L'evoluzione di un Automa Cellulare Toroidale

Stefano Sinigardi

27 Gennaio 2005

1 Premessa

Tra i dieci temi d'esame proposti dal Docente abbiamo scelto l'analisi dello sviluppo di un automa cellulare. Per la realizzazione abbiamo scelto di affiancare, alla visualizzazione testuale in console, una riproduzione dei risultati in forma grafica.

Lo sviluppo del codice relativo alla produzione dei dati numerici sugli stati dell'automa cellulare è stato effettuato su piattaforma Macintosh, su un sistema PowerPC G4 (RISC), con sistema operativo Mac OS X Panther (10.3.7) e ambiente di sviluppo Xcode 1.5; il compilatore era GCC 3.3.

Lo sviluppo del codice relativo alla visualizzazione grafica e il debug dell'automa cellulare sono avvenuti su un sistema PC ix86-32, processore Intel Pentium M 745, sistema operativo Red Hat Linux Fedora Core 3, ambiente di sviluppo Emacs e compilatore GCC 3.4.3, quest'ultimo personalizzato per poter eseguire il linking nei programmi che sfruttano la libreria Passe-partout fornita dal Docente.

I linguaggi di programmazione utilizzati sono stati l'ANSI UNI ISO C e l'ANSI UNI ISO C++.

Questa relazione è stata scritta utilizzando il sistema LaTeX 2_ε

Università di Bologna - Dipartimento di Fisica

2 Creazione ed evoluzione

Richiesta del Docente:
Scrivere un programma che implementi un automa cellulare toroidale. Si definisce automa cellulare un insieme composto di un numero finito N di individui disposti su un reticolo discreto di uno spazio d-dimensionale: in un automa toroidale gli N individui saranno pertanto disposti lungo una circonferenza e ciascuno avrà perciò accanto a sé altri due individui. Ogni individuo si trova, in un determinato istante t, in uno tra un numero finito K di possibili stati S_i(t) ciascuno identificato con un numero compreso tra 0 e K-1 e lo stato S_i(t+1) dell'individuo all'istante seguente è determinato dallo stato presente dell'individuo stesso e dei suoi L vicini di destra e di sinistra:S_i(t+1) = f(S_(i-L)(t), S_(i-L+1)(t), ..., S_i(t), ..., S_(i+L)(t)). Implementare il programma con K=2 e L=1 e calcolare l'evoluzione dell'automa a partire da uno stato iniziale arbitrario per una generica regola di evoluzione f da potersi scegliere tra le 256 possibili regole dipendenti solo dalla somma dei valori degli stati dei tre individui coinvolti. Facoltativamente generalizzare al caso di L (con L minore di N/2) e K qualsiasi.

La prima fase di progettazione è stata necessaria per approfondire la conoscenza dei concetti di automa cellulare toroidale e di regola di evoluzione applicata ad un insieme di elementi.
Successivamente siamo passati alla stesura delle funzioni che acquisiscono i dati necessari, come il numero di individui, il numero di stati possibili associabili ad ogni individuo, il numero di passaggi evolutivi da calcolare e il range di influenza per l'evoluzione stessa, e che elaborano di conseguenza un casuale automa che soddisfi queste richieste dell'utente. Nella funzione principale abbiamo invece applicato la regola di evoluzione con l'immediata rappresentazione a video dei risultati sia in modalità testuale che grafica.

Breve analisi delle funzioni

#include "passe_par_tout.h" : include nel codice sorgente le funzioni grafiche implementate nella libreria fornita dal Docente.

int numero_individui() : all'interno della prima funzione si acquisiscono il numero di individui, che non deve essere né minore di 3 per necessità logiche, né maggiore di 760 a causa dei limiti imposti alla struttura del programma.

int inizializza_colori() : dato che la libreria passe_par_tout usa per default esclusivamente gli 8 colori fondamentali di X (tra i quali l'ottavo (bianco) è scartato dal programma perché appunto invisibile...) in questa

funzione vengono creati e resi disponibili tanti colori quanti sono gli stati richiesti dall'utente.

int numero_stati() : chiede all'utente il numero di stati possibili per gli elementi dell'automa cellulare.

int numero_passaggi() : chiede all'utente il numero di step evolutivi, che saranno logicamente come minimo 2, e, sempre a causa dei limiti imposti al programma, al massimo 760.

int range_influenza() : acquisisce un valore molto importante ai fini dell'evoluzione del sistema, ossia il numero di elementi a sinistra e a destra che influenzano lo sviluppo dell'elemento in questione.

