ROMA TRE
UNIVERSITÀ DEGLI STUDI

Facoltà di Scienze Matematiche Fisiche e Naturali

Graduation Thesis in Mathematics

Evolving Algorithms and Computability (sintesi)

Author:
Luca Colangelo
246718

Supervisor:
Prof. Marco Pedicini

Academic year: 2011/2012

MSC AMS: 03B70, 03F05, 03B40, 68N18, 68Y05.

KEYWORDS: Theory of computation, Genetic Algorithms, Small Universal Turing Machines

1 Introduzione

Dall' avvento della Tesi di Church-turing si é lavorato molto per semplificare computazionalmente i sistemi universali. Oggi, ottant'anni dopo, la dimensione dei più semplici sistemi universali è sorprendente. Il problema di trovare sistemi universali semplici ha un interesse intrinseco e un grande numero di applicazioni. Ad esempio, la più ovvia quella di capire i confini tra universalità e non-universalità.

Uno dei primi e più importanti modelli in questo contesto é la *Macchina di Turing*. Fu introdotta dal logico inglese Alan Turing nel 1936, ed é composta da tre parti: un nastro unidimensionale infinito, una testina in grado di leggere e scrivere sul nastro e un elemento di controllo. Il nastro é diviso in una serie di celle, ognuna delle quali può contenere un simbolo di un alfabeto finito esteso con un carattere speciale denominato "blank". Inizialmente una parola (l'input) viene scritta sul nastro, le altre celle contengono il simbolo blank, la testina legge il simbolo più a sinistra dell'input e la macchina si trova nello stato iniziale. Ad ogni passo, in accordo con lo stato corrente e il simbolo letto dalla testina, il simbolo viene modificato, la testina si muove a destra o sinistra e lo stato é modificato. La computazione si arresta quando viene raggiunto uno stato speciale, lo stato finale.

Turing fornisce inoltre un esempio del suo modello, una *Maccina di Turing universale*, che simula il comportamento di una qualsiasi altra macchina , data una codifica appropriata della stessa col suo input. Questo riduce il problema di simulare tutte le macchine di Turing al problema di simularne una universale.

Indipendentemente da Turing altri modelli con la proprietà dell'universalità sono stati costruiti circa nello stesso periodo. Post ideò una macchina simile a quella di Turing ma non diede dettagli su come la macchina potesse risolvere problemi specifici o come codificarli come input. Nel 1930 Alonzo Church introdusse il λ-*calcolo*, come un metodo per formalizzare il concetto di effetiva computabilità. Il λ-calcolo é universale poiché ogni funzione computabile può essere espressa e calcolata usando il suo formalismo, ed é quindi equivalente alle

macchine di Turing. Vennero introdotte inoltre le *funzioni ricorsive parziali*, una famiglia di funzioni costruita a partire da tre funzioni base (lo zero,il successore e la proiezione) e chiusa rispetto a tre operazioni (composizione, ricorsione primitiva e minimizzazione). Le funzioni calcolate dalle macchine di Turing sono esattamente le funzioni ricorsive parziali.

L'equivalenza tra i primi e successivi sistemi che mostrano proprietà di universalità, porta ad ipotizzare che ci siano delle regole intrinseche comuni a tutti questi modelli. Se pensiamo ai processi naturali come a delle computazioni è quindi possibile che la proprietà di universalità potrebbe presentarsi non solo in sistemi astratti ma anche in natura.

L'obiettivo di questa tesi è quindi quello di proseguire nell'esplorazione dei sistemi universali, ma con un approccio diverso: invece di costruire macchine finalizzate ad essere universali, la ricerca verrà effettuata a partire da macchine semplici ed usando una particolare classe di algoritmi chiamati "algoritmi genetici" che simulano un chiaro processo di ottimizzazione presente in natura, quello dell'evoluzione.

2 Algoritmi Genetici

Gli algoritmi genetici furono introdotti da Holland nel 1975 nel suo libro "Adaption in natural and artificial systems".

Un algoritmo genetico è un algoritmo di ottimizzazione che appartiene ad una particolare classe di algoritmi utilizzati in diversi campi, tra cui l'intelligenza artificiale. un metodo euristico di ricerca ed ottimizzazione, ispirato al principio della selezione naturale di Charles Darwin che regola l'evoluzione biologica.

Un algoritmo genetico parte da un certo numero di possibili soluzioni(individui) chiamate popolazione e provvede a farle evolvere nel corso dell'esecuzione: a ciascuna iterazione, esso opera una selezione basata su una funzione chiamata *fitness* di individui della popolazione corrente, impiegandoli per generare nuovi elementi della popolazione stessa, che andranno a sostituire un pari numero d'individui già presenti, e a costituire in tal modo una nuova popolazione per

l'iterazione (o generazione) seguente. Tale successione di generazioni evolve verso una soluzione ottimale (locale o globale) del problema assegnato.

