UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA

Laurea triennale in Informatica

Fondamenti di Intelligenza Artificiale

Lezione 7 - Algoritmi di ricerca locale (2)

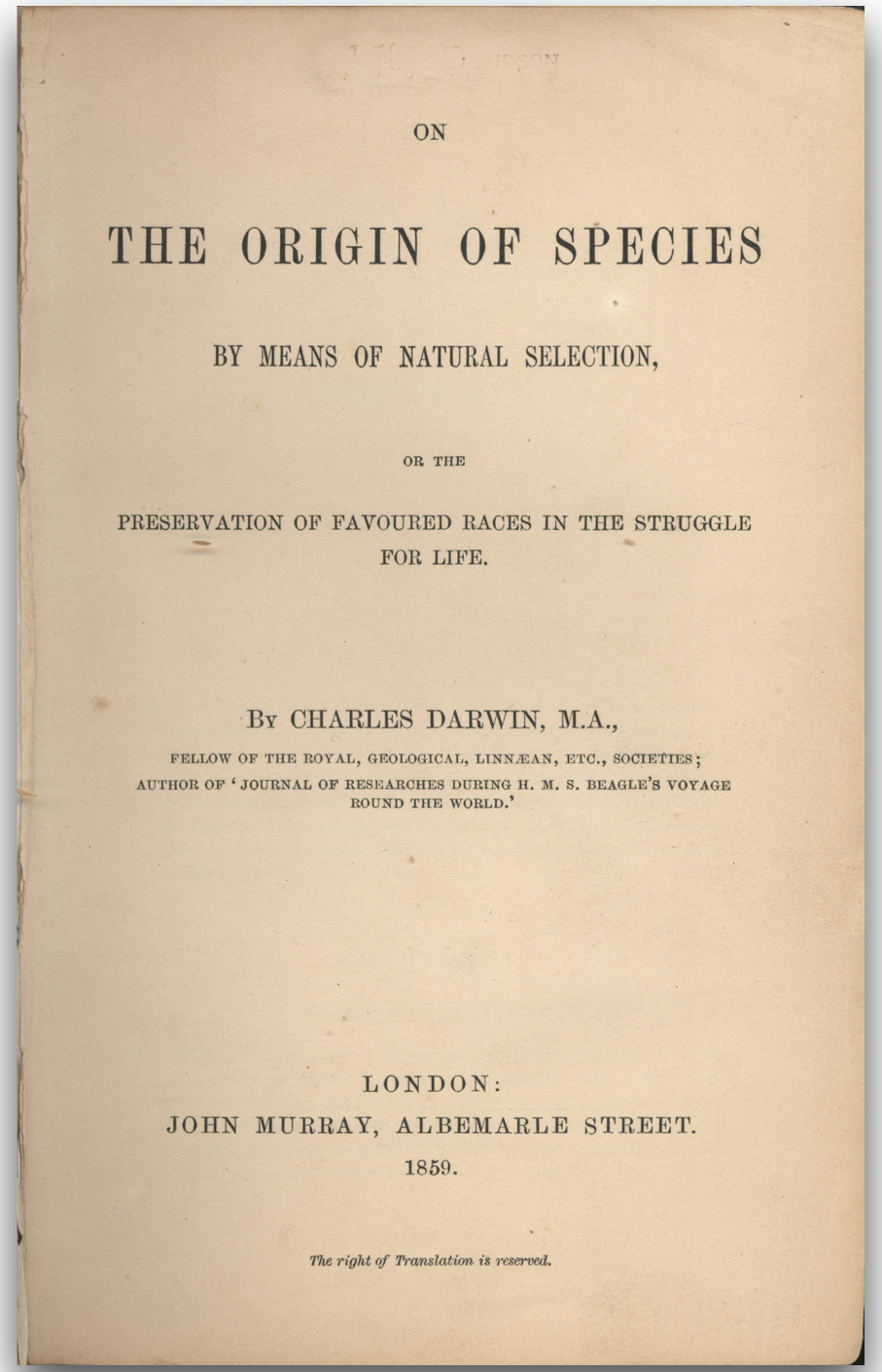
UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA

Laurea triennale in Informatica
Anno accademico 2020/2021

Link al PadLet di oggi… ma serve un* segretari* che mi aiuti!

Algoritmi di Ricerca Locale (2), Gli Algoritmi Genetici

Gli algoritmi genetici, un po' di storia



ON

THE ORIGIN OF SPECIES

BY MEANS OF NATURAL SELECTION,

OR THE

PRESERVATION OF FAVOURED RACES IN THE STRUGGLE FOR LIFE.

By CHARLES DARWIN, M.A.,

FELLOW OF THE ROYAL, GEOLOGICAL, LINNÆAN, ETC., SOCIETIES;
AUTHOR OF 'JOURNAL OF RESEARCHES DURING H. M. S. BEAGLE'S VOYAGE ROUND THE WORLD.'

LONDON:
JOHN MURRAY, ALBEMARLE STREET.
1859.

The right of Translation is reserved.

Charles Darwin ne l'*Origine della Specie* (1859), introduce il concetto di *teoria dell'evoluzione*, dove organismi della medesima specie **evolvono** tramite un processo di **selezione naturale**.

Senza entrare in dettagli che non ci competono la teoria di Darwin può essere riassunta in 4 principi:

(1) Gli individui con tratti ereditabili che **ben si adattano all'ambiente** tendono a sopravvivere;

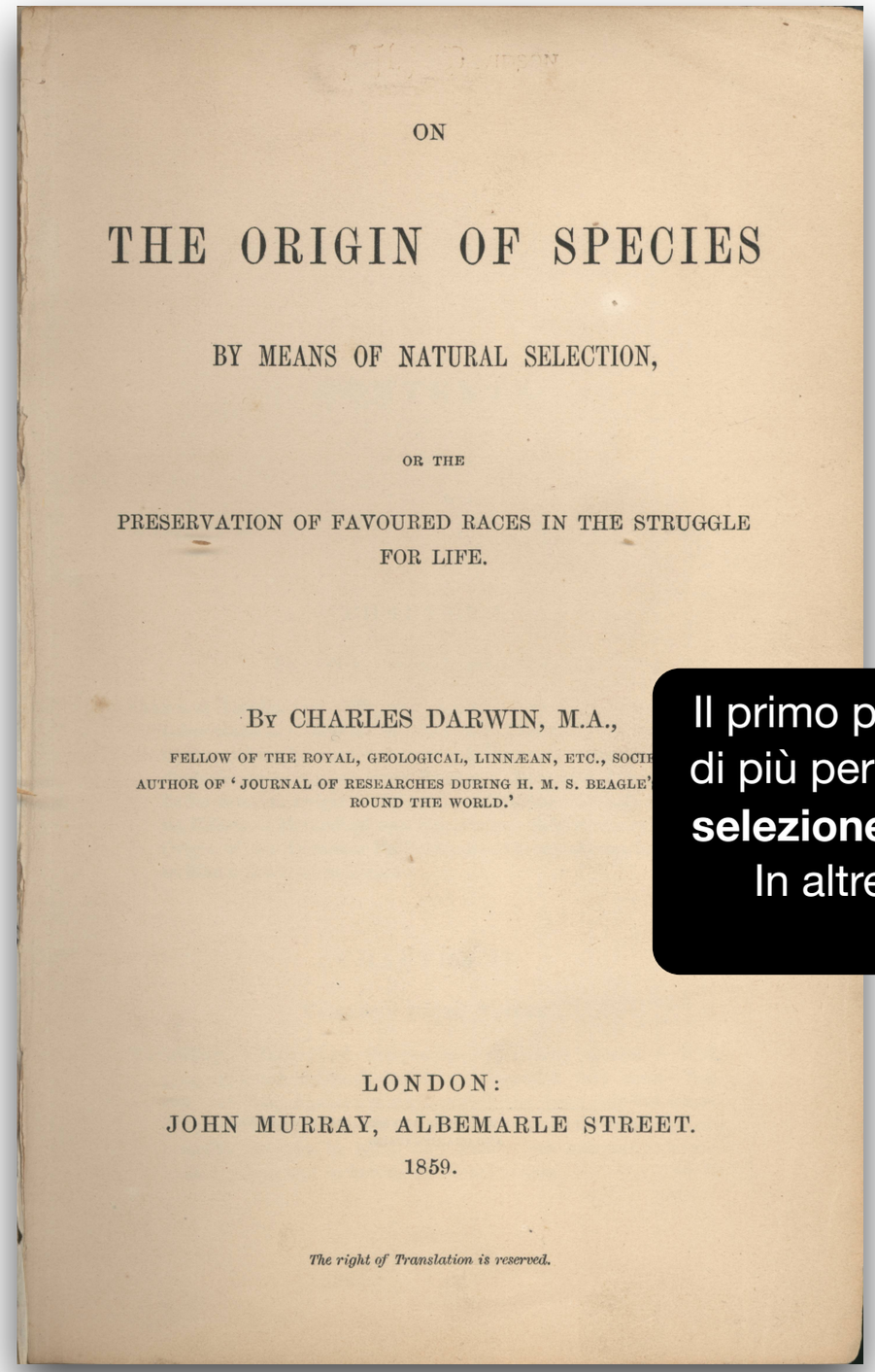
(2) Una **generazione** di individui che sopravvive tende a **produrre più individui**;

(3) Fra gli individui esiste una variazione di caratteristiche osservabili (fenotipo) che può essere passata dai genitori ai figli (*i.e.,* **ereditabile**);

(4) Se popolazioni della medesima specie smettono di interagire tra loro (*i.e.,* isolamento riproduttivo) allora avviene la **speciazione**: le popolazioni avranno accumulato così tante differenze da essere classificate come specie diverse.

Algoritmi di Ricerca Locale (2), Gli Algoritmi Genetici

Gli algoritmi genetici, un po' di storia



ON

THE ORIGIN OF SPECIES

BY MEANS OF NATURAL SELECTION,

OR THE

PRESERVATION OF FAVOURED RACES IN THE STRUGGLE FOR LIFE.

By CHARLES DARWIN, M.A.,

FELLOW OF THE ROYAL, GEOLOGICAL, LINNÆAN, ETC., SOCI

AUTHOR OF 'JOURNAL OF RESEARCHES DURING H. M. S. BEAGLE' ROUND THE WORLD.'

LONDON:

JOHN MURRAY, ALBEMARLE STREET.

1859.

The right of Translation is reserved.

Charles Darwin ne l'*Origine della Specie* (1859), introduce il concetto di *teoria dell'evoluzione*, dove organismi della medesima specie **evolvono** tramite un processo di **selezione naturale**.

Senza entrare in dettagli che non ci competono la teoria di Darwin può essere riassunta in 4 principi:

(1) Gli individui con tratti ereditabili che **ben si adattano all'ambiente** tendono a sopravvivere;

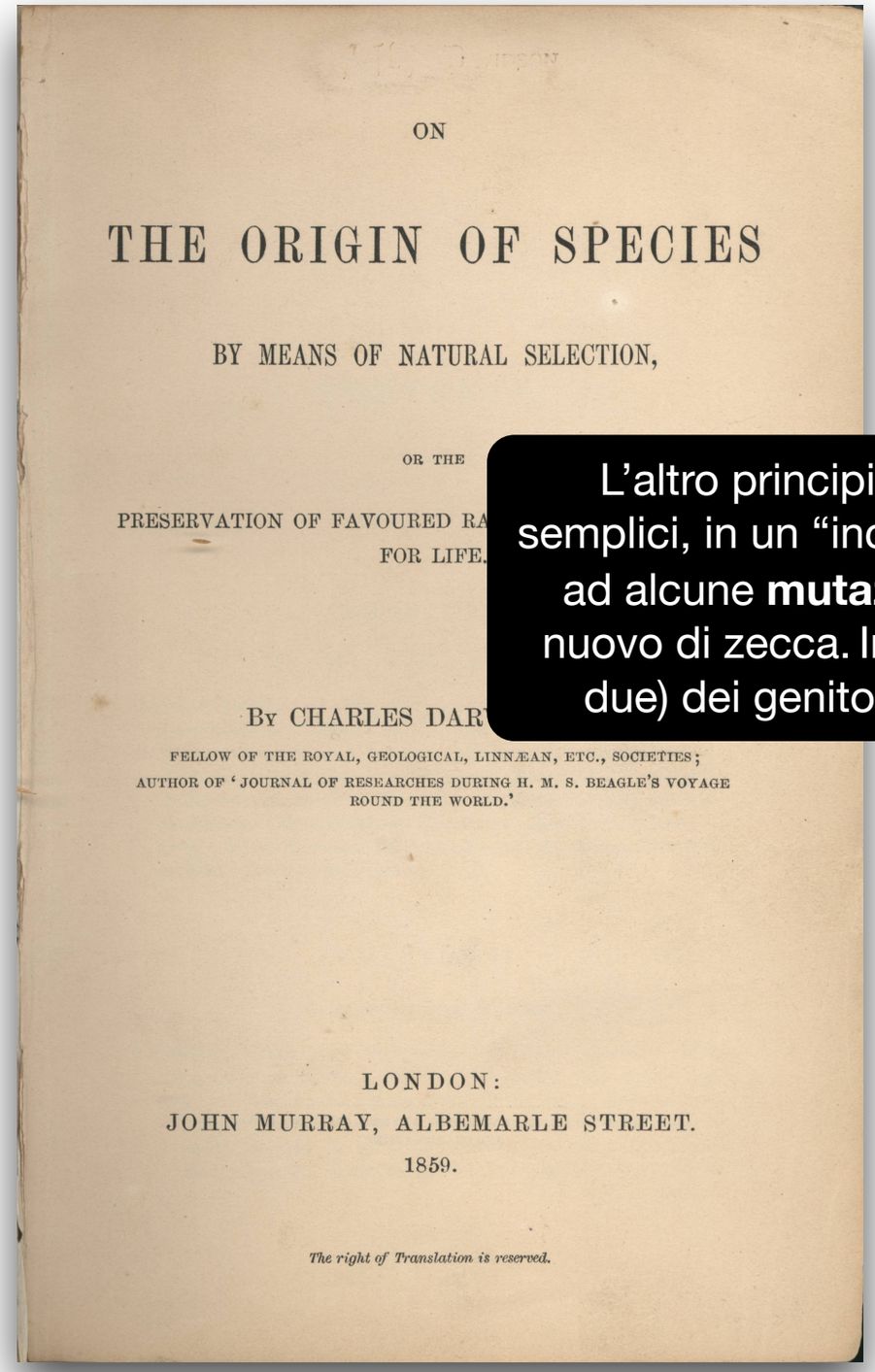
Il primo principio non solo è quello più importante ma è quello che ci interesserà di più per comprendere l'argomento. Esso rappresenta il principio alla base della **selezione naturale**, *a.k.a.,* sopravvivenza del più adatto (*"survival of the fittest"*). In altre parole: gli individui che meglio si adattano all'ambiente circostante sopravvivono e si riproducono, "portando avanti" la specie.

passata dai genitori ai figli (*i.e.,* **ereditabile**);

(4) Se popolazioni della medesima specie smettono di interagire tra loro (*i.e.,* isolamento riproduttivo) allora avviene la **speciazione**: le popolazioni avranno accumulato così tante differenze da essere classificate come specie diverse.

Algoritmi di Ricerca Locale (2), Gli Algoritmi Genetici

Gli algoritmi genetici, un po' di storia



ON

THE ORIGIN OF SPECIES

BY MEANS OF NATURAL SELECTION,

OR THE

PRESERVATION OF FAVOURED RA

FOR LIFE.

By CHARLES DAR

FELLOW OF THE ROYAL, GEOLOGICAL, LINNÆAN, ETC., SOCIETIES;
AUTHOR OF 'JOURNAL OF RESEARCHES DURING H. M. S. BEAGLE'S VOYAGE ROUND THE WORLD.'

LONDON:
JOHN MURRAY, ALBEMARLE STREET.
1859.

The right of Translation is reserved.

Charles Darwin ne l'*Origine della Specie* (1859), introduce il concetto di *teoria dell'evoluzione*, dove organismi della medesima specie **evolvono** tramite un processo di **selezione naturale**.

Senza entrare in dettagli che non ci competono la teoria di Darwin può essere riassunta in 4 principi:

L'altro principio che ci interessa molto è il terzo. La riproduzione consiste, in termini semplici, in un "incrocio" del materiale genetico (*i.e.,* genotipo) dei genitori, che, in aggiunta ad alcune **mutazioni**, produrrà nuovi individui della medesima specie, con un genotipo nuovo di zecca. In altre parole: un figlio possiede i tratti migliori (o peggiori o un misto dei due) dei genitori, nella speranza che esso sopravviva di più e "porti avanti" la specie.

