Intelligenza Artificiale

Introduzione al Genetic Programmimg

Marco Piastra

Classificazione del GP

- In base alle dimensioni dei sistemi di calcolo evolutivo

Rappresentazione	Strutture ad albero
Ricombinazione	Scambio di sotto-alberi
Mutazione	Cancellazione e ricostruzione di sotto-alberi
Selezione	Torneo
Processo evolutivo	Generazionale

- Vi sono comunque molte varianti ...

Strutture sintattiche e alberi

- Universalità
  - Qualsiasi cosa abbia una sintassi formale (una grammatica) può essere rappresentata in forma di albero

1. Espressioni aritmetiche

$$2 \cdot \pi + \left((x+3) - \frac{y}{5+1} \right)$$

2. Formule logiche

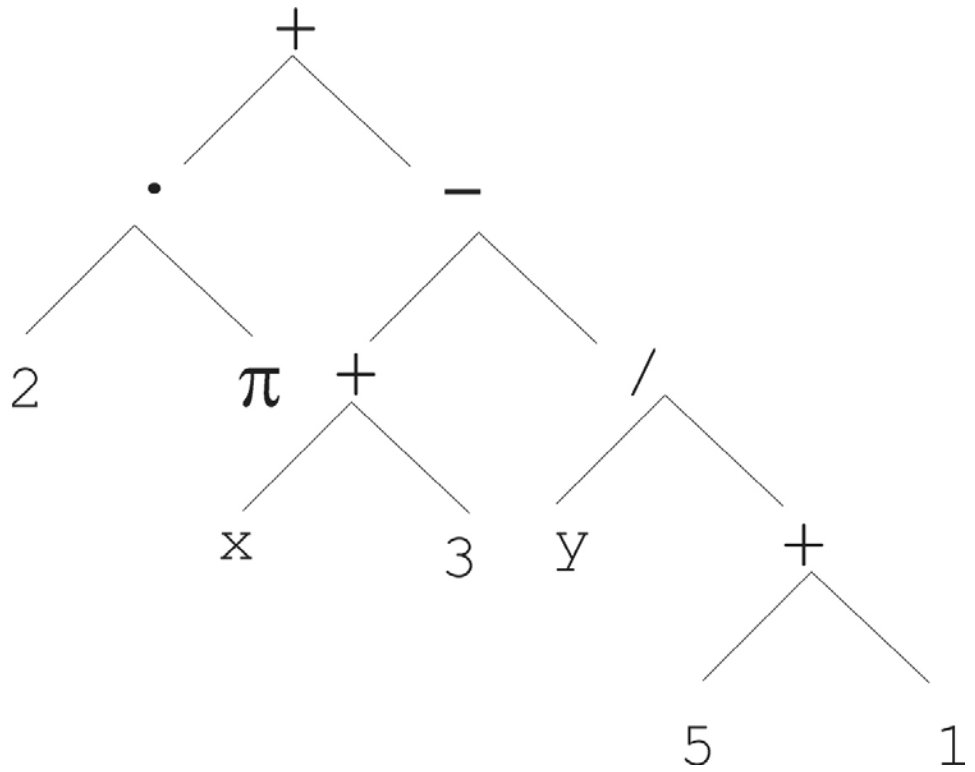
$$(\mathrm{x} \wedge \mathrm{z}) \rightarrow ((\mathrm{x} \vee \mathrm{y}) \vee (\mathrm{z} \leftrightarrow (\mathrm{x} \wedge \mathrm{y})))$$