L'evoluzione della popolazione viene ottenuta attraverso una parziale ricombinazione delle soluzioni, ogni individuo trasmette parte del suo patrimonio genetico ai propri discendenti, e con l'introduzione di mutazioni casuali nella popolazione di partenza, sporadicamente quindi nascono individui con caratteristiche non comprese tra quelle presenti nel corredo genetico della specie originaria. La mutazione serve, in genere, per inserire un po' di caos tra le soluzioni ed evitare che si cada in un minimo locale.

Più precisamente il ciclo di un algoritmo genetico è il seguente:

Algorithm 1

1. Genera una popolazione casuale di n cromosomi (soluzioni idonee per il problema).

2. Calcola la fitness $f(x)$ per ogni cromosoma x nella popolazione.

3. Crea una nuova popolazione ripetendo i seguenti passi fino a che la nuova popolazione è completa :

   - [Selezione] scegli due cromosomi genitori dalla popolazione sulla base della loro fitness (maggiore è la fitness, maggiore sarà la probabilità di essere scelti);
   - [Ricombinazione] con probabilità di ricombinazione p_C, ricombina i genitori per formare nuovi figli. Se la ricombinazione non avviene, i figli saranno l'esatta copia dei genitori;
   - [Mutazione] con probabilità di mutazione p_M, muta i nuovi figli;
   - [Accettazione] posiziona i nuovi figli nella nuova popolazione.

4. Usa la nuova popolazione generata per una valutazione dell'algoritmo.

5. Se la condizione di arresto è soddisfatta, stop, e restituisci la migliore soluzione della popolazione corrente.

6. vai al passo (2) per la valutazione della fitness.

Un ruolo chiave è interpretato dalla popolazione iniziale, poiché la sua ampiezza e etereogenità determina la riuscita dell'algoritmo. Chiaramente maggiori saranno queste due caratteristiche, minore sarà lo spazio di ricerca.

La funzione di fitness gioca anch'essa un ruolo fondamentale poiché non solo indica quanto è "buona" la soluzione, ma corrisponde anche a quanto la soluzione sia vicina a quella ottimale.

2.1 Selezione

La selezione è il processo di scelta di due individui per la ricombinazione. Il processo di selezione deve far sì che i migliori individui producano più figli. La selezione deve essere inoltre bilanciata con la variazione determinata dalla ricombinazione e dalla mutazione; infatti una pressione selettiva troppo forte implica che individui sub-ottimali prendano il sopravvento riducendo la biodiversità, mentre una pressione selettiva troppo debole porta ad una evoluzione eccessivamente lenta.

I principali metodi per effettuare la selezione sono la *random selection* in cui gli individui sono scelti in maniera casuale; la *roulette wheel selection* in cui ad ogni individuo è assegnato un settore di una disco di una roulette, la cui ampiezza è proporzionale alla sua fitness. La ruota viene girata N volte, dove N è la cardinalità della popolazione, ed ad ogni iterazione l'individuo che si trova nel settore selezionato viene scelto per la riproduzione. Inoltre vi è la *tournament selection* in cui viene effettuato un torneo tra gli individui, il cui vincente ad ogni fase è determinato dalla fitness, o *Boltzmann selection* che simula un processo di raffreddamento di un metallo, garantendo una variazione della pressione selettiva.

2.2 Ricombinazione

La ricombinazione (o crossover) è il processo che prende due soluzioni come genitori e produce da esse un figlio. Viene effettuata con probabilità p_C, con

l'auspicio che crei individui migliori.

I vari tipi di ricombinazione (assumendo una codifica della popolazione in stringhe binarie) sono: *single point crossover* dove un punto di crossover viene scelto e i bit dopo di esso vengono scambiati tra i genitori, *two points crossover* dove vengono scelti due punti e il contenuto tra questi viene scambiato, *N-points crossover* che consiste nella generalizzazione dei due metodi precedenti. Inoltre è possibile effettuare *uniform crossover* dove ogni gene trasmesso al figlio viene scelto tra i due genitori in acoordo ad una stringa binaria casuale (per esempio se c'è uno 0 il gene viene preso dal primo genitore, altrimenti dal secondo); oppure *ordered crossover* dove le stringhe vengono divide in tre parti(sinistra, centrale e destra) e i figli ereditano quelle esterne dal primo genitore ed eredtani i geni di quelle centrali dal primo genitore ma nell'ordine in cui appaiono nel secondo.

2.3 Mutazione

Dopo la ricombinazione, gli individui vengono sottoposti alla mutazione con probabilità p_M. La mutazione previene la possibilità di incorrere in un minimo locale e disturba casualmente la diversità genetica. Mentre la ricombinazione è utile per il miglioramento degli individui, la mutazione è utile per l'esplorazione dello spazio di ricerca.

La mutazione piú semplice consiste nel cambiare ogni gene di solito con probabilità $1/L$ dove L è la lunghezza della stringa. Altimenti si può cambiare ogni gene sulla base di un cromosoma generato casualmente (ovvero si flippa ogni posizione corrispondente ad un 1 nel cromosoma). Si può oppure scambiare due posizioni nel cromosoma o invertire l'intero cromosoma.

