DOCUMENTAZIONE DEL PROGETTO DI RETI NEURALI E VISIONE ARTIFICIALE AA 2003/2004

Giulio Manzonetto[a] *, Giulio Marcon[a] †, Anna Sperotto[a] ‡.

[a]{gmanzone, gmarcon, asperott}@dsi.unive.it

Abstract. Con questa documentazione si intende dare una descrizione del problema di Visione Artificiale affrontato e delle scelte implementative fatte. Si vuole altresì dare una sintetica spiegazione delle modalità di esecuzione del software (un piccolo manuale per l'utente).

Keywords: Figure ground discrimination, edge detection, perceptual grouping, mean field annealing, simulated annealing.

Indice

*Giulio Manzonetto, matricola 794652

†Giulio Marcon, matricola 793939

‡Anna Sperotto, matricola 787327

1. INTRODUZIONE

Innanzitutto definiamo il problema di raggruppamento percettivo che intendiamo risolvere tramite il nostro software. Quello che ci viene richiesto è di effettuare una trasformazione da un'immagine raster a un'immagine contenente solo zero o uno in quei punti in cui si presentino dei contorni che, in qualche modo, ne identificano la figura. Per fare ciò si deve passare per una fase intermedia: il riconoscimento dei contorni, conosciuta in letteratura come *edge detection*.

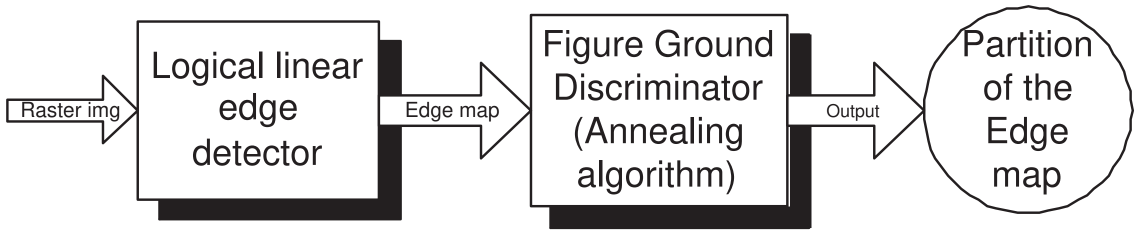


Figura 1. Descrizione del problema e identificazione dei due step.

Questo processo consiste nell'identificare i fronti dell'immagine in cui l'intensità dell'immagine cambia in modo brusco. Per essere precisi, i nostri contorni hanno anche un verso ed una direzione: la direzione corrisponde proprio alla normale del fronte in cui l'intensità cambia.

Dopo questo primo passo si può passare a determinare una partizione dell'insieme dei contorni trovati dividendoli in due insiemi distinti: sfondo e figura.

Per aumentarne la facilità d'utilizzo abbiamo, ordunque, deciso di dividere il nostro software in due programmi distinti le cui funzionalità rispecchino quelle delle due operazioni descritte a monte.

L'algoritmo utilizzato per il riconoscimento degli contorni è, essenzialmente, quello disponibile in rete all'indirizzo [5] che viene presentato come una implementazione portabile del metodo descritto nell'articolo [4].

2. OPERATORI LOGICO LINEARI PER IL RICONOSCIMENTO DEI CONTORNI

L'approccio adottato da Iverson in [6] permette di definire operatori locali che rispondano solamente in punti dell'immagine in cui passi un curva con una certa geometria. Come in tutte le metodologie classiche di riconoscimento dei contorni, ed in generale di elaborazione dell'immagine, gli operatori si basano su convoluzioni lineari ma aboliscono il post processing (a parte una semplice sogliatura) formalizzando in un algebra il ragionamento sui risultati delle convoluzioni. L'algebra utilizza delle operazioni logiche (ed è in effetti un algebra booleana con $\underset{+}{\wedge}$ come *meet*, $\overset{+}{\vee}$ come *join* e dotata di complemento); queste sono approssimate per ottenere una tolleranza al rumore.

Gli operatori definiti per riconoscere i contorni ricercano punti dell'immagine che soddisfino una condizione normale (un forte contrasto rispetto ad una certa sezione) ed una tangenziale (continuità). Ogni operatore ha un supporto spaziale di forma ovale al cui centro si trova il punto dell'immagine considerato; nel nostro caso l'ovale aveva una dimensione di circa 20 pixel lungo la componente tangenziale e 10 pixel lungo quella normale.