(1) Ogni individuo tende a **produrre più individui**;

(3) Fra gli individui esiste una variazione di caratteristiche osservabili (fenotipo) che può essere passata dai genitori ai figli (*i.e.,* **ereditabile**);

(4) Se popolazioni della medesima specie smettono di interagire tra loro (*i.e.,* isolamento riproduttivo) allora avviene la **speciazione**: le popolazioni avranno accumulato così tante differenze da essere classificate come specie diverse.

Algoritmi di Ricerca Locale (2), Gli Algoritmi Genetici

Gli algoritmi genetici, verso il concetto di “ottimizzazione”

Il concetto chiave è chiaramente quello di *miglioramento naturale* di una specie. La parola miglioramento è parente stretta alla parola *ottimizzazione*.

> **Ottimizzazione.** Un tipo di ricerca volto a trovare un punto di ottimo (secondo un obiettivo ben specificato, e.g., una funzione) tra le diverse alternative (i.e., all’interno di un dominio ben specificato).

Sulla base del lavoro di Darwin, negli anni ’50 è iniziata la diffusione dei cosiddetti *algoritmi evolutivi*, cioè algoritmi di ricerca/ottimizzazione ispirati proprio dal meccanismo dell’evoluzione Darwiniana, che quindi usano tutti i concetti di cui abbiamo parlato: *popolazioni*, *individui*, *evoluzione*, *genotipo* e *fenotipo*, ecc.

E’ importante notare che il termine “algoritmi evolutivi” (EA) è un ombrello che comprende molteplici algoritmi, tra cui, l’oggetto della nostra lezione.

Gli algoritmi evolutivi appartengono alla categoria degli algoritmi *nature-inspired*, ovvero algoritmi ispirati dai processi della natura. Per conoscenza, vale anche la pena menzionare l’esistenza degli algoritmi di *swarm intelligence*, ovvero algoritmi che simulano il comportamento collettivo, decentralizzato e auto-organizzante.

Quando si parla di *Ant Colony Optimization* o *Particle Swarm Optimization*, si parla di algoritmi ispirati alle colonie di formiche o agli stormi di uccelli/banchi di pesci.

Algoritmi di Ricerca Locale (2), Gli Algoritmi Genetici

Gli algoritmi genetici, verso il concetto di “ottimizzazione”

Il concetto chiave è chiaramente quello di *miglioramento naturale* di una specie. La parola miglioramento è parente stretta alla parola *ottimizzazione*.

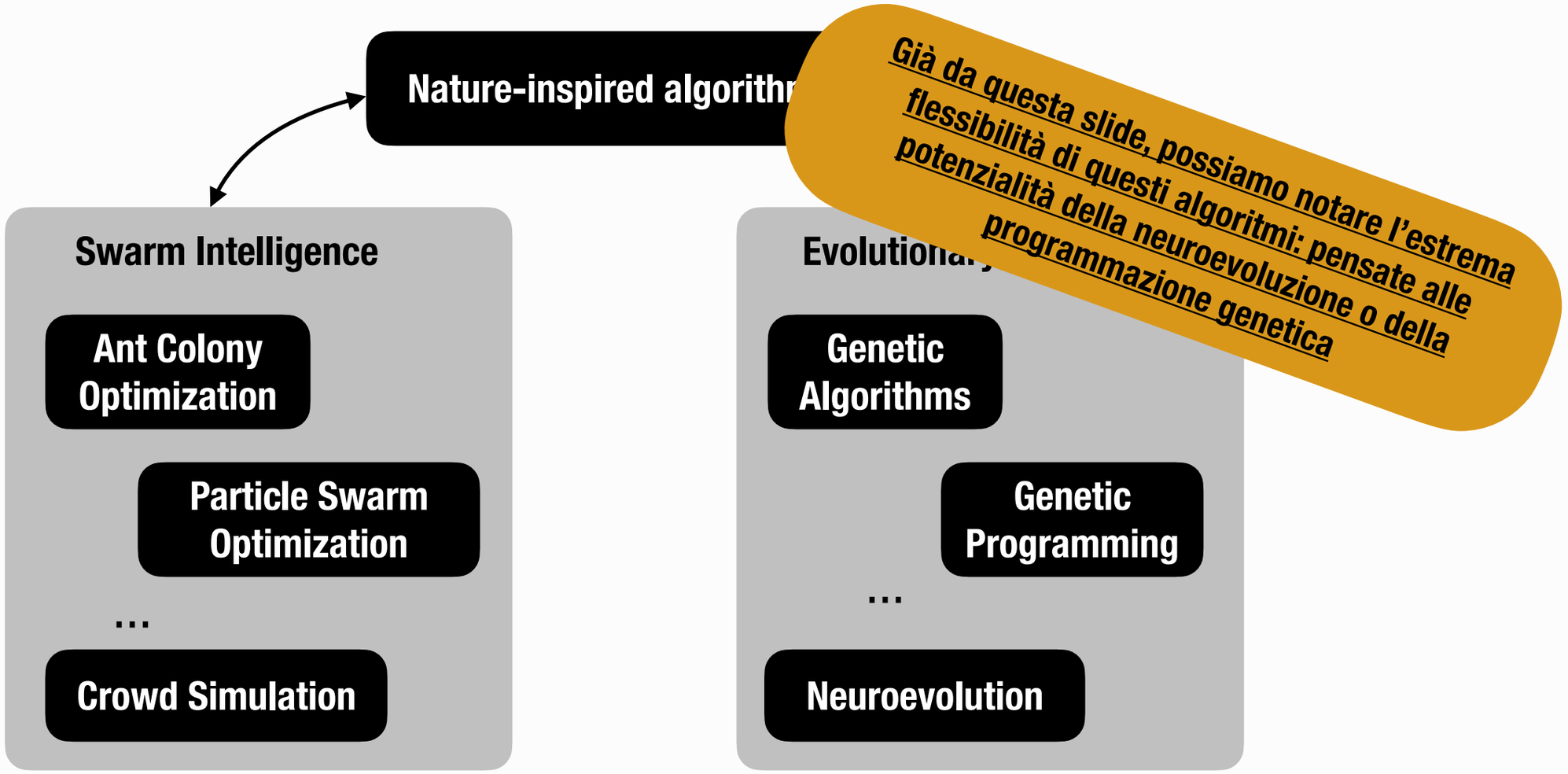
> **Ottimizzazione.** Un tipo di ricerca volto a trovare un punto di ottimo (secondo un obiettivo ben specificato, e.g., una funzione) tra le diverse alternative (i.e., all’interno di un dominio ben specificato).

cosiddetti
dal
i di cui
ecc.

o che

nspired,
anche la pena
ritmi che
te.

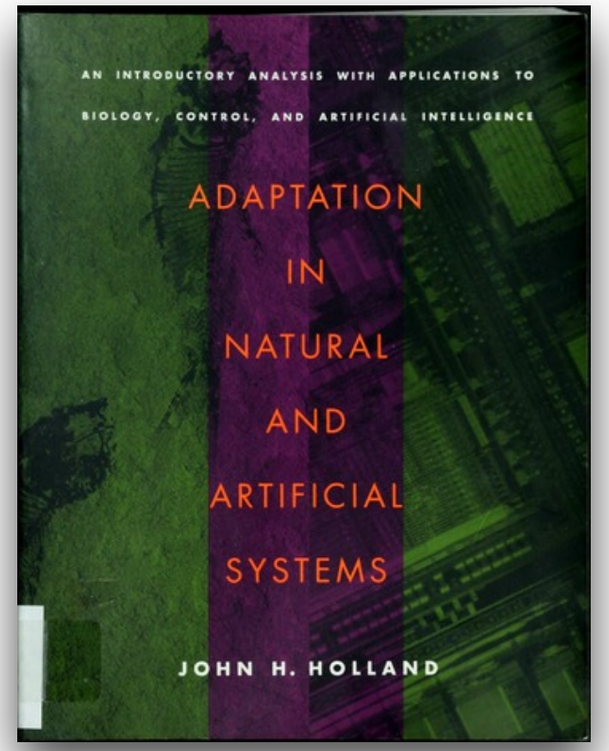
on, si parla di
i pesci.



Algoritmi di Ricerca Locale (2), Gli Algoritmi Genetici

Gli algoritmi genetici, definizione

Nel 1975, John Holland presenta i cosiddetti *genetic plans*, quelli che poi si sarebbero chiamati algoritmi genetici già dopo un paio di anni.



Algoritmi Genetici (GA). Procedura di alto livello (meta-euristica) ispirata alla genetica per **definire** un algoritmo di ricerca.

Facciamo subito attenzione: GA NON è il nome di un algoritmo di ricerca, ma è una procedura che *ci consente di definire* algoritmi di ricerca. Per questo si usa spesso il plurale: ci riferiamo a qualsiasi algoritmo che segue le regole dei GA.

Facciamo attenzione anche all'uso del termine "meta-euristica": questo denota l'uso di un approccio euristico di tipo generale.

Quindi, analizzeremo un approccio alla creazione di algoritmi.

Da queste osservazioni possiamo subito dare alcune proprietà dei GA:

(1) **I GA non sono problem-specific**. Non risolvono una singola classe di problemi di ricerca, ma molteplici, anche di natura molto differente (dai giochi al testing).

(2) **I GA sono flessibili**. E' semplice modificarli, adattarli e creare varianti.

(3) **I GA non garantiscono l'ottimalità**. Di norma, producono soluzioni sub-ottimali.

Algoritmi di Ricerca Locale (2), Gli Algoritmi Genetici

Gli algoritmi genetici, definizione

Algoritmi Genetici (GA). Procedura di alto livello (meta-euristica) ispirata alla genetica per **definire** un algoritmo di ricerca.

Elaborando sulla definizione di GA, possiamo dire che:

Evolve una **popolazione** di **individui** (soluzioni candidate) producendo di volta in volta soluzioni sempre migliori rispetto ad una **funzione obiettivo**, fino a raggiungere l'**ottimo** o un'altra **condizione di terminazione**. La creazione di nuove **generazioni** di individui avviene applicando degli **operatori genetici**, precisamente **selezione**, **crossover** e **mutazione**.

I quattro aspetti caratterizzanti di un GA sono:

(1) **Codifica** gli individui come stringhe di lunghezza finita.

(2) E' **population-based**: non fa evolvere una singola soluzione, ma un insieme di esse contemporaneamente, passo passo. Questo li contrappone ai tradizionali algoritmi di *ricerca locale*.

(3) E' **cieco**: La funzione obiettivo non necessita di informazioni di basso livello del problema (*i.e.,* problem-specific) che si sta risolvendo.

(4) Ha elementi di **casualità** che guidano la ricerca, pur non arrivando ai livelli di una random search.

Algoritmi di Ricerca Locale (2), Gli Algoritmi Genetici

Gli algoritmi genetici, definizione

Algoritmi Genetici (GA). Procedura di alto livello (meta-euristica) ispirata alla genetica per **definire** un algoritmo di ricerca.

Elaborando sulla definizione di GA, possiamo dire che:

Evolve una **popolazione** di **individui** (soluzioni candid… …ndo di volta in volta soluzioni sempre migliori rispetto ad una **funzione obi**… …ungere l'**ottimo** o un'altra **condizione di terminazione**. La creazi… …**oni** di individui avviene applicando degli **operatori genetici**, precisame… **cro**…

Come vedremo, quindi, gli algoritmi genetici consentono molteplici manipolazioni che supportano la risoluzione di numerosi problemi

NB: Questo è vero da un punto di vista teorico, ma da quello pratico, per ragioni di efficienza e semplicità, qualche volta si codifica in altri modi.

I qu… …

(1) **Codifica** gli individui come stringhe di lunghezza finita.

(2) E' **population-based**: non fa evolvere una singola soluzione, ma un insieme di esse contemporaneamente, passo… …ai tradizionali algoritmi di *ricerca locale*.

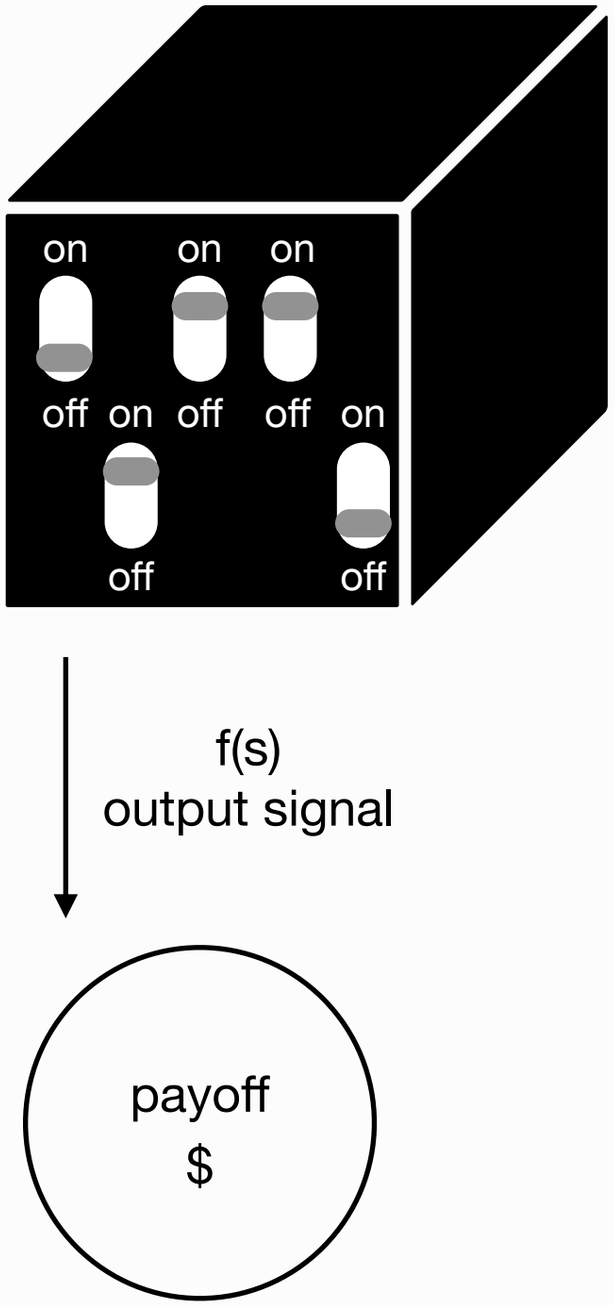
Talvolta, però, qualche informazione di basso livello può aiutare la ricerca

(3) E' **cieco**: La funzione obiettivo non necessita di informazioni di basso livello del problema (*i.e.,* problem-specific) che si sta risolvendo.

(4) Ha elementi di **casualità** che guidano la ricerca, pur non arrivando ai livelli di una random search.

Gli algoritmi genetici, un esempio

Per capire meglio di cosa stiamo parlando, iniziamo da un esempio giocattolo. Consideriamo una scatola nera con cinque interruttori.



Non sappiamo nulla sull'interno della scatola (è nera!), soltanto che può essere azionata utilizzando le levette on-off di ciascun interruttore.

A seconda della configurazione on-off degli interruttori, la scatola produrrà un segnale di output e, quindi, agirà in maniera diversa.