Chiaramente altre ci sono altre possibilità per effettuare la mutazione in accordo con la codifica scelta per la popolazione.

2.4 Rimpiazzo

Poiché vogliamo tenere la dimensione della popolazione fissa, per ogni nuovo individuo generato, un altro deve essere eliminato secondo un metodo fissato.

Essenzialmente ci sono due tipi di metodi per mantenere la popolazione fissa: *generational update* e *steady state update*.

Il generational update consiste nel produrre N figli da una popolazione di dimensione N, e questi rimpiazzeranno completamente i genitori.

Nello steady state updates, nuovi individui sono inseriti nella nuova popolazione appena vengono creati rimpiazzando,ad esempio, l'individuo peggiore o il più vecchio.

2.5 Condizioni di arresto

L'algoritmo si arresta quando avviene una delle seguenti condizioni:

- **Generazioni massime** Viene raggiunto un numero massimo di generazioni prefissato.
- **Tempo passato** Viene superato un tempo limite prefissato.
- **Nessun cambio della fitness** Non c'è incremento della fitness dopo un certo numero di generazioni.
- **Individuo migliore** La minima fitness nella popolazione è minore di un valore di convergenza.
- **Individuo peggiore** Gli individui peggiori della popolazione hanno una fitness minore di un valore di convergenza.
- **Fitness media** Almeno metà degli individui hanno una fitness maggiore o uguale di un valore di convergenza.

2.6 The Schema Theorem

Per i convenzionali metodi di ottimizzazione deterministici, usualmente si hanno risultati che garantiscono che la sequenza di iterazioni converga verso una soluzione

ottimale in tempi e velocità noti. Per gli algoritmi genetici ci sono alcune circostanze che rendono difficile capirne il comportamento. Dunque non siamo in grado di fornire teoremi forti sulla convergenza ma solo qualche indizio sul perchè gli algoritmi genetici funzionano su alcuni problemi. Per semplicità restringiamo il campo ad algoritmi con un numero fissato m di stringhe di lunghezza n.

Innanzitutto diamo alcune definizioni.

Definitions

1. Una stringa $H = (h_1, \ldots, h_n)$ da un alfabeto $\{0, 1, *\}$ viene definita *schema* di lunghezza n. Una $h_i \neq *$ è detta *specification* di H , una $h_i = *$ è detta *wildcard*. Se consideriamo la seguente funzione che mappa uno schema nel suo alfabeto associato

$$\begin{aligned} i : \{0, 1, *\}^n &\longrightarrow \mathcal{P}\left(\{0, 1\}^n\right) \\ H &\longmapsto \{S \mid \forall i : 1 \leq i \leq n \; (h_i \neq *) \Rightarrow (h_i = s_i)\} \end{aligned}$$

   gli schemata possono essere considerati come sottoinsiemi di $\{0, 1\}^n$.

2. Una stringa $S = (s_1, \ldots, s_n)$ dell'alfabeto $\{0, 1\}$ *completa* lo schema $H = (h_1, \ldots, h_n)$ se e solo se è uguale ad H in ogni sua posizione diversa dalle wildcard:

$$\forall i \in \{j \mid h_j \neq *\} : s_i = h_i$$

   Coerentemente con sopra scriveremo $S \in H$.

3. Il *numero di specification* di uno schema H è chiamato *ordine* e si denota:

$$\mathcal{O}(H) = \mid \{i \in \{1, \ldots, n\} \mid h_i \neq *\} \mid$$

4. La distanza tra la prima e l'ultima specification

$$\delta(H) = \max\{i \mid h_i \neq *\} - \min\{i \mid h_i \neq *\}$$

   è detta *lunghezza significativa* di uno schema H.

Per enunciare il teorema abbiamo bisogno delle seguenti notazioni:

1. La generazione al tempo t è una lista di m stringhe che denoteremo con

$$\mathcal{B}_t = (b_{1,t}, b_{2,t}, \ldots, b_{m,t})$$

2. Il numero di individui che completano lo schema H al tempo t sono

$$r_{H,t} = \mid \mathcal{B}_t \cap H \mid$$

3. L'espressione $\overline{f}(t)$ si riferisce alla fitness media osservata al tempo t:

$$\overline{f}(t) = \frac{1}{m} \sum_{i=1}^{m} f(b_{i,t})$$

4. L'espressione $\overline{f}(H,t)$ sta per la fitness media osservata in$t$ passi:

$$\overline{f}\,(H,t) = \frac{1}{r_{H,t}} \sum_{i \in \{j \mid b_{j,t} \in H\}} f(b_{i,t})$$

