' individui uguali (copie).

Altrettanto possibile che degli individui di $P(t)$ non siano presenti in $P'(t)$, quindi si estinguino.

La distribuzione della estrazione non e' uniforme ma dipendera' alla fitness degli individui.

Possibili modi di selezione sono:

- La probabilita' proporzionale alla fitness, la probabilita' dell'individuo i sara'

  $$p_i = \frac{f_i}{\sum_{j=1}^{n} f_j}$$

  dove f e' la fitness dell'individuo.

- Selezione per tornei

- Selezione per rang

Quest'ultimi saranno discussi in seguito.

4.3.2 Crossover

In questa fase si prendono gli n individui (genitor) di $P'(t)$ per creare gli n individui (discendenti) di $P''(t)$.

Per fare cio' si prendereanno gli individui a coppie e ogni 2 genitori creara' 2 discendenti.

Per fare cio' le coppie saranno create cosi:

$$(s_1, s_2), (s_3, s_4)...(s_i, s_{i+1})...(s_{n-1}, s_n)$$

Dove s_j e' un individuo di $P'(t)$.

Cosi si avranno $\frac{n}{2}$ coppie di genitori uniche.

Si difinisce una probabilita' di crossover p_{cr} per ogni coppia:

$$\begin{array}{ccc} (s_1, s_2) & ... & (s_{n-1}, s_n) \\ \downarrow p_{\text{cr}} & ... & \downarrow p_{\text{cr}} \\ (s'_1, s'_2) & ... & (s'_{n-1}, s'_n) \end{array}$$

Con probabilita' 1-p_{cr} per la coppia (s_i, s_{i+1}) la coppia non produrra discendenti ma verra' trasferita uguale in $P''(t)$.

I discendenti sono prodotti a partire dei genotipi dei genitori mescolandoli tra loro:

$$\begin{array}{rl} s_i & [A_i, B_i] \\ s_{i+1} & [A_{i+1}, B_{i+1}] \\ & \text{Allora } i \text{ discendenti saranno} \\ s'_i & [A_i, B_{i+1}] \\ s'_{i+1} & [A_{i+1}, B_i] \end{array}$$

Secondo il tipo di problema il crossover puo' essere creato in modi diversi (per esempio per il TSP).

4.3.3 Mutazione

Su ciascun degli n individui di $P''(t)$ si applicano delle trasformazioni casuali al codice genetico con probabilita' p_{mt}.

Per esempio si potrebbero invertire ciascun bit con probabilita' p_{mt}.

4.3.4 Diversificazione VS Intensificazione

La Selezione favorisce l'intenficazione mentre la mutazione favorisce alla diversificazione. Infine il crossover contribuisce ad entrambi i fattori.

Gl **AG** hanno piu' parametri di guida:

- n la dimensione della popolazione
- p_{cr} la probabilita' di crossover
- p_{mt} la probabilita' di mutazione

4.4 Esempio

Vogliamo un **AG** per risolvere il problema **MaxOne**, ovvero trovare la seguenza di bit $x_1, ..., x_l$ che massimizza;

$$f(x) = \sum_{i=1}^{l} x_i$$

Ovviamente l'ottimo e' $x = 1111...1$.

Studieremo in dettaglio un **AG** con $n = 6$ individui e per un problema **MaxOne** di dimensione $l = 10$ bits.

Si supponga la seguente popolazione iniziale:

$P(0)$	fitness	Probabilita'
$s_1 = 1111010101$	$f(s_1) = 7$	$p_1 = 0.206$
$s_2 = 0111000101$	$f(s_2) = 5$	$\vdots$
$s_3 = 1110110101$	$f(s_3) = 7$	$\vdots$
$s_4 = 0100010011$	$f(s_4) = 4$	$\vdots$
$s_5 = 1110111101$	$f(s_5) = 8$	$\vdots$
$s_6 = 0100110000$	$f(s_6) = 3$	$p_6 = 0.09$
	$f(P(0)) = 34$	

Per la **selezione** usiamo la probabilita' proporzionale al fitness:

$$p_i = \frac{f(s_i)}{f(P(0))} = \frac{f(s_i)}{34}$$

Si estrae dunque $P^1(0)$

$$P^1(0) = [s'_1 = s_1, s'_2 = s_3, s'_3 = s_5, s'_4 = s_2, s'_5 = s_4, s'_6 = s_5]$$

Si noti che s_5 e' stato selezionato 2 volte mentre s_6 0.

Per la fase di **crossing** per ogni coppia (s_i, s_{i+1}) si scegliera' di di fare un crossing con probabilita' p_{cr} e dunque generare una coppia di figli, altrimenti questi verranno trasferiti in $P''(0)$.

Prenderemo in questo caso una $p_{\text{cr}} = 0.6$ e supponiamo che solo le coppie $(s'_1,\ s'_2)$ e $(s'_5,\ s'_6)$ generaranno degli ibridi.

Per la prima coppia avremo un punto di incrocio $k = 2$:

$$s'_1 = 11 \cdot 11010101$$
$$s'_2 = 11 \cdot 10110101$$

Dunque dopo il crossing otteneremo

$$s''_1 = 11 \cdot 10110101 = s'_2$$
$$s''_2 = 11 \cdot 11010101 = s'_1$$

Non ci sara' duqneu nessun ammiglioramente in quanto la sequenza di bit prima del punto di incrocio e' uguale per tute e due gli individui.

Per la seconda coppia di il punto di incrocio $k = 5$

$$s'_5 = 01000 \cdot 10011$$
$$s'_6 = 11101 \cdot 11101$$

Dopo il crossing

$$s''_5 = 01000 \cdot 11101$$
$$s''_6 = 11101 \cdot 10011$$

Questa volta i due individui generati sono davvero differenti.

Ricapitolando la popolazione $P^2(0)$ sara dunque:

$$\begin{array}{rcl} P^2(0) & & P^3(0) = P(1) \\ s_1'' = 1111010101 & \rightarrow & s_1''' = 1111110101 \rightarrow f_1 = 8 \\ s_2'' = 1110110101 & \rightarrow & s_2''' = 1110100101 \rightarrow f_2 = 6 \\ s_3'' = 1110111101 & \rightarrow & s_3''' = 1110101111 \rightarrow f_3 = 8 \\ s_4'' = 0111000101 & \rightarrow & s_4''' = 0111000101 \rightarrow f_4 = 5 \\ s_5'' = 0100011101 & \rightarrow & s_5''' = 0100011101 \rightarrow f_5 = 5 \\ s_6'' = 1110110011 & \rightarrow & s_6''' = 1110110011 \rightarrow f_6 = 6 \end{array}$$

Per la **mutazione** ogni bit della catena avra' una probabilita' di essere invertito di $\frac{1}{10}$.