3. Programmi

```
i =1;
while (i < 20) {
     i = i +1
}
```

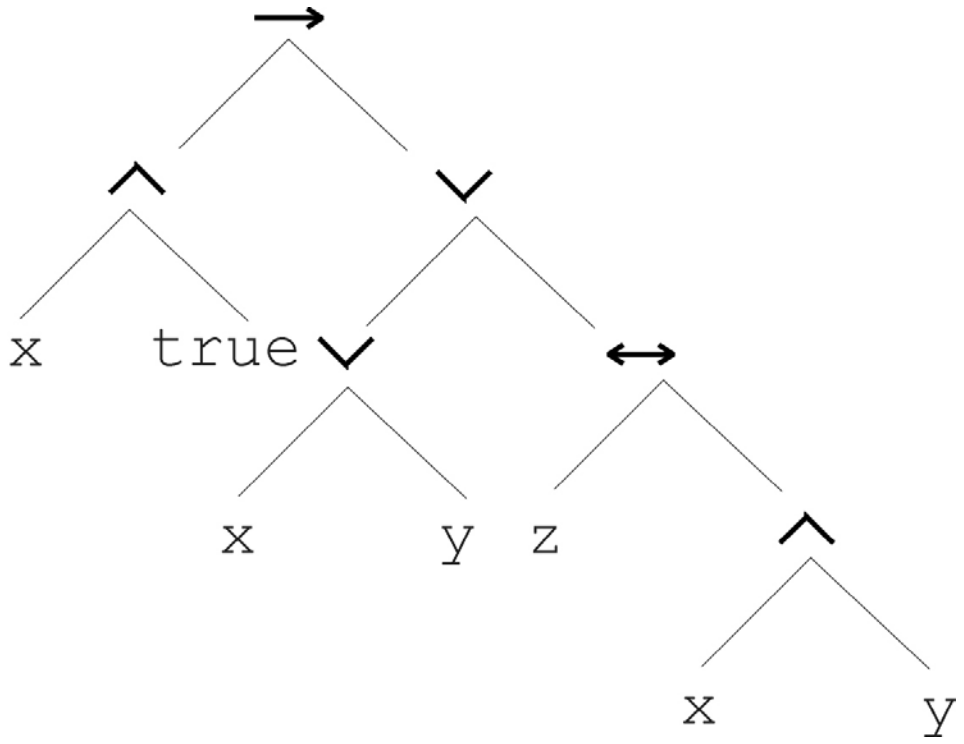
Esempio 1

$$2 \cdot \pi + \left((x+3) - \frac{y}{5+1} \right)$$



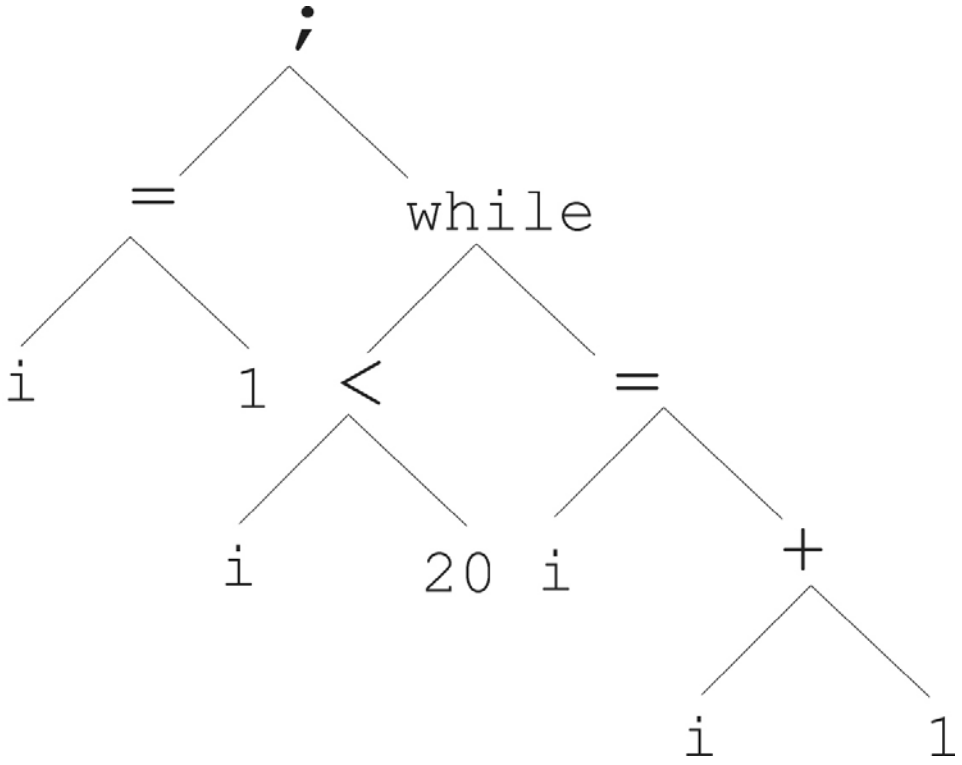
Esempio 2

$$(x \wedge z) \rightarrow ((x \vee y) \vee (z \leftrightarrow (x \wedge y)))$$



Esempio 3

```
i =1;
while (i < 20) {
      i = i +1
}
```

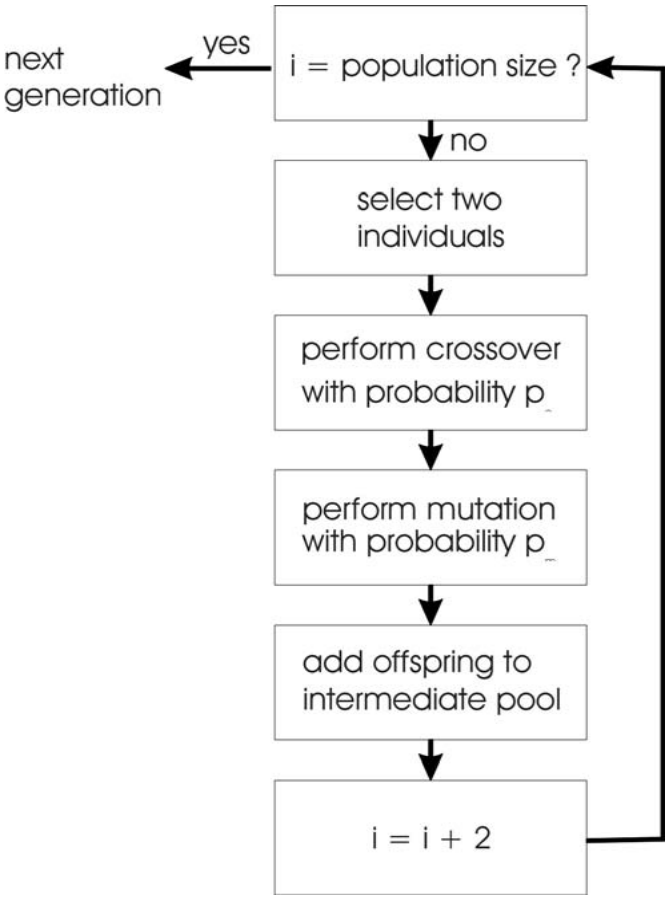


GP ed alberi

- Una tecnica di calcolo evolutivo per le **strutture ad albero**
  - Le strutture ad albero rappresentano *espressioni* in una *sintassi*
  - Gli operatori genetici (*mutazione*, *ricombinazione*) agiscono direttamente sulle strutture ad albero