double * inizializza_automa() : come si può notare è una funzione che restituisce un puntatore: è adibita a generare casualmente un automa che soddisfi le richieste dell'utente.

double somma_stati_funzione() : la legge evolutiva che abbiamo ideato si basa sulla somma degli stati delle cellule vicine. Questa funzione somma gli stati richiesti.

int main() : è la parte fondamentale di ogni programma C ed è anche quella che viene eseguita per prima; da essa vengono lanciate altre funzioni, che restituiscono valori utili per l'esecuzione generale del processo. Raccolti tutti i dati viene eseguito l'output a video sia testuale che grafico dello stato iniziale del sistema, e quindi, all'interno di un ciclo, viene eseguita l'elaborazione e l'evoluzione del sistema. Infine il risultato è mostrato all'utente. Per quanto riguarda l'output sul server X questo viene distinto a seconda che il numero di individui o di passaggi sia minore o maggiore di 200: nel primo caso gli individui dell'automa sono disegnati come cerchietti, nel secondo caso invece viene abilitata una cosiddetta pixmap ed ogni individuo è rappresentato da un singolo pixel sul monitor. Quest'ultima tecnica permette la gestione di un numero molto più elevato di individui, limitato solo (in questo caso nel quale la grafica è statica) dalla risoluzione del monitor in uso (il programma limita comunque la dimensione massima della finestra ad 800x800).

3 Problemi incontrati e soluzioni adottate

Vettori di lunghezza n : la richiesta del docente indicava esplicitamente di realizzare il programma in modo da poter decidere in fase di runtime alcuni parametri quali la lunghezza dell'automa, il numero di stati possibili, il range di influenza e il numero di passaggi evolutivi. In principio avevamo intenzione di rappresentare l'automa come un array di int. Purtroppo questo non avrebbe reso il programma abbastanza flessibile da permettere di decidere a posteriori la lunghezza dell'automa: un array infatti ha una lunghezza definita, indicata nel codice sorgente e fissata in modo irreversibile in fase di compilazione. La soluzione ovvia è stata utilizzare i puntatori al posto degli array, che possono essere inzializzati a una lunghezza opportuna in fase di runtime, in questo caso proprio alla lunghezza richiesta dall'utente.

Circolarità : l'operatore modulo (%) del C++ si è rivelato molto utile nell'individuare la posizione sul vettore automa durante l'esecuzione della trasformazione indicata dalla regola di evoluzione; trattandosi infatti di un automa di tipo toroidale, gli elementi del vettore andavano immaginati disposti in cerchio, e quindi, avvicinandosi ad uno degli estremi del vettore reale, e dovendo obbedire ad un range di influenza maggiore della distanza dall'estremo del vettore, bisognava necessariamente *rientrare* dall'estremo opposto.

Problema di evoluzione 1 : nella prima stesura del programma la regola di evoluzione che trasformava ogni elemento del vettore automa prendeva in considerazione, allo scopo della trasformazione, l'automa parzialmente trasformato, invece dell'automa fissato al tempo *t - 1*. In altre parole avevamo inserito il codice per duplicare ogni elemento trasformato dell'automa all'interno del ciclo di evoluzione, anzichè subito dopo, in modo da avere una evoluzione non influenzata da se stessa.