L'operatore da noi utilizzato è il seguente:

$$\Phi = T \times N = (t^{-} \underset{+}{\wedge} t^{+}) \times (n'_c \underset{+}{\wedge} n''_l \underset{+}{\wedge} n''_r \underset{+}{\wedge} n^4_l \underset{+}{\wedge} n^4_r)$$

Con queste funzioni lineari:

- $t^{-}(x) = G_{2.5}(x)\sigma_{5.0}(-x) + 0.2G'_{2.5}(x)$
- $t^{+}(x) = G_{2.5}(x)\sigma_{5.0}(x) - 0.2G'_{2.5}(x)$
- $n'_c(x) = G'_{1.4}(x)$
- $n''_l(x) = G''_{1.4}(x + 0.8)$
- $n''_r(x) = -G''_{1.4}(x - 0.8)$
- $n^4_l(x) = -G''_{1.4}(x + 0.8)$
- $n^4_r(x) = G''_{1.4}(x - 0.8)$

$G_\sigma(x), G'_\sigma(x), \ldots$ sono la gaussiana e le sue derivate, e $\sigma_\rho(x)$ è definita nell'articolo [4].

L'operatore è a due soli componenti tangenziali perché i risultati qualitativi ottenuti con quattro, come nell'implementazione di Iverson, non sono migliori a tal punto da giustificare il costo computazionale supplettivo.

La reimplementazione è stata piuttosto difficoltosa, pur avendo a disposizione il codice originale di Iverson. In modo particolare questa differisce dalle descrizioni dell'articolo ed utilizza piuttosto delle definizioni e delle tecniche introdotte nella tesi di dottorato. Inoltre i parametri utilizzati e il modo in cui vengono approssimate diverse funzioni suggerisce che sia stato utilizzato un approccio *trial and error* in luogo di un tentativo di ottimizzazione: ad esempio un esperimento fatto calcolando in modo esatto tali funzioni ci ha portato a risultati palesemente pessimi.

Particolarmente significativo è la presenza, nel codice originale, di un operatore $\underset{+}{\wedge}$ approssimato per cinque argomenti che si comporta in modo totalmente diverse dall'applicazione dello stesso più volte, ossia:

$$\underset{+}{\wedge}_5(x_1, x_2, x_3, x_4, x_5) \neq \underset{+}{\wedge}(x_1, \underset{+}{\wedge}(x_2, \underset{+}{\wedge}(x_3, \underset{+}{\wedge}(x_4, x_5))))$$

Il framework degli operatori logico/lineari è stato reimplementato nella sua interezza e la versatilità del linguaggio di programmazione utilizzato ci ha permesso di farlo in sole 350 righe di codice contro le 5000 dell'originale.

Poiché utilizziamo due funzioni per la componente tangenziale e cinque per la componente normale, per ogni pixel dell'immagine applichiamo dieci operatori di convoluzione di dimensione media di 20×10 pixel. Inoltre è importante sottolineare che tale operatore individua i contorni che abbiano un determinato angolo: è necessario per cui fare queste operazioni per ogni angolazione in cui si vogliono trovare contorni (il numero di parti in cui quantizzare l'angolo giro è un parametro del programma). Data l'alta mole di calcoli, sono state fatte alcune ottimizzazioni, le cui più significative sono:

- *contorni di verso opposto*: per l'individuazione di contorni con la stessa direzione ma verso opposto si può utilizzare la proprietà di complemento dell'algebra booleana dimezzando i calcoli; le convoluzioni vengono applicate solo una volta, i $\underset{+}{\wedge}$ e $\overset{+}{\vee}$ vengono applicati così come sono per un verso, e invertiti per il verso opposto;

- *convoluzione*: le maschere utilizzate lavorano a interi piuttosto che a valori in virgola mobile come nella versione originale; le maschere vengono normalizzate opportunamente per dare una risposta adatta alle necessità;

- *loop unrolling*: i cicli, pur essendo mantenuti in paradigma funzionale (sono funzioni ricorsive), sono stati adattati in modo tale da ridurre al minimo il numero di operazioni di base effettuate (funzioni ricorsive dentro a funzioni ricorsive); ulteriori ottimizzazioni del compilatore (inlining) hanno concorso a rendere il programma così efficiente.