    - Praticamente assente la differenza tra *genotipo* e *fenotipo*
- Valutazione della *fitness*
  - Le strutture ad albero hanno una *semantica operazionale*
    - Descrivono operazioni da compiere
      - p.es. il programma di controllo di un robot
  - La valutazione della *fitness* è indiretta
    - Si valuta la struttura ad albero
    - La funzione di fitness si applica ai risultati della valutazione
- Lo spazio delle possibili soluzioni
  - p.es. lo spazio delle espressioni aritmetiche
  - Troppo vasto ed articolato per uno studio analitico esaustivo

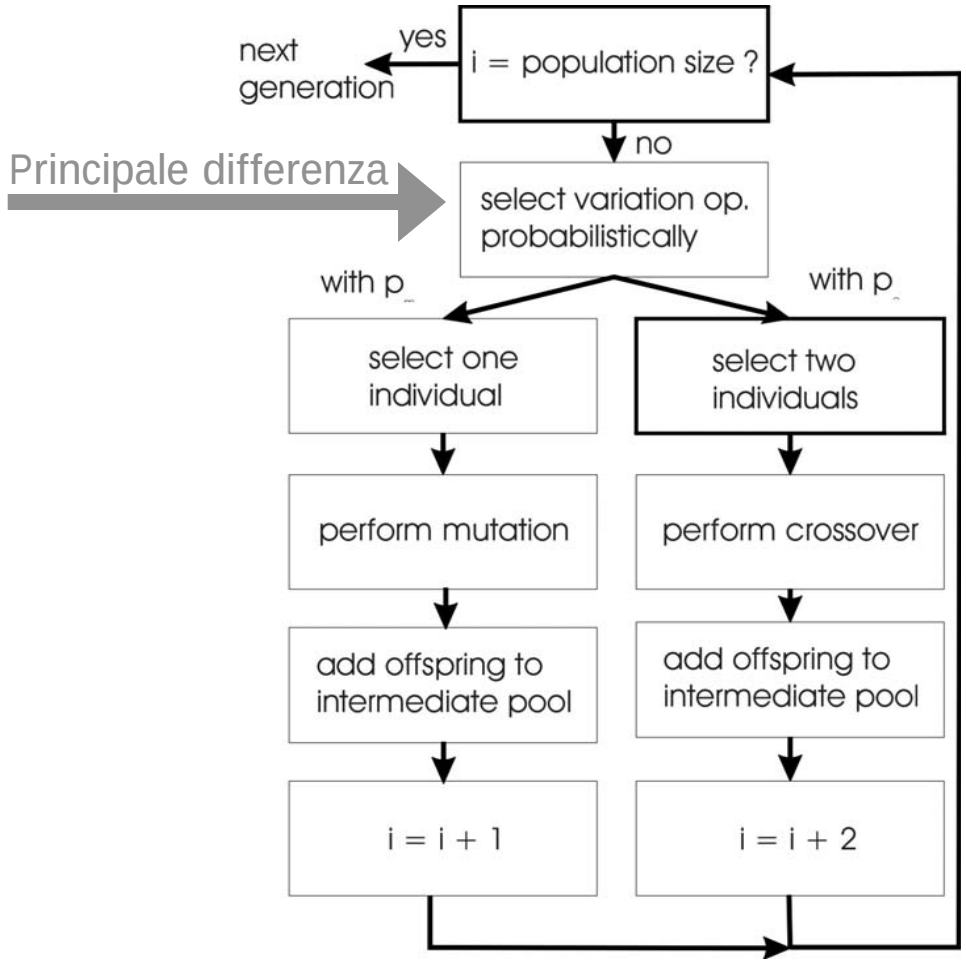
GP e processo evolutivo

Genetic Algorithms (tipico)



