Algoritmi Euristici

Corso di Laurea in Informatica e Corso di Laurea in Matematica

Roberto Cordone

DI - Università degli Studi di Milano

Lezioni:	Lunedì 13.30 - 15.30 in Aula G30
	Giovedì 13.30 - 15.30 in Aula G30
Ricevimento:	su appuntamento
Tel.:	02 503 16235
E-mail:	roberto.cordone@unimi.it
Web page:	http://homes.di.unimi.it/~cordone/courses/2018-ae/2018-ae.html

Euristiche di ricombinazione

Le euristiche costruttive e di scambio gestiscono una soluzione alla volta (eccettuato l'*Ant System*)

Le euristiche di ricombinazione gestiscono molte soluzioni in parallelo

- partono da un insieme (popolazione) di soluzioni (individui) ottenuto in qualche modo
- ricombinano individui producendo una nuova popolazione

Il loro aspetto originale è l'uso di operazioni che lavorano su più soluzioni, ma spesso includono elementi delle altre euristiche (a volte ribattezzati)

Alcune sono quasi o del tutto deterministiche

- Scatter Search
- Path Relinking

altre fanno forte uso di passi casuali (spesso in base a metafore biologiche)

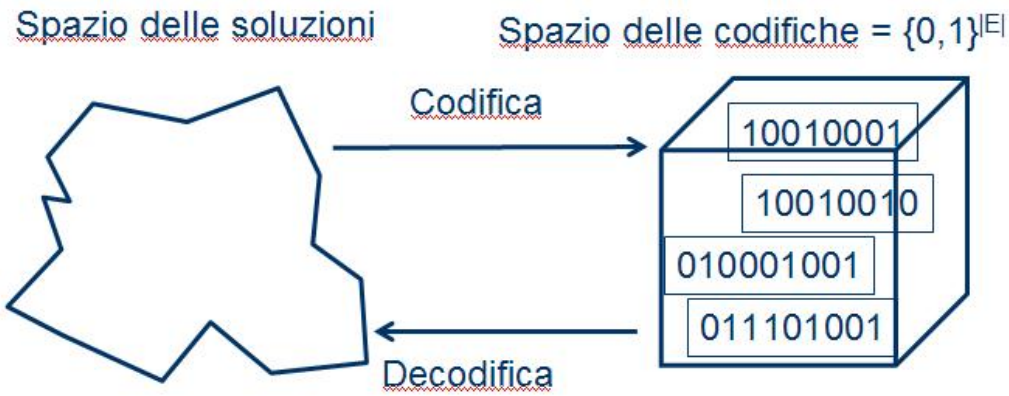
- algoritmi genetici
- algoritmi memetici
- strategie evoluzionistiche

Ovviamente l'efficacia di un metodo non dipende dalla metafora fondante

Algoritmi operanti su codifiche

Molte euristiche di ricombinazione lavorano su codifiche delle soluzioni (rappresentazioni compatte), anziché sulle soluzioni stesse

- gli operatori di scambio e ricombinazione sono definiti sulle codifiche



Le ragioni di questo approccio sono

- astrazione: distinguere concettualmente il metodo dal problema cui viene applicato
- generalità: costruire operatori di scambio e ricombinazione validi per ogni problema con un dato insieme di codifiche

A rigore, ogni rappresentazione di una soluzione è una codifica:
si tende a definire codifiche le rappresentazioni più involute e compatte

La differenza è sfumata

Algoritmo genetico

L'algoritmo genetico, proposto da Holland (1975) è l'esempio più noto

Prevede la costruzione di una popolazione $X^{(0)}$ seguita ripetutamente da

1. selezione: genera una nuova popolazione a partire da quella corrente
2. crossover: ricombina sottoinsiemi di due o più individui
3. mutazione: modifica singoli individui

