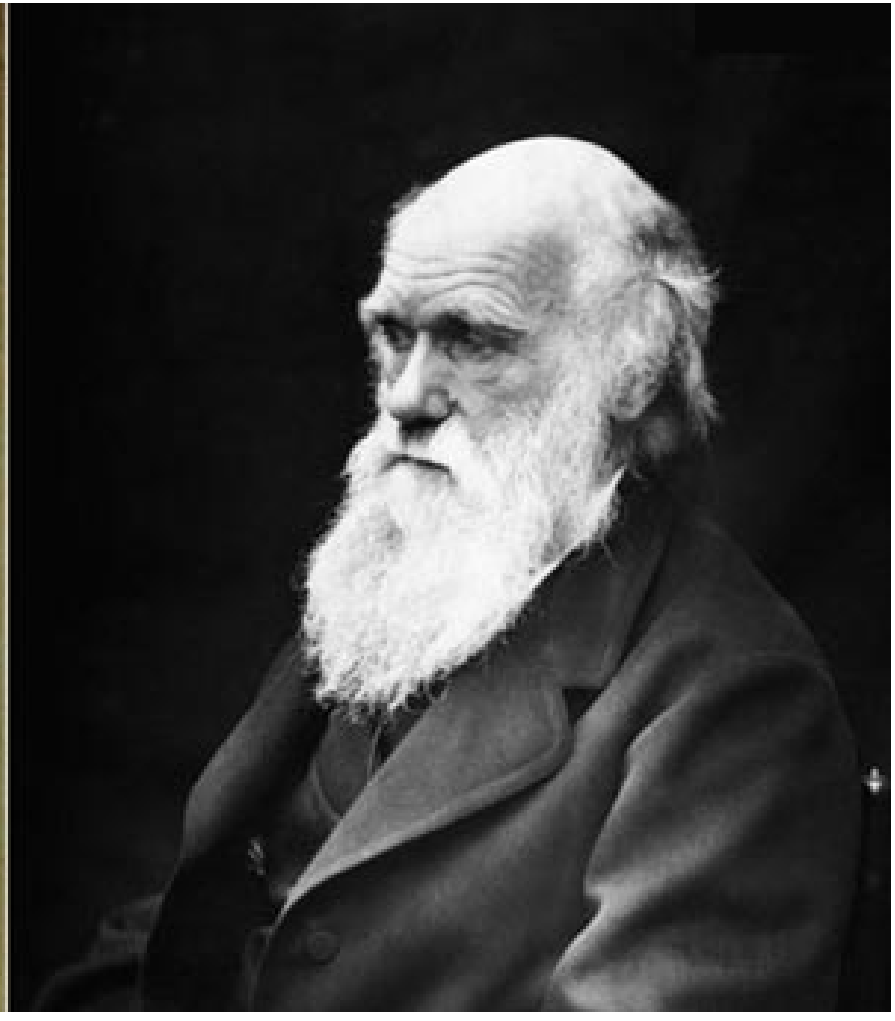
L'evoluzione nelle macchine

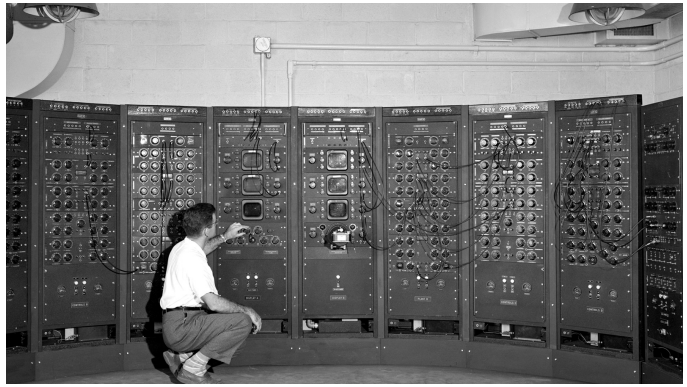
Edmondo Trentin (DIISM)

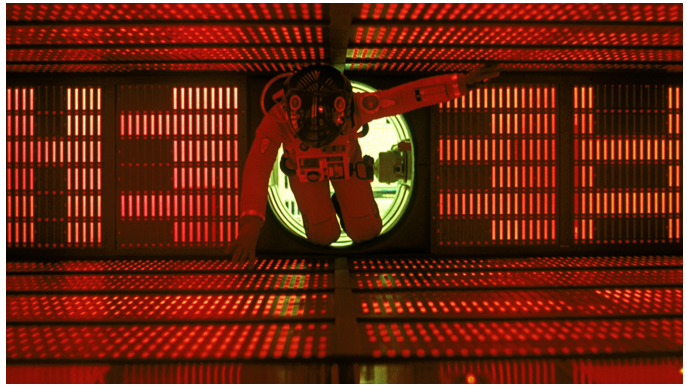
Jean-Baptiste Lamarck (1744 - 1829)



Charles Darwin (1809 - 1882)

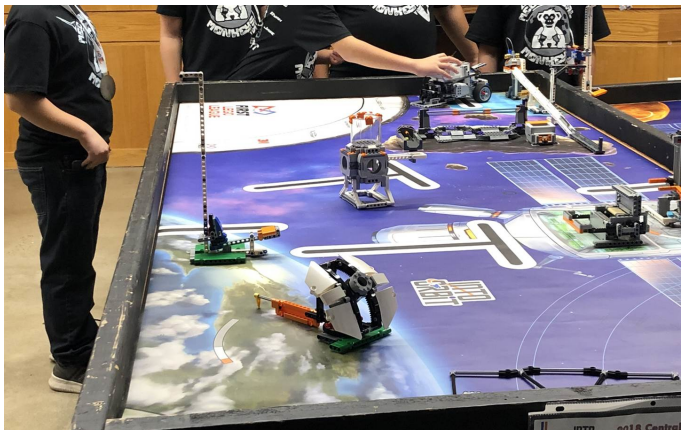
“Giro girotondo ...”





Con la canzoncina *Daisy* intonata dal sintetizzatore vocale Xerox e da HAL9000, *2001 Odissea nello spazio* suggerisce l’idea di una “memoria generazionale” che passa da una **generazione** di computer alla successiva, per via forse genetica o addirittura memetica.

Veicolo 6: la selezione naturale



Prepariamoci con grandi quantità di filo di rame, sensori, elementi a soglia, plastica, motori, ruote, ecc.

- Su un tavolo ricco di stimoli sensoriali collochiamo diversi veicoli tra i più interessanti costruiti finora
- i veicoli si muoveranno (e interagiranno) in base alle proprie circuiterie e in risposta alle stimolazioni dei sensori
- lasciamo perdere quei veicoli che cadono dal tavolo, andando in frantumi: evidentemente essi **sono inadatti alla sopravvivenza nel loro ambiente e a sostenere la lotta con altri veicolui per le risorse** (materia organica, calore, ossigeno, ...)

- quando invece un veicolo ancora sul tavolo ci transita davanti, cerchiamo rapidamente di **copiarne la circuiteria riproducendola nel cervello di un veicolo costruito ex novo**, che metteremo sul tavolo
- **errori di copia** produrranno a volte veicoli meno adatti, che cadranno a terra; altre volte, veicoli **più intelligenti** e vincenti (*ricorda la def. di intelligenza della* ***scuola ecologista***)
- non facendo in tempo a copiare l'intera circuiteria, spesso dovremo copiarne una parte da un veicolo e una seconda parte da un altro, generando **per incrocio** discendenti che potranno essere più o meno intelligenti dei loro genitori
- i veicoli più adatti all'ambiente resteranno più a lungo sul tavolo e daranno vita a un maggiore numero di discendenti, **quelli meno adatti si estingueranno**.

Computazione evolutiva (*evolutionary computation*)

Il Veicolo 6 è implicitamente un esempio di **computazione evolutiva**.

Nella computazione evolutiva si fanno evolvere generazioni successive di macchine, algoritmi o programmi in base a operazioni ispirate ai fondamenti della biologia evolutiva:

- Aleatorietà
- Riproduzione
- Selezione naturale

Esistono numerose branche della computazione evolutiva. Qui ci limiteremo a tre di esse, particolarmente significative:

- Algoritmi genetici
- Programmazione genetica
- Algoritmi memetici

Le tecniche di computazione evolutiva sono specializzazioni del seguente **schema algoritmico generale**:

1. Viene originata una popolosa ($\sim 10^3 - 10^6$) **generazione 0** di individui (macchine, programmi, ...) creati **aleatoriamente**
2. *Loop*: per $t = 1, 2, \ldots$ si produce la **generazione** t**-ima** nel seguente modo
   - 2.1 **sopravvivenza**: una frazione (es. il 10%) degli individui *più adatti* della generazione $(t-1)$-ima sopravvive ed entra a fare parte della generazione t-ima
   - 2.2 una piccola frazione ($< 5\%$) degli individui della generazione t-ima viene ottenuta per effetto di una **mutazione** casuale da altrettanti individui tra i *più adatti* della generazione $(t-1)$-ima
   - 2.3 i rimanenti individui della generazione t-ima vengono ottenuti per incrocio (**crossover**) da due genitori selezionati tra gli individui *più adatti* della generazione $(t-1)$-ima.