Theorem 1 (Schema theorem) *Se consideriamo un algoritmo genetico del tipo (1), allora la seguente disuguaglianza vale per ogni schema H:*

$$E[r_{H,t+1}] \geq r_{H,t} \cdot \frac{\overline{f}(H,t)}{\overline{f}(t)} \cdot \left(1 - p_C \cdot \frac{\delta(H)}{n-1}\right) \cdot (1-p_M)^{\mathcal{O}(H)} \qquad (1)$$

Da un punto di vista qualitativo lo Schema theorem prevede che schemata con fitness superiore alla media e con lunghezza significativa corta, producono più figli degli altri. Chiamiamo queste schemata *building blocks*. Ciò ci porta al seguente risultato:

Building Block Hypothesis

Un algoritmo genetico crea progressivamente soluzioni migliori ricombinando,incrociando e mutando schemata corte, di basso ordine e con fitness alto

3 Algoritmi genetici nel λ-calcolo

Nel λ-calcolo le funzioni sono rappresentate da λ-termini. Il più semplice λ-termine è una variabile. Per costruire λ-termini più complessi si possono usare

solo due costrutti: astrazione e applicazione. La sintassi di un λ-termine è:

$$\langle \lambda - termine \rangle := \langle variabile \rangle | \langle astrazione \rangle | \langle applicazione \rangle \tag{2}$$

$$\langle astrazione \rangle := \lambda \langle variabile \rangle \langle \lambda - termine \rangle \tag{3}$$

$$\langle applicazione \rangle := (\langle \lambda - termine \rangle) \langle \lambda - termine \rangle \tag{4}$$

Le astrazioni introducono un parametro $\langle variabile \rangle$, ad esempio x e trasformano un λ-termine dato in una funzione unaria. Ad esempio x è un λ-termine in virtù di (1) a astraendo x attraverso (2) si ottiene λxx che è la funzione identità.

In un termine del tipo $\lambda \langle variabile \rangle \langle \lambda - termine \rangle$ tutte le occorrenze della variabile nel λ-termine vengono dette "legate". le variabile non legate sono dette "libere".

La valutazione delle funzioni avviene tramite una relazione di riduzione tra i λ-termini chiamata "sostituzione":

$$(\lambda x P) \Rightarrow [Q/x]P$$

dove P e Q sono λ-termini e x è una variabile. $[Q/x]P$ significa che Q viene sostituita a tutte le occorrenze di x in P. Un termine che non contiene sottotermini di questo tipo viene chiamato "forma normale" e il processo di riscrivere un termine in forma normale si chiama "normalizzazione". Non tutti i termini hanno una forma normale, infatti si possono incontrare serie infinite di riduzioni che corrispondono a una computazione che non termina.

Useremo ora gli algoritmi genetici per trovare la funzione predecessore. L'algoritmo comincia con una popolazione di termini chiusi generati in maniera casuale. Un termine è scelto per la riproduzione con probabilità proporzionale alla sua fitness. La riproduzione produce una copia, un mutante, o una ricombinazione con un altro termine. La pressione selettiva avviene mantenendo la dimensione della popolazione costante.

3.1 Mutazione

La mutazione avviene introducendo o rimuovendo i due costrutti astrazione e applicazione. Schematizzando si possono inserire o cancellare le parti sottolineate nel contesto di termini chiusi:

(i) $\underline{\lambda x}\,\bullet$ dove $\bullet$ sta per un termine, e x è una variabile;

(ii) $\underline{(\textit{simple})}\bullet$ dove *simple* è la funzione identità o una variabile legata;

(iii) $(\bullet)\underline{\textit{simple}}$ dove *simple* è la funzione identità o una variabile legata.

3.2 Crossover

La naturale ricombinazione di λ-termini corrisponde allo scambio di sottotermini, ma questo potrebbe produrre delle variabili libere, violando la proprietà di chiusura. Quindi lo scambio avviene solo tra "combinatori" che sono unità le cui azioni sono indipendenti dal contesto.

3.3 Selezione

in una popolazione di n λ-termini, un termine $expr_i$ viene scelto per la riproduzione con probabilità:

$$p_r(expr_i) = f(expr_i)/\sum_{k=1}^{n} f(expr_k)$$

dove $f(expr_i)$ è la fitness di $expr_i$.

Il termine da rimuovere può essere scelto in due modi: il metodo *non-elitario* che sceglie il termine da rimuovere con probabilità indipendente

$$p_d(expr_i) = 1/n$$

o il metodo *elitario* che fa lo stesso, prevenendo però che il termine con la fitness più alta venga rimosso.

3.4 Fitness

La fitness viene calcolata sulla base del comportamento che un termine ha applicandolo a un numerale:

$$(expr)numeral_i \Rightarrow result_i \tag{5}$$

dove $result_i$ è la forma normale risultante.

La fitness valuta per un numero finito di casi fitness, la differenza tra $result_i$ e l'obiettivo desiderato $target_i$, confrontandoli dal punto di vista aritmetico (facendo la differenza tra i due o nel caso $result_i$ non sia un numerale sottraendo il più grande numerale contenuto in esso), dal punto di vista sintattico e facendo il prodotto dei confronti.