Genetic Programming (tipico)

Principale differenza



Generazione di strutture ad albero

- Algoritmo di generazione
  - Due insiemi di simboli:
    - Simboli non teminali NT, con *arità* $\{+, -_{(2)}, *, /, -_{(1)}\}$
    - Simboli terminali T $\{x, y, z, \pi, ...\}$
  - Limite di profondità massima D_{max}
  - Si genera l'albero in modo incrementale
    - A partire dal nodo radice, scegliendo i nodi a caso in NT e T
- *Full method*
  - Ogni nodo a profondità < D_{max} è scelto in NT
  - Ogni nodo a profondità = D_{max} è scelto in T
- *Grow Method*
  - Ogni nodo a profondità < D_{max} è scelto in NT $\cup$ T
  - Ogni nodo a profondità = D_{max} è scelto in T
  - Gli alberi risultanti possono avere profondità < D_{max}

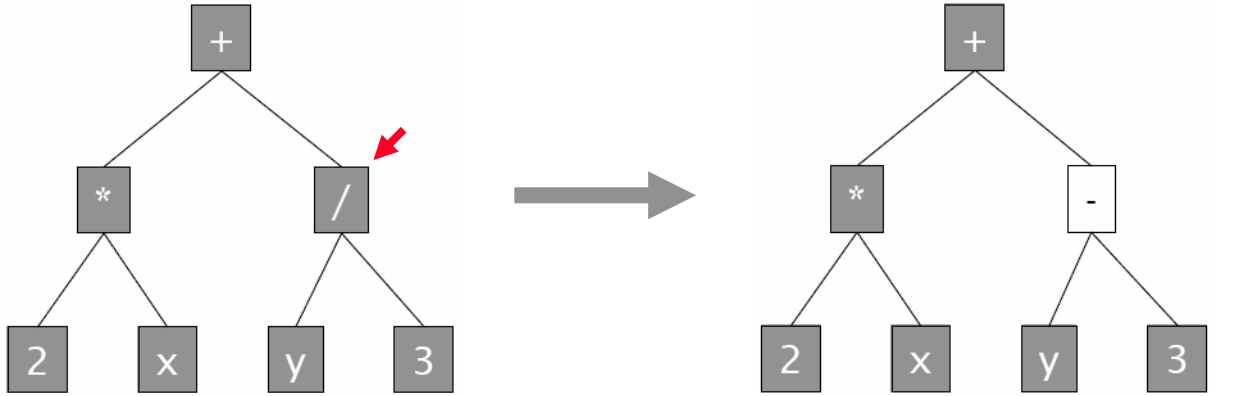
Strategie di generazione

- Popolazione iniziale
  - Strategia classica nel GP: *ramped half-and-half*
    - 50% di individui generati con il *Full Method*
    - 50% di individui generati con il *Grow Method*

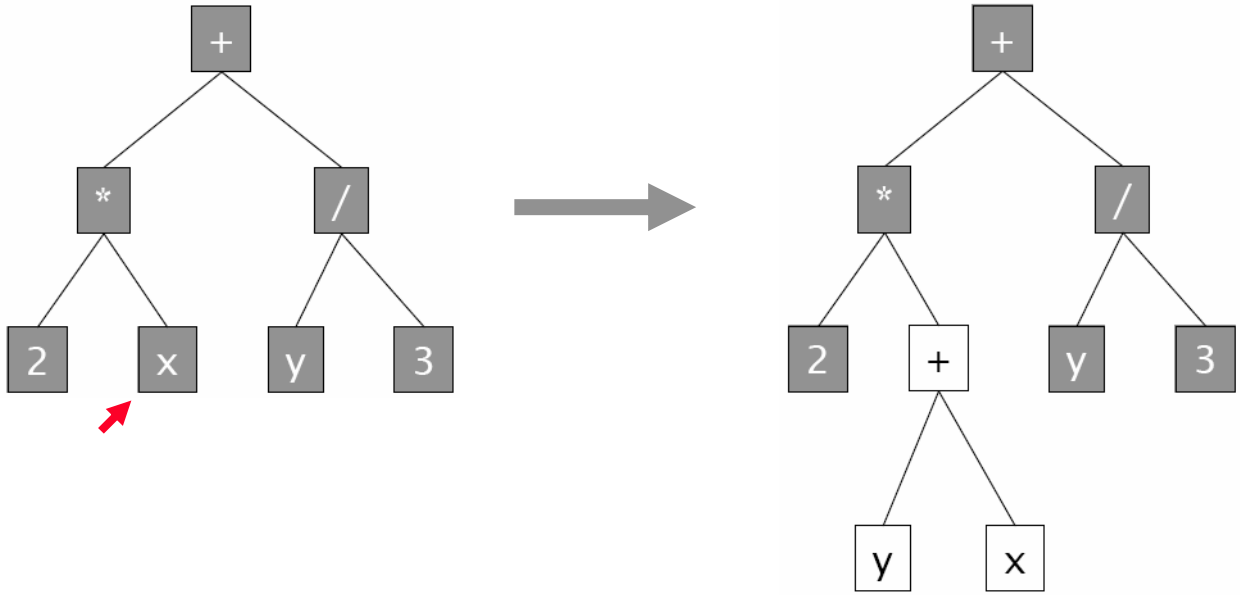
- Sintassi semplici e limitazioni semantiche
  - L'algoritmo di generazione funziona solo per le sintassi **typeless**
    - A parte i limiti di profondità
    - Qualsiasi elemento di NT ∪ T può essere scelto ad ogni passo dell'algoritmo di generazione
    - Le operazioni devono essere protette (p.es. divisione per zero)
  - La maggior parte dei linguaggi di programmazione sono invece **strongly-typed** (p.es. C, C++, Java, SQL)