Il ciclo viene ripetuto a lungo, fino al raggiungimento di un obiettivo (*goal*) o fino a esaurimento delle risorse di calcolo.

• Saggiamente, si mantiene un **equilibrio demografico** per la durata dell'intero processo (onde evitare estinzione di massa o esplosioni demografiche).

- La misura quantitativa di quanto un individuo sia adatto al proprio ambiente avviene invocando una *funzione di idoneità* (**fitness**) definita ad hoc per lo specifico *task* richiesto (essa è l'equivalente di quella che è la *funzione criterio* per le reti neurali).
- La **selezione degli individui "più adatti"** all'interno di una data generazione può avvenire in diversi modi, sempre valutando la fitness di ciascuno dei candidati.
    - Un metodo popolare e efficace è quello della **selezione "a torneo"**: estrai *a caso* un campione di pochi individui candidati (ad es. $\sim 7 - 23$) e seleziona il vincitore in base al massimo valore della corrispondente fitness.
- La specifica realizzazione (/implementazione) degli operatori di *mutazione* e *incrocio* dipende strettamente dal **come sono rappresentati gli individui**.

Algoritmi genetici

Assumiamo che ogni macchina sia rappresentabile **biunivocamente**[1] attraverso il suo "**DNA**":

0	1	1	1	0	0	1	1	1	0	1	...	0

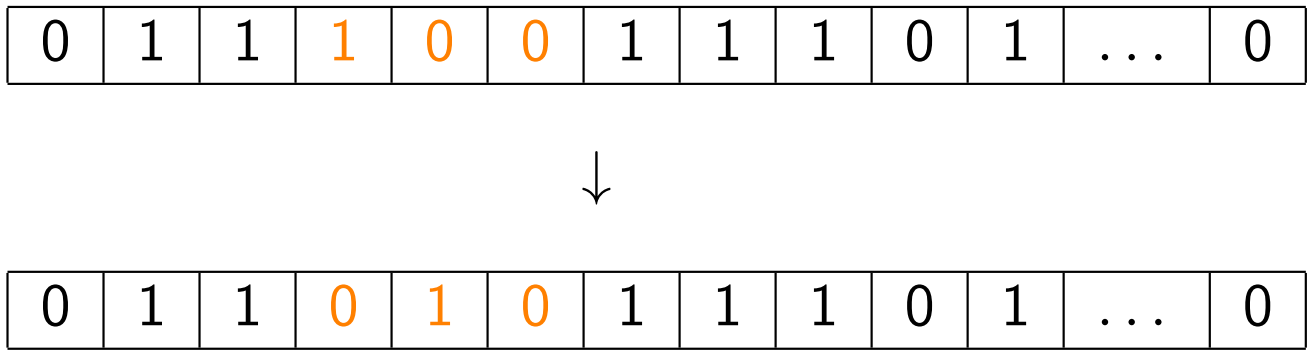
dove il "DNA" è una stringa di caratteri su un certo alfabeto.

Ti pare troppo semplicistico? ☺

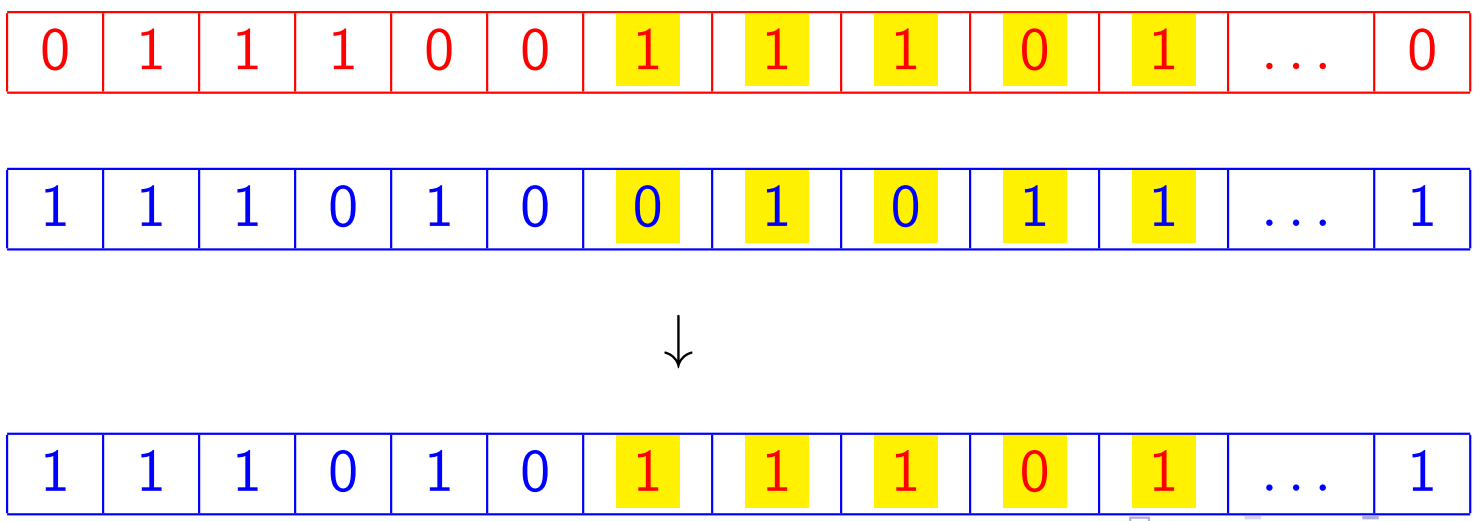
[1] Ogni sequenza di DNA rappresenta dunque una ben precisa macchina.

Operatori

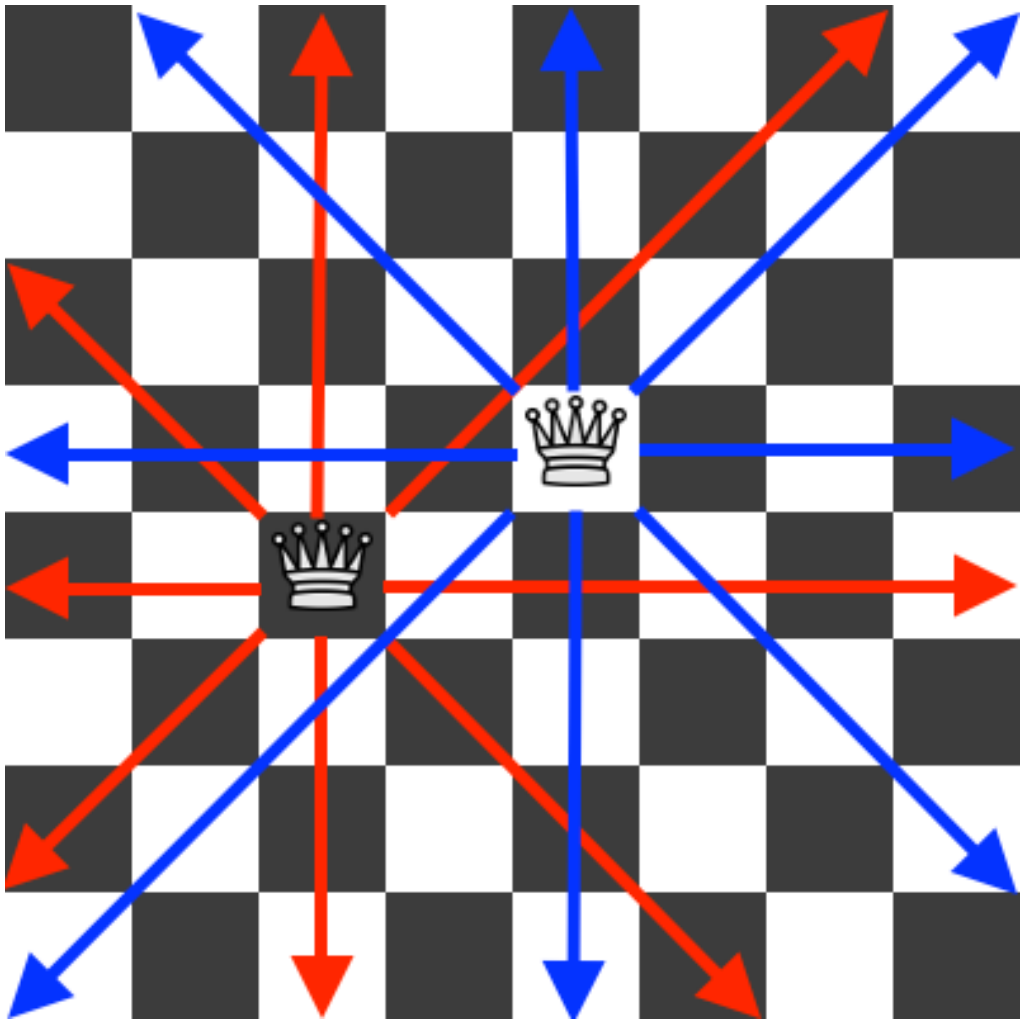
- **Mutazione**: una piccola sotto-stringa del DNA viene scelta a caso e alterata aleatoriamente



- **Crossover**: una sotto-stringa del DNA del *genitore 1* viene scelta a caso e trascritta al posto dell'equivalente sotto-stringa del DNA del *genitore 2*, generando un *discendente*