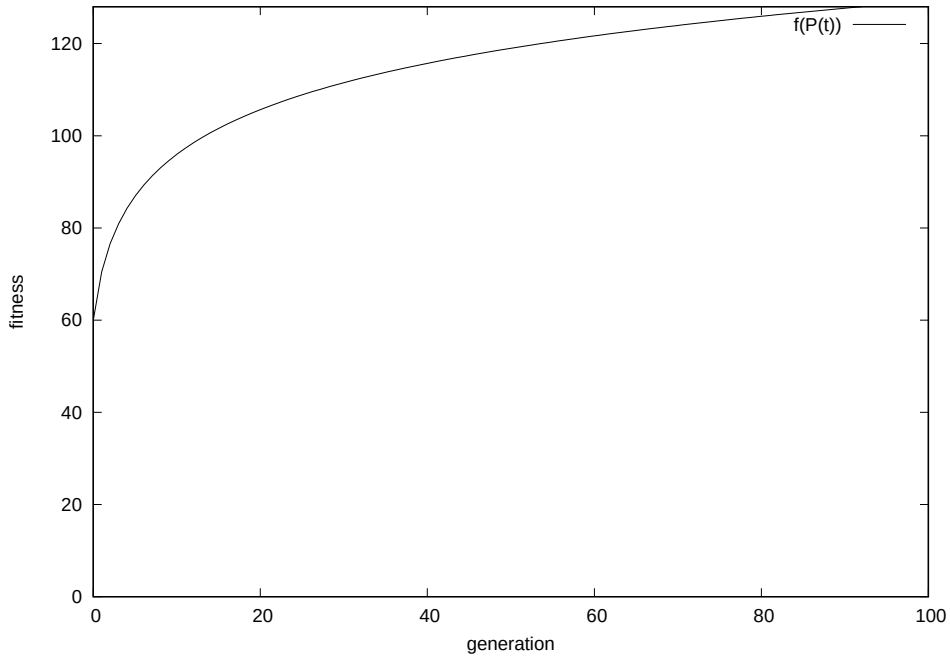
Questo significa che in media saranno invertiti 6 bit sui 60 totatli della popolazione.

Dunque la fitness totale di $P(1) = 36$ mentre quella di $P(0) = 34$, ovvero $\backsim 10\%$ di ammigliorazione.

4.4.1 Esempio piu' realistico

Un esempio piu' realistico puo' essere uno con la seguente configurazioni:

$n = 100$, $l = 128$, **MaxOne**, $p_{\text{cr}} = 0.8$ e $p_{\text{mt}} = 0.05$



In circa 60 generazioni l'algoritmo converge e trova la soluzione migliore.

4.5 Ottimizzazione di funzioni continue

Il funzionamento del **AG** e' buono anche nei casi di funzioni continue reali con molti massimi e minimi locali.

Esempio: le funzioni di Rastrigin e Schwefeld che sono dei benchmark tradizionali per gli AG:

$$f_R(x) = A\,n + \sum_{i=1}^{n} \left[x_i^2 - A\cos(2\pi x_i)\right] \quad \text{in } \mathbb{R}^n$$

$$f_S(x) = B\,n + \sum_{i=1}^{n} \left[x_i \sin\left(\sqrt{|x_i|}\right)\right] \quad \text{in } \mathbb{R}^n$$

$f_S(x)$ per $\mathbb{R}$ semplice e' $f_S(x) = \left| x \sin\left(\sqrt{|x|} \right) \right| + C$

La difficolta' delle funzioni reali e' quello di definire lo spazio di ricerca, le possiblita' sono due:

- Aritmetica a virgola fissa, ogni punto di $S \subset \mathbb{R}^n$ e' codificato come una sequenza di bit di lunghezza fissa. Si divide dunque l'intervallo di ricerca per il numero di valori rappresentabili dalla sequenza di bit e dunque questa definisce la precisione.

- Aritmetica a virgola mobile, i numeri in S sono rapresentati come

  $$x = s \times r^t$$

  dove r e' la base (in generale 2), s e' il valore mentre t e' l'esponente.

  $$s = b_0 + \sum_{k=1}^{p-1} b_k \, 2^{-k} \quad \text{dove } b_i \in [0, 1]$$

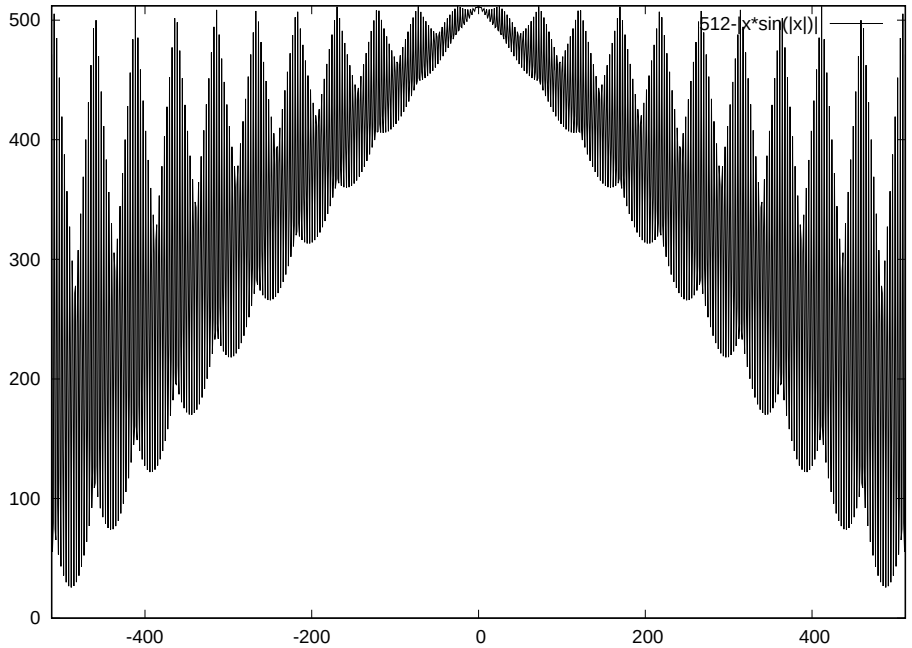
  $$t = \sum_{k=p}^{n-1} b_k \, 2^{k+1-p}$$

  Dunque ci sono p bits per rapresentare il valore (*mantisse*) e $n - p$ bit per rappresentare l'esponente. n e' il numero di bit totali.

  Lo standard IEEE 754 propone come valori $n = 64$, $p = 52$ e $q = 11$ bits e un bit di segno

  Questo ci permette di avere 16 cifre significative e un esponente compreso tra -1024 e 1024.

Esempio: sia la funzione di fitness $f(x) = -|x \sin(|x|)| + C$ con $x \in \mathbb{R}$ e vogliamo trovare il minimo della funzione f nel intervallo [-512,512].




```
set samples 1000; plot [-512:512][0:512] abs(x*sin(abs(x)))
```

Data la simmetria possiamo analizzare solo il lato positvo [0,512].

Se scegliamo di rapresentare S con l'aritmetica a virgola fissa, e scegliamo come lunghezza della catena $n=10$ possiamo rapresentare 1024 soluzioni.

Lo spazio di ricerca sara' dunque:

$$x=\frac{512}{1023}S \qquad S\in\{0,1,...,1023\}$$

La precisione dunque e' solo di ~ 0.5 ($\frac{513}{1023}$).

Per aumentare la precisione bisogna dunque il numero di bit, inoltre con S piu' grandi il problema sarebbe ancora piu' grande.

Per il nostro esempio su $f(x)$ l'AG avra' $N=50$ individui i quali risultati sono per una esecuzione specifica:

Generazione	Migliore Fitness	Fitness Media
0	104	268
3	52	78
9	0.00178	32
⋮	⋮	⋮
69	0.00178 (global)	0.15

Abbiamo trovato il minimo globale dopo solo 9 generazioni, lo sforzo di calcolo e' dunque solo 50×9.

Vediamo anche in questo esempio che la fitness media tende a quella ottimale, questo riflette il fatto che tutti gli individui tendono a quello ottimale.

Per problemi piu' complessi bisogno utilizzare rparessentazioni a virgola flottante. In questo cas le mutazioni e il crossover verranno applicate si sulla base che sull'esponente. Le probabilita' della base e del esponente non saranno necessariamente uguale.