Più precisamente

$$case(result_i, target_i) = \underbrace{\frac{primitives(num_i)}{primitives(result_i)}}_{\text{distanza sintattica}} \cdot \underbrace{\frac{1}{\mid num_i - target_i \mid + \epsilon}}_{\text{distanza aritmetica}} \tag{6}$$

$primitives(expr)$ indica la somma del numero di applicazioni, astrazioni e occorrenze di variabili in $expr$.

La fitness totale di un termine $expr$ è dunque:

$$f(expr) = \sum_{i=0}^{C} case(result_i, target_i) \tag{7}$$

dove $C + 1$ è il numero di casi fitness e $result_i$ è ottenuto da (5).

3.5 Il Predecessore

Il precessore ha una storica rilevanza nel λ-calcolo poichè sembra difficile da trovare.

E' definito come:

$$pred(n) = \begin{cases} n - 1 & \text{if } n > 0 \\ 0 & \text{if } n = 0 \end{cases}$$

GA è stato in grado di trovare il predecessore; 4 tentativi su 300 sono riusciti, il numero cresce se si elimina la condizione che $pred(0) = 0$.

Il termine più corto trovato è:

$$\lambda x_1((x_1)\underbrace{\lambda x_2((x_2)\lambda x_3 x_3)\lambda x_4\lambda x_5((x_2)x_4)(x_4)x_5}_{S}\underbrace{\lambda x_6(x_1)\lambda x_7\lambda x_8\lambda x_9 x_9}_{A}$$

che può essere abbreviato nella forma:

$$\mathbf{pred} \stackrel{\text{def}}{=} \lambda x_1((x_1)S)A$$

Se applichiamo questo termine ad u numerale otteniamo:

$$(\mathbf{pred})\mathbf{n} = (\lambda x_1((x_1)S)A)\lambda f\lambda x(f)^n x$$

Il primo passo di riduzione produce $((\lambda f\lambda x(f)^n x)S)A_n$ con
$A_n \equiv \lambda x_6(\lambda f\lambda x(f)^n x)\lambda x_7\lambda x_8\lambda x_9 x_9$.

Il passo successivo produce $(\lambda x(S)^n x)A_n$, ed infine

$$(\mathbf{pred})\mathbf{n} = (S)^n A_n.$$

Quando **pred** viene applicato ad un numerale **n**, il suo sottotermine S viene iterato n volte su A_n. Se $n = 0$, S non viene applicato ad $A_0 \equiv \lambda x_6(\lambda f\lambda xx)\lambda x_7\lambda x_8\lambda x_9 x_9$ e A_0 si normalizza in $\lambda x_6\lambda xx$ che uguale allo 0. Quindi $(\mathbf{pred})\mathbf{0} = \mathbf{0}$.

Se $n > 0$, $A_{n>0} \equiv \lambda x_6(\lambda f\lambda x(f)^n x)\lambda x_7\lambda x_8\lambda x_9 x_9$, la cui forma normale è $A' \stackrel{\text{def}}{=} \lambda x_6\lambda x\lambda x_8\lambda x_9 x_9$ indipendentemente da $n > 0$. Quando S viene applicato ad A' il risultato ottenuto è 0. Un analisi analoga evidenzia che S applicato ad un numerale si comporta esattamente come il successore. Il meccanismo di questo predecessore consiste quindi in una prima applicazione di S ad A' che restituisce lo zero; dopodiché le successive $n-1$ applicazioni semplicemente incrementano 0 fino a $n-1$.

$$(\mathbf{pred})\mathbf{n} = (S)^n A' = \underbrace{(S)\ldots(S)}_{n\,times}A' = \underbrace{(S)\ldots(S)}_{n-1 times}(S)A' = \underbrace{(S)\ldots(S)}_{n-1 times}\mathbf{0} \equiv n-1.$$

Durante gli esperimenti è stat usata una popolazione di 1000 termini, calcolando la fitness su 9 casi fitness(i numerali da 0 a 8 con $\epsilon = 0.1$). Solitamente in ogni esperimento sono stati generati da 10^5 a $2 \cdot 10^5$ termini.

4 La macchina di Turing

Le Macchine di Turing (MT) sono il modello di calcolo di riferimento fondamentale sia nell'ambito della teoria della calcolabilità sia in quello della teoria della complessità computazionale.

Nella sua versione più tradizionale una macchina di Turing si presenta come un dispositivo che accede ad un nastro potenzialmente illimitato diviso in celle contenenti ciascuna un simbolo appartenente ad un dato alfabeto Σ dato, ampliato con il carattere speciale $\square$ (blank) che rappresenta la situazione di cella non contenente caratteri. La macchina di Turing opera su tale nastro tramite una testina, la quale pu' scorrere su di esso in entrambe le direzioni. Su ogni cella la testina pu' leggere o scrivere o caratteri appartenenti allalfabeto Σ oppure il simbolo $\square$.