Esempio applicativo: problema delle 8 regine



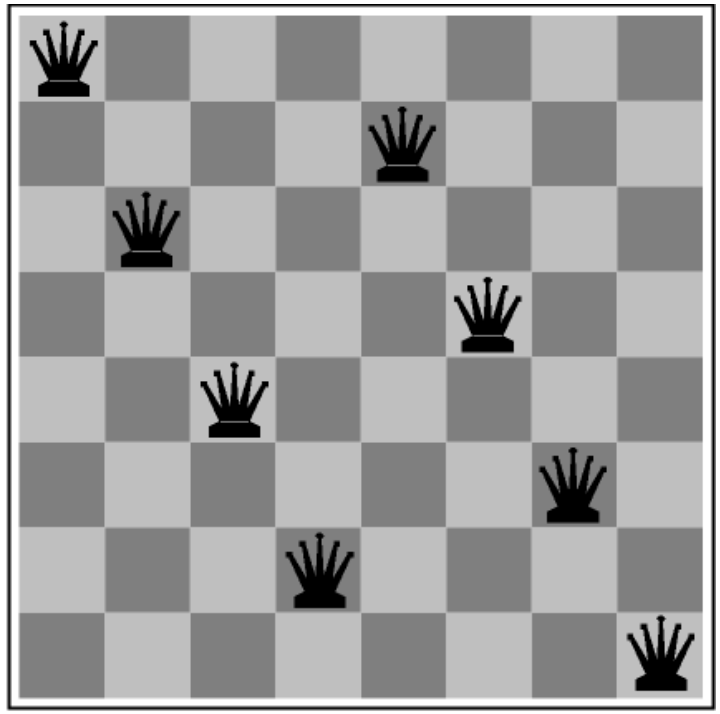
Queste due regine sono disposte sulla scacchiera in modo da **non poter muovere all'attacco** l'una dell'altra.

Un **classico dell'IA** è lo sviluppo di una macchina capace di trovare una soluzione al problema delle **8 regine**:

ovvero: come disporre 8 regine sulla scacchiera in modo che nessuna sia in posizione tale da poterne attaccare un'altra?

- Ci sono 92 soluzioni, ma ...
- ... il numero di **combinazioni possibili (4.426.165.368)** rende impossibile una ricerca esaustiva
- Diverse forme di IA sono state applicate al *task*. Tra di esse, gli algoritmi genetici.

Esercizio: prova a trovare una soluzione al problema delle 8 regine.

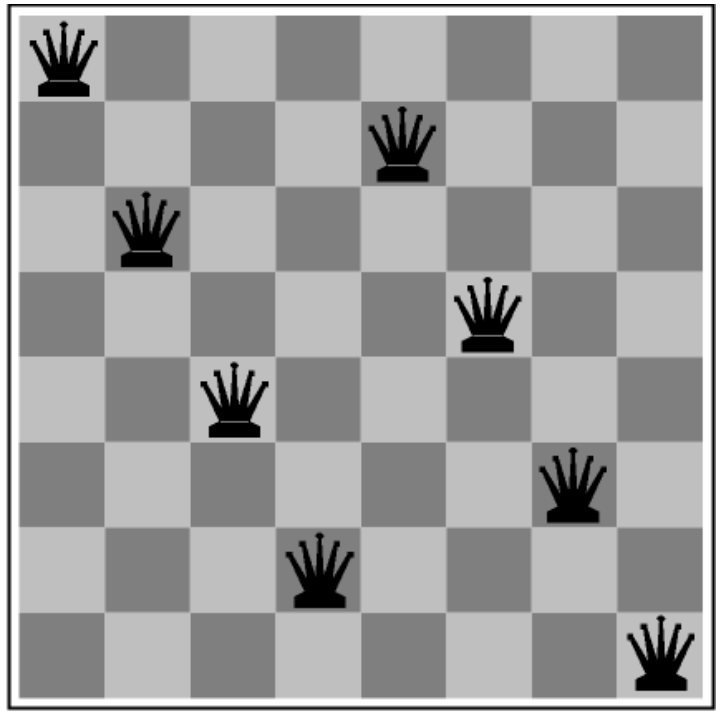


- **DNA**: posizione verticale (a partire dal basso) della regina nella colonna corrispondente

8	6	4	2	7	5	3	1

Alfabeto: $\{1, 2, 3, \ldots, 8\}$

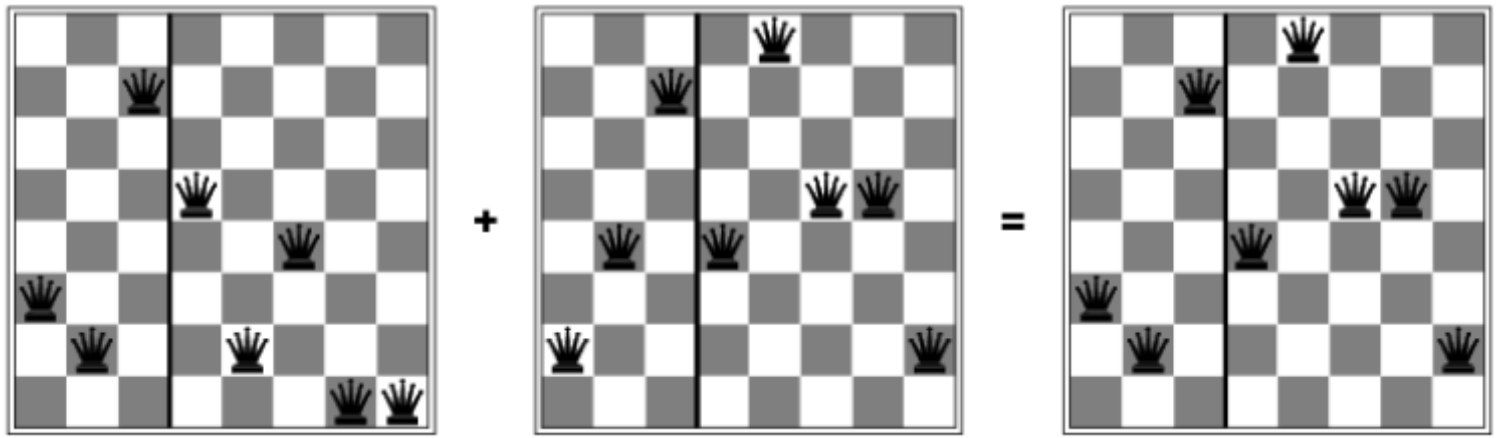
- *Esercizio*: definisci altri tipi di DNA che codifichino univocamente la disposizione delle regine sulla scacchiera.



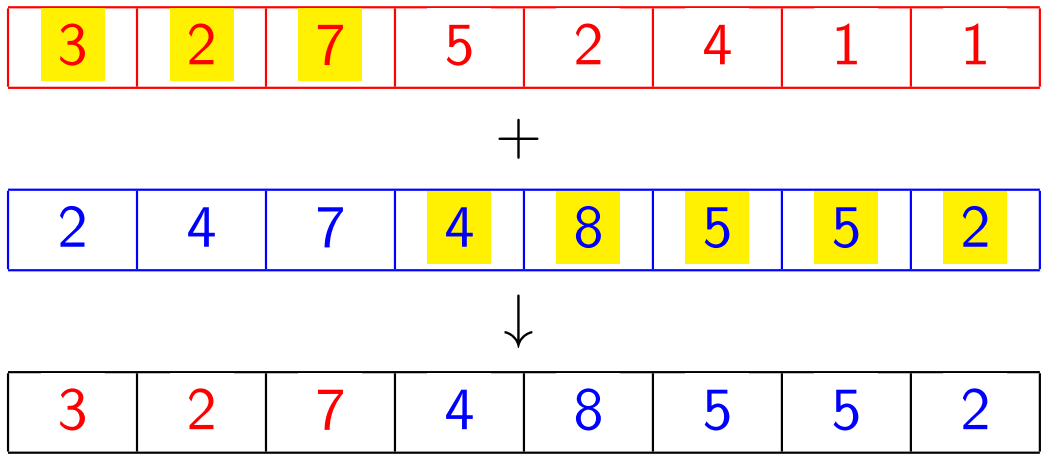
- **Funzione fitness**: numero complessivo di coppie di regine che non sono sotto reciproco attacco.
  - Fitness minima: 0
  - Fitness massima: $\frac{8\times 7}{2}$, cioè 28
  - Nella figura la fitness è 27