Operatori di mutazione

- **Sostituzione**
  - Selezione di un nodo
  - Alterazione del nodo

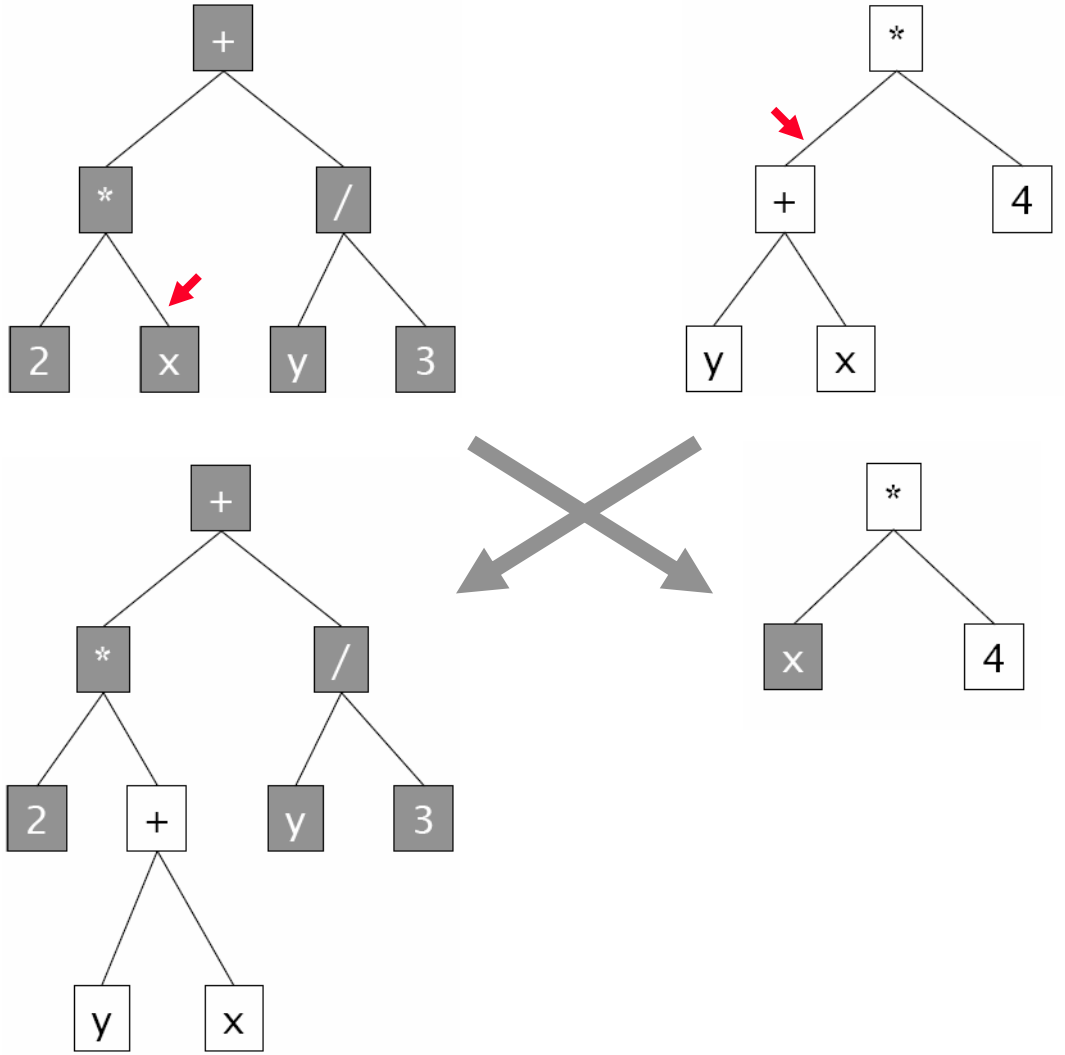


- **Editing**
  - Selezione di un nodo
  - Rimozione del sotto-albero
  - Generazione di un nuovo sotto-albero