Con questo le AG possono muoversi in paesaggi molto complicati.

4.6 Altri metodi di selezione

$$P(t)\underset{\text{selection}}{\longrightarrow} P^1(t)$$

La selezione e' una operazione stocastica che produce n individui in P^1 a partire dagli n individui di P.

- Selezione proporzionale alla fitness:

$$p_i=\frac{f_i}{\sum_j f_j}$$

Se f e' negativa si puo sempre aggiungere una costante C in modo che $f_i+C>0 \quad \forall i$.

Mano a mano che l'evoluzione continua gli individui si somigliano sempre di piu' e quindi diviene molto difficili da distinguere. Dunque per migliorare il processo di selezione si puo' redefinire il la fitness come:

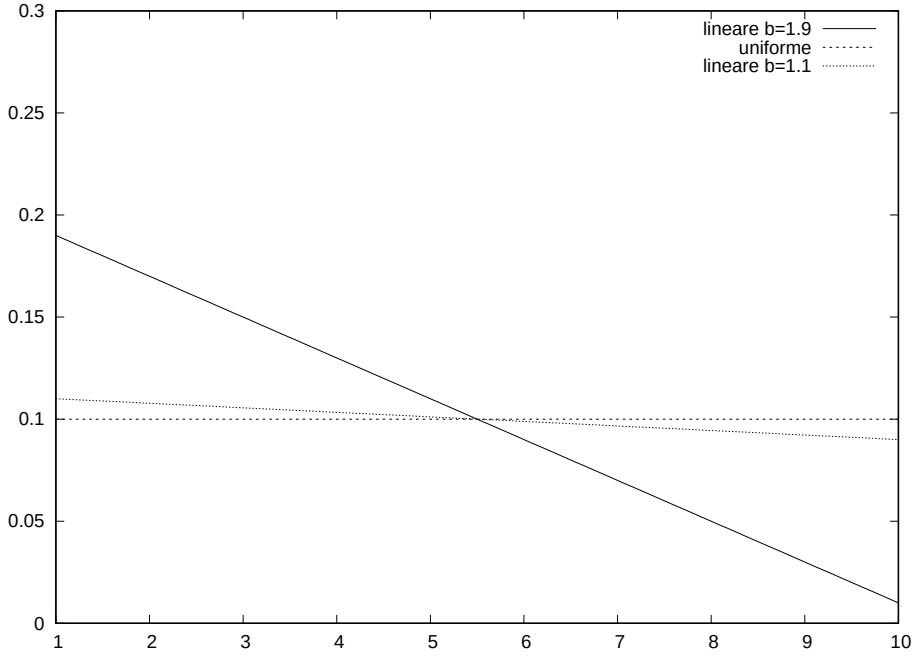
$$f_t=f_i-\min_{j\in P}(f_j)$$

Si ridefinisce l'intervallo dei valori di fitness in uno piu' significativo.

- Selezione per rango:

  Si ordinano gli individui in base alla loro fitness dalla migliore alla meno buona, questo gli dona quindi un rango compreso tra $1...n$, indipendente dal valore numerico della fitness.

  La probabilita' di selezionare l'individuo i sara' una funzione dipendente dal suo rango.



  ovviamente:

$$\sum_{i=1}^{n} p_i = 1.0$$

  Per avere una distribuzione lineare dipendente dal rango si avra':

$$p_i = \frac{1}{n}\left[\beta - 2(\beta-1)\frac{i-1}{n-1}\right]$$

  con $1 \leqslant \beta \leqslant 2$ dove 2 e' il valore che amplifica al meglio la selezione di selezionare di buoni fitness

- Selezione per torneo:

  Per construire $P^1(t)$ si realizzano n tornei costituiti da k individui presi da $P(t)$ presi aleatoriamente e uniformemente con ripescaggio.

  Questi k individui saranno messi in competizione e il migliore messo in $P^1(t)$. Dopo n tornei si avra' P^1 completo.

  Un modo semplice di svolgere il torneo e quello di fare vincere l'individuo con fitness migliore.

  - se $k = 1$ si avra' dunque una selezione uniforme aleatoria

- se $k = n$ si copiera' n volte il migliore individuo in P^1

- in generale $k \in \{2, 3..., 10\}$ e sempre $k \ll n$.

Nota: con i tornei si possono comparare degli individui senza dover specificare una fitness. Per esempio con dei robot da evolvere si potrebbero comparare donando dei compiti da svolgere e selezionarli sulle performance di questi compiti senza dover modellizare queste performance.

I metodi di selezione favoriscono il migliore indviduo. La questione che ci si puo' porre e' di sapere se questo va invadere tutta la popolazione durante il corso dell'evoluzione.

Ovviamente le mutazioni e il crossover cercano di opporsi a questo fenomeno. Ma un metodo di selezione troppo forte diminuira' comunque la diversita' della popolazione.

Il *"takeover time"* e' una misura della intensita' della selezione. Questo si misura sopprimendo il crossover e la mutazione e contanto il numero di iterazioni necessari per rempire la popolazione di copie dell'individuo migliore.

Misuriamo per esempio il tempo di *takeover* τ su degli esempi numerici, il numero di copie cresce in modo esponenziale con i metodi di selezione precedente.

Intutivamente se m e' il numero di copie del migliore individuo, si puo sperare che $m(t)$ abbia la seguente formua:

$$\frac{\partial m}{\partial t} = \alpha \frac{m}{n}\left(1 - \frac{m}{n}\right)$$

dove la prima parte $\left(\alpha \frac{m}{n}\right)$ e' la probabilita' di scegliere il migliore individuo e la seconda il numero di posti disponibile $\left(1 - \frac{m}{n}\right)$.

Questo si chiama una equazione logistica, in un tempo $\tau = \log(n)$ si a che $m(t) \cong \frac{n}{2}$.

Possiamo verificare cio per la selezione proporzionale:

Si hanno n individui di fitness: $\{f_1, f_2 ... f_n\}$ con $f_1 > f_2 > ... > f_n > 0$.

Quindi $m(t)$ e' il numero di copie che dell'individuo con fitntess f_1:

$$m(t+1) = n \frac{m(t) f_1}{F_{\text{tot}}(t)}, \quad F_{\text{tot}}(t) = \sum_{i=0}^{n} f_i(t)$$

dove n e' il numero di estrazioni, $\frac{f_1}{F_{\text{tot}}(t)}$ e' la probabilita' che il migliore individuo venga estratto e $m(t)$ eil numero di copie presenti.

$$F_{\text{tot}}(t) \leqslant m(t) f_1 + (n - m(t)) f_2$$

dunque:

$$\begin{aligned} m(t+1) &\geqslant n \frac{m(t) f_1}{m(t) f_1 + (n - m(t)) f_2} \\ &\geqslant m(t) + n \frac{m(t) f_1}{m(t) f_1 - (n - m(t)) f_2} - m \frac{m(t)\ f_1 + (n - m(t)) f_2}{m(t) f_1 + (n - m(t)) f_2} \\ &\geqslant m(t) + \frac{n\, m(t)(f1 - f2) - m^2(f_1 - f_2)}{m(t)\ f_1 - (n - m(t)) f_2} \end{aligned}$$