- **Crossover**

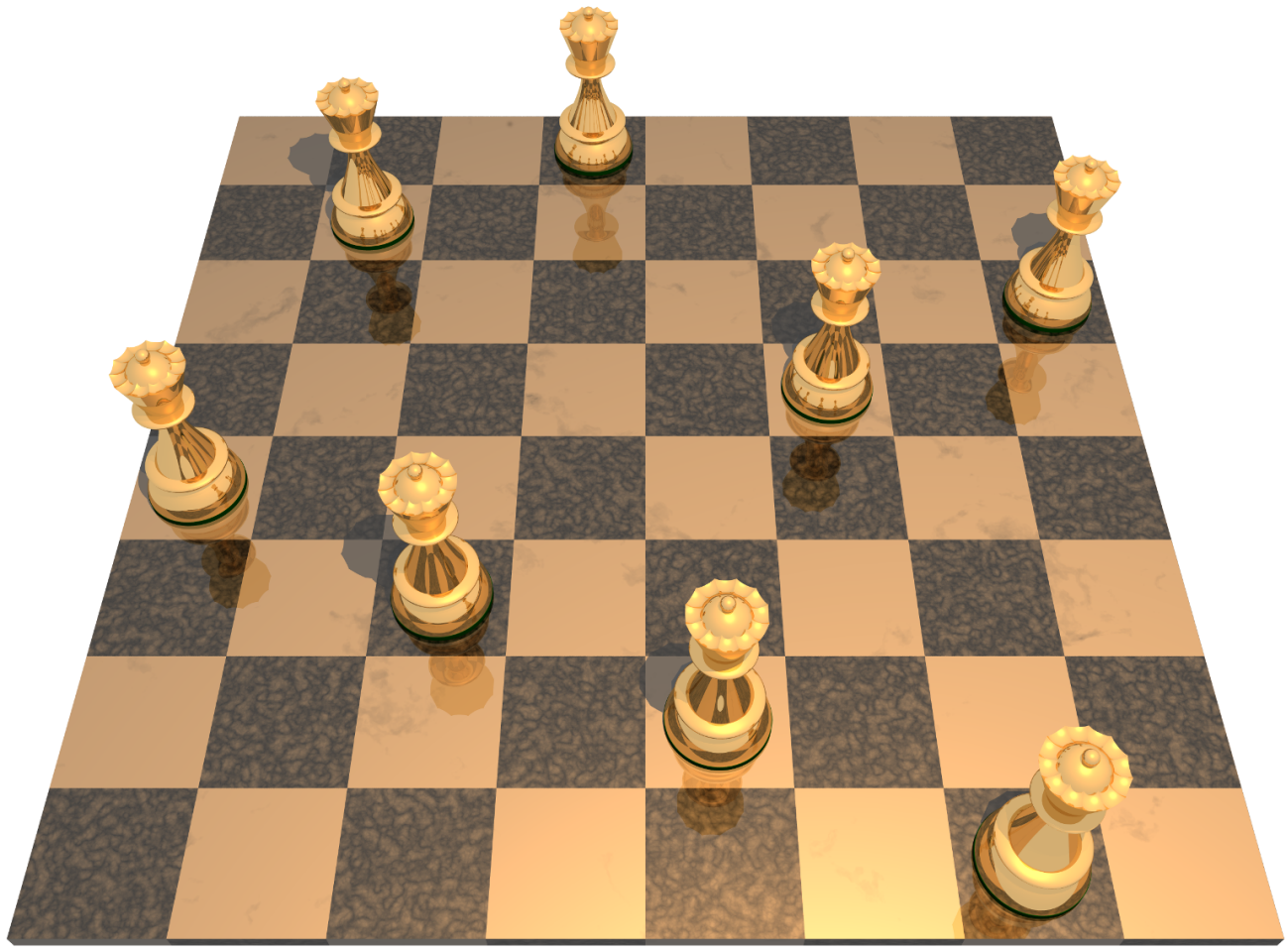
(1) Qualitativamente:



(2) Formalmente:



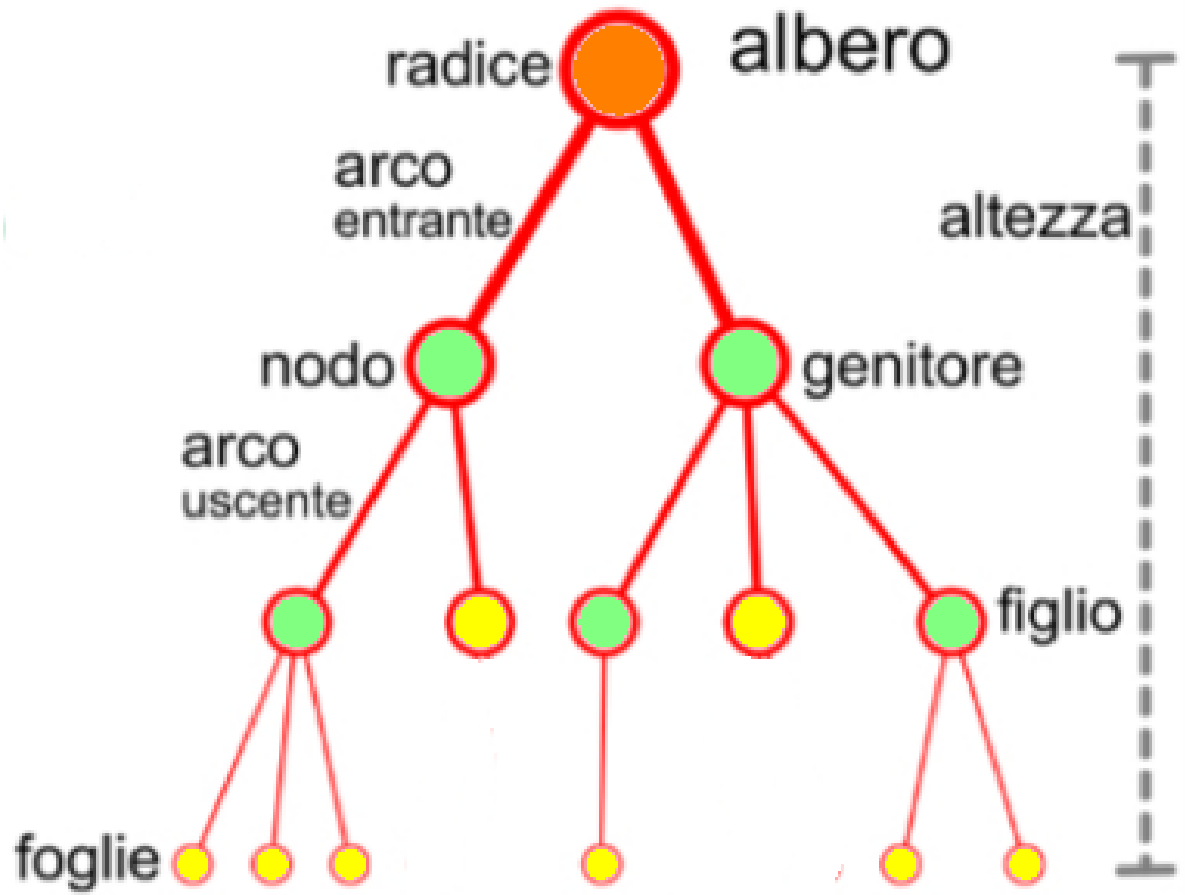
Gli algoritmi genetici sono stati utilizzati con successo per risolvere il problema delle 8 regine, rivelandosi generalmente veloci (poche decine di generazioni sono solitamente sufficienti). Ecco un esempio di soluzione:



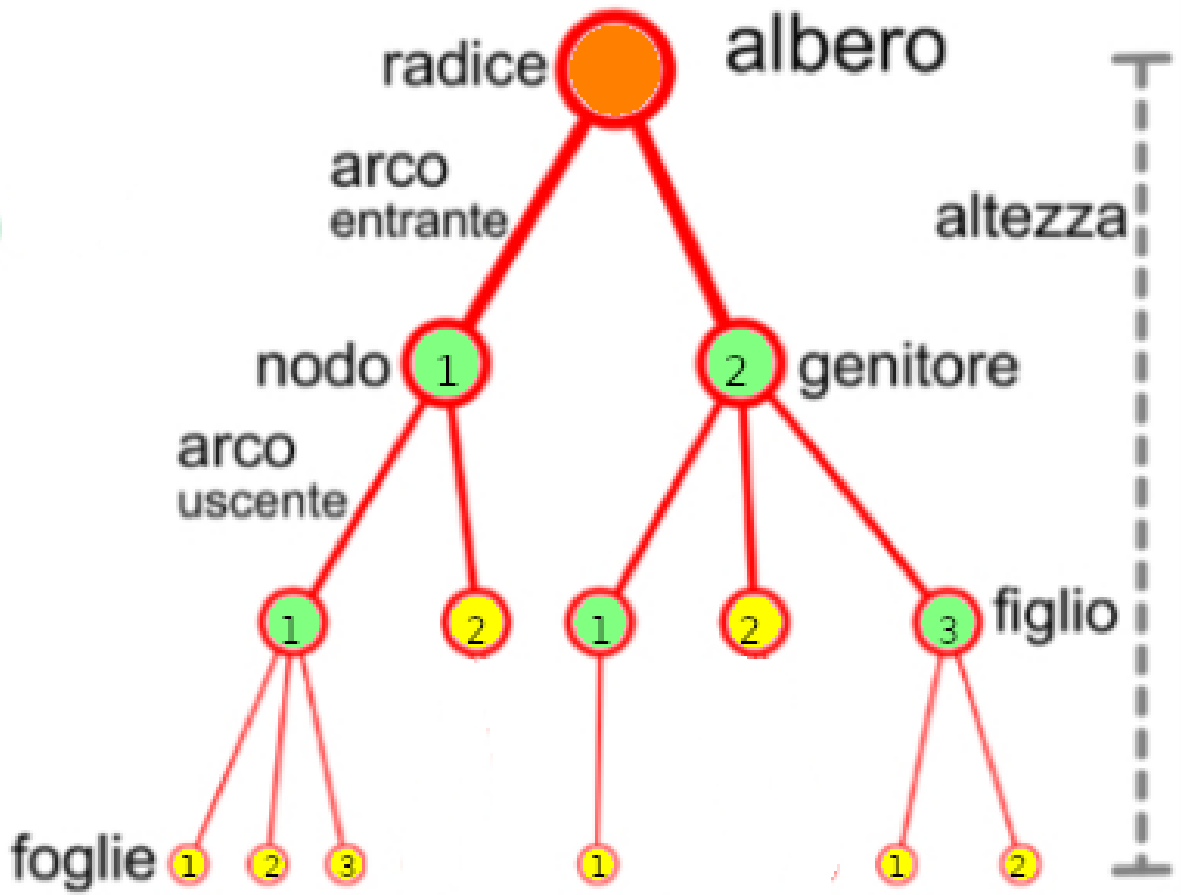
Evoluzione di veicoli di Braitenberg che evitino gli ostacoli