Problema di evoluzione 2 : nella prima stesura del codice l'algoritmo di evoluzione calcolava, per il confronto con gli stati adiacenti, un numero casuale nuovo ad ogni esecuzione del ciclo; questo ha causato un comportamento imprevedibile e sbagliato dell'automa: questo valore di confronto doveva essere unico per tutte le fasi evolutive. Eliminato il passaggio indeterministico, l'automa ha preso a comportarsi come ci si sarebbe aspettati (con grande soddisfazione nostra e sua, *dell'automa*!).

```
AUTOMA CELLULARE
versione 2.0
creato da Stefano Sinigardi e Silvano Bertossa

Scrivere un programma che implementi un automa cellulare
toroidale. Si definisce "automa cellulare" un insieme
composto di un numero finito N di individui disposti su un
reticolo discreto di uno spazio d-dimensionale: in un automa
"toroidale" gli N individui saranno pertanto disposti
lungo una circonferenza e ciascuno avra' percio'
accanto a se' altri due individui. Ogni individuo si trova,
in un determinato istante t, in uno tra un numero finito K di
possibili "stati" S_i(t) ciascuno identificato con un numero
compreso tra 0 e K-1 e lo stato S_i(t+1) dell'individuo all'istante
seguente e' determinato dallo stato presente dell'individuo stesso
e dei suoi L vicini di destra e di sinistra:
S_i(t+1) = f(S_(i-L)(t), S_(i-L+1)(t), ...., S_i(t), ...., S_(i+L)(t))
Implementare il programma con K=2 e L=1 e calcolare l'evoluzione
dell'automa a partire da uno stato iniziale arbitrario per una generica
"regola di evoluzione" f da potersi scegliere tra le 256 possibili
"regole" dipendenti solo dalla somma dei valori degli stati
dei tre individui coinvolti.
Facoltativamente generalizzare al caso di L (L<N/2) e K qualsiasi.


#include <passe_par_tout.h>

int numero_individui()
  {
    int num_ind;
    cout << "Inserisci il numero di individui: ";
    cin >> num_ind;
    if (num_ind<3 || num_ind>760)
      {
         while (num_ind<3 || num_ind>760)
            {
                cout << "Il numero di individui dev'essere come minimo 3 e al massimo 760... " << endl;
                cout << "Reinserisci il numero di individui: ";
                cin  >> num_ind;
            }
      }
    return num_ind;
  }


int inizializza_colori(int num_colori)
  {
    double red, green, blue;
      {
         for (int kcol=9; kcol<=(num_colori+9); kcol++)
            {
                red = (((double)random() / 0x7fffffff) * 255);
                green = (((double)random() / 0x7fffffff) * 255);
                blue = (((double)random() / 0x7fffffff) * 255);
                m_new_color(&red, &green, &blue);
            }
      }
  }


int numero_stati()
  {
    int num_stati;
    cout << "Inserisci il numero di stati: ";
    cin >> num_stati;
    if (num_stati<2 || num_stati>248)
      {
         while (num_stati<2 || num_stati>248)
            {
                cout << "Il numero di stati dev'essere come minimo 2 e massimo 248... " << endl;
                cout << "Reinserisci il numero di stati: ";
                cin  >> num_stati;
            }
      }
    return num_stati;
  }


int numero_passaggi()
  {
    int num_days;
    cout << "Inserisci il numero di passaggi: ";
    cin >> num_days;
```

```
    if (num_days<2 || num_days>760)
      {
         while (num_days<2 || num_days>760)
           {
                cout << "Il numero di stadi evolutivi dev'essere come minimo 2 e massimo 760... " << endl;
                cout << "Reinserisci il numero di passaggi: ";
                cin >> num_days;
           }
      }
    return num_days;
  }


int range_influenza(int nmezzi)
  {
    int width_range;
    cout << "Inserisci il range di influenza: ";
    cin >> width_range;
    if (width_range<1 || width_range>nmezzi)
      {
          while (width_range<1 || width_range>nmezzi)
            {
                cout << "Il range di influenza dev'essere come minimo 1 e non maggiore di "<< nmezzi <<"..." << endl;
                cout << "Reinserisci il range di influenza: ";
                cin >> width_range;
                cout << "\n\n";
           }
      }
    return width_range;
  }


double * inizializza_automa(int numind, int numstati)
  {
    double * cell1 = new double [numind];
    for (int z = 0; z < (numind); z++) { cell1[z] = (numstati * (double)random() / 0x7fffffff); }
    return cell1;
  }


double somma_stati_funzione(int posizione, int * dati_funzione, double * cell2)
  {
    int ii, wrange2 = dati_funzione[1], numind2 = dati_funzione[0], somma_stati_cellule = cell2[posizione], modulo = 0;
    for (ii = (posizione+1); ii <= (posizione+wrange2); ii++)
      {
          modulo = ii % numind2;
          somma_stati_cellule += cell2[modulo];
      }