Algorithm AlgoritmoGenetico$(I, X^{(0)})$

$x^* := \arg \min_{x \in X^{(0)}} f(x)$; { Miglior soluzione trovata sinora }

For $g = 1$ *to* n_g *do*

$X^{(g)} := Selezione(X^{(g-1)})$;

$X^{(g)} := Crossover(X^{(g)})$;

$x_c := \arg \min_{x \in X^{(g)}} f(x)$;

If $f(x_c) < f(x^*)$ *then* $x^* := x_c$; { Aggiorna la miglior soluzione }

$X^{(g)} := Mutazione(X^{(g)})$;

$x_m := \arg \min_{x \in X^{(g)}} f(x)$;

If $f(x_m) < f(x^*)$ *then* $x^* := x_m$; { Aggiorna la miglior soluzione }

EndFor;

Return $(x^*, f(x^*))$;

Caratteristiche di una buona codifica

Le prestazioni di un algoritmo genetico dipendono dalla codifica scelta

Dovrebbero valere le seguenti proprietà (con importanza decrescente)

1. ogni soluzione deve avere una codifica,
   altrimenti vi sarebbero soluzioni non ottenibili
2. ogni codifica deve essere traducibile in una soluzione,
   altrimenti la popolazione conterrebbe individui inutili
3. ogni soluzione dovrebbe corrispondere a un pari numero di codifiche,
   altrimenti vi sarebbero soluzioni indebitamente avvantaggiate
4. le operazioni di codifica e decodifica dovrebbero essere efficienti,
   altrimenti si avrebbe un algoritmo altamente inefficiente
5. località: modifiche piccole alla codifica dovrebbero produrre
   modifiche piccole nella decodifica, altrimenti non si può intensificare

Queste condizioni dipendono moltissimo dai vincoli del problema

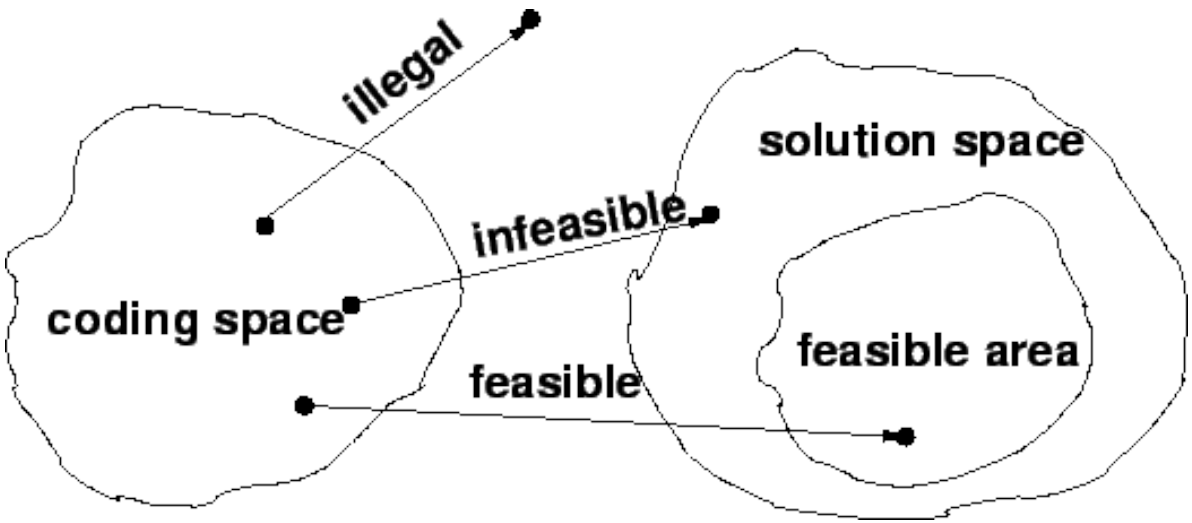
(*alla faccia dell'astrazione...*)

Caratteristiche di una buona codifica

In particolare, operatori di mutazione e crossover generici producono facilmente sottoinsiemi non ammissibili

1. per violazione di vincoli quantitativi (ad es., capacità)
2. per violazione di vincoli strutturali (sottoinsiemi di archi che non formano cicli, alberi, ecc. . .)