Alberi

Un **albero radicato** è un grafo non orientato in cui (1) per ogni coppia di nodi u e v esiste uno e un solo percorso tra u e v, e (2) uno e uno solo dei nodi è detto *radice*:

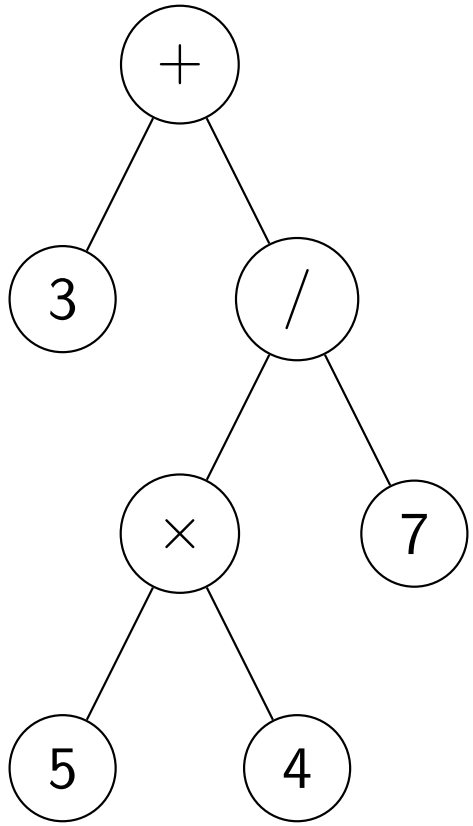


Useremo alberi radicati **ordinati**, cioè alberi radicati per i quali esista un ordinamento (totale) dei discendenti di ciascun vertice. Garficamente rappresentiamo l'ordinamento dei discendenti disegnandoli ordinatamente da sinistra a destra:



Programmazione genetica

Si consideri il seguente albero radicato ordinato:



Esso rappresenta un "programma" per il calcolo di $3 + (5 \times 4)/7$

• *Esercizio*: si scrivano altre espressioni matematiche e si disegnino gli alberi che rappresentano i rispettivi "programmi".

Nella **programmazione genetica** (PG) facciamo evolvere generazioni di **programmi funzionali**, cioè programmi che possono essere rappresentati in forma di albero radicato ordinato in cui

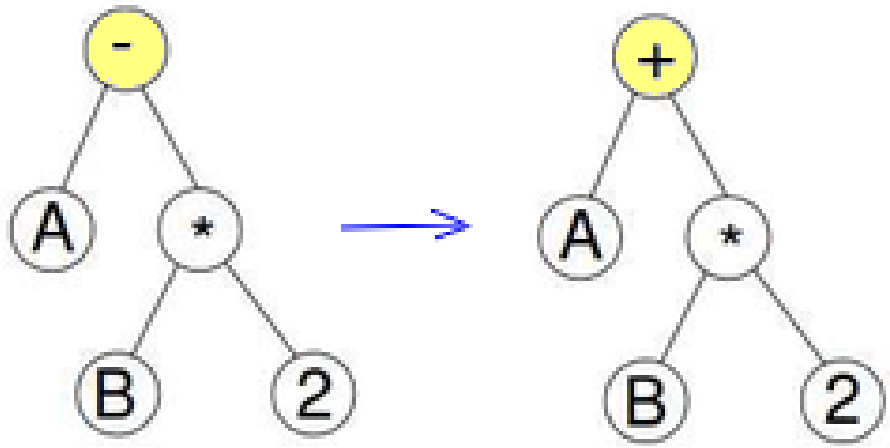
- i nodi interni rappresentano funzioni o azioni definite su uno o più argomenti;
- le foglie rappresentano costanti, oppure funzioni o azioni senza argomenti.

Per usare la PG, è necessario che, **qualunque siano i loro argomenti, tutte le espressioni** e sotto-espressioni presenti in un qualsiasi programma funzionale **assumano un preciso valore** (a meno che l'esecuzione di un'espressione comporti esplicitamente la terminazione del programma).

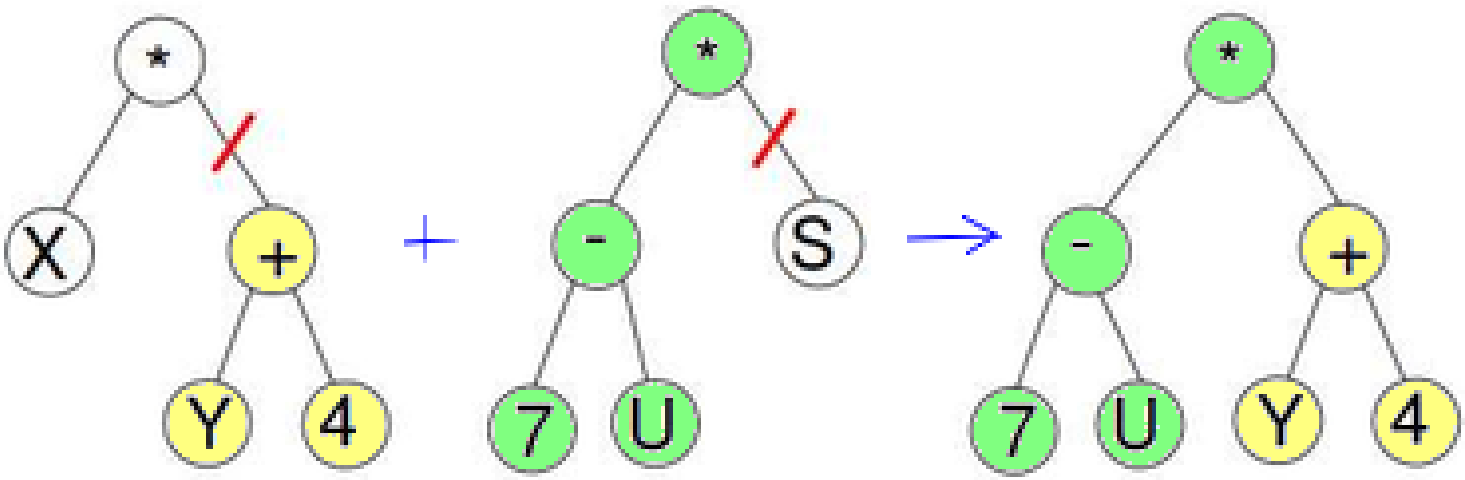
• In questo modo, si garantisce che **un qualsiasi albero** costruito in modo che i nodi relativi alle diverse **funzioni abbiano il corretto numero di figli** (cioè, di argomenti per le rispettive funzioni) **rappresenti un programma** sintatticamente corretto ed effettivamente **eseguibile**.

PG: operatori

Mutazione

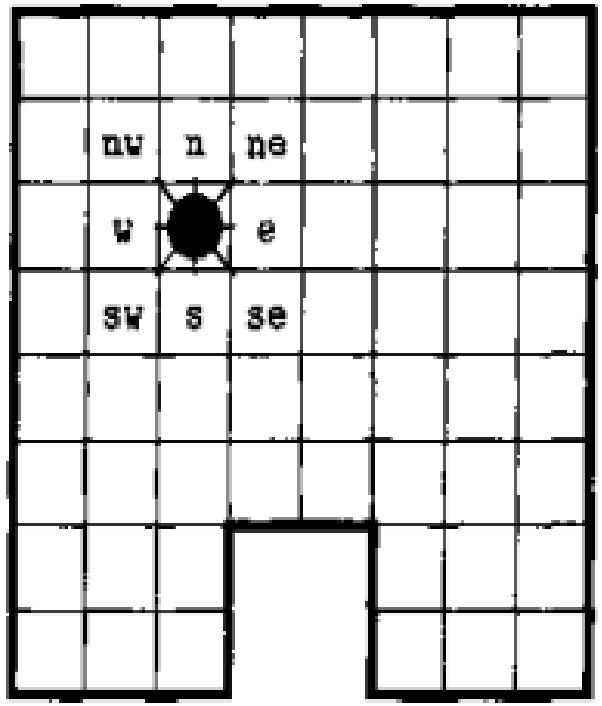


Crossover



Esempio: navigazione autonoma nel mondo a griglia

Il *task* è la circumnavigazione in senso orario del perimetro interno del mondo a griglia.