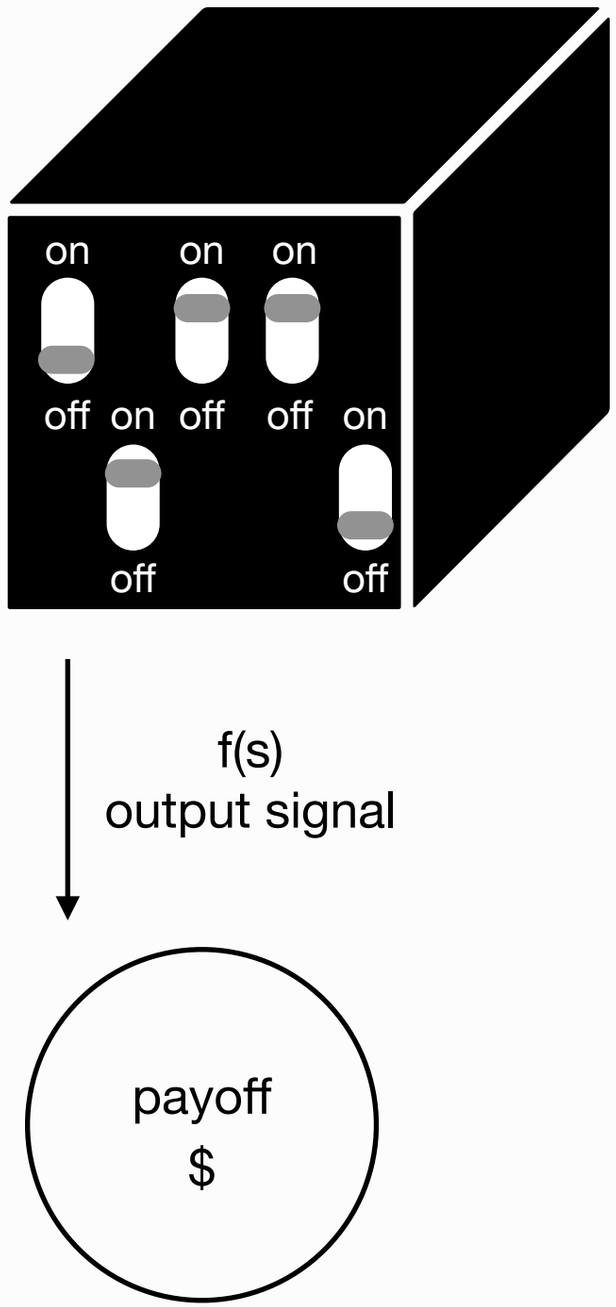
Il comportamento della scatola è modellato dalla funzione *f(s)*, dove *s* è una configurazione degli cinque interruttori. Non sappiamo null'altro di *f(s)*.

Il nostro obiettivo è quello di produrre il valore più alto possibile della funzione *f(s)*. In altri termini, vogliamo ottenere una configurazione delle levette tale per cui il valore di payoff risultante sarà *massimizzato*.

Il primo scoglio per la risoluzione del problema - e quello più importante in assoluto - è trovare una codifica per gli individui del problema. Questa indicherà le soluzioni candidate. Idee?

Gli algoritmi genetici, un esempio

Come detto in precedenza, gli individui sono delle stringhe di lunghezza finita. Di conseguenza, possiamo utilizzare una stringa.



Molto banalmente, avendo 5 interruttori on-off, è ragionevole modellare una possibile soluzione come una stringa binaria lunga cinque bit.

Stando a ciò che vediamo in figura, una soluzione candidata potrebbe essere la stringa 01110.

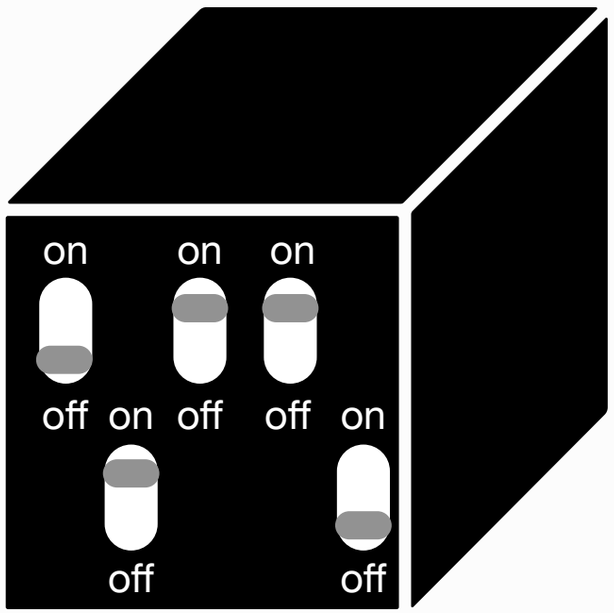
Se usassimo un algoritmo di ricerca locale poco robusto, potremmo partire da questo 01110 e “salteremo” di soluzione in soluzione fino ad un ottimo locale, portando l’algoritmo a terminare e restituire la soluzione pensando sia l’ottimo globale di *f(s)* quando non è detto che sia così, visto che non conosciamo la natura di *f(s)*.

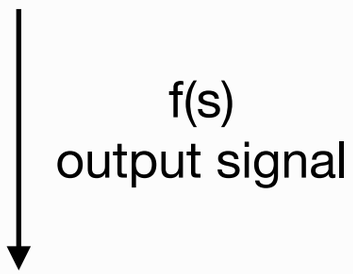
In scenari black-box in cui abbiamo pochissimi dettagli sulla funzione obiettivo, un GA è l’ideale. Vedremo che spesso rappresentano una scelta ideale anche in scenari white-box.

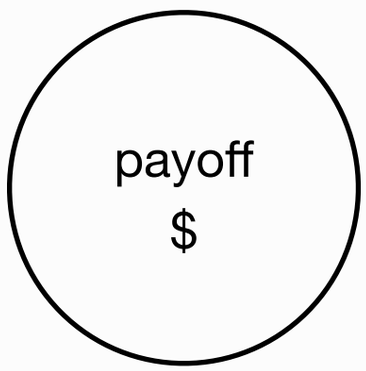
Proviamo quindi ad usare un GA.

Gli algoritmi genetici, un esempio

Il primo passo sarà quello di inizializzare una popolazione di individui, i quali formeranno la prima generazione.







Ricordate infatti che i GA sono population-based e, quindi, richiedono la definizione di una popolazione di individui di partenza. Supponendo che la taglia della popolazione sia fissata a quattro individui, potremmo ottenere, in modo pseudo-casuale, i seguenti individui:

01101
11000
01000
10011

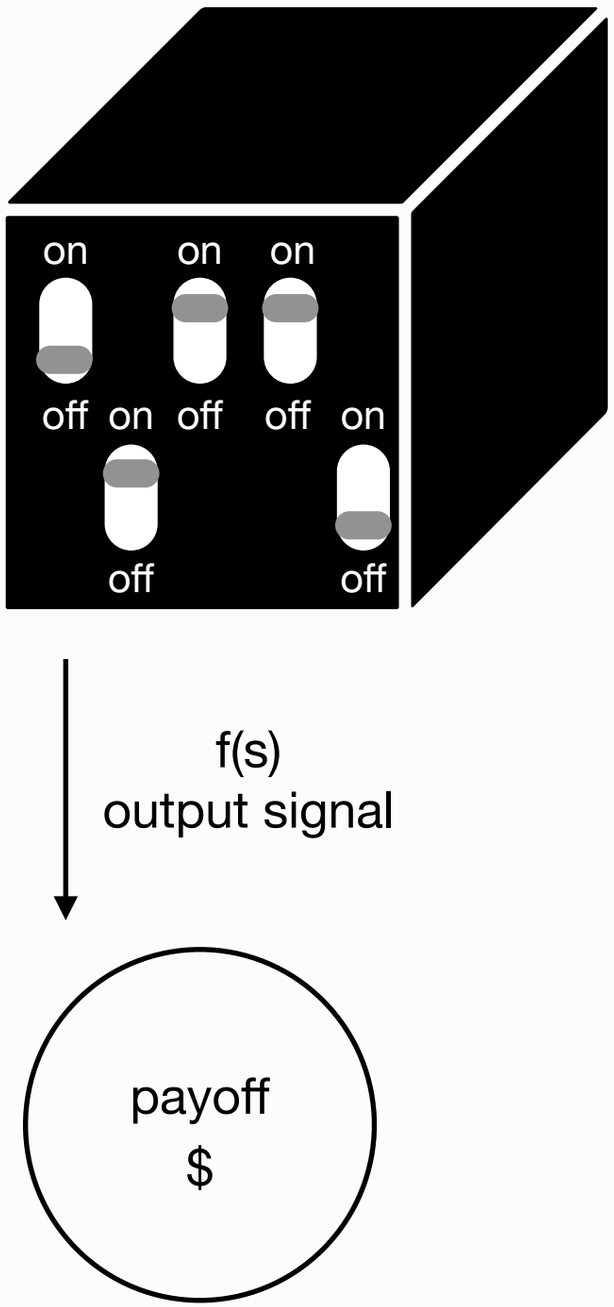
Queste configurazioni assumono significato dal momento in cui vengono date in input ad una *funzione di fitness*, che misura il grado di conformità di una soluzione rispetto al problema.

Approfondiremo il discorso sulla funzione di fitness ma, intanto, nel nostro caso, possiamo considerare la funzione obiettivo *f(s)*, senza alcuna modifica particolare, come funzione di riferimento.

Algoritmi di Ricerca Locale (2), Gli Algoritmi Genetici

Gli algoritmi genetici, un esempio

Il primo passo sarà quello di inizializzare una popolazione di individui, i quali formeranno la prima generazione.



Ricordate infatti che i GA sono population-based e, quindi, richiedono la definizione di una popolazione di individui di partenza. Supponendo che la taglia della popolazione sia fissata a quattro individui, potremmo ottenere, in modo pseudo-casuale, i seguenti individui:

01101 -> f(s) -> 169
11000 -> f(s) -> 576
01000 -> f(s) -> 64
10011 -> f(s) -> 316

Il miglior individuo di questa popolazione è quindi 11000.

Tuttavia, l'algoritmo non può affatto sapere se 11000 sia il punto di ottimo globale o meno, anzi, probabilmente ci sono punti nettamente migliori. Non possiamo, quindi, *fermarci adesso*, dobbiamo continuare.

Come? Evolvendo, ovvero creando una seconda generazione di individui, a parte da questa.

Gli algoritmi genetici, un esempio

Evolvere una seconda generazione di individui. Rispetto agli altri algoritmi di ricerca locale, i GA si distinguono per l'applicazione in sequenza di tre operatori genetici.

Selezione. Una copia di alcuni individui nella successiva generazione.

Vogliamo soluzioni sempre migliori, per cui ha *quasi sempre* senso implementare l'idea del *"il più forte sopravvive"*. Di conseguenza, la probabilità di sopravvivenza di un individuo dipende dal valore di fitness calcolato in precedenza.

Crossover. Accoppiamento di individui (*parents*) per crearne di nuovi (*offsprings*), da aggiungere alla nuova generazione.

Come succede a noi umani, gli individui di una popolazione di un GA possono accoppiarsi, incrociando il loro corredo genetico (*i.e.,* la codifica) per formare i figli.

Mutazione. Una modifica casuale di alcune parti del corredo genetico.

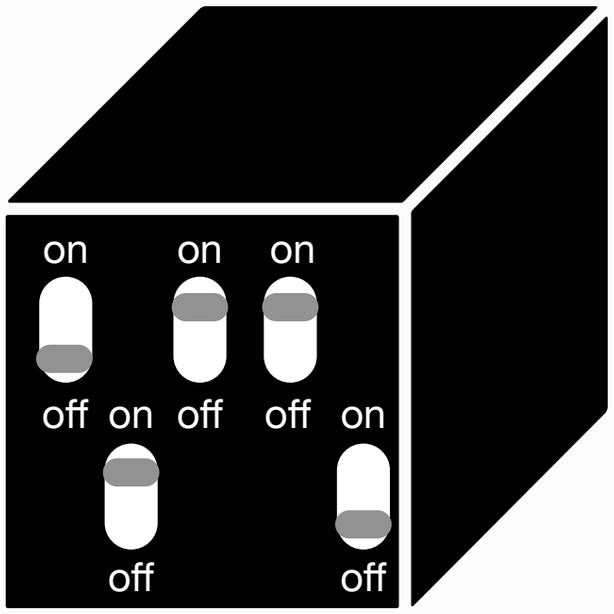
Selezione e crossover tendono a favorire delle caratteristiche non sempre ottimali: anche in natura, non sempre l'unione di due individui "buoni" porta ad un individuo "buono". La mutazione serve a "reintrodurre" ciò che è stato perduto ed *esplorare* meglio lo spazio di ricerca.

Vediamo come usare i tre operatori nell'esempio giocattolo.

Algoritmi di Ricerca Locale (2), Gli Algoritmi Genetici

Gli algoritmi genetici, un esempio

Iniziamo dalla selezione. Eravamo rimasti con la soluzione 11000 identificata come la migliore della prima iterazione dell'algoritmo.

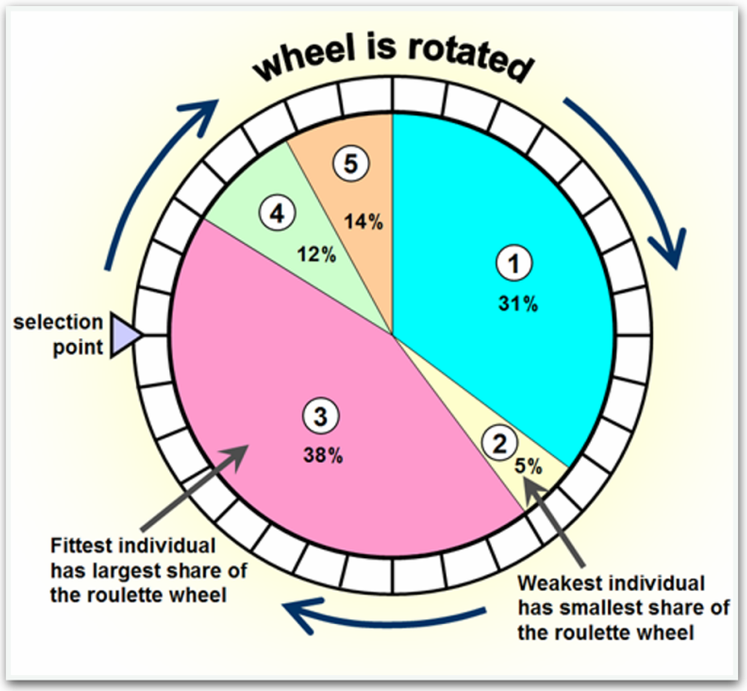


f(s)
output signal

payoff
$

01101 -> f(s) -> 169 (15%)
11000 -> f(s) -> 576 (51%)
01000 -> f(s) -> 64 (6%)
10011 -> f(s) -> 316 (28%)

Per procedere con la selezione, utilizzeremo l'opzione chiamata *Roulette Wheel Selection*.



Roulette Wheel perché è proprio come giocare in un casinò!

Assegnamo una porzione di una ruota a ciascun individuo a seconda della sua fitness relativa.

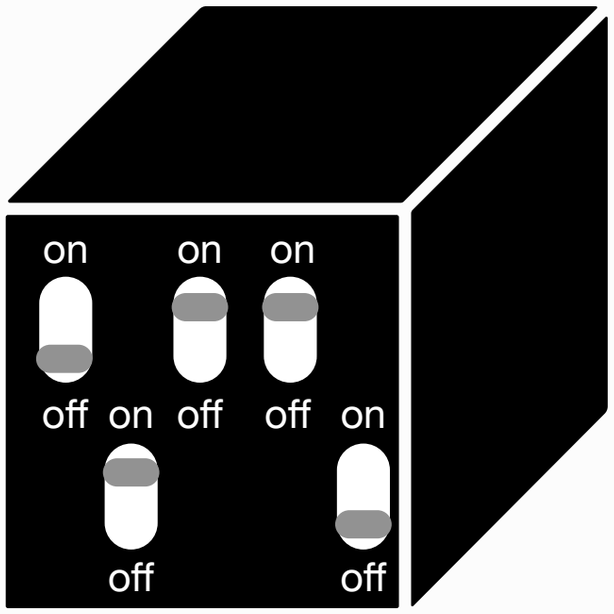
Giriamo la ruota un numero di volte pari al numero di individui (4 volte), e i vincitori saranno ammessi alla riproduzione.

L'immagine non è legata all'esempio, è solo ai fini dimostrativi.

Algoritmi di Ricerca Locale (2), Gli Algoritmi Genetici

Gli algoritmi genetici, un esempio

Iniziamo dalla selezione. Eravamo rimasti con la soluzione 11000 identificata come la migliore della prima iterazione dell'algoritmo.