Si definisce $\Delta f = f_1 - f_2 > 0 \rightarrow \frac{\Delta f}{f_1} < 1$

$$\begin{aligned} m(t+1) &\geqslant m(t) + \frac{n\, m(t)\Delta f - m^2 \Delta f}{m(t)(f_1+f_2) - n\, f_2} \\ &\geqslant m(t) + \frac{n\, m \Delta f \big(1 - \frac{m}{n}\big)}{n\, f_1 + \Delta f (n-m)} \\ &\quad \text{si divide sopra } e \text{ sotto per } n \,/\, f_1 \\ &\geqslant m(t) + m \frac{\Delta f}{f_1} \Big(1 - \frac{m}{n}\Big) \left(\frac{1}{1 - \big(1 - \frac{n}{m}\big)\frac{\Delta f}{f_1}} \right) \end{aligned}$$

dato che $m \leqslant n$ e $\frac{\Delta f}{f_1} < 1$ allora:

$$\frac{1}{1 - \big(1 - \frac{n}{m}\big)\frac{\Delta f}{f_1}} > 1$$

dunque:

$$m(t+1) - m(t) \;\geqslant\; m(t) \frac{\Delta f}{f_1} \left(1 - \frac{m(t)}{n} \right)$$

Dunque la crescenza di m e' piu' rapida di quella predetta dalla equazione logistica. Dunque τ e' logaritmica in n.

Possiamo fare le stesse osservazioni per la selezione per torneo:

Con un torneo di k individui la probabilita' di non selezionare il migliore individuo tra i membri di un torneo e':

$$\left(1 - \frac{m(t)}{n} \right)^k$$

dove $\frac{m(t)}{n}$ e' la probabilita' di selezionare il migliore.

Dunque la probabilita' che il migliore sia scelto all'interno di un torneo e'

$$1 - \left(1 - \frac{m(t)}{n} \right)^k$$

dunque dato che ci sono n tornei:

$$\begin{aligned} m(t+1) &= n \left[1 - \left(1 - \frac{m(t)}{n} \right)^k \right] \geqslant n \left[1 - \left(1 - \frac{n}{m(t)} \right)^2 \right] \\ &\geqslant n \left[1 - \left(1 - \frac{2n}{m(t)} + \frac{n^2}{m(t)^2} \right) \right] \\ &\geqslant n \left(\frac{2m(t)}{n} - \frac{m(t)^2}{n^2} \right) \\ &\geqslant m(t) - m(t) \left(1 - \frac{m(t)}{n} \right) \end{aligned}$$

si nota ancora che la selezione usata e' molto intensa e si rischia di perdere rapidamente la diversita' della popolazione. In seguito vedremo dei metodi di selezione piu' lenti.

4.7 Altre operazioni di crossover e mutazione

1. Si puo' avere un crossover a due o piu' punti, invece di dividere gli individui in due parti (prendendo un punto di crossover casuale) si possono dividere in tre o piu' parti (prendendo sempre dei punti aleatori):

$|A|B|C|,|A'|B'|C'| \Rightarrow |A|B'|C|, |A'|B|C'|$

2. Si puo' invece un crossover uniforme, prendedndo due individui rapresentati nel seguente modo

$[b_0, b_1, ..., b_n]$

$[b'_0, b'_1, ..., b'_n]$

Si possono dunque costruire un individuo figlio scambiando ogni bit b_i, b'_i con probabilita' $\frac{1}{2}$.

Molti problemi, come quelli di tipo combinatorio, un crossover classico non funzion, per esempio per il **TSP**:

$I_0 = 1 \to 3 \to 2| \to 4 \to 5$

$I_1 = 5 \to 2 \to 1| \to 3 \to 4$

Usando il crossover classico il risultato sarebbe il seguente

$I'_0 = 1 \to 3 \to 2| \to 3 \to 4$

$I'_1 = 5 \to 2 \to 1| \to 4 \to 5$

Questi due individui pero' non sono validi!

Una possibilita' e' dunque quella di prendere la prima parte del crossover e aggiungere come seconda parte gli elementi mancanti nel ordine in cui appaiono nella altra soluzione, nel nostro caso:

$I'_0 = 1 \to 3 \to 2| \to 5 \to 4$

$I'_1 = 5 \to 2 \to 1| \to 3 \to 4$

Anche le mutazioni in molti casi non funzionano, ancora un volta nel nostro caso:

$$1 \to 3 \to 2 \to 4 \to 5 \quad \underset{\text{muta il 3 elemento}}{\longrightarrow} \quad 1 \to 3 \to 4 \to 4 \to 5$$

Alcune possibilita' di mutazioni sono semplicemente:

- Permutare due elementi
- Permutare due parti del percorso (quindi + elementi alla volta)
- Spostare un elemento in una altra posizione (shiftando gli elementi)

4.8 I teoremi degli schemi

Come abbiamo visto nell'esempio del posizionamento delle antenne GSM, gli **AG** cercano la soluzione ottimale giustapponendo dei pezzi delle soluzioni piu' promettenti.

I teoremi degli schemi esprime in maneiera matematica come i migliori pezzi della soluzione sono amplificati e preservati.

$\to$ E' un tentativo di dimostrare la convergenza delle **AG**.

Si utilizzera' dinuovo delle catene di bit, uno schema e' una espressione regolare semplice di bits.

Per esempio uno schema puo' essere $S = (1**1*0)$, per esempio la catena (1 0 0 1 0 0) appartiene allo schema S.

L'ordine $o(S)$ di uno schema S e' il numero di posizioni fisse dello schema, nel nostro caso $o(1**1*0) = 3$.

La lunghezza $\delta(S)$ di uno schema e' la distanza tra la prima e l'ultima posizione fissa, nel nostro caso $\delta(1**1*0) = 6 - 1 = 5$, per un caso $\delta(**1*) = 3 - 3 = 0$.

Si cerchera' di stimare l'evoluzione del numero di individui che possiede uno schema S dato.

Si definisce la fitnesse medio di uno schema S

$$\underset{P(t)}{f(t,S)} = \frac{1}{M(t,S)} \sum_{i=1}^{M(t,S)} f(v_i)$$

dove $v_1, v_2, ..., v_{M(S)}$ sono gli $M(t,S)$ individui della popolazione $P(t)$ che soddisfano lo schema S.

Si puo' anche definire la fitness totale della popolazione $P(t)$ al tempo t

$$F(t) = \sum_{v \in P(t)} f(v)$$

La fitness totale media e':

$$\bar{F}(t) = \frac{1}{n} F(t)$$

con n la dimensione totale della popolazione.

Vogliamo calcolare $M(t+1,S)$ il numero di individui che contengono lo schema S al tempo $t+1$ in funzione di $M(t,S)$. La teoria degli schemi calcola questo numero cosi:

$$M(t+1,S) = \alpha M(t,S) \times [\text{Prob } S \text{ sopr al crossover}] \times [\text{Prob } S \text{ sopr alla mutazione}]$$

$\alpha M(t,S)$ e' il numero di individui con S selezionati dal processo di selezione.

Il caclcolo dei termini (che indicheranno se lo schema S e' amplifacato o preservato) sono:

Gli effetti della selezione:

Con una selezione proporzionale

$$M(t+1,S) = n \frac{M(t,S) \cdot f(t,S)}{F(t)} = M(t,S) \cdot \frac{f(t,S)}{\bar{F}(t)}$$

Se $\frac{f(t,S)}{F(t)} > 1$ (ovvero se $f(t,S)$ e' + grande della fitness media totale) lo schema S sara' amplifacato dalla selezione.