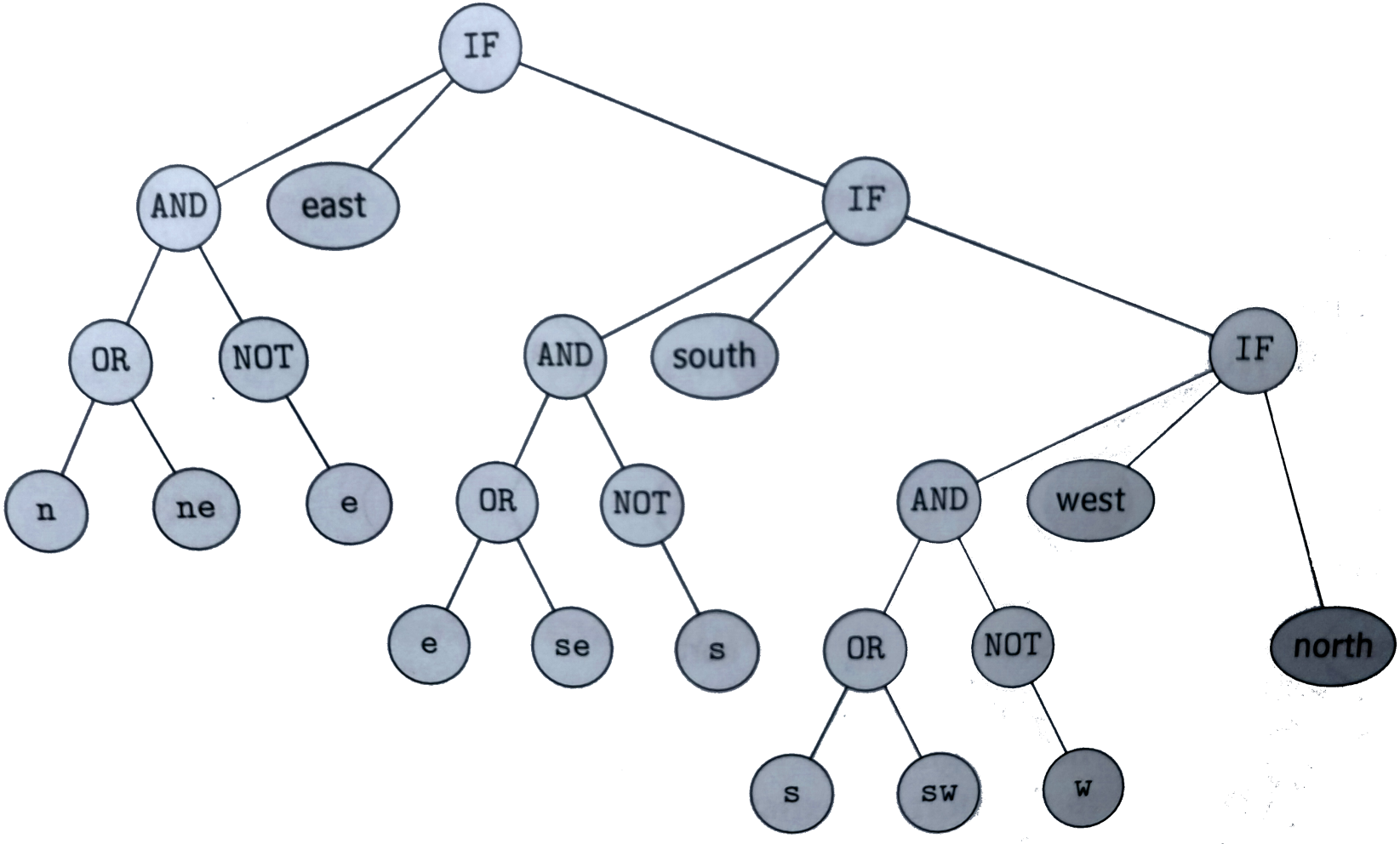


Per motivi mnemonici il Nilsson usa gli acronimi *nw*, *n*, *ne*, ... per denotare i segnali sensoriali booleani $s_1, s_2, s_3, \ldots$.

• PG: faccio evolvere programmi che controllino la navigazione.

• Ogni programma realizza un ciclo percezione - azione (S-R) e viene eseguito ripetutamente per realizzare il *task*.

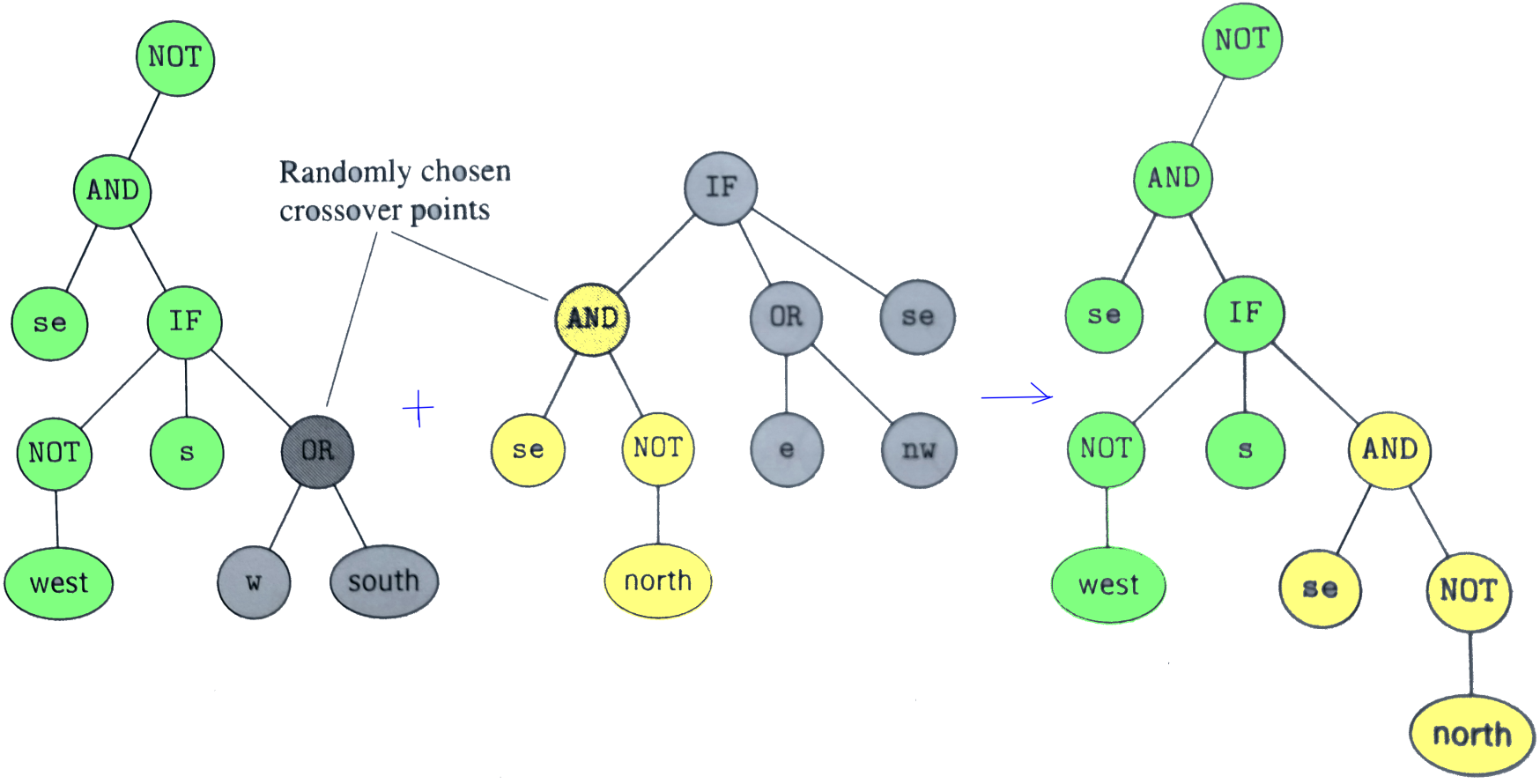
Esempio di programma che realizza la strategia di navigazione:



L'esecuzione delle azioni (*north*,*east*, *south*, e *west*) non produce valori da propagarsi ai nodi genitore nell'albero.