Sono tutte inammissibilità, ma le seconde sono più difficili da riparare, perché riguardano vincoli che interagiscono più fortemente fra loro



Codifiche: il vettore di incidenza

La codifica più diretta per problemi di Ottimizzazione Combinatoria è il vettore binario di incidenza $\xi \in \mathbb{B}^{|B|}$

$$\begin{cases} \xi_i = 1 \text{ indica che } i \in x \\ \xi_i = 0 \text{ indica che } i \notin x \end{cases}$$

Un vettore binario generico corrisponde

- nel *KP* a un insieme di oggetti: potrebbe avere peso eccessivo
- nel *SCP* a un insieme di colonne: potrebbe lasciare righe scoperte
- nel *PMSP* e nel *BPP* a un insieme di assegnamenti di lavorazioni (oggetti) a macchine (contenitori): potrebbe assegnare alcune lavorazioni zero o più volte; nel *BPP*, potrebbe violare la capacità di qualche contenitore;
- nel *TSP* a un insieme di archi: potrebbe non formare un circuito hamiltoniano
- nel *CMST* (*VRP*) a un insieme di lati (archi): potrebbe non formare un albero (insieme di cicli), oppure eccedere la capacità dei sottoalberi (cicli)

Codifiche: le stringhe di simboli

Se l'insieme base è partizionabile in componenti

$$B = \bigcup_{c \in C} B_c \quad \text{con } B_c \cap B_{c'} = \emptyset \text{ per ogni } c \neq c'$$

in modo che le soluzioni contengano un elemento di ogni componente

$$|x \cap B_c| = 1 \text{ per ogni } c$$

(per es., variabili logiche, oggetti, lavorazioni, vertici, nodi, ecc. . .) si può

- definire per ogni componente c un alfabeto di simboli che descrive B_c (per es., valori logici, contenitori, macchine, rami, veicoli, ecc. . .)
- codificare la soluzione con una stringa di simboli $\xi \in B_1 \times \ldots B_{|C|}$

$$\xi_c = \alpha \text{ indica che } x \cap B_c = (c, \alpha)$$

Esempi di codifiche:

- *Max-SAT*: la stringa degli n valori per ogni variabile logica
- *PMSP*: la stringa delle macchine cui assegnare ogni lavorazione
- *BPP*/*CMST*: la stringa dei contenitori/rami cui assegnare ogni oggetto/vertice:
  - soddisfa il vincolo di assegnamento degli oggetti/vertici
  - ignora il vincolo di capacità dei contenitori/rami
- *VRP*: vale lo stesso, ma in più la decodifica del ciclo percorso da ogni veicolo a partire dai nodi serviti è un problema $\mathcal{NP}$-completo
- il *TSP*, il *KP*, il *SCP* non hanno partizioni come soluzioni

Codifiche: le permutazioni

Una codifica di uso comune è data dalle permutazioni di un insieme

- le soluzioni sono permutazioni, è la codifica naturale (nel *TSP*, nodi)
- se le soluzioni sono partizioni e l'obiettivo è additivo sui sottoinsiemi, il metodo *order-first split-second* trasforma permutazioni in partizioni
  (*la relazione fra soluzioni e codifiche non è biunivoca*!)
- se il problema ha un algoritmo costruttivo che ad ogni passo esegue
  - scelta di un elemento
  - (eventualmente) scelta del modo di inserirlo in soluzione

  è possibile scegliere gli elementi nell'ordine dato dalla permutazione
  (*se i modi sono tanti, una soluzione può non avere codifica*)