Ad ogni istante la macchina si trova in uno stato appartenente ad un insieme finito Q ed il meccanismo che fa evolvere la computazione della macchina è una funzione di transizione δ la quale, a partire da uno stato e da un carattere osservato sulla cella del nastro su cui è attualmente posizionata la testina, porta la macchina in un altro stato, determinando inoltre la scrittura di un carattere su tale cella ed eventualmente lo spostamento della testina stessa.

Formalmente,

Definition 1 *Una macchina di Turing (TM) è una 7-pla* $\mathcal{M} = \langle Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject} \rangle$ *dove* Q, Σ, Γ *sono insiemi finiti e* Q *è l'insieme degli stati,* Σ *è l'alfabeto di input (* $\square \notin \Sigma$*),* Γ *è l'alfabeto del nastro, dove* $\square \in \Gamma$ *e* $\Sigma \subseteq \Gamma$*,* q_0 *è lo stato iniziale,* $q_{accept} \in Q$ *è lo stato accettante,* $q_{reject} \in Q$ *è lo stato rifiutante e* δ *è la funzione di transizione definita da*

$$\delta : Q \times \Gamma \mapsto Q \times \Gamma \times \{R, L\}$$

dove R, L *indicano, rispettivamente, il movimento a destra e a sinistra.*

La configurazione iniziale di un TM $\mathcal{M}$ con input w è la configurazione $q_0 w$, che indica che la macchina è nello stato q_0 e la testina e sul carattere più

a sinistra di w. Per ogni input tre risultati sono possibili: la macchina può accettare, rifiutare o non arrestarsi.

La collezione di stringhe accettate dalla macchina $\mathcal{M}$ é il *linguaggio riconosciuto da* $\mathcal{M}$, denotato da $L(\mathcal{M})$.

Definition 2 *Un linguaggio è detto decidibile secondo Turing (T-decidibile) se esiste una macchina di Turing che lo riconosce.*

4.1 TM multinastro

Un'utile variante della macchina di Turing classica è rappresentata dalla macchina di Turing a più nastri o multinastro (MTM), in cui si hanno più nastri contemporaneamente accessibili in scrittura e lettura che vengono consultati e aggiornati tramite più testine (una per nastro).

La definizione rimane la stessa fatta eccezione per la funzione di transizione che diventa :

$$\delta : Q \times \Gamma^k \mapsto Q \times \Gamma^k \times \{L, R, S\}^k$$

dove k è il numero di nastri.

Nonostante apparentemente la MTM sembra più potente dell MT a nastro singolo, abbiamo il seguente risultato.

Theorem 2 *Data una macchina di Turing $\mathcal{M}$ a k nastri, esiste una macchina a un nastro che simula $\mathcal{M}$.*

Che porta al seguente

Corollary 1 *Un linguaggio é Turing-decidibile se e solo se esiste una macchina di Turing multinastro che lo riconosce.*

4.2 La macchina universale e il problema dell'arresto

Una *macchina di Turing universale* è una macchina di Turing capace di simulare una qualsiasi altra MT su un input arbitrario w.

Se indichiamo con $\langle M \rangle$ la codifica di una TM sull'alfabeto 0,1, otteniamo il seguente

Theorem 3 *Esiste una macchina di Turing U "universale", che sull'input $\langle \mathcal{M}, w \rangle$ dove $\mathcal{M}$ è una TM e $w \in \{0,1\}^*$, simula la computazione di $\mathcal{M}$ sull'input w. Più precisamente:*

1. *U accetta $\langle \mathcal{M}, w \rangle$ sse $\mathcal{M}$ accetta w*
2. *U rifiuta $\langle \mathcal{M}, w \rangle$ sse $\mathcal{M}$ rifiuta w*

Al fine di illustrare i limiti del potere computazionale di una macchina di Turing mostriamo ora un esempio di funzione non T-calcolabile. Si consideri il problema decisionale seguente, noto come problema della terminazione (halting problem):date una macchina di Turing M e una stringa w, stabilire se M termina la computazione avendo w come input.

Ad esso possiamo associare il linguaggio A_{TM}, dove

$$A_{TM} = \{\langle \mathcal{M}, w \rangle | \mathcal{M} \text{ è una Tm e } \mathcal{M} \text{ accetta } w\}.$$

ottenendo il seguente teorema.

Theorem 4 *A_{TM} is undecidable.*

L'indecidibilità del problema della terminazione ha delle importanti conseguenze nell'ambito della teoria della programmazione. In particolare, esso indica chiaramente che non è possibile costruire un perfetto sistema di "debugging" che sia in grado di determinare se un programma termini o meno sull'input dato. Si noti che il problema della terminazione, pur non essendo decidibile, risulta semidecidibile.

5 Macchine di Turing universali "piccole"

5.1 2-Tag System

I Tag-system furono introdotti da Post, e Minsky dimostrò la loro universalità, proprietà che successivamente venne dimostrata per i 2-tag systems.