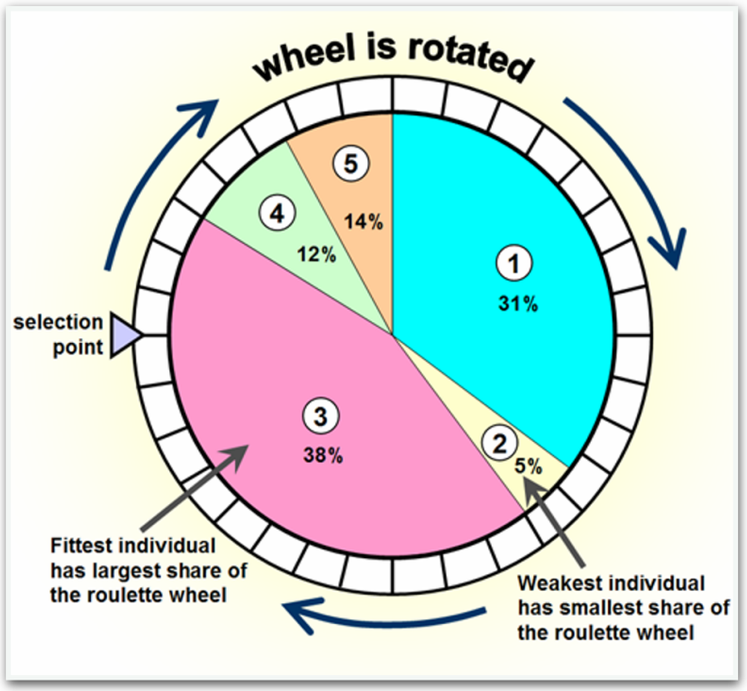


f(s)
output signal

payoff
$

01101 -> f(s) -> 169 (15%)
11000 -> f(s) -> 576 (51%)
01000 -> f(s) -> 64 (6%)
10011 -> f(s) -> 316 (28%)

Per procedere con la selezione, utilizzeremo l'opzione chiamata *Roulette Wheel Selection*.



Roulette Wheel perché è proprio come giocare in un casinò!

Assegnamo una porzione di una ruota a ciascun individuo a
volte pari al numero di individui (4 volte), e i vincitori saranno ammessi alla riproduzione.

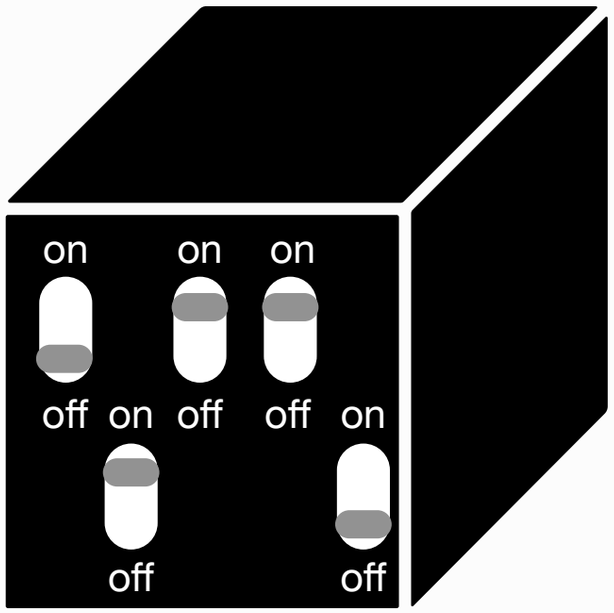
NB: un individuo può vincere più volte! Essere ammessi alla riproduzione consiste nell'entrare nel **mating pool.**

L'immagine non è legata all'esempio, è solo ai fini dimostrativi.

Algoritmi di Ricerca Locale (2), Gli Algoritmi Genetici

Gli algoritmi genetici, un esempio

Iniziamo dalla selezione. Eravamo rimasti con la soluzione 11000 identificata come la migliore della prima iterazione dell'algoritmo.



f(s)
output signal

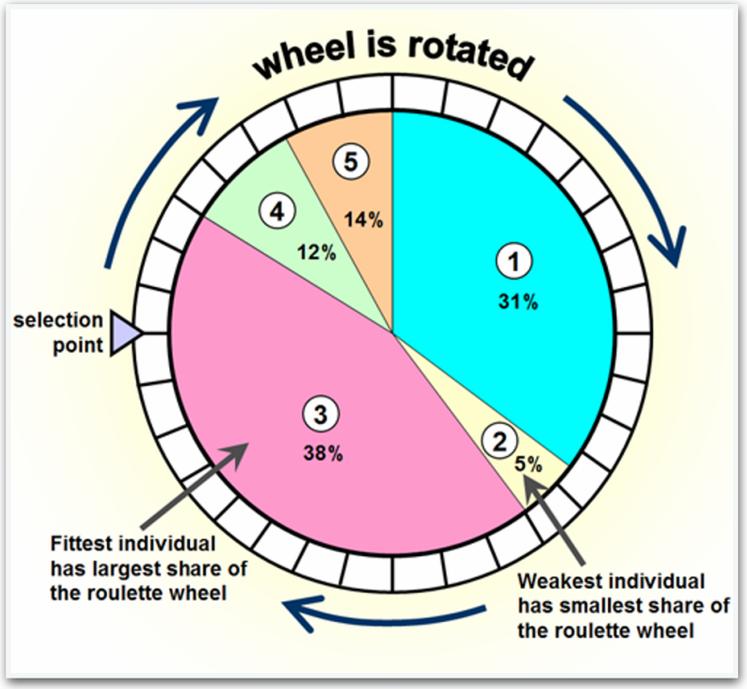
payoff
$

01101 -> f(s) -> 169 (15%)	11000
11000 -> f(s) -> 576 (51%)	11000
01000 -> f(s) -> 64 (6%)	10011
10011 -> f(s) -> 316 (28%)	01000

Supponiamo che i vincitori siano questi individui.

11000 ha vinto due volte, mentre 01101 è fuori.

Per procedere con la selezione, utilizzeremo l'opzione chiamata *Roulette Wheel Selection*.



Roulette Wheel perché è proprio come giocare in un casinò!

Assegnamo una porzione di una ruota a ciascun individuo a[illegible]a.

NB: un individuo può vincere più volte! Essere ammessi alla riproduzione consiste nell'entrare nel **mating pool.**

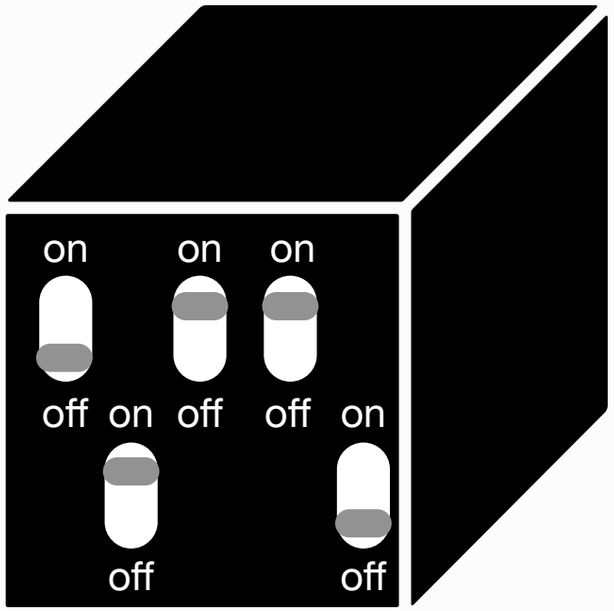
volte pari al numero di individui (4 volte), e i vincitori saranno ammessi alla riproduzione.

L'immagine non è legata all'esempio, è solo ai fini dimostrativi.

Algoritmi di Ricerca Locale (2), Gli Algoritmi Genetici

Gli algoritmi genetici, un esempio

Una volta selezionati gli individui nel mating pool, passiamo alla fase di riproduzione, in cui applicheremo gli operatori di crossover e mutazione.

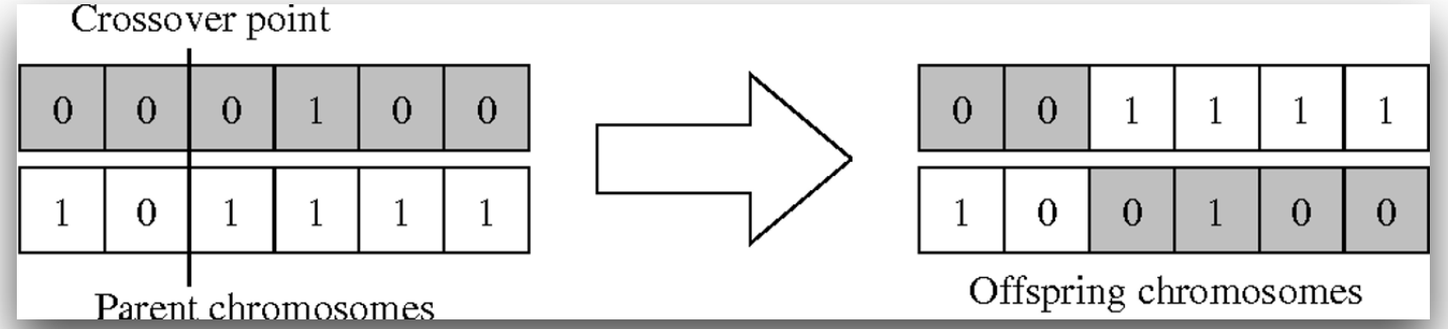


f(s)
output signal

payoff
$

Nel contesto dell'esempio, consideriamo un particolare tipo di crossover chiamato il *single-point crossover*.

Gli individui si accoppiamo in maniera *casuale* - tipicamente senza considerarli più di una volta. Dopodiché, si sceglie un *punto di taglio* (pipe) casuale nella codifica degli individui e si *incrociano* gli individui.



L'immagine non è legata all'esempio, è solo ai fini dimostrativi.

Supponiamo i seguenti abbinamenti, con punto di taglio tra il secondo ed il terzo bit.

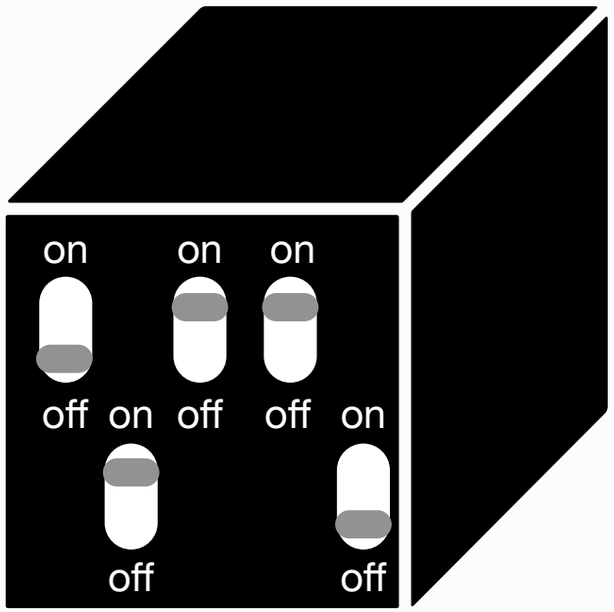
11|000 ♥ 01|000 —> 11000 e 01000 — figli identici ai genitori

11|000 ♥ 10|011 —> 11011 e 10000 — figli diversi dai genitori

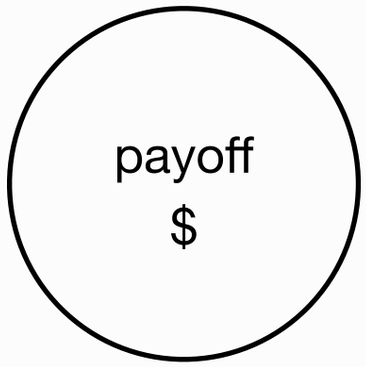
Algoritmi di Ricerca Locale (2), Gli Algoritmi Genetici

Gli algoritmi genetici, un esempio

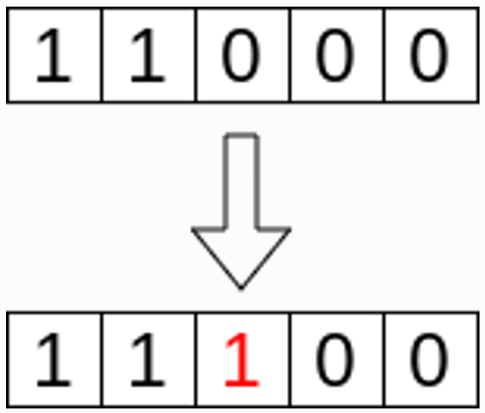
L'operazione di crossover consente di far evolvere una popolazione. In maniera simile a ciò che succede in natura, diamo spazio ai giovani.



f(s)
output signal



Ma non abbiamo ancora finito. L'ultimo passo sarà quello della mutazione. Anche in questo caso, sceglieremo una strategia *single-point*.



Scegliamo casualmente alcuni individui, scegliamo un gene casuale e lo mutiamo.

Nel caso binario spesso si muta con un *bit-flip*, ma in casi con geni interi/caratteri allora abbiamo maggiore libertà di scelta.

L'immagine non è legata all'esempio, è solo ai fini dimostrativi.

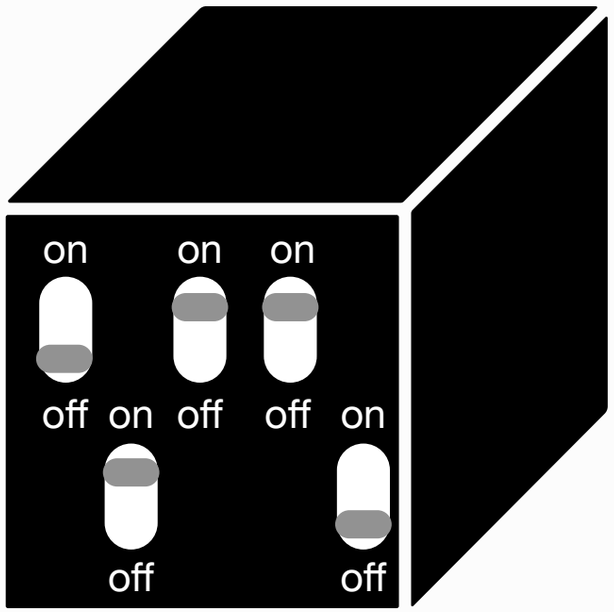
Nel nostro esempio, quindi, avverrà qualcosa del genere:

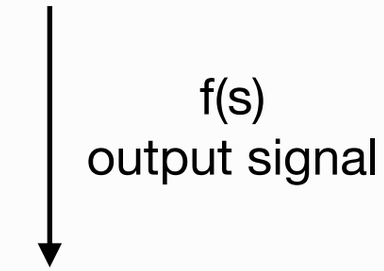
11000 —> 11100
01000
11011
10000

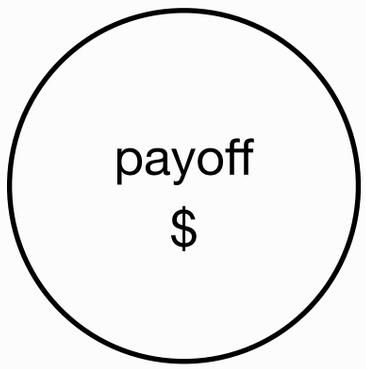
Vale la pena notare che, nel caso specifico, abbiamo deciso di scegliere un solo individuo e mutare un solo gene, tuttavia avremmo potuto optare per una più massiva opera di mutazione, la quale avrebbe potuto portare maggiore diversità nella popolazione… vedremo le implicazioni di una scelta del genere tra poco.

Gli algoritmi genetici, un esempio

Abbiamo concluso con l'applicazione degli operatori genetici. Possiamo a questo punto valutare la generazione corrente sulla base della funzione di fitness.







11100 -> f(s) -> 794
01000 -> f(s) -> 64
11011 -> f(s) -> 729
10000 -> f(s) -> 256

Le differenze con la generazione precedente ci sono: adesso abbiamo due individui molto buoni, uno mediocre ed uno pessimo.

Avendo ottenuto **almeno un individuo con fitness migliore dell'ex migliore individuo della generazione precedente** (576) l'algoritmo, nella sua interezza, ha fatto progresso, quindi abbiamo ragione di continuare ancora.