Operatori di ricombinazione

- **Crossover**
  - Selezione di due punti di taglio
  - Scambio dei sotto-alberi



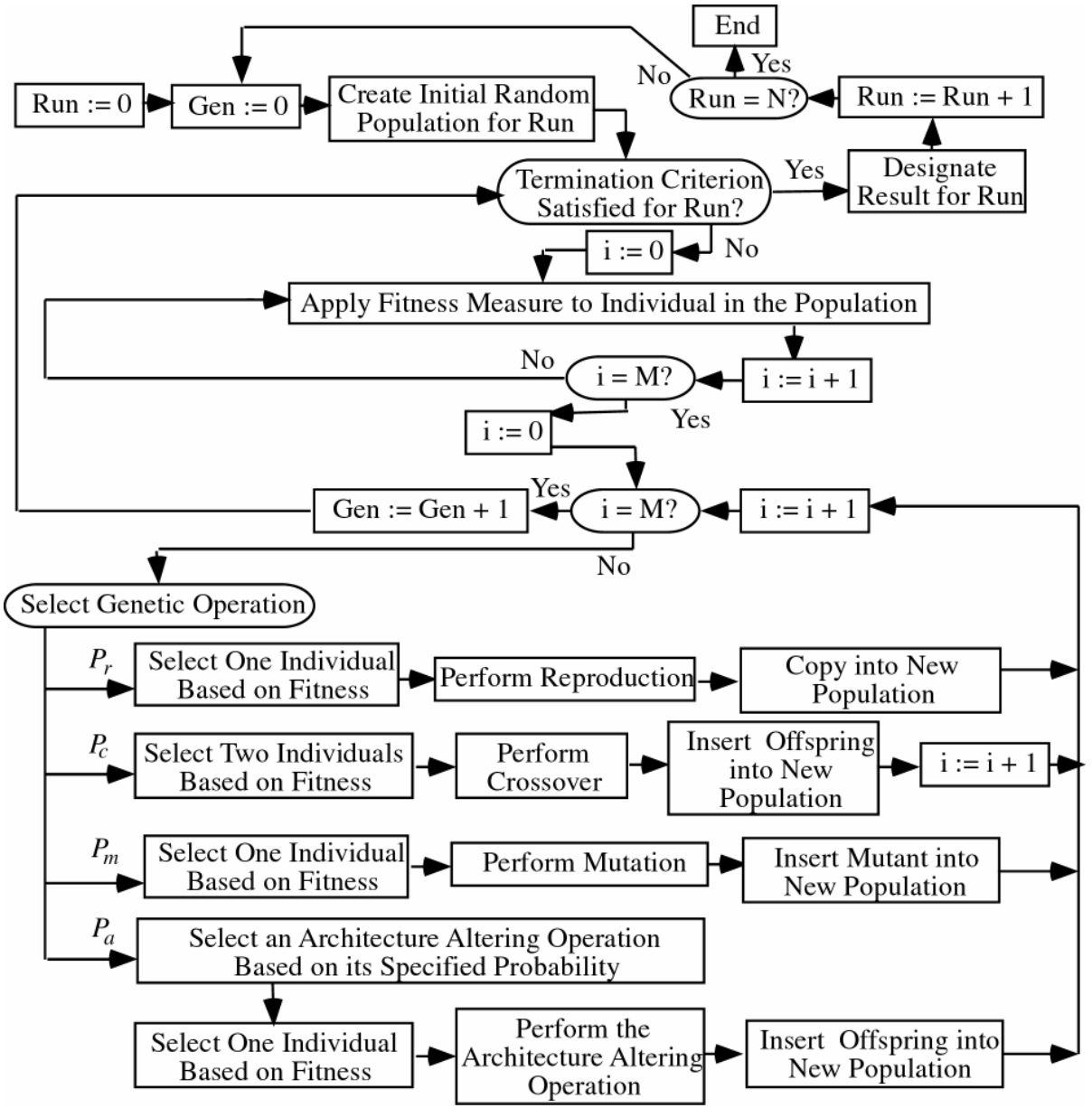
Selezione a torneo (*tournament*)