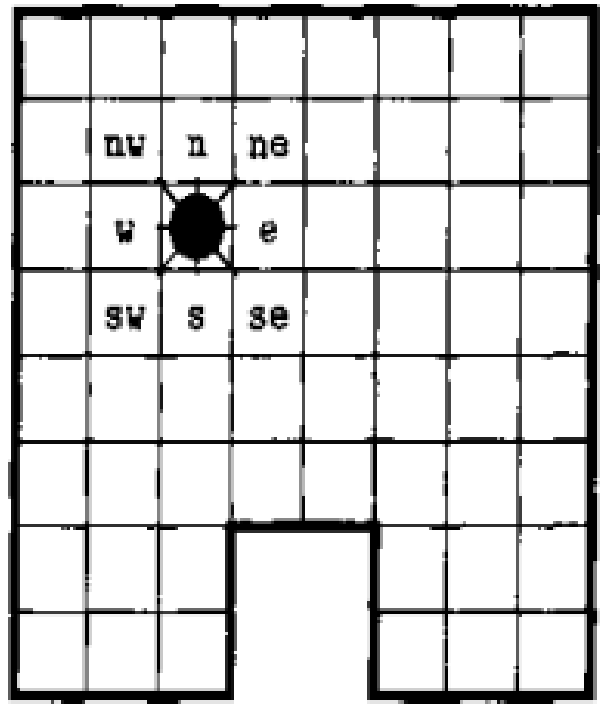
• I nodi "IF" corrispondono alla funzione di 3 argomenti IF(x, y, z) (i 3 figli) che significa "`IF x = 1 THEN y, ELSE z`"

Esempio di crossover tra programmi:



Esercizio: disegnare altri esempi di crossover e verificare che effettivamente si ottengono nuovi programmi eseguibili.

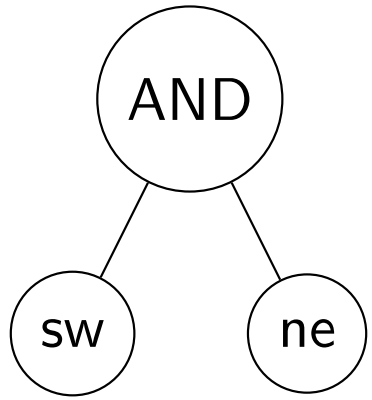
La **fitness** di un programma è definita come segue.



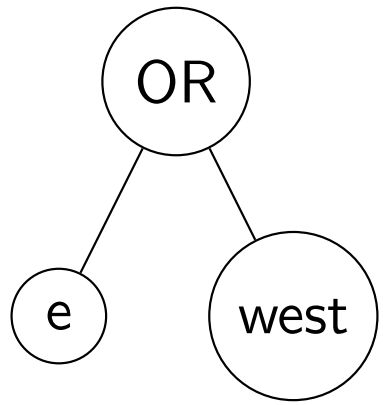
- il **programma viene valutato eseguendolo per 60 volte** di seguito e contando il numero di celle lungo il muro visitate dal robot ($\min = 0, \max = 32$);
- la **valutazione è ripetuta 10 volte**, posizionando ogni volta il robot in una cella iniziale a caso;
- la ***fitness* è definita come la somma delle valutazioni** ottenute sui 10 *run* casuali ($\min = 0, \max -320$).

Risultati della simulazione

- È stata fatta evolvere una **popolazione di 5000 programmi**.
- Esempi di **programmi di generazione 0**:

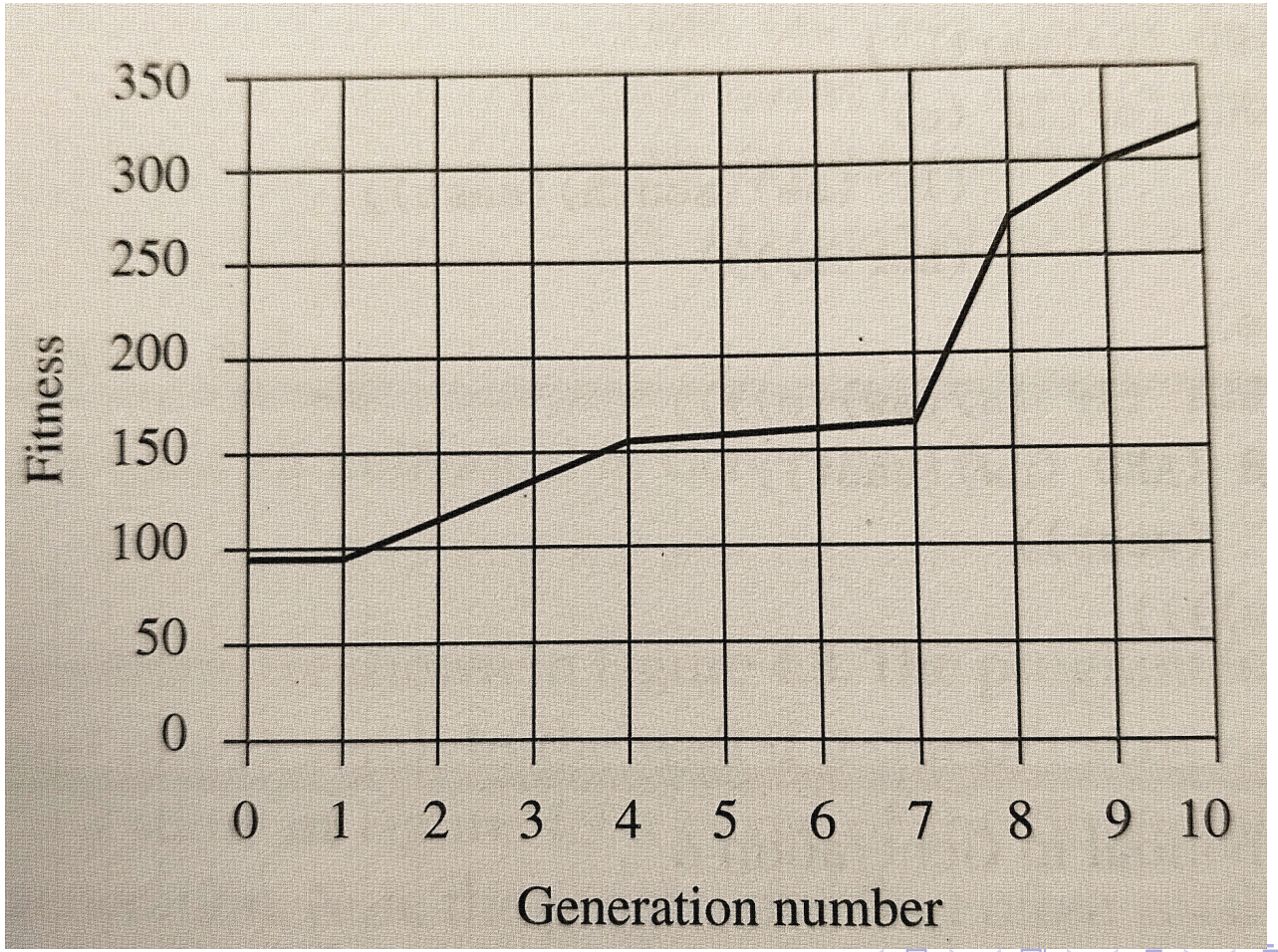


(non fa nulla, *fitness* = 0)



(valuta il valore di "*west*", si muove a ovest e termina; *fitness* = 5)

- **Dopo 10 generazioni** è comparso un programma (molto complicato e ridondante) che si muove verso sud fino a raggiungere il muro e da lì prende a **muoversi perfettamente lungo il muro in senso orario**.
- Grafico dell'andamento della *fitness* del miglior programma di ogni generazione:



Algoritmi memetici

Richard Dawkins (1941 -)

The Selfish Gene, 1976: introduce il concetto di meme

Memetica (evoluzione culturale)

- Il **meme** è, come il gene, un *replicatore*
- Esso è "vivo" e si replica (subendo mutazioni e incroci) passando, come parte di un *pool* memetico, **da un cervello all'altro**
- Ogni meme codifica una singola **unità** funzionale **culturale** (un'idea, una conoscenza pratica, una credenza, un pregiudizio, ...)
- Tale unità si può dunque **trasmettere** tra animali della specie *homo sapiens*, e **tra generazioni successive** di umani
- L'**evoluzione** può portare in parallelo allo sviluppo "simbiotico" di geni e meni, come nel caso del **modulo di Dio** presente nella neocorteccia e del replicarsi della credenza nella vita oltre la morte.

Negli **algoritmi memetici** (AM) si realizzano in parallelo tanto un'evoluzione "genetica" di una popolazione di macchine, quanto un meccanismo di evoluzione "memetica".

La letteratura sugli AM è frastagliata, esistono varianti e tecniche molto diverse tra loro (alcune sono assai complesse).