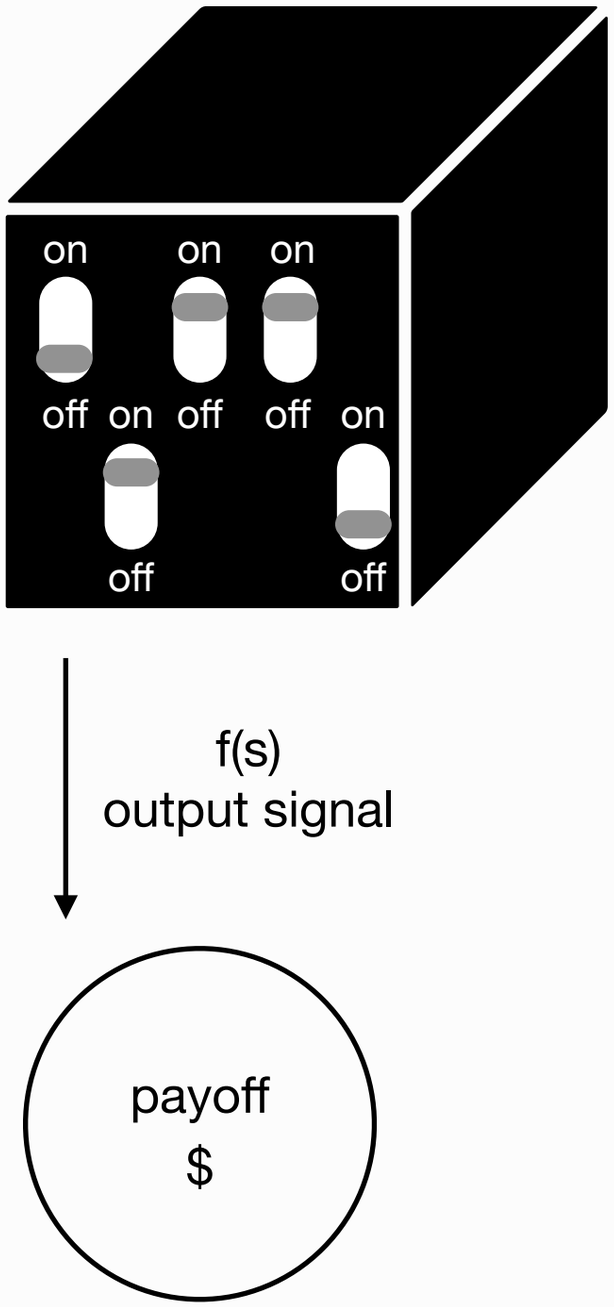
Reiteriamo quindi il processo. Omettendo i dettagli, otteniamo:

11000 -> f(s) -> 576	01100 -> f(s) -> 144
11111 -> f(s) -> 961	11011 -> f(s) -> 729
11110 -> f(s) -> 900	11000 -> f(s) -> 576
11100 -> f(s) -> 784	10110 -> f(s) -> 484

Algoritmi di Ricerca Locale (2), Gli Algoritmi Genetici

Gli algoritmi genetici, un esempio

Abbiamo concluso con l'applicazione degli operatori genetici. Possiamo a questo punto valutare la generazione corrente sulla base della funzione di fitness.



11100 -> f(s) -> 794
01000 -> f(s) -> 64
11011 -> f(s) -> 729
10000 -> f(s) -> 256

Le differenze con la generazione precedente ci sono: adesso abbiamo due individui molto buoni, uno mediocre ed uno pessimo.

Avendo ottenuto **almeno un individuo con fitness migliore dell'ex migliore individuo della generazione precedente** (576) l'algoritmo, nella sua interezza, ha fatto progresso, quindi abbiamo ragione di continuare ancora.

Reiteriamo quindi il processo. Omettendo i dettagli, otteniamo:

11000 -> f(s) -> 576
11111 -> f(s) -> 961
11110 -> f(s) -> 900
11100 -> f(

01100 -> f(s) -> 144
11011 -> f(s) -> 729

Questa volta siamo peggiorati. Decidiamo di terminare perché non abbiamo portato miglioramenti dalla scorsa iterazione. Quindi, scartiamo questa generazione, torniamo alla precedente e prendiamo il miglior individuo, *i.e., 11111*

Algoritmi di Ricerca Locale (2), Gli Algoritmi Genetici

Gli algoritmi genetici, pseudocodice

```
function ALGORITMO-GENETICO-SEMPLICE(funzione_fitness, stopping_condition)
                                    returns una soluzione

    P <— popolazione iniziale

    repeat

       valuta(funzione_fitness, P)
       P_1 <— selezione(P)
       P_2 <— crossover(P_1)
       P_3 <— mutazione(P_2)
       P <— P_3

    until stopping_condition è vera

    b <— ottieni_migliore_individuo(fitness_function, P)
    return b
```

/* E' importante notare che la funzione non prevede il caso in cui l'algoritmo fallisca */

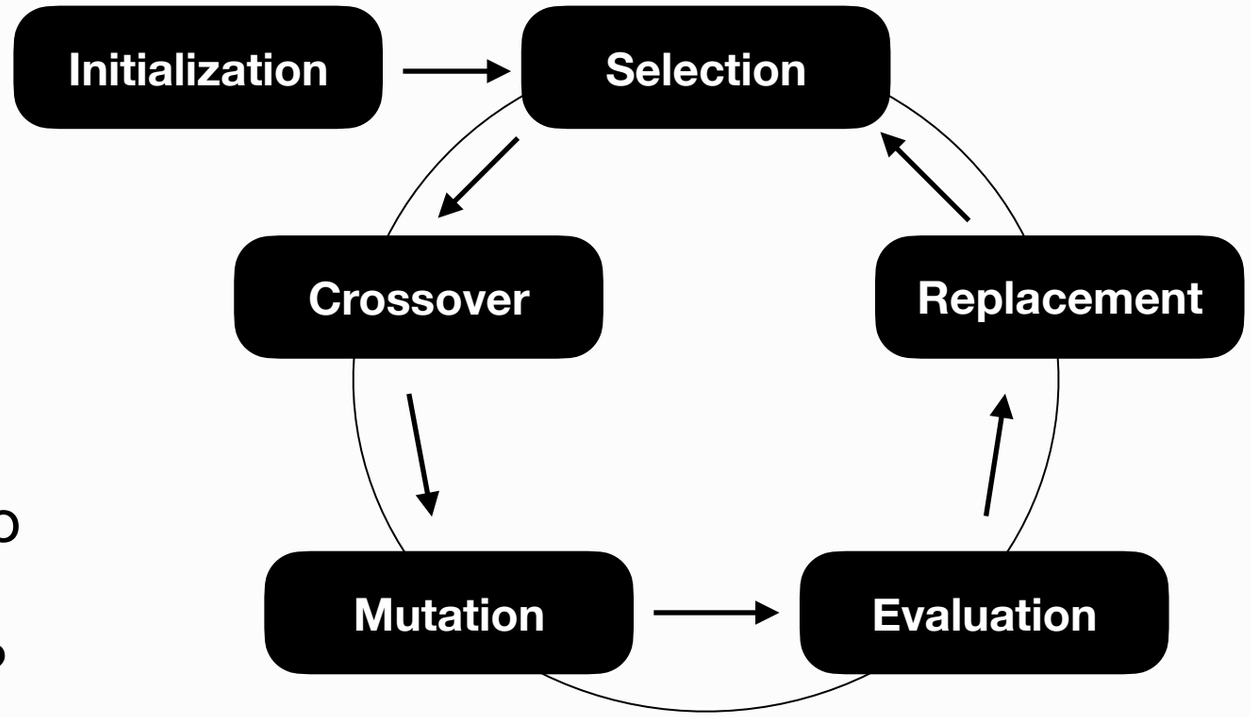
Potenzialmente, infatti, un GA può "non fallire mai", nel senso che una soluzione, seppur non ottimale, la può sempre restituire.

In realtà, il concetto di fallimento dipende fortemente dal problema che stiamo risolvendo e da come decidiamo di risolverlo. Nell'esempio giocattolo, ci siamo mossi solo nello *spazio delle soluzioni ammissibili*, ma possono esserci GA che viaggiano in spazi di soluzioni inammissibili. Dopo vediamo un esempio.

Algoritmi di Ricerca Locale (2), Gli Algoritmi Genetici

Gli algoritmi genetici, recap e considerazioni

In primo luogo, quindi, possiamo rappresentare le fasi dell'algoritmo come in figura.



Nell'esempio, abbiamo restituito l'individuo 11111 poiché lo abbiamo considerato ottimo.

Tuttavia, per quanto l'algoritmo giocattolo termini, non è detto che 11111 sia effettivamente il punto di ottimo. Perché?

Lo abbiamo detto prima: un GA è **cieco**. Per tutta l'esecuzione, non abbiamo mai conosciuto né discusso della natura di *f(s)*, ovvero la monotonia, continuità e derivabilità.

Togliamo il cappuccio, e scopriamo che *f(s)* altro non era che: ***val(s)²***: converte *s* in un intero senza segno e lo eleva al quadrato. Sapendo che questa funzione è monotona crescente, e che 11111 è il massimo intero senza segno su 5 bit, soltanto adesso possiamo dire con sicurezza che 11111 è il punto di ottimo globale!

In ogni caso, l'esempio ci ha aiutato a comprendere le potenzialità dei GA: da un lato, possono essere utilizzati in situazioni di incertezza; dall'altro lato, gli operatori genetici consentono di diversificare le soluzioni, esplorando meglio il panorama degli stati.

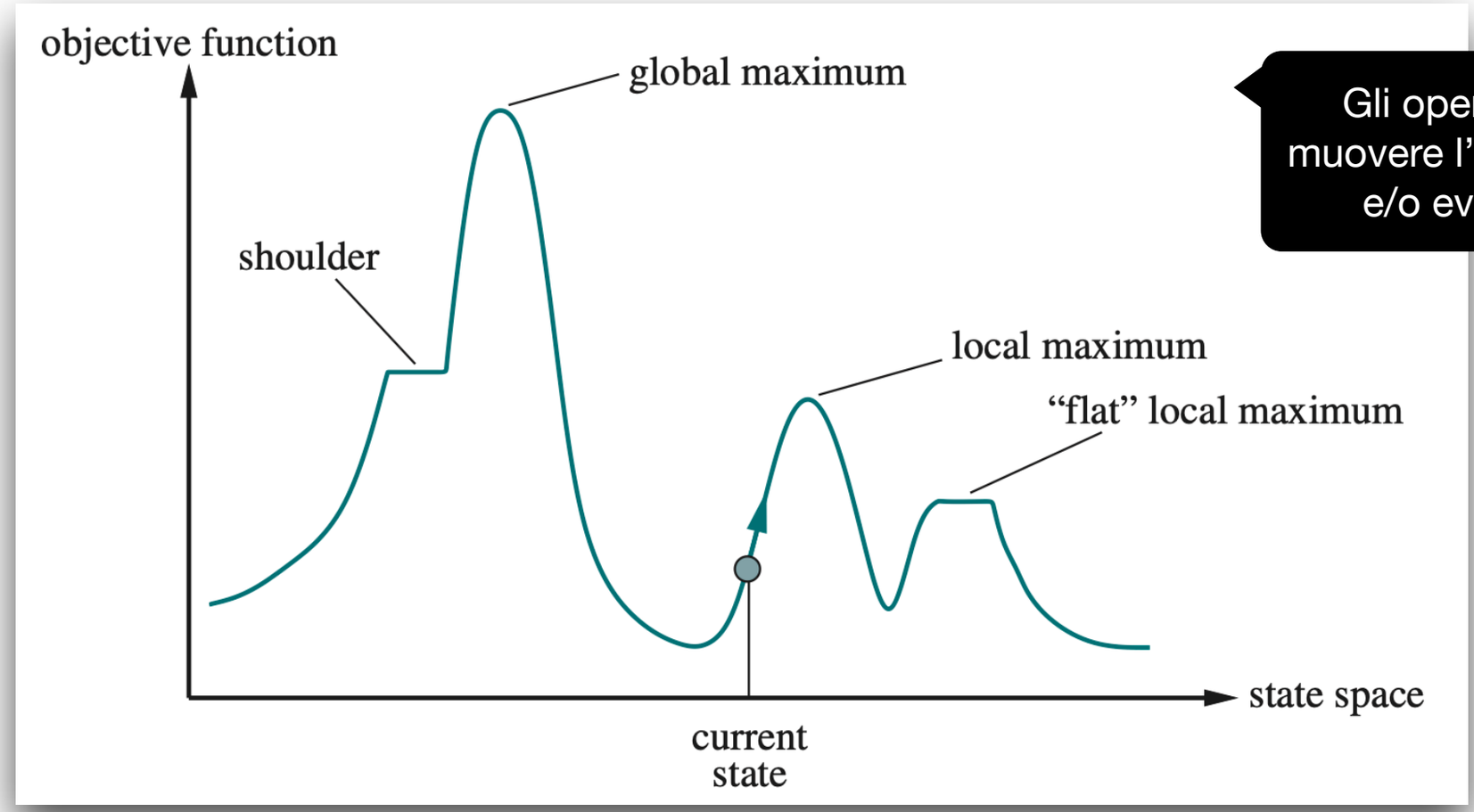
Algoritmi di Ricerca Locale (2), Gli Algoritmi Genetici

Gli algoritmi genetici, alcuni termini rilevanti

Avendo compreso il meccanismo generale di un algoritmo genetico, passiamo adesso a capire come possiamo configurare i singoli passi dell'algoritmo e quale effetto le nostre scelte di progettazione possono avere sull'efficacia dell'algoritmo.

Nel prosieguo della lezione, faremo riferimento a tre termini che vale la pena definire.

Innanzitutto, ricordiamo che ci stiamo muovendo lungo un panorama degli stati al fine di ottimizzare soluzioni iniziali verso un ottimo globale.



Gli operatori genetici hanno l'obiettivo di muovere l'algoritmo verso stati più interessanti e/o evitare stagnazione in ottimi locali.

Algoritmi di Ricerca Locale (2), Gli Algoritmi Genetici

Gli algoritmi genetici, alcuni termini rilevanti

Avendo compreso il meccanismo generale di un algoritmo genetico, passiamo adesso a capire come possiamo configurare i singoli passi dell'algoritmo e quale effetto le nostre scelte di progettazione possono avere sull'efficacia dell'algoritmo.

Nel prosieguo della lezione, faremo riferimento a tre termini che vale la pena definire.

Innanzitutto, ricordiamo che ci stiamo muovendo lungo un panorama degli stati al fine di ottimizzare soluzioni iniziali verso un ottimo globale.

Convergenza

Per *convergenza* si intende la capacità di un GA di migliorare iterativamente le soluzioni candidate verso il punto di ottimo.

Quando gli individui tendono a somigliarsi troppo, bloccando *troppo presto* il progresso globale dell'intera popolazione, parliamo di *convergenza prematura*.

Diversità

Per *diversità* si intende la capacità di un GA di definire individui che possano navigare il panorama degli stati in maniera efficace, evitando la stagnazione verso punti di ottimo locale.

Algoritmi di Ricerca Locale (2), Gli Algoritmi Genetici

Gli algoritmi genetici, alcuni termini rilevanti

Funzione di fitness

Menzionata più volte già durante l'esempio, rappresenta un elemento chiave per la definizione di un algoritmo genetico.

Formalmente parlando, la funzione di fitness è una *funzione in grado di associare un valore a ogni soluzione*.

Misura il livello di adeguatezza degli individui rispetto al problema considerato e, inevitabilmente, guida il processo di selezione.

A livello ingegneristico, progettare una buona funzione di fitness è difficile quanto definire delle buone euristiche negli algoritmi di ricerca informata.

Chiaramente, la funzione dipenderà dal problema in esame e dovrà ben rappresentare il grado di adeguatezza di una soluzione.

Nota importante: nell'esempio, abbiamo visto che la funzione fosse composta da un unico elemento, ovvero $val(s)^2$, tuttavia nulla vieta di combinare più elementi.

Ad esempio, una combinazione lineare di più fattori. In tutti i casi, indipendentemente da come impostiamo la funzione di fitness, stiamo ancora parlando di funzioni *mono-obiettivo*, in quanto l'obiettivo è formalizzato usando un'unica funzione obiettivo. Vedremo a breve che possiamo anche fare altro…

objective

Gli algoritmi genetici, parametri

L'esempio ci ha illustrato il funzionamento di base dei GA, ma ci sono dei punti che abbiamo tralasciato. Ad esempio:

(1) Perché abbiamo usato Roulette Wheel Selection? Non potevamo usare altro?
(2) Crossover e Mutation hanno delle varianti?
(3) Perché la popolazione aveva una size fissata a quattro individui?
(4) Perché ci siamo fermati dopo soltanto una iterazione che non ha dato miglioramenti? Se avessimo aspettato un altro po' che sarebbe successo?
(5) Quali altre stopping condition esistono?

Tutte queste domande trovano risposta nel *setup* di un GA. Abbiamo a disposizione diversi parametri che consentono al progettista di modificare la struttura dell'algoritmo.

Size Popolazione Numero individui di ciascuna generazione.

- Se la size è fissa, bisogna decidere il numero;
- Se la size è variabile, è opportuno decidere una dimensione massima. La ricerca potrebbe diventare eccessivamente lenta.

Size Mating Pool Numero di individui selezionati e che partecipano alla riproduzione.