- Motivazione
  - I metodi *fitness proportionate* richiedono strutture dati di popolazione
    - Può essere scomodo su popolazioni vaste e/o distribuite
- Metodo generale
  - Si scelgono a caso *k* individui della popolazione
  - Tra i *k* individui, si sceglie quello a *fitness* più alta
- Controllo
  - Tramite il valore di *k* (finestra di selezione)
    - Per *k* = 1 la selezione è puramente casuale (la *fitness* non conta più)
    - Per *k* = dim(popolazione) la selezione non è più casuale
  - Maggiore il valore di k, maggiore è la pressione selettiva
    - Gli individui a minore fitness hanno probabilità minori di essere selezionati
  - Maggiore è la pressione selettiva, più breve è la durata della diversità
    - L'individuo migliore (della popolazione) prende il sopravvento
    - Un alto tasso di mutazione non può compensare questo effetto

Riproduzione ed elitismo

- Motivazione
  - Nei processi evolutivi generazionali, è pericoloso produrre nuove generazioni solo per mutazione e ricombinazione
    - Si rischia di perdere i risultati acquisiti
    - In quanto gli individui migliori potrebbero andar persi
- **Riproduzione**
  - Gli individui selezionati vengono copiati nella nuova generazione
  - Un operatore genetico 'degenere': non fa nulla
- **Elitismo**
  - Selezione non casuale dei migliori *n* individui e riproduzione
    - Si ha la certezza di non perdersi i migliori di ogni generazione
    - Usare con cautela: abbrevia la durata della diversità

Schema di flusso



Parametri di processo: esempio

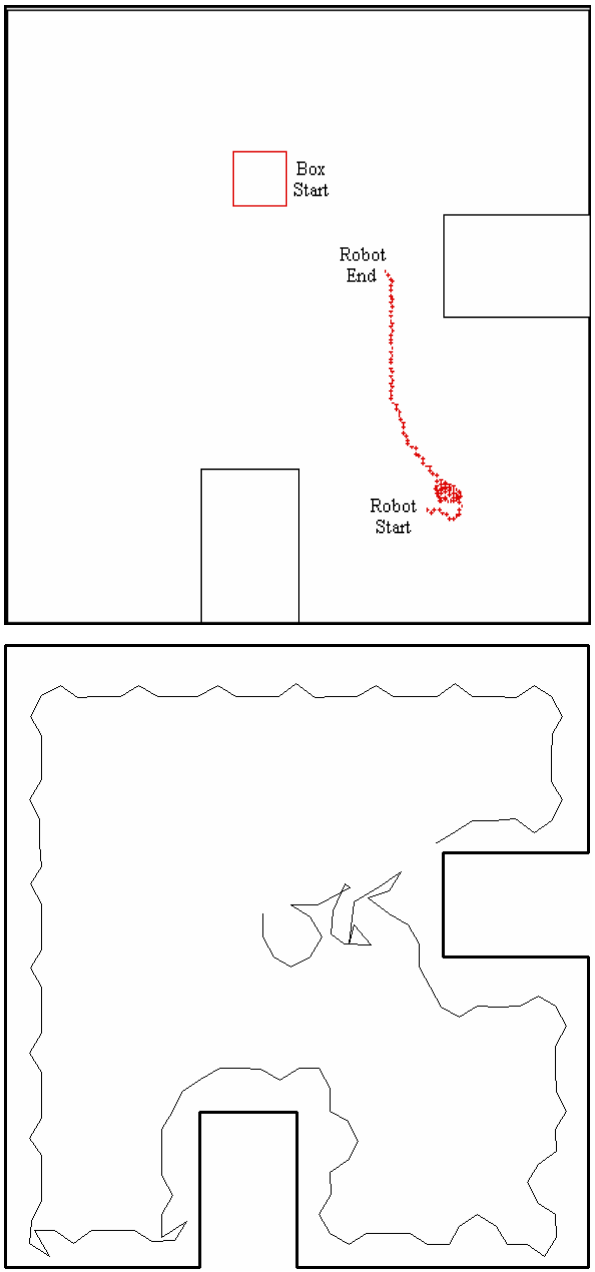
- Popolazione
  - Dimensione: da qualche decina a decine di migliaia
    - Dominante, in molti casi, è il costo di valutazione della *fitness*
    - Popolazioni molto ampie danno maggiori garanzie ma rallentano il processo
  - Profondità D_{max}: tipicamente 7 o 8 (dipende dal problema)
- Selezione operatori genetici
  - Riproduzione: $p = 0.1$
  - Crossover: $p = 0.7$
  - Mutazione: $p = 0.2$
    - La mutazione per *editing* è meno distruttiva
  - Elitismo: poche unità
- Pressione selettiva
  - Finestra di selezione k: tipicamente 5 su una popolazione di 1024