3. SEPARAZIONE DELLA FIGURA DALLO SFONDO COME PROBLEMA DI OTTIMIZZAZIONE

Gli algoritmi di *annealing* sono nati come metodi di simulazione della tempra dei solidi. L'*annealing* è il processo con il quale un solido, portato allo stato fluido, mediante riscaldamento ad alte temperature, viene riportato poi di nuovo allo stato solido o cristallino a temperature basse controllando e riducendo gradualmente la temperatura.

Ad alte temperature, gli atomi nel sistema si trovano in uno stato altamente disordinato e quindi l'energia del sistema è elevata. Per portare tali atomi in una configurazione cristallina altamente ordinata (statisticamente), deve essere abbassata la temperatura del sistema. Riduzioni veloci della temperatura possono causare difettosità nel reticolo cristallino con conseguente metastabilità, con fessurazioni e fratture del reticolo stesso (stress termico).

L'annealing evita questo fenomeno procedendo ad un graduale raffreddamento del sistema, portandolo ad una struttura globalmente ottima stabile.

Per entrambi i metodi di annealing che descriveremo vi è un certo numero di parametri da stimare consentendo ampia libertà di scelta e quindi elevata applicabilità. Tuttavia c'è un prezzo da pagare: la taratura di un numero elevato di parametri causa un duro lavoro iniziale affinché il metodo possa convergere.

3.1. Da edge a edgel

Un primo problema comune ad entrambi gli algoritmi di annealing che abbiamo considerato è quello di trasformare la mappa di contorni che viene presa in input in una mappa di *edgel* che sono il tipo di dato su cui lavorano correttamente questi algoritmi. Un contorno o *edge* può essere pensato, intuitivamente, come un segmento completamente contenuto all'interno di una cella di lato unitario e passante per il suo centro. Tale cella sarà posizionata in un qualche punto all'interno dell'immagine.

Possiamo quindi affermare che un edge e è caratterizzato dalla sua posizione sullo schermo (x_e, y_e)[4], dalla sua angolazione d_e e da un'intensità i_e.

Un edgel è qualcosa di molto simile: infatti è caratterizzato dalle stesse quantità e può essere pensato come una retta di lunghezza arbitraria[5].

Come descritto in [1], nel momento in cui esista all'interno della edge map un insieme di edge adiacenti in modo tale che vengano a formare una retta, essi possono essere sostituiti da un edgel che ha come posizione il loro punto medio, come coefficiente angolare della retta il coefficiente angolare degli edge[6] e come intensità la media delle intensità.

Per ogni edge in input dobbiamo quindi calcolare la posizione di un edge che potrebbe essere un suo *prolungamento*, controllarne l'esistenza, determinare se l'angolazione è la medesima e scoprire se ne esistono altri che possono essere raggruppati in quell'edgel. Per eseguire questa operazione in modo efficiente utilizziamo funzioni *tail recursive* e manteniamo gli edge in una lista ordinata con la relazione lessicografica sulle triple $(angolo, x, y)$[7].

Viene poi memorizzata la corrispondenza fra edgel e l'insieme degli edge che l'hanno generato per poterli ripristinare alla fine (ricordiamo che vogliamo ottenere come output una partizione degli edge non degli edgel).

Una possibilità fornita dal nostro programma è la rimozione degli edgel formati da meno di un certo numero di edge prima di effettuare l'ottimizzazione.

3.2. Matrice di interazione

Un'altro passo comune ad entrambi gli algoritmi è la creazione della matrice contenente i coefficienti di interazione fra gli edge. Questa matrice è, per costruzione, simmetrica. Abbiamo dunque deciso di scrivere una piccola libreria per la gestione ottimizzata di questo tipo di dato che quindi si preoccupi di memorizzare solo metà matrice. Per evitare spiacevoli sorprese ne abbiamo implementata una prima versione totalmente funzionale che non faceva uso di side effect di sorta a scapito della velocità. Nel momento in cui abbiamo constatato che funzionava correttamente l'abbiamo riscritta in modo più efficiente.

---

[4] La coordinata si riferisce alla posizione della cella.

[5] La coordinata si riferisce alla posizione della cella in cui si trova il centro della retta.