    for (ii = (posizione-1); ii >= (posizione-wrange2); ii--)
      {
          modulo = ii % numind2;
          somma_stati_cellule += cell2[modulo];
      }
    return somma_stati_cellule;
  }


int main(int narg, char ** args, char ** env)
  {
    int nind, ndays, stato_int, nstati, nindmezzi, sum_states, casual, wrange, m, i;
    int * dati = new int [4];
    double xmin = 0.0, ymin = 0.0, xmax, ymax, xy[2], semiassi[2], raggio, dim_massima, somma_stati = 0;
    char * uscita;

//  DICHIARAZIONI VARIABILI GRAFICHE
    int LARG, ALT, ALLOC = 120000, Lx = 95, Ly = 95, attesa = 'q', all_windows = -1, pixmap_window = 1;
    int color = 7, k, modo_ellisse = 1, redraw_null = 0;;
    double LARG_d, ALT_d, aumento = (100.0 / 95.0);
    char * titolo_finestra = "Automa Cellulare";
//  FINE DICHIARAZIONI VARIABILI GRAFICHE

    system("clear");
    srand(time(NULL));

    cout << "-------------------------------------------------------\n";
    cout << "AUTOMA CELLULARE v2.0\n";
    cout << "copyright 2004-2005 Silvano Bertossa Stefano Sinigardi\n";
    cout << "-------------------------------------------------------\n\n";
    cout << "Fornisci al programma i dati richiesti: il numero di \n";
    cout << "individui che compongono l'automa, gli stati che essi \n";
    cout << "possono assumere, quanti step evolutivi compiere e l' \n";
    cout << "influenza che le cellule adiacenti hanno nello sviluppo \n";
    cout << "di ognuna; in base ad essi verr simulato e riprodotto \n";
```

```
    cout << "lo sviluppo di un automa cellulare toroidale \n\n";

    nind = numero_individui();
    nindmezzi = (nind/2);
    nstati = numero_stati();
    ndays = numero_passaggi();
    wrange = range_influenza(nindmezzi);
    casual = ((2 * wrange + 1) * nstati * ((double) random() / 0x7fffffff));

    cout << "\n\n";

    double * cell = new double [nind];
    double * new_cell = new double [nind];
    int * cellula_intera = new int [nind];

    dati[0] = nind; dati[1] = wrange; dati[2] = nstati; dati[3] = ndays;
    cell = inizializza_automa(nind, nstati);
    if (nind>(ndays+1)) dim_massima = nind;
    else dim_massima = (ndays+1);
    raggio = ((760.0/dim_massima)/2.0);
    if (nind>=200 || ndays>=200)        { xmax = nind; ymax = (ndays+1); }
    else                                { xmax = ((raggio * nind) * 2); ymax = ((raggio * (ndays+1)) * 2); }
    if (nind>=200 || ndays>=200)        { LARG = nind * aumento; ALT = (ndays+1) * aumento; }
    else                                {
                                          LARG_d = (raggio * nind * 2) * (100.0 / 95.0);
                                          ALT_d = (raggio * (ndays+1) * 2) * (100.0 / 95.0);
                                          LARG = LARG_d +1; ALT = ALT_d +1;
                                        }
    if (nind<200 && ndays<200)          {
                                          semiassi[0] = raggio, semiassi[1] = raggio;
                                          xy[0] = raggio, xy[1] = (ymax-raggio);
                                        }
    else                                { xy[0] = 1, xy[1] = (ndays+1); }
    m_prestartg(&ALLOC);
    m_startg(titolo_finestra, &LARG, &ALT);
    m_flush();
    m_window(&Lx, &Ly);
    m_frame(&xmin, &ymin, &xmax, &ymax);
    if (nind>=200 || ndays>=200) m_use_as_pixmap(&pixmap_window, &color);
    if (nstati>7) inizializza_colori(nstati);

    for (m = 0; m <= (ndays); m++)
      {
          for (i = 0; i < (nind); i++
    {
              if (m > 0)
                {
                   sum_states = somma_stati_funzione(i,dati,cell);
                   if (sum_states > casual) new_cell[i] = cell[i];
                   else new_cell[i] = (sum_states % nstati);
                   cellula_intera[i] = new_cell[i];
                }
              else cellula_intera[i] = cell[i];
              if (nstati>7) color = (cellula_intera[i]+9);
              else color = (cellula_intera[i]);
              m_color(&color);
              if (nind<200 && ndays<200) m_ellypse(xy, semiassi, &modo_ellisse);
              else m_fill_pixmap (&pixmap_window, xy, &color);
//            cout << cellula_intera[i];
//            if (i<nind) cout << "\t";
              if (nind<200 && ndays<200)  { if (i<nind) xy[0] += (raggio*2); }
              else                        { if (i<nind) xy[0] += 1; }
            }
          for (i = 0; i < (nind); i++) { cell[i] = new_cell[i]; }
//        cout << "\n\n";
          if (nind<200 && ndays<200)       { xy[0] = raggio; xy[1] -= (raggio*2); }
          else                             { xy[0] = 1; xy[1] -= 1; }
      }
    m_redraw(&redraw_null);
    m_flush();
    m_loop_for_events(&attesa,&all_windows);
    m_endg();
    return 0;
  }
```