Selezione

Ad ogni generazione g si costruisce una nuova popolazione $X^{(g)}$

$$X^{(g)} := \text{Selezione}\left(X^{(g-1)}\right);$$

estraendo $n_p = |X^{(g)}|$ individui dalla popolazione corrente $X^{(g-1)}$

L'estrazione segue due criteri fondamentali

1. è lecito estrarre più volte lo stesso individuo
2. la probabilità di estrazione è più alta per gli individui migliori

$$\varphi(\xi) > \varphi(\xi') \Rightarrow \pi_\xi \geq \pi_{\xi'}$$

dove la fitness $\varphi(\xi)$ è una misura di qualità dell'individuo ξ, legata al valore dell'obiettivo per la corrispondente soluzione

- per un problema di massimizzazione, in genere si usa

$$\varphi(\xi) = f(x(\xi))$$

- per un problema di minimizzazione, in genere si usa

$$\varphi(\xi) = \frac{1}{f(x(\xi))} \text{ oppure } \varphi(\xi) = -f(x(\xi))$$

Sono stati proposti diversi schemi di selezione

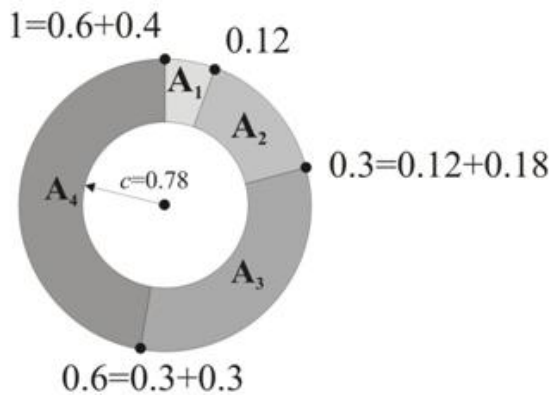
Selezione proporzionale

Lo schema proposto originariamente da Holland (1975) assumeva una probabilità proporzionale alla fitness

$$\pi_\xi = \frac{\varphi(\xi)}{\sum_{\xi \in X^{(g-1)}} \varphi(\xi)}$$

Si parla di *roulette-wheel selection* o *spinning wheel selection*:

- data la popolazione $X^{(g)} = \{\xi_1, \ldots, \xi_{n_p}\}$
- si costruiscono gli intervalli $\Gamma_i = \left(\sum_{k=1}^{i-1} \pi_{\xi_i}; \sum_{k=1}^{i} \pi_{\xi_i} \right]$
- si estrae un numero casuale $r \in U(0; 1]$
- l'individuo i^* scelto è quello tale che $r \in \Gamma_{i^*}$



Selezione per rango

La selezione proporzionale soffre di

- convergenza prematura: se i migliori individui sono cattivi e gli altri pessimi, la selezione produce rapidamente una popolazione cattiva
- stagnazione: a lungo termine, tutti gli individui tendono ad avere una buona fitness, dunque uguale probabilità di selezione

Per ovviare a questi difetti occorre al tempo stesso

- limitare la differenza di probabilità tra individui diversi
- differenziare la probabilità tra individui simili

La selezione per rango

- ordina gli individui per fitness non decrescente

$$X^{(g)} = \{\xi_1, \ldots, \xi_{n_p}\} \text{ con } \varphi_{\xi_1} \leq \ldots \leq \varphi_{\xi_{n_p}}$$

- assegna all'individuo k-esimo probabilità

$$\pi_{\xi_k} = \frac{k}{\sum_{k=1}^{n} k} = \frac{2k}{n(n-1)}$$

È computazionalmente più gravoso che usare direttamente la fitness

Selezione per torneo

È un compromesso che non richiede l'ordinamento completo delle fitness

- estrae n_p sottoinsiemi $\bar{X}_1, \ldots, \bar{X}_{n_p}$ di dimensione α
- seleziona in ciascuno l'individuo di fitness massima

$$\xi_k := \arg\max_{\xi \in \bar{X}_k} \varphi(\xi)$$

Il parametro α modula la forza della selezione:

- $\alpha \approx n_p$ favorisce gli individui migliori
- $\alpha \approx 2$ lascia buone probabilità anche agli individui cattivi

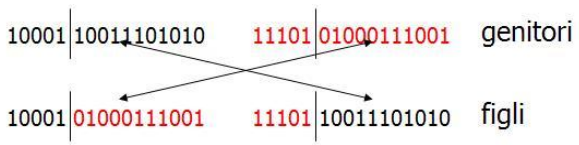
Tutte le procedure di selezione ammettono la variante elitista, che che include nella nuova popolazione l'individuo migliore di quella corrente (*cioè l'individuo migliore trovato sinora*)

Crossover

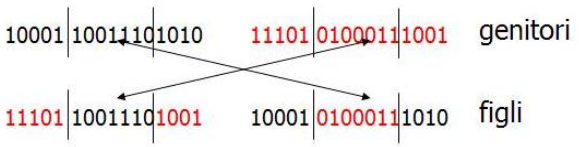
L'operatore di crossover combina due individui producendone altri due

I più comuni sono

- crossover semplice:
  - si estrae una posizione a caso con probabilità uniforme
  - si spezza la codifica in due parti in corrispondenza a tale posizione
  - si scambiano le parti finali delle codifiche dei due individui



- crossover doppio:
  - si estraggono due posizioni a caso con probabilità uniforme
  - si spezza la codifica in tre parti in corrispondenza a tali posizioni
  - si scambiano le parti estreme delle codifiche dei due individui



Crossover

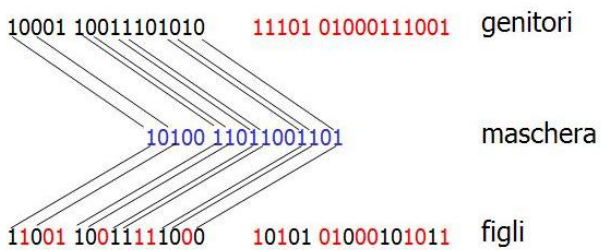
Generalizzando, si ottiene il

- crossover ad α punti:
  - si estraggono α posizione a caso con probabilità uniforme
  - si spezza la codifica in $\alpha + 1$ parti in corrispondenza a tale posizione
  - si scambiano le parti dispari delle codifiche dei due individui (prima, terza, ecc. . .)

Per valori di α bassi si ha polarizzazione posizionale (positional bias): simboli vicini nella codifica tendono a rimanere insieme

Per evitarlo completamente, si può adottare il

- crossover uniforme:
  - si costruisce un vettore binario casuale $m \in U(\mathbb{B}^n)$ ("maschera")
  - se $m_i = 1$ i simboli in posizione i dei due individui restano invariati, se $m_i = 0$ si scambiano fra loro



Crossover, Scatter Search e Path Relinking

L'operatore di crossover ha forti legami con la ricombinazione di *SS* e *PR*

Le principali differenze sono che

- ricombina i simboli delle codifiche, invece di
  - ricombinare le soluzioni (*SS*)
  - eseguire catene di scambi sulle soluzioni (*PR*)
- opera sull'intera popolazione, invece che solo sull'insieme R delle soluzioni di riferimento
- opera su coppie di individui casuali, invece che metodicamente su tutte le coppie di soluzioni di R
- produce una coppia di nuovi individui, invece di
  - generare una sola soluzione intermedia (*SS*)
  - visitare metodicamente le soluzioni intermedie e sceglierne una (*PR*)

Classificare un operatore sotto una denominazione piuttosto che un'altra è spesso una questione di gusto

Mutazione

L'operatore di mutazione modifica un individuo per generarne uno simile

- scorre la codifica ξ simbolo per simbolo
- decide con probabilità π_m se modificare il simbolo corrente

Il tipo di modifica dipende dalla codifica

- per codifiche binarie, la mutazione è un flip da ξ_i a $\xi_i' := 1 - \xi_1$

- per codifiche simboliche in B_c, spesso si estrae casualmente il nuovo simbolo ξ_i' da $B_c \setminus \{\xi_i\}$
- per le permutazioni di B, ci sono molte proposte
  - scambio di due elementi
  - inversione del tratto compreso fra due posizioni
  - ...