- Più grande è la size, più lento sarà l'algoritmo. Più piccola è la size, più limitata sarà la diversità che l'algoritmo garantirà.

Algoritmi di Ricerca Locale (2), Gli Algoritmi Genetici

Gli algoritmi genetici, parametri

Probabilità crossover Con quale probabilità avviene il crossover tra due genitori.

- Possiamo stabilire una probabilità che l'algoritmo userà per valutare se effettuare l'operazione di crossover o meno. Più alta è la probabilità, più spesso i genitori produrranno nuovi individui e, quindi, più diversità ci sarà in termini di soluzioni.

- Anche in questo caso, però, bisogna stare attenti. Più alta sarà la probabilità, più è probabile che l'algoritmo navighi lo spazio di ricerca senza andare a convergenza. Più bassa sarà la probabilità, più è probabile una convergenza prematura.

Probabilità mutazione Con quale probabilità avviene una mutazione.

- Considerazioni simili alle precedenti.

Inizializzazione Con quale criterio si crea la prima generazione.

- Possiamo creare una popolazione iniziale in maniera totalmente casuale.

- Oppure, possiamo basarci su *euristiche*. Se riusciamo a rendere i GA meno ciechi (dando informazioni problem-specific) possiamo evitare di iniziare con individui quasi sicuramente pessimi (se non addirittura inammissibili).

- Se possibile, basarsi su euristiche è desiderabile: la ricerca può essere immediatamente guidata verso soluzioni ammissibili/migliori, riducendo il carico computazionale.

Algoritmi di Ricerca Locale (2), Gli Algoritmi Genetici

Gli algoritmi genetici, parametri

Rappresentazione individui Tipo di codifica degli individui.

- Abbiamo visto, nell'esempio, l'uso di una rappresentazione basata su stringa binaria.
- Ma possiamo rappresentare gli individui come meglio crediamo per il problema in esame. Altre rappresentazioni molto usate sono le stringhe di interi/reali o addirittura gli alberi.

Algoritmo di selezione Come avviene la selezione degli individui.

- *Casuale*: sconsigliata in molti casi, poiché aumenta le chance di non convergenza.
- *Roulette Wheel*, visto nell'esempio: gli individui ricevono una probabilità di selezione pari al valore della loro fitness relativa all'intera popolazione. Molto fedele alla natura, ma un individuo "molto forte" verrà selezionato troppe volte, rischiando la convergenza prematura.
- *Rank:* si compie un ordinamento totale degli individui rispetto alla fitness e si assegna a ciascuno individuo il rango in base alla posizione (*e.g.,* il primo otterrà rango 1, il secondo rango 2, e così via). Ciascun individuo riceve una probabilità di selezione inversamente proporzionata al rango.

$$p(x) = \frac{|P| - rank(P)}{|P|}$$

Come per Roulette Wheel, si compiono selezioni sulla base di queste probabilità. Per quanto molto simile, essa ha minor rischio di convergenza prematura rispetto a Roulette Wheel.

Occhio al tempo di esecuzione: ordinare gli individui può costare!

Algoritmi di Ricerca Locale (2), Gli Algoritmi Genetici

Gli algoritmi genetici, parametri

Algoritmo di selezione Come avviene la selezione degli individui.

- *Truncation:* si compie un ordinamento totale in maniera analoga ad una Rank Selection, ma la selezione non avviene su base casuale quanto con una selezione rigida dei migliori *M (< n)* individui. Non possono esserci selezioni ripetute.

- *K-way tournament: K (< n)* individui sono selezionati casualmente (formando il *torneo*) e il migliore tra questi *K* passa la selezione definitivamente. Si ripete finché non si arriva ad *M* tornei (e quindi *M* vincitori). Si possono avere selezioni ripetute.

Algoritmi di Ricerca Locale (2), Gli Algoritmi Genetici

Gli algoritmi genetici, parametri

Algoritmo di selezione Come avviene la selezione degli individui.

- *Truncation:* si compie un ordinamento totale in maniera analoga ad una Rank Selection, ma la selezione non avviene su base casuale quanto con una selezione rigida dei migliori *M (< n)* individui. Non possono esserci selezioni ripetute.

- *K-way tournament: K (< n)* individui sono selezionati casualmente (formando il *torneo*) e il migliore tra questi *K* passa la selezione definitivamente. Si ripete finché non si arriva ad *M* tornei (e quindi *M* vincitori). Si possono avere selezioni ripetute.

E' poco complessa ed ha interessanti somiglianze con la natura (legge del più forte, letteralmente), tutti motivi per cui è una scelta molto popolare. Tuttavia impone l'onore di una buona scelta di *K* ed *M*.

Si può adottare una *pressione di selezione*: invece di far vincere sempre il migliore (assoluto) di un torneo, ogni individuo ha una probabilità di vittoria proporzionata alla sua fitness. In sostanza, i singoli tornei diventano delle piccole Roulette Wheel.

Si possono *impedire le ripetizioni* (in tal caso, se il mating pool è grande quanto la popolazione, ogni individuo ha la garanzia di partecipare ad esattamente *K* tornei).

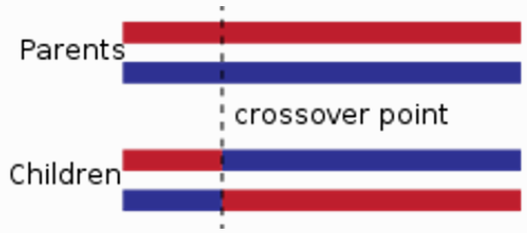
Osserviamo che con *K*=1, si ha una Truncation Selection. Di fatti, si procederebbe selezionando gli individui migliori fino ad M, escludendo gli altri.

Algoritmi di Ricerca Locale (2), Gli Algoritmi Genetici

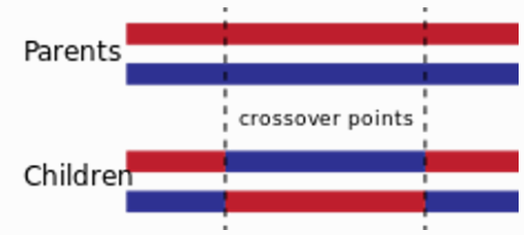
Gli algoritmi genetici, parametri

Algoritmo di crossover Come avviene il crossover tra individui.

- *Single Point:* visto nell'esempio, si seleziona un punto del patrimonio genetico dei genitori e si procede alla generazione di due figli tramite scambio di cromosomi.



- *Two-Point:* simile al single point, va a considerare però due punti di incrocio. Aumenta chiaramente la diversità dei geni degli individui generati.



- *K-Point:* Generalizzazione dei precedenti, è poco utilizzato dal momento che richiede la definizione di una strategia di combinazione dei K cromosomi dei genitori.

- *Uniform:* Ciascun gene *i*-esimo viene scelto casualmente tra i due geni *i*-esimi dei genitori. Ha il massimo grado di casualità. E' un particolare caso di K Point con *K*=Size_Individuo.

- *Arithmetic:* Ciascun gene *i*-esimo è il valor medio dei geni *i*-esimi dei genitori. Valido solo per rappresentazioni numeriche (precisamente, per le rappresentazione per cui ha senso calcolare il mid-point). Ne segue che i due figli saranno *gemelli*.

—> Per differenziare i due figli, è possibile assegnare un peso (fisso o casuale) ai due genitori e fare la media pesata invece che quella aritmetica. Quindi, il primo figlio usa peso *α* per il primo genitore e 1-*α* per il secondo, viceversa per il secondo figlio.

Algoritmi di Ricerca Locale (2), Gli Algoritmi Genetici

Gli algoritmi genetici, parametri

Algoritmo di crossover Come avviene il crossover tra individui.

ne richiede la

simi dei genitori.
ze_Individuo.

ori. Valido solo
cui ha senso

suale) ai due
no figlio usa
o figlio.

Quasi sempre usiamo 2 genitori per la riproduzione. Tuttavia, alcuni studi empirici hanno mostrato che le possibilità con $n > 2$ genitori non devono essere sottovalutate!

Applied Mathematical Modelling 37 (2013) 2737–2746

Contents lists available at SciVerse ScienceDirect

Applied Mathematical Modelling

journal homepage: www.elsevier.com/locate/apm

Multiple parents crossover operators: A new approach removes the overlapping solutions for sequencing problems

Shih-Hsin Chen [a], Min-Chih Chen [b], Pei-Chann Chang [c,*], V. Mani [d]

[a] Department of Electronic Commerce Management, Nanhua University, No. 55, Sec. 1, Nanhua Rd., Zhongkeng, Dalin Township, Chiayi County 62248, Taiwan, ROC
[b] Department of Information Management, WuFeng University, Chiayi County 62153, Taiwan, ROC
[c] Department of Information Management, Yuan-Ze University, 135 Yuan-Dong Rd., Taoyuan 32026, Taiwan, ROC
[d] Department of Aerospace Engineering, Indian Institute of Science, Bangalore, India

ARTICLE INFO

Article history:
Received 3 October 2010
Received in revised form 20 May 2012
Accepted 4 June 2012
Available online 15 June 2012

Keywords:
Diversity
Removing redundant solutions
Multi-parents crossover operator
Flowshop scheduling problems

ABSTRACT

Maintaining population diversity throughout generations of Genetic Algorithms (GAs) is key to avoid premature convergence. Redundant solutions is one cause for the decreasing population diversity. To prevent the negative effect of redundant solutions, we propose a framework that is based on the multi-parents crossover (MPX) operator embedded in GAs. Because MPX generates diversified chromosomes with good solution quality, when a pair of redundant solutions is found, we would generate a new offspring by using the MPX to replace the redundant chromosome. Three schemes of MPX will be examined and will be compared against some algorithms in literature when we solve the permutation flowshop scheduling problems, which is a strong NP-Hard sequencing problem. The results indicate that our approach significantly improves the solution quality. This study is useful for researchers who are trying to avoid premature convergence of evolutionary algorithms by solving the sequencing problems.

1. Introduction

Genetic Algorithms (GAs) have been widely used to solve many optimization problems because of their ease of use with promising results. GAs maintain and evolve solutions through the selection and variation in each generation. Through the generations of GAs, the population converges to better solution space while the population diversity is decreased in the same time. Premature convergency is well-recognized for GAs [1–3], which causes the problem of staying at local optimal instead of the global optimum. It is because that evolutionary algorithms attempt to converge to a optimal solution so that the solution space is narrowed down to a small region. Hence, it is a key to improve the population diversity when the population diversity is poor. As a result, many researchers studied some approaches to avoid the premature convergence of GAs, including:

1. Restart Strategy [4,5].
2. Immigrants [6–10].
3. Multiple crossover operators [11–16].
4. Adaptive Strategy [17–19].

* Corresponding author.
E-mail addresses: shihhsin@mail.nhu.edu.tw (S.-H. Chen), cthunter@mail.wfc.edu.tw (M.-C. Chen), iepchang@saturn.yzu.edu.tw (P.-C. Chang), mani@aero.iisc.ernet.in (V. Mani).

0307-904X/$ - see front matter © 2012 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.apm.2012.06.005

Algoritmi di Ricerca Locale (2), Gli Algoritmi Genetici

Gli algoritmi genetici, parametri

Algoritmo di mutazione Come avviene la mutazione degli individui.

- *Bit Flip:* visto nell'esempio, consiste nella modifica di un singolo gene binario.
- *Random Resetting:* cambio casuale di un gene ad un altro valore ammissibile. La distribuzione da cui campionare il valore è a scelta: uniforme/normale/ecc.
- *Swap:* scambio di due geni, scelti casualmente.
- *Scramble:* si sceglie un subset di geni in modo casuale e lo si permuta casualmente.
- *Inversion:* si sceglie un subset di geni in modo casuale e lo si ribalta.

In altri termini, in termini di mutazione abbiamo molta flessibilità. Qualsiasi variazione va bene in linea di principio, ma è bene che la mutazione sia *ammissibile* (*i.e.,* da un individuo ammissibile si passi ad un altro ammissibile, ovvero che la mutazione non vada a violare i vincoli del problema).

Alcuni algoritmi di mutazione lavorano *localmente*, ovvero su singoli geni dell'individuo, altri a livello *globale*. Chiaramente, gli algoritmi globali creano maggiore diversità, ma aumentano il rischio di mancata convergenza.

Tipicamente, gli algoritmi locali sono preferibili proprio perché limitano l'effetto di casualità che la mutazione può avere sulla convergenza dell'algoritmo.

Algoritmi di Ricerca Locale (2), Gli Algoritmi Genetici

Gli algoritmi genetici, parametri

Stopping condition

Con quale criterio decidiamo di terminare l'evoluzione oppure proseguire.

- *Tempo di esecuzione:* l'evoluzione termina se si è superato un tempo di esecuzione *T*. Allo stop o si decide di restituire l'ultima generazione ottenuta o la migliore ultima (preferibile).

- *Costo:* se è presente una funzione di costo (*i.e.,* una funzione---diversa dalla fitness---che associa ad individuo un costo secondo un certo criterio), allora al raggiungimento o superamento di quel costo l'evoluzione termina.

- *Numero di iterazioni:* si itera per un massimo di *X* generazioni.

- *Assenza di miglioramenti:* visto nell'esempio, se per *Y* generazioni consecutive non ci sono miglioramenti (significativi), allora l'evoluzione termina.

- *Ibride:* ovvero, basate su una combinazione delle precedenti.

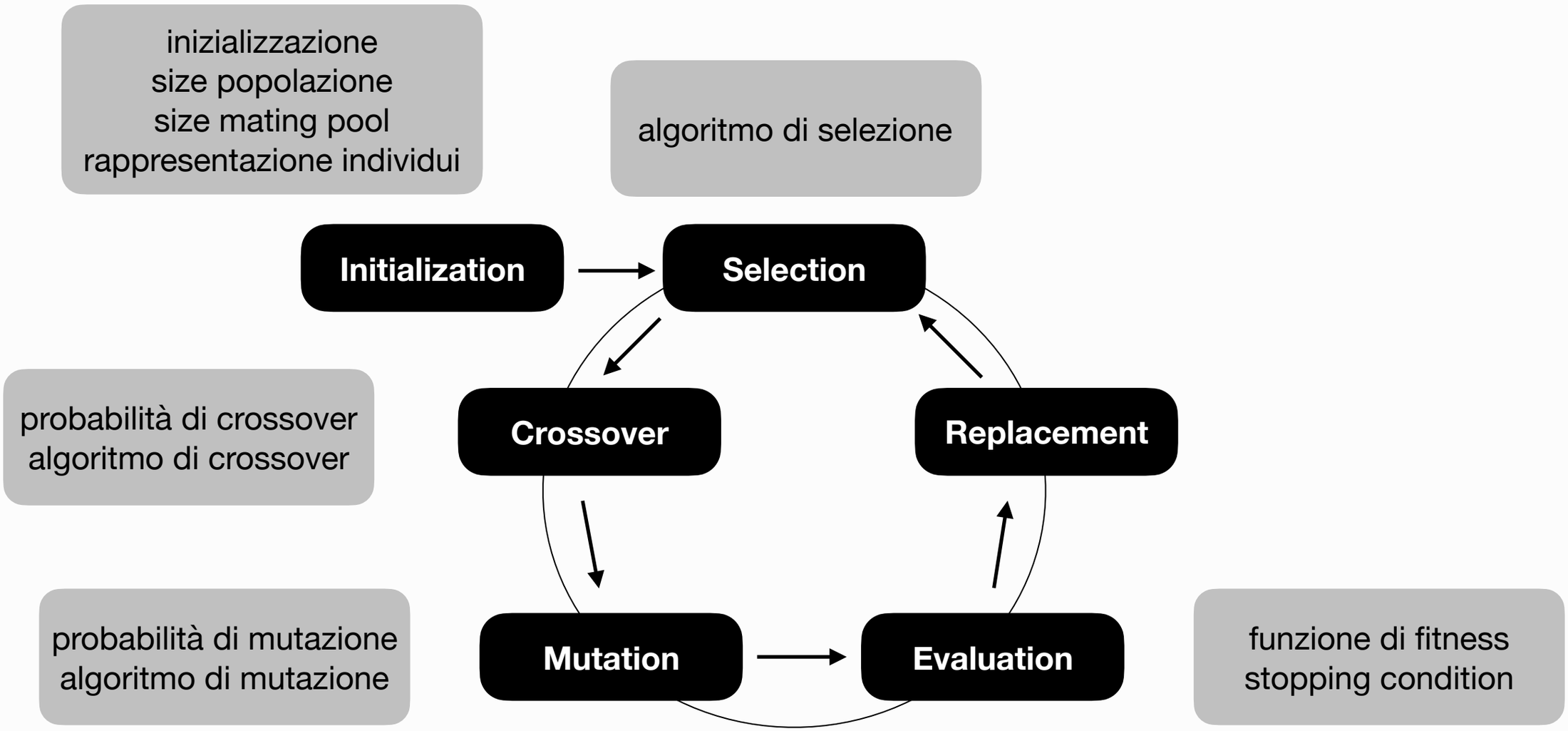
- *Problem specific:* conoscendo i dettagli del problema, possiamo definire condizioni ad-hoc.