Probabilita' che S sopravviva al crossover

Uno schema S e' distrutto dal crossover se i punti di incrocio cade all'interno dello schma. Con uno schrma S di lungheza $\delta(S)$ ci saranno dunque $\delta(S)$ punti di incrocio distruttivi.

Se gli individui sono catene di m bits ci sono dunque $m-1$ possibili punti di incroci equiprovabili, la probabilita' dunque di distruggere lo schema S e' inferiore a:

$$\frac{\delta(S)}{m-1}$$

Inferiore perche' nel caso due individui abbiamo entrambi lo schema S nella stessa posizione lo schema sopravvivera' in tutti i casi.

Dunque lo schema S soprivvevera' con una probabilita':

$$1 - P_c \cdot \frac{\delta(S)}{m-1}$$

dove P_c e' la probabilita' di crossover.

A questo punto abbiamo ottenuto questa formula:

$$M(t+1,S) = M(t,S) \cdot \frac{f(t,S)}{\bar{F}(t)} \cdot \left[1 - P_c \cdot \frac{\delta(S)}{m-1}\right] \cdot [\text{prob di sop. alle mutazioni}]$$

Probabilita' che S sopravviva alle mutazioni

Uno schema S e' distrutto se si muta uno dei suoi punti fissi.

Sia P_m la probabilita' di mutare ciascuno degli m bits dell'individuo.

La probabilita' di non mutare $o(S)$ bits fissi di S e'

$$(1 - P_m)^{o(S)}$$

In generale $P_m \ll 1$ e quindi possiamo approssimare

$$(1 - P_m)^{o(S)} \approx 1 - o(S) \cdot P_m$$

Possiamo anche fare la seguente approsimazione:

$$P_m \cdot P_c \approx 0$$

e quindi quando facciamo il prodotto

$$\left[1 - P_c \cdot \frac{\delta(S)}{m-1}\right] \cdot [1 - o(S) \cdot P_m] = \left[1 - P_c \cdot \frac{\delta(S)}{m-1} - o(S) \cdot P_m\right]$$

E finalmente si ottiene che

$$M(t+1,S) = M(t,S) \cdot \frac{f(t,S)}{\bar{F}(t)} \cdot \left[1 - P_c \cdot \frac{\delta(S)}{m-1} - o(S) \cdot P_m\right]$$

Se $\frac{f(t,S)}{\bar{F}(t)} \cdot \left[1 - P_c \cdot \frac{\delta(S)}{m-1} - o(S) \cdot P_m\right] > 1$ allora si avra' una amplificazione di S. Dunque per ottenere una amplificazione di S bisognera' avere una fitness media dello schema $f(t, S)$ maggiore della fitness media totale $\bar{F}(t)$. Inoltre che lo schema S abbia lunghezza $\delta(S)$ e ordine $o(S)$ piccola.

Dunque S per essere amplificato deve essere "buono" e corto.

Si spera dunque che la soluzione ottimale sia ottenuta per giustapposizione di schemi corti e buoni, chiamati building blocks. Purtorppo ci sono degli esempi che mostrano che gli **AG** si sbaglino e siano attratti da zone sub-ottimali.

4.9 Popolazioni strutturate

Fino ad adesso abbiamo sempre supposto che tutti gli individui della popolazione possano interagire l'un con l'altro, senza costrizioni, per esempio geografiche. In biologia questo tipo di popolazioni si chiamano panmictiche.

Ma nell'evoluzione delle specie, non si potra' veramente riprodursi con qualsiasi individui aribitrariamente e indipendentemente dalla distanza.

Di conseguenza avremo una popolazione piu' diversa di quella che otterremmo con una popolazione panmictica.

Per questo speriamo che questo tipo di popolazione sia benefico per gli algoritmi **AG**. Questo permettera' di mantenere la popolazione e qundi di migliorare il tempo di convergenza verso l'individuo massimale.

Inoltre questo permettera' una parallelizzazione piu' semplice del processo di evoluzione.

Ci sono due modi principali per costruire delle popolazioni strutturate:

1. le *multi-popolazioni*
2. le popolazioni *cellulari*

4.9.1 Le multi-popolazioni

Ci si dona un insieme di popolazioni $P_1, P_2 ... P_k$ che co-evolve. Possiamo immaginarli come facenti parte di un grafo di interazione

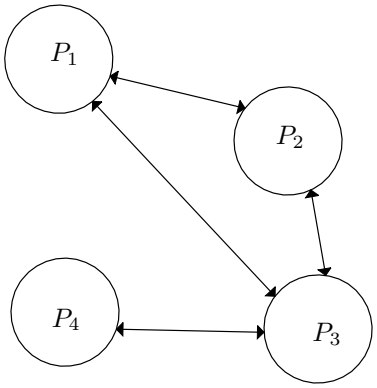


Figura 1.

Queste linee rapressentano le possibilita' di scambio tra gli individui delle diverse popolazioni (migrazioni).

Si applica l'**AG** a ciascuna popolazione come nei casi precedenti.

Di tanto in tanto i migliori elementi di ciascuna poplazione P_i saranno copiati e rimpiazati con degli individui di una popolazione P_j collegata a P_i.

In generale le popolazioni sono collegate con una topologia ad anello di questo tipo:

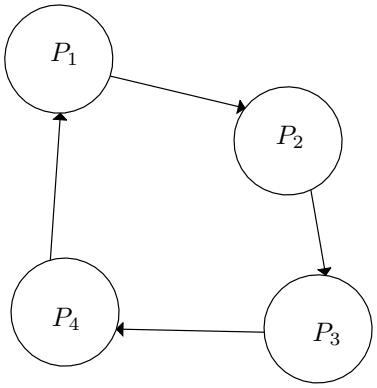


Figura 2.

Bisogna specificare alcuni punti:

- Mantenere un numeri di individui per popolazione assai grande (tipicamente 50 individui)
- Scegliere tra il 5% e il 10% di popolazione migrante
- Le migrazioni avvengono ogni 10 generazioni

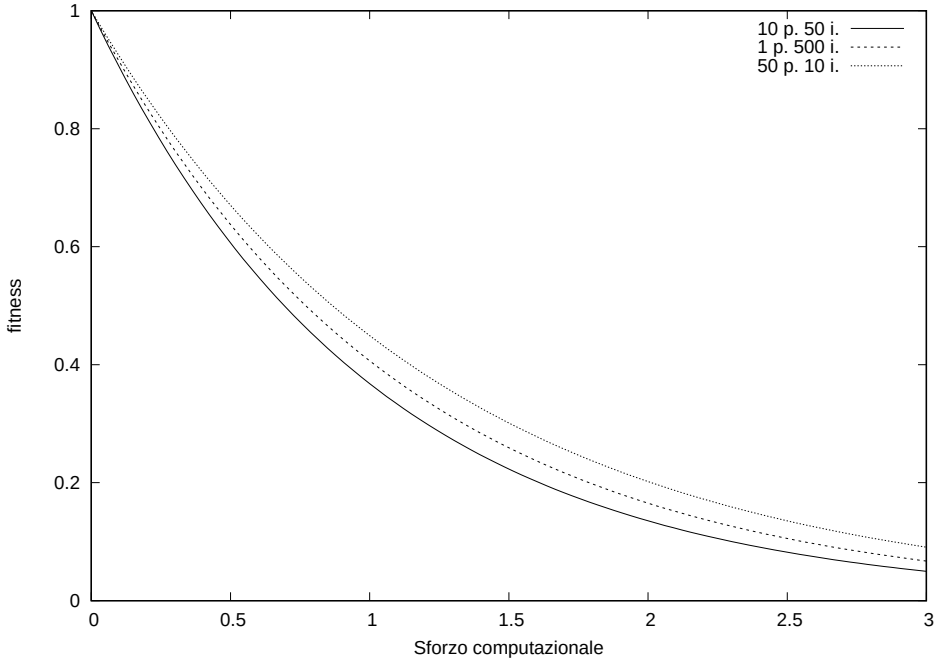
Gli scambi si fanno in modo asincrono: e' la popolazione emettitrice che decide chi e quando inviare i migranti. La popolazione ricevente accetta i migranti quando pronta. Questo permette di implementare una parellilizzazione molto efficace (un processo per popolazione) e senza problematiche di equilibrio della carica.