Mutazione e ricerca locale

L'operatore di mutazione ha forti legami con la ricerca locale

Le principali differenze sono che

- modifica i simboli delle codifiche, invece di eseguire scambi sulle soluzioni
- opera casualmente, invece che metodicamente
- opera su un numero casuale di simboli, invece che eseguire un solo scambio

Classificare un operatore sotto una denominazione piuttosto che un'altra è spesso una questione di gusto

Il problema dell'ammissibilità

Se la corrispondenza fra soluzioni e codifiche non è completa nei due sensi, crossover e mutazioni possono facilmente produrre codifiche che non corrispondono a soluzioni ammissibili

Questo produce diversi svantaggi:

- inefficienza, perché si perde tempo computazionale manipolando oggetti privi di significato
- inefficacia, perché è possibile che l'euristica non ottenga soluzioni
- difficoltà progettuali, perché bisogna definire la funzione fitness in modo ragionevole anche per sottoinsiemi non ammissibili

Ci sono tre approcci principali per contrastare questo problema

1. rappresentazioni e operatori speciali
2. procedure di riparazione
3. funzioni di penalità

Rappresentazioni e operatori speciali

Questi approcci si basano sull'idea di indagare

- rappresentazioni che producano solo o quasi codifiche ammissibili
  Ne sono esempi:
  - le rappresentazioni per problemi di partizione (*CMSTP*, *VRP*, ecc. . .)
    basate sul metodo order-first split-second
  - le rappresentazioni per i problemi di scheduling (*PMSP*,. . .)
    basate su euristiche costruttive alimentate da una permutazione
- operatori che conservano l'ammissibilità
  Esempio: gli operatori di crossover per il *TSP* (*PMX*)
  gli operatori di mutazione per il *TSP* che simulano k-scambi

Questi metodi

- abbandonano l'idea di astrarre il metodo dal problema specifico,
  che era fondamentale negli algoritmi genetici classici
- tendono ad avvicinarsi molto a metodi di scambio e ricombinazione
  basati sul concetto di intorno

Procedure di riparazione

Data una codifica ξ, corrispondente a un sottoinsieme x non ammissibile, la procedura di riparazione produce una soluzione x' ammissibile con cui eventualmente aggiorna la miglior soluzione nota x^*

Nella popolazione $X^{(g)}$, a seconda dei metodi, si può

- conservare la codifica ξ
- sostituirla con la codifica ξ' della soluzione riparata x'

Il secondo metodo produce sempre soluzioni ammissibili, ma ha lo svantaggio di introdurre

- una forte polarizzazione verso le codifiche ammissibili
- uno sbilanciamento verso le soluzioni ammissibili più facili da ottenere con la procedura di riparazione

Funzioni di penalità: misurare l'inammissibilità

Chiameremo per brevità codifiche ammissibili (inammissibili) le codifiche che corrispondono a soluzioni ammissibili (che non vi corrispondono)

Se esistono codifiche inammissibili, non si può definire la fitness partendo dal solo obiettivo, come nel caso $\varphi(\xi) = 1/f(x(\xi))$

Supponiamo che ogni codifica ξ corrisponda a un sottoinsieme $x \subseteq B$ e che esista una misura di inammissibilità $\psi(x)$ per ogni sottoinsieme, con

$$\begin{cases} \psi(x) = 0 \text{ se } x \in X \\ \psi(x) > 0 \text{ se } x \notin X \end{cases}$$

Se i vincoli del problema sono espressi da uguaglianze o disuguglianze, si può definire $\psi(x)$ come la somma pesata delle loro violazioni

Come si definiscono i pesi?
Sono fissi, variabili o adattivi?