Spesso, invece di parlare esplicitamente di stopping condition, si preferisce la locuzione *budget di ricerca* (search budget), per indicare meglio il concetto di risorse a disposizione (tempo/costo/iterazioni/ecc.).

In molti contesti, una stopping condition composta da tempo di esecuzione/costo e assenza di miglioramenti tende ad essere sufficientemente buona.

Algoritmi di Ricerca Locale (2), Gli Algoritmi Genetici

Gli algoritmi genetici, ottimizzazione dei parametri

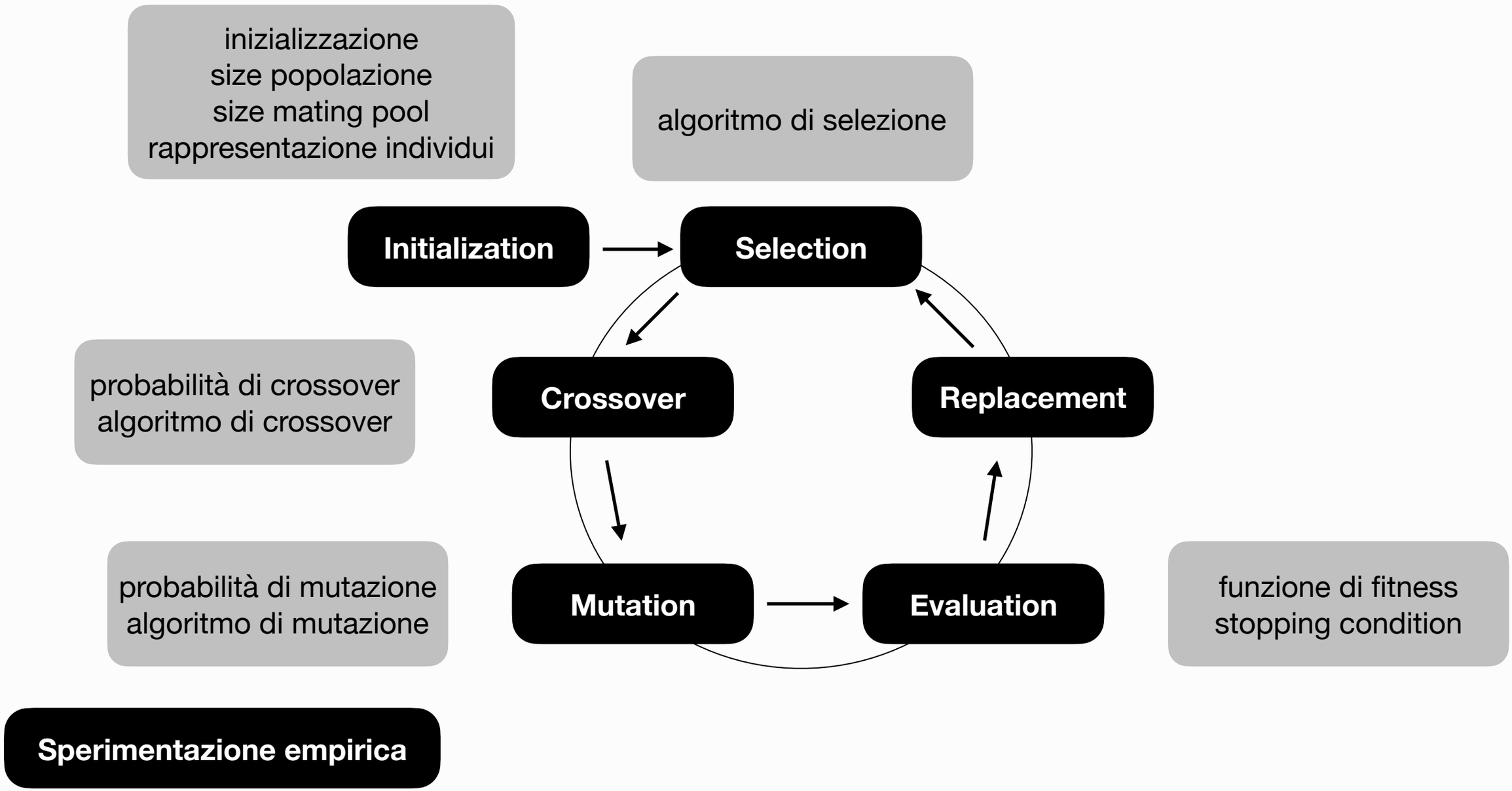


L'avere a disposizione tutti questi parametri ci consente una flessibilità enorme nella fase di progettazione. Tuttavia, flessibilità implica anche enorme capacità di errore!

Il modo in cui i parametri vengono impostati impatta sia l'efficacia che l'efficienza dell'algoritmo risultante e, per questo, sarà necessaria una fase di ottimizzazione dei parametri, ovvero una fase di verifica dell'impatto dei parametri sulle prestazioni.

Algoritmi di Ricerca Locale (2), Gli Algoritmi Genetici

Gli algoritmi genetici, ottimizzazione dei parametri

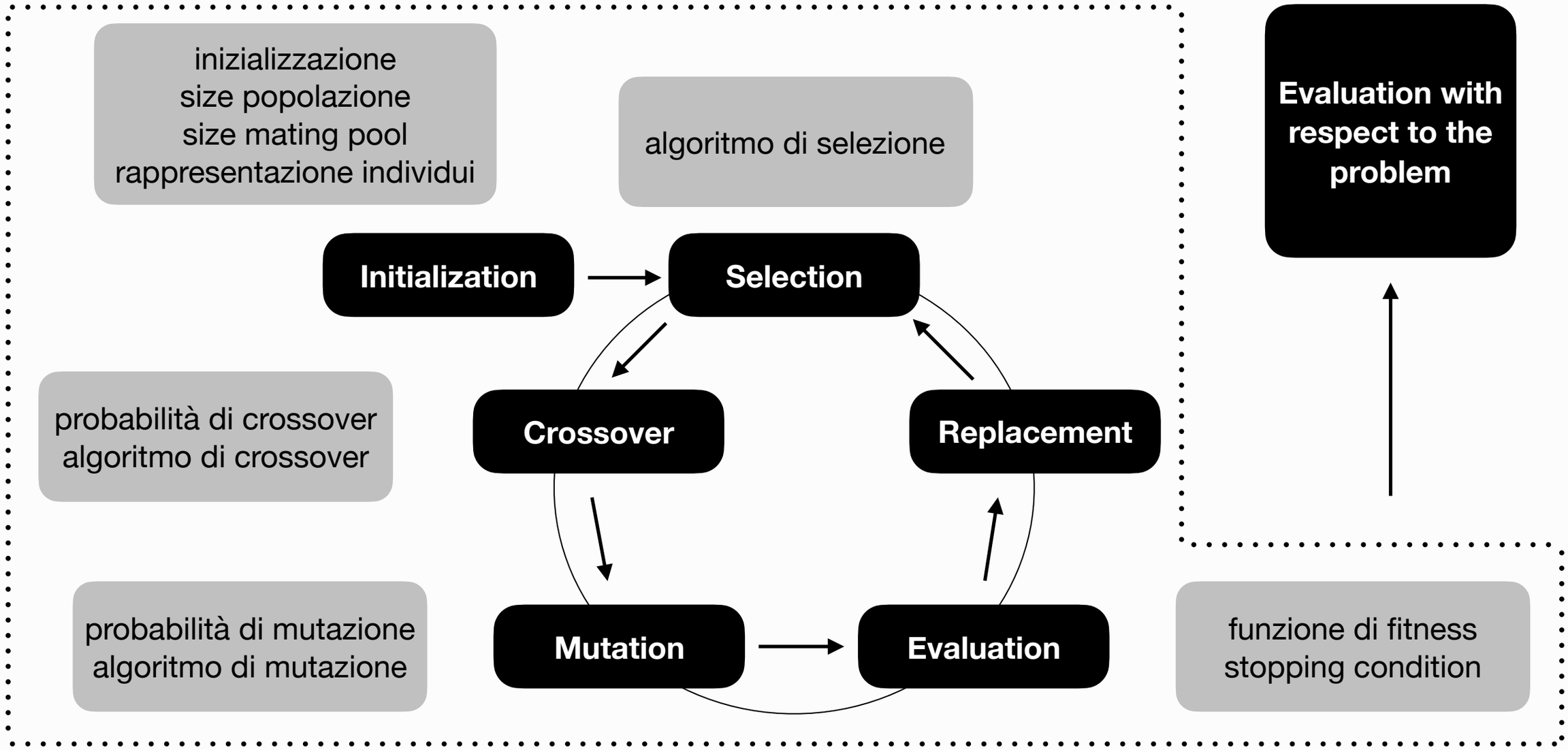


Come fare? La soluzione migliore al problema consiste nella sperimentazione empirica, ovvero nella verifica di diverse configurazioni dell'algoritmo rispetto alle prestazioni.

La sperimentazione empirica non significa testare configurazioni a caso…

Algoritmi di Ricerca Locale (2), Gli Algoritmi Genetici

Gli algoritmi genetici, ottimizzazione dei parametri



Normalmente, il modo migliore è quello di procedere per incrementi successivi, valutando l'impatto di **un singolo parametro alla volta**.

Cambiando più parametri, infatti, non avete controllo: non riuscirete a comprendere se il cambiamento osservato nelle prestazioni dipende da uno o più fattori.

Il modo di procedere va documentato insieme al razionale dietro le scelte fatte!

Algoritmi di Ricerca Locale (2), Gli Algoritmi Genetici

Gli algoritmi genetici, campi di applicazione

Come intuito, i GA risultano essere molto flessibili. Ragion per cui, nella letteratura scientifica ci sono moltissime applicazioni nei più disparati campi.

(1) Telecomunicazioni: design ottimale di antenne;
(2) Aerodinamica: design ottimale di corpi aerodinamici;
(3) Circuiti Elettronici: Evolvable Hardware (EH);
(4) Bioinformatica: analisi DNA;
(5) **Machine Learning**: Feature Selection, ottimizzazione iperparametri, *AutoML* (*i.e.,* far evolvere un insieme di blocchi matematici di base per costruire in modo ottimale i vari step della pipeline ML)
(6) **Deep Learning**: *Neuroevoluzione i.e.,* far evolvere reti neurali come "alternativa" al loro addestramento (praticamente, invece che studiare per un esame, facciamo riprodurre tantissime persone finché non esce un genio che sa superarlo senza studiare);
(7) **Algoritmi**: Commesso viaggiatore (TSP) e sue applicazioni;
(8) **Programmazione**: *Genetic Programming* (GP) *i.e.,* far evolvere programmi come "alternativa" alla loro scrittura manuale).

Ma i GA sono utilizzati tanto anche in combinazione con l'ingegneria del software.

Ad esempio, automatic remodularization (e.g., ottimizzazione di coesione e accoppiamento), automated test case generation, o software transpantation.

Gli algoritmi genetici, campi di applicazione

Come intuito, i GA risultano essere molto flessibili. Ragion per cui, nella letteratura scientifica ci sono moltissime applicazioni nei più disparati campi.

(1) Telecomunicazioni: design ottimale di antenne;
(2) Aerodinamica: design ottimale di corpi aerodinamici;
(3) Circuiti Elettronici: Evolvable Hardware (EH);
(4) Bioinformatica: analisi DNA;
(5) **Machine Learning**: Feature Selection, ottimizzazione iperparametri, *AutoML* (*i.e.,* far evolvere un insieme di blocchi matematici di base per costruire in modo ottimale i vari step della pipeline ML)
(6) **Deep Learning**: *Neuroevoluzione i.e.,* far evolvere reti neurali come "alternativa" al loro addestramento (praticamente, invece che studiare per un esame, facciamo riprodurre tantissime persone finché non esce un genio che sa superarlo senza studiare);
(7) **Algoritmi**: Commesso viaggiatore (TSP)
(8) **Programmazione**: *Genetic Programmin* mmi come "alternativa" alla loro scrittura manuale).

Alla fine del corso, quando parleremo di usabilità, discuteremo anche dell'utilizzo di GA nel contesto di smart development

Ma i GA sono utilizzati tanto anche in combinazione con l'ingegneria del software.

Ad esempio, automatic remodularization (e.g., ottimizzazione di coesione e accoppiamento), automated test case generation, o software transpantation.

Algoritmi di Ricerca Locale (2), Gli Algoritmi Genetici

Gli algoritmi genetici, un'analisi critica

A conclusione di questa prima disamina sugli algoritmi genetici, facciamo un recap dei principali pro e contro nell'uso di algoritmi genetici.

- **Fitness molto flessibile.** Spesso si basa sul concetto di *goal*, ovvero un "requisito" che gli individui devono soddisfare per ottenere una maggiore fitness.
- Ottimizza sia nel **continuo** che nel **discreto**.
- Valido per problemi **multi-obiettivo.**
- **Esplora rapidamente** lo spazio di ricerca ed ottiene in poco tempo soluzioni "buone".
- Ben parallelizzabile.
- Buona explainability.

- Questa sua flessibilità **non si presta** molto bene **per problemi di decisione** (*i.e.,* problemi che pone "domande con risposte si/no" ai suoi input).

Workaround: convertirli nei corrispondenti problemi di ottimizzazione.

- Durante l'evoluzione, la funzione di fitness è valutata moltissime volte, e le **prestazioni possono risentirne** se essa e/o la codifica sono complesse.

Tip: mettere in campo delle approssimazioni, se possibile.

Algoritmi di Ricerca Locale (2), Gli Algoritmi Genetici

Gli algoritmi genetici, un'analisi critica

A conclusione di questa prima disamina sugli algoritmi genetici, facciamo un recap dei principali pro e contro nell'uso di algoritmi genetici.

- **Fitness molto flessibile.** Spesso si basa sul concetto di *goal*, ovvero un "requisito" che gli individui devono soddisfare per ottenere una maggiore fitness.
- Ottimizza sia nel **continuo** che nel **discreto**.
- Valido per problemi **multi-obiettivo.**
- **Esplora rapidamente** lo spazio di ricerca ed ottiene in poco tempo soluzioni "buone".
- Ben parallelizzabile.
- Buona explainability.

- Esposti al problema dell **convergenza prematura.**

Tip: bisogna aumentare la **diversity**, ovvero mettendo in campo strategie che creino individui i più diversi tra loro (e.g., con la mutazione e altri elementi di casualità). In generale, è bene che un GA faccia buona **exploitation** (i.e., favorire individui che sono già buoni, promettenti) e buona **exploration** (i.e., saltare in più zone possibili dello spazio di ricerca, così da aggirare eventuali ottimi locali e liberarsi da convergenze premature).

Algoritmi di Ricerca Locale (2), Gli Algoritmi Genetici

Gli algoritmi genetici, un’analisi critica

A conclusione di questa prima disamina sugli algoritmi genetici, facciamo un recap dei principali pro e contro nell’uso di algoritmi genetici.

- **Fitness molto flessibile.** Spesso si basa sul concetto di *goal*, ovvero un “requisito” che gli individui devono soddisfare per ottenere una maggiore fitness.
- Ottimizza sia nel **continuo** che nel **discreto**.
- Valido per problemi **multi-obiettivo.**
- **Esplora rapidamente** lo spazio di ricerca ed ottiene in poco tempo soluzioni “buone”.
- Ben parallelizzabile.
- Buona explainability.

- Esposti al problema dell **convergenza prematura.**
- Non raggiungono sempre **l’ottimo**, ma una buona configurazione può portare a risultati quasi sempre buoni.
- There is no **magic number.** E’ difficile fare una scelta ottimale dei parametri, ma una buona fase di valutazione empirica può rendere l’algoritmo molto performante.

Algoritmi di Ricerca Locale (2), Gli Algoritmi Genetici

Gli algoritmi genetici, tecniche di improvement

Non finisce qui, però. I GA sono migliorabili sono tantissimi punti di vista.

Improvement #1 Il concetto di *elitismo*.