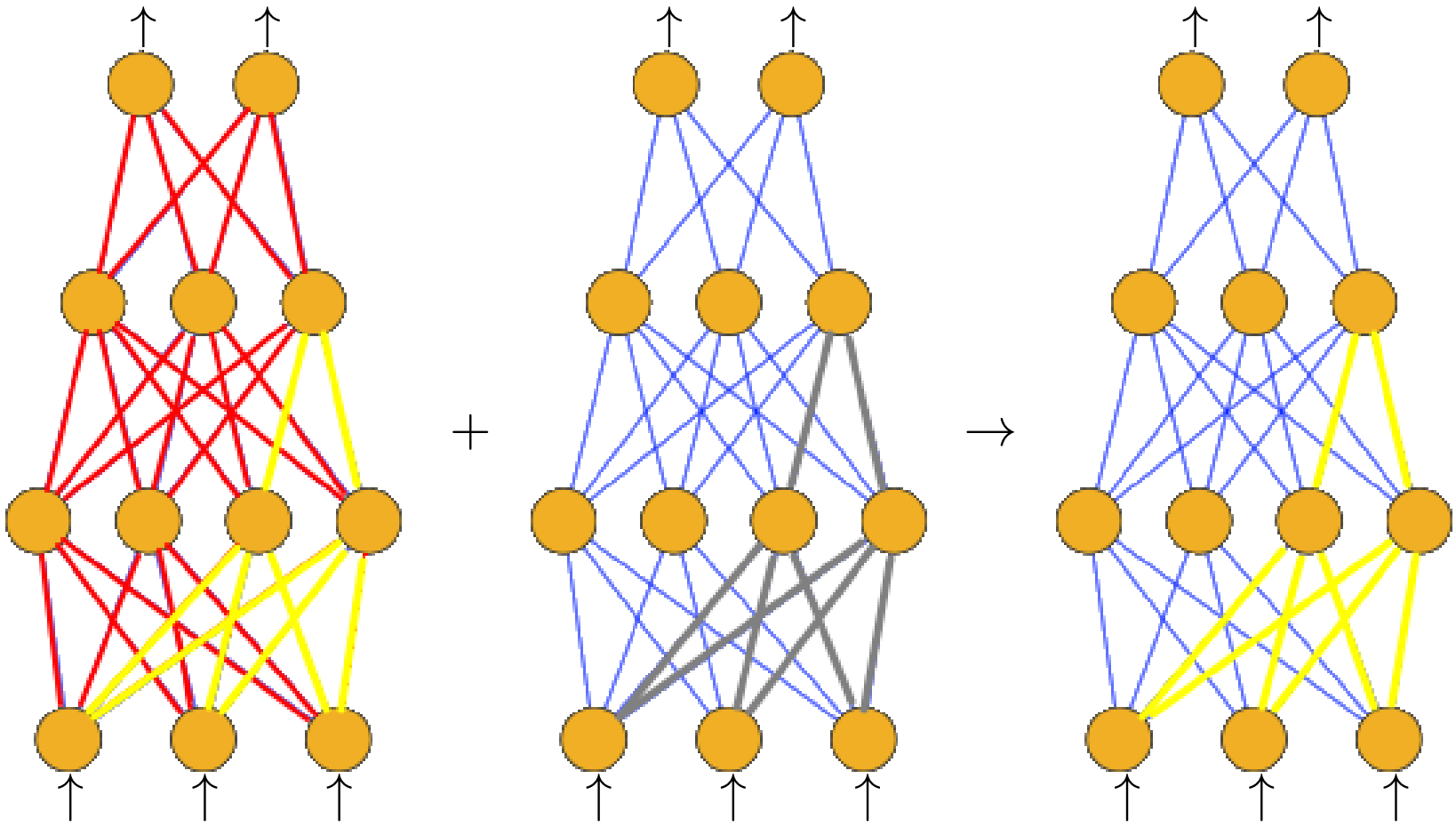
Qui ci limitiamo allo schema base dei cosiddetti **AM multi-meme**. Lo schema algoritmico è una variante dell'algoritmo generale della computazione evolutiva.

Esercizio: immagina un certo numero di memi che potresti avere inconsapevolmente ricevuto, e immagina in che modo potrebbero esserti stati passati.

Schema generale dello AM multi-meme:

1. Viene creata una **generazione 0** di individui con geni e memi generati aleatoriamente
2. *Loop*: per t $= 1, 2, \ldots$ si produce la generazione t-ima per **sopravvivenza, crossover e mutazione** da individui selezionati (ad es. a torneo) nella generazione $(t-1)$-ima
   - 2.1 I **memi vengono "ereditati" da una generazione alla successiva attraverso il genotipo** (il DNA), secondo un **criterio Lamarckiano**
   - 2.2 tra i nuovi individui di generazione t viene **selezionato un gruppo ristretto** (ad es. a torneo) che viene **raffinato attraverso un processo individuale di apprendimento** di uno dei seguenti due tipi:
     - 2.2.1 **per imitazione** di memi scelti a caso da individui selezionati della generazione $(t-1)$-ima
     - 2.2.2 **per ottimizzazione di una funzione criterio** (che può coincidere con la *fitness*)

AM: **evoluzione di popolazioni di reti neurali**. Es. di *crossover*:



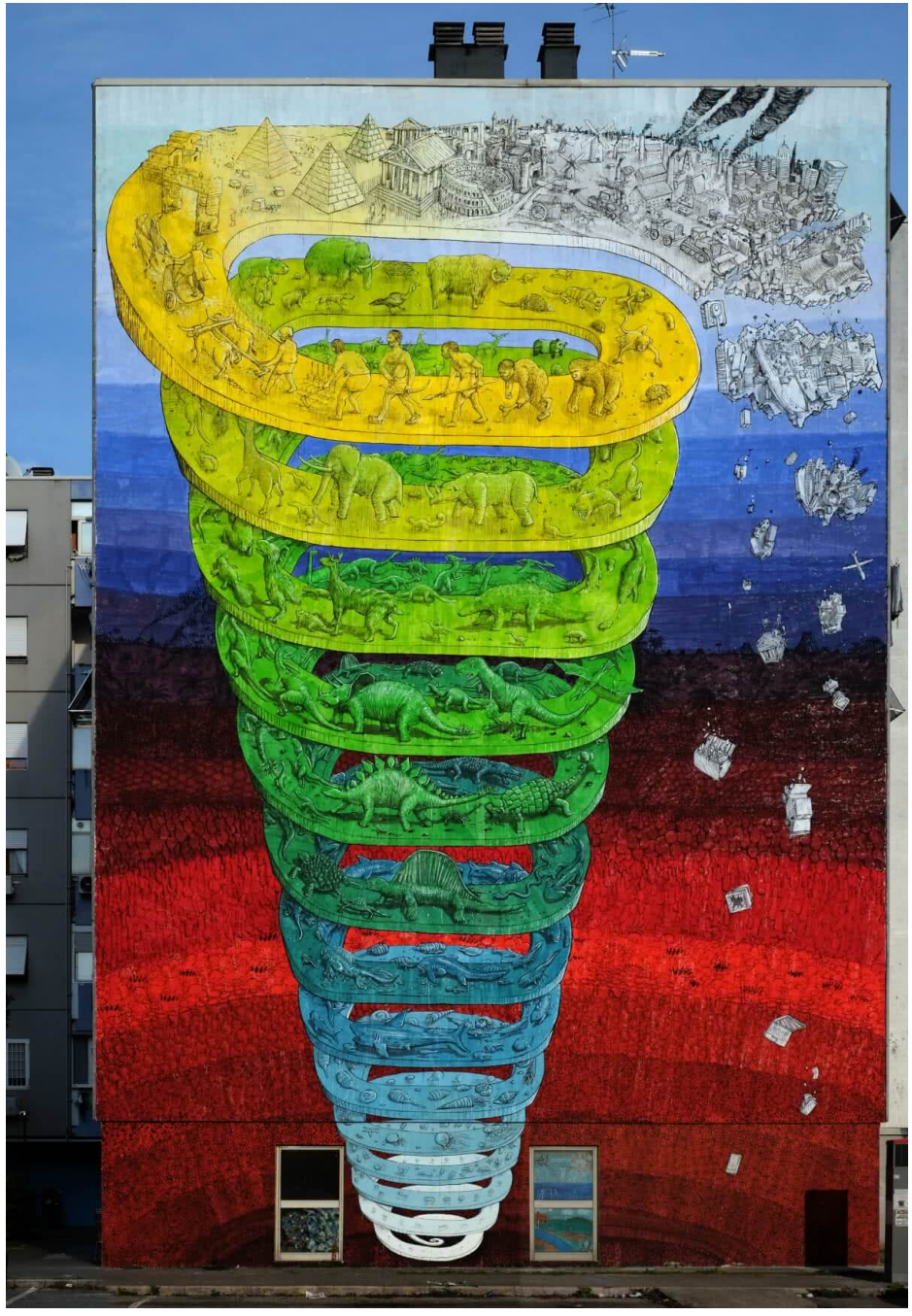
• Una **sotto-rete** del genitore 1 viene sostituita a una **sotto-rete** del genitore 2

• I **memi** sono codificati nei **pesi** di rete 2 e **sotto-rete 1**

• Come nella PG, la **sotto-rete 1** e la **sotto-rete 2** possono essere **grafi diversi**.

Applicazioni degli AM

- **Pianificazione dei flussi** nelle filiere tecnologico-produttive
- **Riconoscimento di forme** (*pattern recognition*) quali caratteri manoscritti o figure geometriche
- **Pianificazione della navigazione autonoma** e della manipolazione di oggetti da parte di piattaforme robotiche
- **Controllo dell'orientazione dei fasci di radiazioni** ionizzanti in radioterapia
- **Progettazione automatica di circuiti** integrati VLSI
- **Generazione automatica di orari** (ad es. l'orario delle lezioni di una scuola) o di un calendario di eventi (ad es. quello della *National Hockey League*)
- **Pianificazione dei turni** di lavoro in grandi aziende
- **Selezione di *feature*** (per ridurre la dimensionalità dello spazio delle *feature*)



Spirale (Blu, murale - Roma sud)

Esercizio: guarda su YouTube i *murales* animati *Big Bang Big Boom* e *Evoluzione* (“Evolution of Men”) di Blu.