Funzioni di penalità: combinare obiettivo e inammissibilità

La fitness $\varphi(\xi)$ è un'opportuna combinazione di

- funzione obiettivo $f(x(\xi))$
- misura di inammissibilità $\psi(x(\xi))$

Le estensioni più tipiche sono

- penalità assoluta: date due codifiche ξ e ξ'
  - se entrambe sono ammissibili, è meglio quella con f minore
  - se una sola è ammissibile, quella ammissibile è meglio dell'altra
  - se entrambe sono inammissibili, è meglio quella con ψ minore

- penalità legata al costo di riparazione: data una procedura che, applicata a x, produce una soluzione $x' = r(x)$ ammissibile

$$\varphi(\xi) = \frac{1}{f(x')} = \frac{1}{f(r(x(\xi)))}$$

cioè si usa il valore della funzione obiettivo nella soluzione "riparata"

Funzioni di penalità: includere l'inammissibilità nella fitness

- penalità uniforme: le codifiche inammissibili hanno una penalità fissa

$$\varphi(\xi) = \begin{cases} \dfrac{1}{f(x(\xi))} & \text{per le codifiche ammissibili} \\ \dfrac{1}{f(x(\xi)) + \alpha} & \text{per le codifiche inammissibili} \end{cases}$$

- penalità proporzionale: la penalità cresce con la violazione

$$\varphi(\xi) = \frac{1}{f(x(\xi)) + \alpha\psi(x(\xi))}$$

Funzioni di penalità: tarare la penalità

Sperimentalmente, conviene usare la penalità minima efficace

- se la penalità è troppo bassa, non si trovano soluzioni ammissibili
- se la penalità è troppo alta, si rimane confinati in una parte della regione ammissibile

Per trovare il valore corretto del parametro α che modula la penalità sono state proposti

- metodi dinamici: α cresce nel tempo
- metodi adattivi: α cresce se nella popolazione prevalgono le codifiche inammissibili, cala se prevalgono quelle ammissibili
- metodi evoluzionistici: ogni individuo codifica anche α, che viene selezionata e raffinata dall'euristica insieme alla soluzione

Algoritmi memetici

Gli algoritmi memetici (Moscato, 1989) si ispirano al concetto di meme (Dawkins, 1989) come unità base di informazione culturale riproducibile

A livello di espressione fenotipica, i geni vengono solo selezionati, mentre i memi possono anche cambiare, come nell'evoluzione lamarckiana

Fuori della metafora, gli algoritmi memetici combinano

- gli operatori "genotipici" che manipolano le codifiche (crossover e mutazione)
- gli operatori "fenotipici" che manipolano le soluzioni (ricerca locale)

Le soluzioni vengono migliorate con scambi prima di essere ricodificate

Una serie di parametri stabiliscono come applicare la ricerca locale

- ogni quante generazioni (sempre, o dopo sufficiente diversificazione)
- a quali individui (tutti, i migliori, i più diversi)
- per quanto tempo (fino al primo ottimo locale, oltre, oppure prima)
- con quale metodo (steepest descent, VNS, ILS, ecc. . .)

Strategie evoluzionistiche

Proposte da Rechenberg e Schwefel (1971), sono molto simili agli algoritmi genetici

Le principali differenze sono:

- le soluzioni sono codificate in vettori di numeri reali
- la popolazione è ridotta a un singolo individuo, che ne produce λ (*oggi si preferisce usarne un piccolo numero* μ)
- esistono due varianti fondamentali:
  - nella strategia $(1 + \lambda)$ i nuovi individui competono col progenitore e il migliore di tutti sopravvive
  - nella strategia $(1, \lambda)$ i nuovi individui competono e il migliore fra loro sostituisce il progenitore, anche se quello lo domina
- l'operatore di mutazione somma un disturbo casuale con distribuzione normale di media nulla: $\delta \in N(0, \sigma)$
- non c'è operatore di crossover (*ma oggi si preferisce usarlo*)