In fase di selezione, un individuo con alta fitness ha alte chance di sopravvivenza, ma la selezione può essere comunque crudele contro di lui… Vogliamo davvero perdere un individuo potenzialmente importante?

Elitism. Operazione per la quale salviamo il migliore (o i migliori *k*) individuo e lo copiamo nella generazione successiva.

In altri termini, evitiamo all'individuo più forte di incappare nel processo di selezione.

Questa operazione può essere applicata a prescindere dall'algoritmo di selezione e *garantisce* che la nuova generazione non sia peggiore della precedente (rispetto al miglior individuo).

Improvement #2 Uso di euristiche problem-specific.

E' vero che i GA sono ciechi, ma ciò non toglie che l'utilizzo di informazioni aggiuntive possano rivelarsi utili. Queste possono velocizzare i miglioramenti, aumentare la diversità e ridurre il rischio di convergenza prematura.

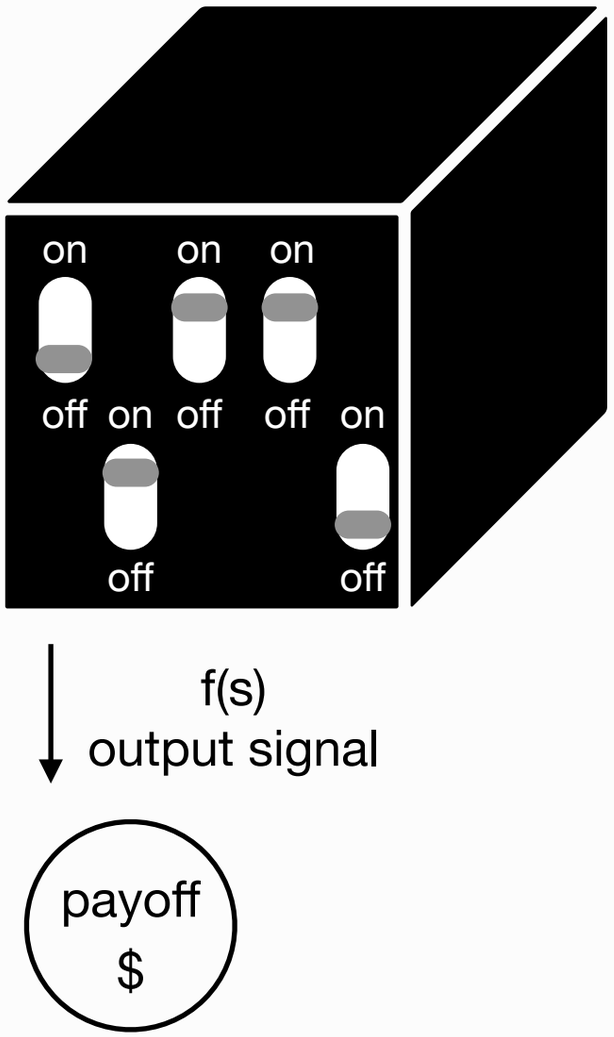
Algoritmi di Ricerca Locale (2), Gli Algoritmi Genetici

Gli algoritmi genetici, tecniche di improvement

Non finisce qui, però. I GA sono migliorabili sono tantissimi punti di vista.

Improvement #3 Problemi multi-obiettivo.

Nel parlare della funzione di fitness, abbiamo sempre menzionato - implicitamente o esplicitamente - di funzioni *mono-obiettivo*.



Ad esempio, nell'esempio giocattolo il nostro obiettivo era quello di massimizzare il valore di output dato da una configurazione delle levette on-off.

Supponiamo però che ogni levetta attivi un interruttore della corrente. Vorremmo magari tenere sotto controllo il consumo energetico e i costi, soprattutto nei giorni correnti…

Di conseguenza, l'obiettivo potrebbe essere riformulato come quello di massimizzare il valore di output minimizzando il numero di levette in posizione on, dando vita ad un problema multi-obiettivo.

In questi problemi, ci saranno *n* funzioni di fitness. Un individuo, quindi, non avrà più un singolo valore di fitness, ma *n*. O meglio, un *vettore di fitness lungo n.*

Gli algoritmi genetici, tecniche di improvement

Non finisce qui, però. I GA sono migliorabili sono tantissimi punti di vista.

Improvement #3 Problemi multi-obiettivo.

Sembra una semplice generalizzazione, ma c'è una sostanziale differenza rispetto ai problemi mono-obiettivo. Consideriamo un problema di *minimo* con due funzioni di fitness non negative. La domanda è: *"chi è il migliore tra un individuo x con fitness <1,2> e un individuo y con fitness <0, 5>?"*

Rispetto alla prima funzione sarebbe meglio *y*, ma rispetto alla seconda è meglio *x.* E quindi? Nei problemi multi-obiettivo *non possiamo fornire un ordinamento totale*.

Di conseguenza, in fase di valutazione dovremmo ragionare diversamente rispetto a quanto fatto nei problemi mono-obiettivo.

Per arrivare al punto, facciamo un ulteriore caso di un individuo *z* con fitness <10, 20>: in questo caso, potremmo sicuramente dire che *x* è migliore poiché *x* ha (i) **tutte le fitness corrispondenti non peggiori** e (ii) **almeno una strettamente migliore**.

Fronte di Pareto. Insieme degli individui di una popolazione non migliori tra loro (secondo le due regole suddette) ma migliori di tutti gli individui fuori dal fronte.

x e *y* sono nel fronte di Pareto, mentre *z* no. *x* e *y* sono *ottimi di Pareto*.

Gli algoritmi genetici, tecniche di improvement

Non finisce qui, però. I GA sono migliorabili sono tantissimi punti di vista.

Improvement #3 Problemi multi-obiettivo.

Dati due individui nel fronte di Pareto, come decido quale di questi sia il migliore?

Preference Sorting. Tecnica che permette di fornire un ordinamento totale tra gli individui nel fronte di Pareto attraverso l'uso di una funzione di preferenza.

La funzione di preferenza è diversa dalla funzione di fitness. Quasi sempre, il criterio che governa questa funzione *deriva necessariamente dalla conoscenza del problema*.

Nell'esempio delle levette, potremmo definire una funzione di preferenza diversa a seconda del contesto o del periodo storico in cui la scatola nera è usata.

- Se la scatola fosse utilizzata in un momento storico in cui l'energia elettrica costasse poco, potremmo preferire una soluzione principalmente focalizzata sul valore di output.
- Altrimenti, potremmo voler "accontentarci" di una soluzione con valore minore ma che rappresenti un compromesso migliore.

Possiamo introdurre quanti criteri di preferenza vogliamo. Tutto dipende da quante informazioni riusciamo a ricavare dal problema e da ciò che desideriamo.

Gli algoritmi genetici, tecniche di improvement

Non finisce qui, però. I GA sono migliorabili sono tantissimi punti di vista.

Improvement #3 Problemi multi-obiettivo.

Dati due individui nel fronte di Pareto, come decido quale di questi sia il migliore?

Preference Sorting. Tecnica che permette di fornire un ordinamento totale tra gli individui nel fronte di Pareto attraverso l'uso di una funzione di preferenza.

La funzione di preferenza è diversa dalla funzione di fitness. Quasi sempre, il criterio che governa questa funzione *deriva necessariamente dalla conoscenza del problema*.

Nell'esempio delle levette, potremmo definire una funzione di preferenza diversa a seconda del contesto o del periodo storico in cui la scatola nera è usata.

- Se la scatola fosse utilizzata in un momento storico in cui l'energia elettrica costasse poco, potremmo preferire una soluzione principalmente focalizzata sul valore di o
- Altrimenti, p na soluzione con valore minore ma che rappresen

Non è comunque detto che avremo sicuramente l'ordinamento totale: dipende tutto dalla funzione di preferenza. Spesso, non ci riusciremo, ma ci accontentiamo.

Possiamo introdurre quanti criteri di preferenza vogliamo. Tutto dipende da quante informazioni riusciamo a ricavare dal problema e da ciò che desideriamo.

Gli algoritmi genetici, tecniche di improvement

Non finisce qui, però. I GA sono migliorabili sono tantissimi punti di vista.

Improvement #4 Il concetto di archivio.

Immaginiamo di avere dieci funzioni di fitness non negative. Ad un certo punto dell'evoluzione, otteniamo un individuo *x* in grado di minimizzare le prime cinque (dalla 1 alla 5). Nella successiva iterazione perdiamo *x* ma otteniamo *y*, che minimizza le funzioni dalla 3 alla 7.

Possiamo dire che: abbiamo "perso" la 1 e la 2, per "guadagnare" la 6 e la 7. Peccato aver perso *x*, se ci fosse un modo per non perderlo…

Archive Strategy. Strategia che mantiene una popolazione aggiuntiva che *non evolve*, contenente gli individui che riescono a soddisfare degli obiettivi mai soddisfatti nelle precedenti iterazioni.

Facciamo un esempio per capire meglio l'utilità dell'archivio.

- 1a iterazione: *x* minimizza da 1 a 5, *y* minimizza da 3 a 7, *z* minimizza solo 1 => Copio *x* e *y* nell'archivio
- 2a Iter.: *a* minimizza 8, *b* minimizza da 1 a 4, *c* niente => Copio *a* nell'archivio
- …
- 100a Iter.: *p* minimizza 9 e 10, *q* ed *r* niente => Copio *p* nell'archivio.

Gli algoritmi genetici, tecniche di improvement

Non finisce qui, però. I GA sono migliorabili sono tantissimi punti di vista.

Improvement #4 Il concetto di archivio.

Immaginiamo di avere dieci funzioni di fitness non negative. Ad un certo punto dell'evoluzione, otteniamo un individuo *x* in grado di minimizzare le prime cinque (dalla 1 alla 5). Nella successiva iterazione perdiamo *x* ma otteniamo *y*, che minimizza le funzioni dalla 3 alla 7.

Possiamo dire che: abbiamo "perso" la 1 e la 2, per "guadagnare" la 6 e la 7. Peccato aver perso *x*, se ci fosse un modo per non perderlo…

> **Archive Strategy.** Strategia che mantiene una popolazione aggiuntiva che *non evolve*, contenente gli individui che riescono a soddisfare degli obiettivi mai soddisfatti nelle precedenti iterazioni.

Facciamo un esempio per capire meglio l'utilità dell'archivio.

Al termine del budget, invece di restituire soltanto l'ultima generazione, **restituisco l'archivio**, perché so già che contiene un insieme di individui **molto forti quanto unici**.

Come nel preference sorting, per applicarlo è necessario avere conoscenze approfondite sulle fitness (*e.g.,* sapere quando ho raggiunto l'ottimo globale).

- ..
- 100a Iter.: *p* minimizza 9 e 10, *q* ed *r* niente => Copio *p* nell'archivio.

Algoritmi di Ricerca Locale (2), Gli Algoritmi Genetici

Gli algoritmi genetici, tecniche di improvement

Non finisce qui, però. I GA sono migliorabili sono tantissimi punti di vista.

Improvement #5 Il concetto di dynamic target selection.

In problemi complessi, potrei avere centinaia/migliaia di funzioni di fitness. Il fronte di Pareto inizia ad ingrandirsi; un criterio di preferenza "debole" non mi sarà di aiuto.

In generale, per evitare di avere troppe fitness, è consigliato usare *funzioni indipendenti tra loro*, ma dove ciò non è possibile si ha situazione del genere: *"se non minimizzo prima f_1 non potrà mai sperare di minimizzare f_2"*.

Di conseguenza, se nella mia popolazione non ho alcun individuo che minimizzi f_1, a che serve che nella successiva iterazione consideri anche f_2 durante la valutazione?

> **Dynamic Target Selection.** Tecnica che, ad ogni iterazione, ci permette di restringere l'insieme delle funzioni di fitness da valutare a seconda del risultato dell'iterazione precedente.

E' un po' come dire: ho 100 problemi, e so che gli ultimi 30 non possono essere risolti finché non risolvo i primi 70. Inizio col risolvere prima i primi 70 e poi inizio a concentrarmi sui restanti 30, *facendo attenzione a non perdere la soluzione ai primi 70!* Motivo per cui, questa tecnica si sposa bene con la strategia dell'archivio.

Algoritmi di Ricerca Locale (2), Gli Algoritmi Genetici

Gli algoritmi genetici, varianti

In questa ultima fase, aumentiamo il livello di granularità e parliamo di metaeuristiche. I GA che abbiamo visto finora sono anche noti come **Generational GA**, perché, banalmente, creano effettivamente nuove generazioni ad ogni iterazione.

Una sua variante semplificata è **Steady-State GA** che, invece, non genera nuove generazioni ma mantiene sempre la medesima popolazione. Ad ogni iterazione:

(1) Non si effettua selezione;
(2) Una sola coppia di genitori si accoppia e produce due figli;
(3) I due figli sono soggetti a mutazione;
(4) I due figli rimpiazzano i genitori.

Questa variante dà vita quindi ad un'evoluzione molto più lenta e meno caotica.

E' vero che così facendo l'algoritmo **evolverà lentamente**, ma il **costo computazionale** di ogni iterazione sarà decisamente **minore**.

Per questa ragione, lo si potrebbe preferire in alcuni problemi multi-obiettivo, i quali possono soffrire in termini di prestazioni.

Allo stesso tempo, questa variante è particolarmente utile nei problemi per cui abbiamo modo di fornire indicazioni problem-specific precise: in altri termini, se la popolazione iniziale rappresenta già una soluzione accettabile, una evoluzione lenta potrebbe portare a migliorare gli individui senza distaccarsi troppo da essi.

Algoritmi di Ricerca Locale (2), Gli Algoritmi Genetici

Gli algoritmi genetici, varianti

Un algoritmo genetico può persino integrare al suo interno **delle strategie di ricerca locale** al fine di migliorare l'exploitation, *e.g.,* lanciando un Hill Climbing sul migliore individuo ottenuto al termine di tutta l'evoluzione, oppure disseminando delle ricerche locali durante le varie iterazioni, e così via.

Quando la presenza della ricerca locale è ben integrata in un GA, parliamo di **Memetic Algorithm** (MA). Lo scopo degli MA è quello di ridurre la probabilità di convergenza prematura combinando metodi di ricerca multipli.

Ulteriori varianti

Trovare la combinazione ottimale di parametri è difficile, soprattutto perché dobbiamo deciderlo a monte, prima di avviare il GA. E se ciò avvenisse a runtime? Ad esempio, viene raddoppiata la probabilità di mutazione se non ci sono miglioramenti dopo 10 generazioni. In questi casi parliamo di **Adaptive GA**.

Alcune volte l'intervento umano può tornare utile. Ad esempio, dopo 100 iterazioni l'algoritmo si sospende, chiedendo una verifica manuale. Se l'utente è soddisfatto, l'algoritmo può essere terminato e restituire la soluzione, altrimenti si modifica il valore di qualche parametro e si riprende l'evoluzione. Qui invece, si parla di **Interactive GA**.

UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA

Laurea triennale in Informatica

Fondamenti di Intelligenza Artificiale

Lezione 7 - Algoritmi di ricerca locale (2)

Algoritmi di Ricerca Locale (2)

Materiale aggiuntivo ed esercizi consigliati

Il codice sorgente che implementa un algoritmo genetico sarà disponibile sulla piattaforma e-learning.

A partire da questo (o anche implementandone uno nuovo), risolvere il problema del commesso viaggiatore. Spiegare poi le differenze, in termini di prestazioni, con la soluzione implementata con Simulated Annealing.

Genetic Algorithms in Search, Optimization & Machine Learning, David E. Goldberg, 1989

Link utili:
-https://en.wikipedia.org/wiki/Darwinism
-https://en.wikipedia.org/wiki/Natural_selection
-https://en.wikipedia.org/wiki/Evolutionary_computation
-https://en.wikipedia.org/wiki/Genetic_algorithm
-https://en.wikipedia.org/wiki/Metaheuristic
-https://en.wikipedia.org/wiki/List_of_genetic_algorithm_applications
-https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_introduction.htm
-https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3
-https://towardsdatascience.com/evolution-of-a-salesman-a-complete-genetic-algorithm-tutorial-for-python-6fe5d2b3ca35
-https://www.researchgate.net/post/What_is_meant_by_the_term_Elitism_in_the_Genetic_Algorithm
-https://www.researchgate.net/post/Whats_the_difference_between_the_steady_state_genetic_algorithm_and_the_generational_genetic_algorithm#:~:text=The%20steady%20state%20GA%20is,typically%20half%20the%20individuals)%2C%20the