Un "tag system" é un'elaboratore di stringhe, che opera su parole di un alfabeto finito Σ. Per ogni lettera $\sigma \in \Sigma$, è data una parola (eventualmente

vuota) $p(\sigma)$ chiamata "produzione" di σ; le lettere la cui produzione è vuota si chiamano *halting*. Formalmente:

Definition 3 *Un tag system è composto da un alfabeto finito Σ, un insieme finito di regole $R : \Sigma \to \Sigma^*$ e un numero di cancellazione $\beta \in \mathbb{N}$, $\beta \geq 1$. Per un 2-tag system $\beta = 2$.*

Minsky trovò una macchina universale a 7 stati e 4 simboli che simulava i 2-tag systems. Quindi simulava le MT tramite la seguente sequenza di simulazioni:

$$\text{Turing machine} \to \text{2-tag system} \to \text{small UTM}$$

dove $A \to B$ indica che A è simulata da B.

Il che ci porta al seguente teorema.

Theorem 5 *Sia M una macchina di Turing deterministica a nastro singolo, allora esiste un 2-tag system che simula la computazione di M*

Quindi una TM che simula un tag-system è universale poiché le MT possono essere simulate dai 2-tag systems.

5.2 UTM che simulano 2-tag system

Il nastro iniziale della macchina sarà

$$\mathcal{T} = \ldots\ldots \mathcal{P} \ldots \mathcal{S} \ldots\ldots$$

dove

- i punti denotano zone di soli blanks;
- la stringa $\mathcal{S}$ codifica la parola w da processare;
- la stringa $\mathcal{P}$ è della forma:

$$\mathcal{P} = P_{n+1} P_n \cdots P_1 P_0 \tag{8}$$

dove P_j codifica la produzione $p(\sigma_j)$.

Associeremo ad ogni $\sigma_j \in \Sigma$ un intero positivo N, la cui definizione sarà data dopo.

Una volta scelto N, fissiamo due simboli μ, ν, con $\mu \neq \nu$ e fissiamo:

$$\begin{cases} \text{se } w = b_1 b_2 \dots b_k \text{ è la parola iniziale,} \\ \text{scegliamo } \mathcal{S} = \nu^{N(b_1)} \mu \nu^{N(b_2)} \mu \dots \mu \nu^{N(b_k)} \end{cases} \tag{9}$$

in maniera tale che l'esponente di ν specificherà con quale lettera stiamo lavorando.

In particolare sia w tale che w comincia con un lettera non halting σ, la cui produzione è $p(\sigma) = \sigma_{j_1} \sigma j_2 \dots \sigma_{j_p}$; per elaborare tale w dobbiamo appendere alla destra di $\mathcal{S}$ la stringa $\Sigma(\sigma) = \mu \nu^{N_{j_1}} \mu \nu^{N_{j_2}} \mu \dots \nu^{N_{j_p}}$ e eliminare la parte più a sinistra di $\mathcal{S}$. Per appendere $\Sigma(\sigma)$ è utile avere un sua immagine speculare $\Sigma'(\sigma)$ che sarà la parte più a destra di $P(\sigma)$. Chiaramente prima di copiare la stringa dobbiamo "raggiungerla"; in altre parole neutralizzeremo la parte di $\mathcal{P}$ che si trova alla destra di $P(\sigma)$ usando la parte iniziale $\nu^{N(\sigma)}$ di $\mathcal{S}$. Ciò verrà realizzato scegliendo intelligentemente la funzione N che sarà scelta in modo che

$$\begin{cases} \text{se la lettera iniziale della parola } w \text{ è } \sigma_l, \\ \text{la corrispondente stringa iniziale } \nu^{N_l} \text{ in } \mathcal{S} \text{ forzerà la MT} \\ \text{a "neutralizzare" } P_0, P_1, \dots, P_{l-1}. \end{cases} \tag{10}$$

Questa neutralizzazione sarà la prima parte della computazione della macchina; la seconda parte copierà la stringa e la terza fase recupererà la parte neutralizzata di $\mathcal{P}$ e cancellerà la parte di $\mathcal{S}$ corrispondente alle prime due lettere della parola iniziale w.

5.2.1 Una 7×4 UTM con 25 comandi

Diamo ora un esempio di tale macchina, fornendo però solo la tabella di transizione, senza una discussione dei passi di computazione (comunque basati sulla discussione precedente).