Un qeustione importante rimane la scelta del numero delle popolazioni e la loro dimensione, con un totale di 1000 individui si potrebbero avere le seguenti configurazioni:

Popolazioni	Individui
1	1000
2	500
⋮	⋮
10	100
⋮	⋮
1000	1

Tabella 1.

Possiamo scegliere la configurazione che minimizza il tempo di convergenza, o piu' precisamente il numero di evoluzioni totatali della fitness, per esempio:

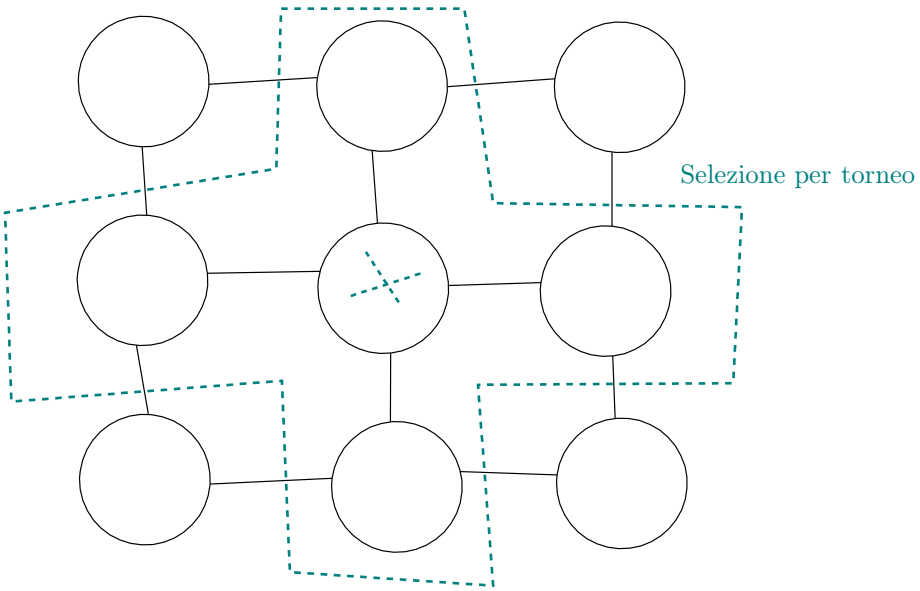


In questo caso la configurazione 50 popolazioni per 10 individui ottine fitness migliori con meno sforzo delle altre.

4.9.2 Popolazioni cellulari

Questo e' un secondo modo di creare popolazioni strutturate.

L'idea e' di mettere ciascun individuo su un griglia ou un grappolo



In questo caso abbiamo 9 individui su una griglia

- La selezione si fara' per torneo: Ogni individuo entra in competizione con i suoi vicini e rimipazzera il vicino
- Il crossover si fara' scegliendo a caso uno dei suoi vicini con cui scambiera' una parte del suo codice genetico
- La mutazione si fara' come sempre.

Con queste regole possiamo fare una aggiornamenteo sincrono di ogni individuo.

Il vantaggio di questo modello e' di mantenere una diversita' per un maggiore tempo, cosa che favorisce la esplorazione.

E' possibile studiare il *takeover time*. per una configurazione ad anello $\tau = O(n)$ dove n e' il numero di individui, mentre per un caso a griglia 2D (come in esempio) $\tau = O(\sqrt{n})$.

5 Programmazione Genetica (GP)

5.1 Introduzione

E' un dominio sviluppato negli anni 90, in particolare da *Kozu*. L'idea e' quella di utilizzare l'evoluzione darwiniana a dei programmi informatici.

Invece di avere dei vettori numerici come negli **AG** si utilizzeranno delle popolazioni di codice che potra duqnque mutare, incrociare e essere selezionato.

Questi programmi dovranno produrre un output ragionabile a partire di un input dato. Possiamo vedere dunqui la **GP** come un problema di apprendimento, o come un problema di minimizzazione tra l'output desiderato e quello ottenuto (l'errore).

Spesso ci si dona un insieme di coppie <input, output> che costituiscono l'insieme dell'apprendimento, anche chiamato test-set.

Altre volte possiamo mettere il programma in condizioni reali e valutare le performance: ex. programma di pilotaggio automatico testato con un simulatore di volo.

Si spera che il programma che emergera' da questa procedura di evoluzione sara' capace di generalizzare. (ovvero reagira' in modo corretto con nuovi dati, non presenti nel *test-set*)

Se il *test-set* non e' corretamente scelto (per esempio e' troppo specifico) il risultato non sara' corretto.

5.2 Scrittura di un programma

Come possiamo rappresentare un programma informatico che sia consistente dopo l'insieme di mutazioni e di crossover?

I linguaggi classici (C,java, python..) sono troppo fragili per sopportare delle mutazioni aleatorie o del crossover: semplicemente non saranno piu' corretti ne sintatticamente ne semanticamente.

Dunque si useranno dei linguaggi ad hoc:

- S-expressions (o alberi) proposti da Koza
- Programmazione a catasta (pila)

Questi programmi sono robusti alle mutazioni e sono sempre eseguibili. Grazie alla selezione si spera di ottenere dei programmi con risultati sperati.

Per valutare la qualita' di un programma $p(x)$ bisogna definire la sua fitness in funzione della sua capacita' di riprodurre l'insieme dei dati del *training set* A:

$$A = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$$

Dove x_i e' l'input e y_i e' l'output atteso, dunque la fitness $f(p)$ e' definita come:

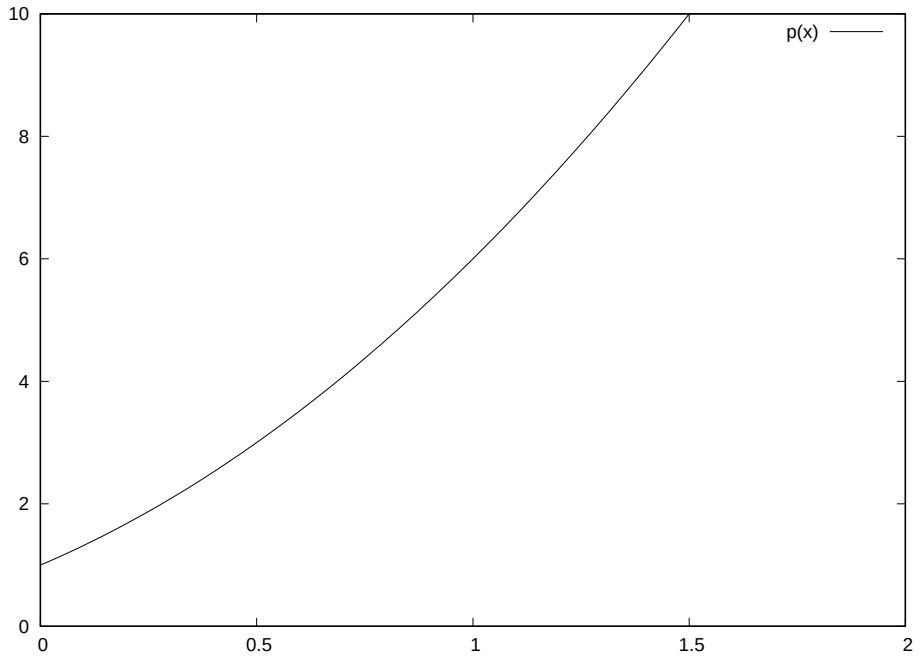
$$f(p) = \sum_{x_i, y_i \in A} |y_i - p(x_i)|$$

Questo e' dunque l'errore e quindi bisogna minimizzare il problema.