Problemi tipici

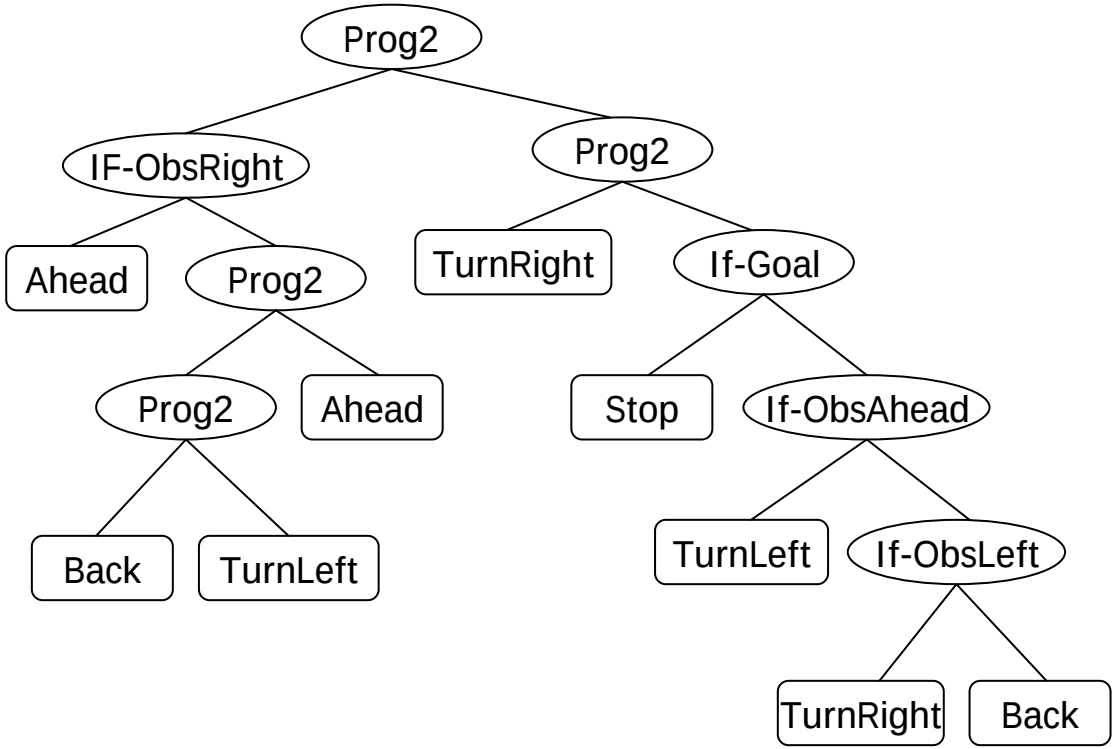
- Non convergenza (valori di *fitness* non accettabili)
- Perdita della diversità della popolazione
  - Aumento eccessivo della pressione selettiva per accelerare il processo
- Gigantismo degli individui (*bloating*)
  - Di solito il valore D_{max} viene usato solo per la generazione
  - Gli operatori genetici (p.es. *crossover*) non limitano le dimensioni dell'*offspring*
  - Individui più grandi (con parti ridondanti) preservano meglio la fitness
    - E` più alta la probabilità che gli operatori genetici alterino le parti ridondanti
  - Risultato: esplosione delle dimensioni degli individui
    - Fenomeno molto più marcato nel caso di processi non convergenti (può essere un indizio ...)
  - Rimedi:
    - In un processo convergente, spesso il problema si risolve da solo
    - Uso della *parsimony pressure*: la fitness penalizza gradualmente gli individui più grandi

Applicazione: esempio

- Un robot che si muove in un ambiente
  - L'obiettivo è evitare gli ostacoli
  - La logica di controllo utilizza un ciclo *sense-eval-act*:
    - Lettura dei sensori (*sense*)
    - Valuta le informazioni (*eval*)
    - Comandi agli attuatori (act)
  - L'obiettivo dell'evoluzione è una logica di controllo
    - Esplorazione efficace e completa
    - Senza urtare gli ostacoli
  - La *fitness* valuta il comportamento
    - + Area esplorata, – Collisioni
    - La valutazione della fitness prende tempo ...



Applicazione: tipico individuo



Tipico individuo: osservazioni

- I nodi sono *typeless*
  - Questo facilita la generazione incrementale
- Non c'è passaggio di valori
  - La struttura dei nodi descrive solo il flusso dell'esecuzione
  - I sotto-alberi non passano un valore alla loro radice
    - Si confronti con il caso delle espressioni aritmetiche
- Non si usa memoria
  - Ogni ciclo sense-eval-act è un episodio indipendente

- Problemi per l'estensione
  - Il passaggio dei valori tende a introdurre la gestione dei *tipi*
    - Si consideri il caso: IF <sense> THEN <eval> ELSE <act>
  - La gestione della memoria può non essere semplice
    - Si può incorporare nei nodi ...