New names	0	α	β	γ
T	$0\mathcal{L}T$	$0\mathcal{L}T$	$\beta\mathcal{L}U$	Halt
U	$\alpha\mathcal{R}U$	$0\mathcal{L}T$	$\beta\mathcal{R}U$	$\alpha\mathcal{L}V$
V	$\alpha\mathcal{R}Z$	$\alpha\mathcal{L}V$	$\beta\mathcal{L}W$	$\gamma\mathcal{L}V$
W	$\alpha\mathcal{L}X$	$\alpha\mathcal{L}V$	$\square$	$\square$
X	$\alpha\mathcal{R}Y$	$0\mathcal{R}X$	$\beta\mathcal{R}X$	$0\ \mathcal{R}T$
Y	$\gamma\mathcal{L}V$	$\alpha\mathcal{R}Y$	$\beta\mathcal{R}Y$	$\gamma\mathcal{R}Y$
Z	$\alpha\mathcal{L}V$	$\alpha\mathcal{R}Z$	$\beta\mathcal{R}Z$	$\gamma\mathcal{R}Z$

Table 1: A 7×4 UTM with 25 commands and easy recover of the output.

6 GA e Macchine universali piccole

In questo capitolo illustriamo come far evolvere una popolazione di MT per ottenere una macchina universale.

L'idea di base è quella di partire con una popolazione iniziale di macchine sempici ad 1 stato e di alfabeto $\mathcal{A} = \{0, 1\}$ della forma

Stato	0	1
1	0 D Halt	1 D Halt

dove D è fissato in maniera casuale come destra o sinistra, e farle evolvere.

Ad ogni generazione ogni MT viene sottoposta ai seguenti processi:

1. incremento degli stati o ampliamento dell'alfabeto,
2. mutazione,
3. crossover
4. selezione and riproduzione.

6.1 Incremento degli stati (o alfabeto)

In questa fase, con probabilità p_s, la MT passa da N a $N+1$ stati con l'aggiunta dello stato

Stato	0	1
$N+1$	0 D Halt	1 D Halt

Inizialmente il nuovo stato non sarà mai richiamato, l'unico modo per attivarlo è tramite un processo di mutazione che cambia la chiamata di un altro stato.

Similmente, con probabilità p_a, l'alfabeto può essere esteso con un nuovo simbolo n (per esempio $n = |\mathcal{A}| + 1$). In tal caso la nuova tabella di transizione sarà:

Stato	0	1	n
1	0 D Halt	1 D Halt	n D Halt

6.2 Mutazione

La mutazione puó essere effettuata in vari modi.

Possiamo cambiare l'ingresso di ogni stato con probabilità p_m. La nuova entrata potrà essere scelta tra:

- un simbolo $n \in \mathcal{A}$ per gli ingressi di scrittura;
- Destra o Sinistra per gli ingressi di movimento;
- Lo stato Halt o un intero tra 1 e il numero degli stati N della macchina per gli ingressi delle chiamate.

Alternativamente possiamo permettere un cambiamento del numero degli stati, aggiungendo o rimuovendone uno. Nel caso della rimozione, si elimina lo stato meno visitato, e tutti i riferimenti a tale stato vengono cambiati casualmente all'interno della nuova tabella di transizione.

Nel caso dell'aggiunta, il nuovo stato viene appeso alla fine della tabella, riempendone le celle con valori casuali; inoltre in uno degli altri stati viene aggiunto un riferimento a quello appena creato.

Infine potremmo usare un operatore che elimina una MT data, rimpiazzandola con una nuova il cui numero di stati é determinato dall'efficienza delle generazioni precedenti.

6.3 Crossover

Un crossover a 2-punti portebbe essere usato (con probabilità p_c), scegliendo due MT nella popolazione con probabilità proporzionale alla loro fitness e scambiandone le righe contigue delle tabelle di transizione comprese tra i due punti scelti di crossover.

6.4 Selezione e riproduzione

La fitness, a sua volta, puó essere definita in diversi modi.

La fitness potrebbe misurare quanto l'output della macchina si avvicini a un prefissato output "obiettivo". Poiché vogliamo trovare MTU potremmo osservare il comportamento di ogni MT simulando un insieme di 2-tag systems prefissato. Quindi incrementare la fitness per ogni valore dell'output uguale a quello dell' "obiettivo" e diminuirla per ognuno diverso (Potremmo anche semplicemente calcolarne la distanza di Hamming, minimizzando la fitness).

Oppure potremmo usare la tecnica appena descritta, ma utilizzando come input per le MT un insieme di codifiche di MT prefissate con i rispettivi input.

Alternativamente, potremmo confrontare ogni cella delle tabelle di transizione delle MT con le tabelle di macchine universali giá note. In tal caso la fitness sarebbe il numero di celle uguali.

Come processo di selezione si puó usare il "torneo di dimensione 2". Due MT vengono scelte casualmente dalla popolazione, viene calcolata la fitness, la macchina con la fitness maggiore vince, e crea due copie di se stessa nella nuova popolazione. Se la fitness risulta uguale ognuna delle due MT crea una copia.

6.5 Criteri di arresto

La computazione di ogni MT termina quando si verifica una delle seguenti condizioni:

1. la macchina non si arresta entro un numero massimo di passi;
2. la macchina non agisce più;
3. la macchina fa riferimento ad uno stato non esistente.

La ricerca si ferma in accordo ai criteri esposti nel paragrafo 2.5.