Spesso il programma p che cerchiamo dovra' generare una espressione algebrica $y_i = p(x_i)$

P. esempio $p(x)$ potrebbe essere un polinomio:

$$p(x) = 2x^2 + 3x + 1$$



Se si resce a ricavare la formula a partire dei dati di training il programma sara' generale e potra' funzionare correttamente anche su punti al di fuori di A.

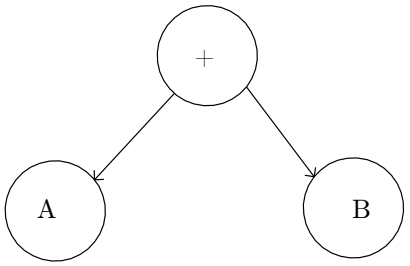
5.2.1 S-Expression

Kozu a proposto questa rapresentazione funzionale per questo tipo di programmi:

$$\begin{aligned} A+B &\rightarrow (+A, B) \\ 2x^2+3x-1 &\rightarrow (-(+(\times(\times x, x), 2), (\times x, 3)), 1) \end{aligned}$$

Il vantaggio di questa rapresentazione che cambiandio qualsiasi operazione si ottiene comunque una operazione valida. Questo dunque ci permettera' di eseguire le funzioni di mutazione e crossover.

Per semplificare questa notazione puo' essere espressa sotto forma di grafo:



(+ A,B) o A+B

In particolare per espressioni piu' complicate questa rapresentazione e' molto + efficace.

5.2.2 Function-Set e Terminal-Set

Per specificare un insieme di programmi genetici ci si basa su qeusta rapresentazione e si definisce:

- Function-Set F: l'insieme di operatori che scegliamo di usare, ex:

$$F = \{+, \times, \div, -\}$$

- Terminal-Set T: L'insieme degli operandi scelti (o il numero di variabili), ex

$$T = \{A, B, C, D, ...\}$$

  I valori di queste variabili saranno presi dal training set. Inoltre e' possibile che siano definite delle costanti.

Per esempio se volessimo trovare una espressione booleana potremmo definire

$$F = \{\text{NOT}, \text{AND}, \text{OR}\}$$

$$T = \{b_1, ..., b_n, \text{TRUE}, \text{FALSE}\}$$

dove TRUE e FALSE sono delle costanti mentre $b_1 ... b_n$ sono variabili.

Tutto il programma sara' quindi costrutio usando unicamente gli operatori F e i termini di T. E' importante che gli operatori di F siano calcolabili su tutti i valori del problema (ex. in caso di un problema reale la divisione deve poter accettare una divisione per 0).

Un altro problema e' il fatto che non tutti gli operatori hanno necesariamente 2 operandi ma per esempio il NOT ne ha uno solo. Per risolvere il problema possiamo o estendere l'operatore in modo che tratti sempre due operandi (ex (NOT A, B) $\rightarrow$ NOT A), quindi si ignora il secodo termine), altrimenit si puo' implementare il programma in modo piu' intellegente che tiene in conto di questo nella costruzione dell'albero.

In questo caso ad ogni tappa il programma riempe il nodo con un termine o con un operando, se e' un termine il ramo si conclude, altrimenti l'algoritmo continua ad avanzare creando uno o piu' rami a seconda dell'operatore scelto.

A mano a mano che la profondita' dell'albero aumenta la probabilita' di scegliere un operando invece di un operatore dovra' essere piu' grande (e ovviamente anche il contrario).

Nota: Un programma genetico qundi non ha lunghezza fissa ma albitraria (a differenze degli **AG**). Inoltre con le operazioni di mutazione e crossover il programma tendera' ad aumentare di dimensione. Per ovviare a questo problema bisogna in genere fissare una lunghezza massima, altrimenit si puo' avere il problema di "*bloat*".

Bisogna in oltre semplificare gli alberi dagli elementi inutili come per ex:

$$(-X, X)$$

Inoltre i programmi genetici potranno avere un IF all'interno del programma e questo avra' la forma:

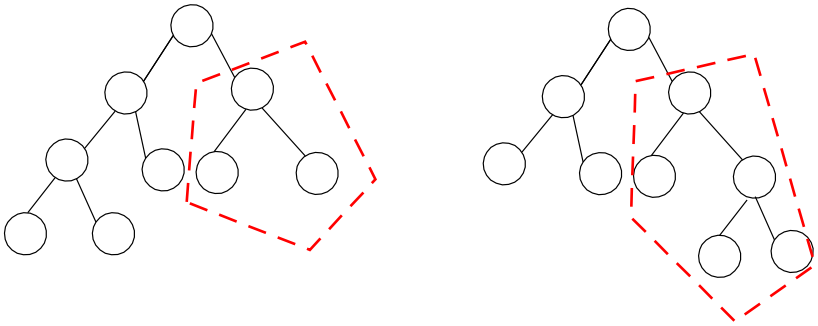
$$(\text{IF } A, B, C)$$

Questo signgifica:

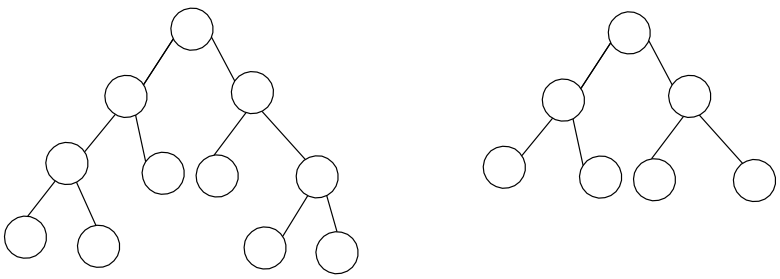
$$\text{IF } A \rightarrow B \quad \text{ELSE} \rightarrow C$$

5.2.3 Operatori Genetici

Ovvero come applicare la matuazione e il crossover a questo tipo di programmi, ex:



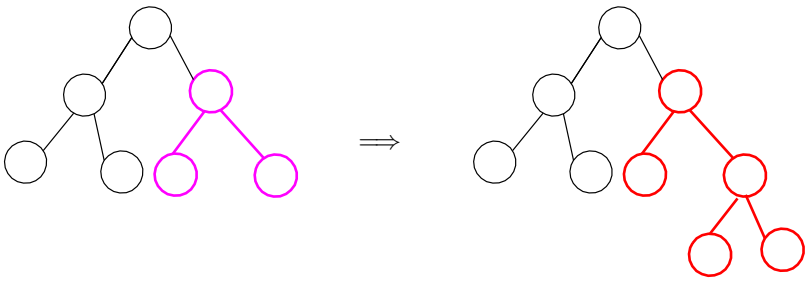
Per eseguire il corssover si scambiano semplicemente due sotto alberi delle soluzioni. Quindi risultera' in :



In generale i nodi sono scelti in modo aleatorio con una probabilita' non uniforme in modo da favorire il crossover sui nodi interni e non sulle foglie.

Come per gli **AG** il crossover si avverra' con una probabilita' p_c per la coppia di genitori scelti.

Per quanto riguarda la mutazione invce si seleziona un nodo e si genera a caso un sotto albero ex:



La parte viola e' la parte che e' stata rimpizzata dal sotto albero casuale rosso.

5.2.4 Programmi genetici sequenziali a "pila"

Piu' simili ai programmi procedurali, la rappresentazione consiste in una lista di istruzioni eseguite in modo sequenziale.

Per garantire dei programmi sintatticamente correti e robusti sia al crossover che a mutazione, l'approccio a pila (stack-based) e' utile (vedi notazione polacca inversa).

Si ha una pila di valori numerici e di istruzioni che consumano un certo numero di valori, fanno il calcolo e ripongono il risultato in cima della lista, ex:

La variabile `A` opera come una funzione e aggiunge il valore della variabile in cima alla pila

LA funzione `ADD` somma i primi due valori della pila, dunque:

`A B ADD C MUL 2 SQRT DIV` $\rightarrow$ `A+B C MUL 2 SQRT DIV` $\rightarrow$ `(A+B)*C 2 SQRT DIV` $\rightarrow$

`(A+B)*C` $\sqrt{2}$ `DIV` $\rightarrow \frac{(A+B)*C}{\sqrt{2}}$

In questo caso bisogna essere sicuri che le istruzioni siano robuste alla mancanza di operandi nella pial:

$$\texttt{A ADD} \rightarrow ? = \begin{cases} 0 \\ \texttt{A} \end{cases}$$

I programmi sono delle liste composto da istruzioni disposte in un ordine albitrario.

Il corssover e la mutazione sono molto simili a quelle che abbiamo visto nel caso di una catena di bit, esempio:

p_1: `A B ADD | C MUL 2 SQRT DIV` $\rightarrow \frac{(A+B)*C}{\sqrt{2}}$

p_2: `1 B SUB | A A MUL ADD 3` $\rightarrow$ `(1-B)+A`^{2}`, 3`

crossover

p_1^1: `A B ADD | A A MUL ADD 3` $\rightarrow$ `(A+B)+A`^{2}`, 3` (il 3 rimane sulla pila)

p_2^1: `1 B SUB | C MUL 2 SQRT DIV` $\rightarrow \frac{(1-B)*C}{\sqrt{2}}$

si nota che i programmi incrociati hanno sempre la stessa dimensione dei genitori.

Per le mutazioni si sceglie di modificare ogni istruzione del programma con probabilita' $p_m \ll 1$ e si trasforma con un altra istruzione scelta' a caso. (si possono avere probabilita' diverse per le istruzioni di tipo "Variabile" e quelle di tipo "Operazione")

Esempio

Ci piacerebbe trovare l'espressione booleana di $x_1, x_2, x_3 \in \{0, 1\}$ definita dalla tavola di verita':

x_1	x_2	x_3	output
0	0	0	0
0	0	1	0
0	1	1	0
0	1	0	0
1	1	0	0
1	0	0	0
1	0	1	0
1	1	1	1

Tabella 2. tabella rapresstnta x_1 and x_2 and x_3

quindi $T = \{x_1, x_2, x_3\}$ e $F = \{\text{AND}, \text{OR}, \text{NOT}\}$

Abbiamo quindi 6 istruzioni possibili, 3 in F e 3 in T. Inoltre considremo un programma di lunghezza fissa 5. In totale abbiamo 6^5 programmi possibili.

L'outup del programma e' il valore incima della pila.

La popolazione scelta sara' di soli 12 programmi e una evoluzione di 50 iterazioni. L fitness sara' semplicemente il numero di valori corretti corrispondenti alla tabelal di verita', quindi si tratta di un problema di massimizzazione (dove $f(\max) = 8$, ovvero tutti gli output corretti).

5.2.5 Struttre di controllo nella programmazione a pile

Con questo metodo e' possibile inoltre implementare strutture di controllo piu' avanzate come `IF` e `LOOP`:

- `IF`: si aggiungono due istruzioni, `IF` e `ENDIF`, a F. Quando si incontra un `IF` se il valore nella pila e' positivo si esegue il codice dopo l'if normalmente. Altrimenti il codice compreso tra `IF` e `ENDIF` sara' saltato. Per fare cio' il programma avra' bisogno di un contatore per sapere se eseguire o no il codice all'interno del if (un contatore per contare il caso degli `IF` concatenati)

- `LOOP`: anche in questo caso si aggiungo due istruzioni, `LOOP` e `ENDLOOP`, a F, ma in questo caso ci sara' bisogno di creare un nuovo stack (control stack) per controllare il numero di iterazioni del loop. Quando il codice trova un `LOOP` si trasferisce il valore in cima alla pila nel *control stack*. Se quando si incontra `ENDLOOP` si controlla che la *control stack* non sia vuota e se non e' vuota si decrementa il valore nella *control stack*. Se il valore del *control stack* sara' $\leqslant 0$ il loop sara' finito. Anche questa operazione e' robusta a mutazioni / crossover

In tutti i due casi se l'istruzioni di `ENDIF` o `ENDLOOP` non sono presenti tutto il codice dopo `IF` o `LOOP` sara' considerato parte del if or loop.

6 Evolutionary Strategy

Anch'esso (**ES**) parte degli algortimi evolutivi, come l'**AG** e la **PG**. E' stato proposto negli anni 60 da Rechenberg e Schwefeld.

Inizialmente e' stata proposta come una metaeursitica a un individuo in $x \in \mathbb{R}^d$. L'evoluzione si effettuava solo per mutazione. In seguito **ES** e' stata amigliorata aggiungendo una popolazione (e non solo un individuo) e il *crossover* tra gli individui.

Ci accontenteremo di fare un rapido sorvolo di questa metaeuristica, di cui esistono varie varianti.

6.1 $(1+1)-$ ES

Questa e' la prima variante, $(1+1)$ indica che c'e un solo successore che rimpiazza il predecessore, e quindi questa e' la versione **ES** senza popolazione.

Se $x(t) \in \mathbb{R}^d$ e' la soluzione attuale si creera' un figlio $x'(t)$ per semplice mutazione:

$$x'(t) = x(t) + N(0, \sigma)$$

con $N(0, \sigma)$ una distribuzione gaussiana (o normale) centrata in $\mu = 0$ con varianza σ.

In seguito si sceglie il successore $x(t+1)$:

$$x(t+1) = \begin{cases} x'(t) & f(x'(t) \text{ migliore dif}(x(t)) \\ x(t) & \text{altrimenti} \end{cases}$$

Si ricorda che stiamo lavorando in $\mathbb{R}^d$ e quindi $x(t)$ e' un vettore a d dimensioni: $x(t) = \{x_1, x_2, ..., x_d\}$. Quindi anche la distribuzione $N(0, \sigma)$ e σ sono vettori a d dimensioni.

Inoltre σ puo' evolvere nel corso delle generazioni, se le mutazioni sono scartate troppo frequentemente allora si aumentra' σ. Invece se le mutazioni sono troppo spesso selezionate si farea diminuire σ. Un buon valore di accettazione della mutazione e' $\frac{1}{5}$.

Questo rapporto di accettazione e' calcolato sulle ultime k selezioni (con k un parametro).

6.2 ES a poplazione