[6] Si noti che per formare un edgel gli edge devono avere lo stesso coefficiente angolare

[7] Facendo attenzione a considerare uguali angoli che differiscono di π, e scartando quindi il verso degli edge

Per calcolare i valori della matrice di interazione (necessari per il computo della funzione energia) è necessario calcolare i coefficienti di:

- cocircolarità: indice della similarità tra la curva passante per i due edgel e un cerchio perfetto;
- morbidezza della curva: aumenta l'interazione di un edgel con edgel posti longitudinalmente piuttosto che con edgel posti latitudinalmente, favorendo in questo modo curve meno accentuate;
- vicinanza: aumenta l'interazione tra edgel relativamente vicini;
- contrasto: aumenta l'interazione di edgel con contrasto similare.

Nel caso del Meanfield annealing è necessario creare anche il vettore dei campi esterni δ.

3.3. Mean field annealing algorithm

L'algoritmo *mean field annealing* è, in realtà piuttosto semplice.

- viene generato in modo casuale un vettore soluzione $\mu^{(0)}$ dal quale partire,
- ad ogni iterazione i si aggiorna il vettore $\mu^{(i)}$ concordemente con la modalità (sincrona o asincrona) che si è deciso di utilizzare e si decrementa la temperatura,
- ogni 6 iterazioni si esegue il test sulla terminazione: se le componenti del vettore $\mu^{(i)}$ sono in media abbastanza vicine a 1 o a -1 l'algoritmo termina, in caso contrario il procedimento si itera,
- quando si raggiunge la convergenza le componenti del vettore risultato $\mu_i^{(r)}$ vengono poste a 1 o a -1 a seconda che $\mu_i^{(r)}$ sia maggiore o minore di 0.

La differenza fra l'aggiornamento sincrono e asincrono del vettore soluzione consiste nel fatto che nel caso sincrono è necessario aggiornare tutte le sue componenti μ_i ad ogni iterazione mentre nel caso asincrono solo una componente μ_i scelta casualmente.

A questo proposito segnaliamo un errore nella descrizione dell'algoritmo proposta nell' appendice B dall'articolo [1] che propone per la modalità asincrona l'aggiornamento di tutte le componenti del vettore ma in ordine casuale andando contro quanto detto all'interno dell'articolo stesso. Questa implementazione, come è facile immaginare, produce soluzioni molto simili al caso sincrono, peggiorandone addirittura l'efficienza in quanto non permette di sfruttare caching. Per un numero di edgel non triviale il numero di iterazioni necessarie per convergere è perfino uguale.

3.4. Simulated annealing algorithm

Il Simulated annealing è un algoritmo di Monte Carlo. Ad alte temperature l'algoritmo si comporta più o meno come una *random search*. La ricerca salta da un punto all'altro dello spazio delle soluzioni individuandone le caratteristiche e quindi le direzioni o le aree in cui è più probabile individuare il minimo globale della funzione energia. A basse temperature l'SA è simile ai metodi *steepest descent*. Le soluzioni vengono localizzate nella zona del dominio maggiormente promettente. Un vantaggio fondamentale dell' SA è che lo si può applicare a problemi di ottimizzazione dei quali si non ha una conoscenza profonda. L'algoritmo SA per un problema di ottimizzazione può essere riassunto nel seguente generico algoritmo:

- si genera una configurazione iniziale o soluzione x_0 con energia o valore della funzione energia E_0. Si seleziona un valore iniziale t_0 per la temperatura,
- Per ogni stadio della temperatura si effettuano i seguenti passi:
  - si genera una configurazione candidata ammissibile tramite una piccola perturbazione casuale della configurazione corrente (una componente del vettore soluzione viene cambiato da -1 a 1 o viceversa). Si valuta la differenza di energia ΔE fra le due configurazioni.
  - Se $\Delta E \leq 0$, la configurazione candidata ha un valore della funzione obiettivo inferiore rispetto a quello della configurazione corrente. La nuova soluzione viene accettata e sostituita a quella corrente,
  - Se $\Delta E > 0$, la configurazione candidato ha un valore della funzione obiettivo peggiore rispetto quello della configurazione corrente, la sia accetta con probabilità $exp\left(-\frac{\Delta E}{T}\right)$ e si aggiorna la configurazione corrente se necessario
- Questo procedimento va iterato finché non viene raggiunto l'equilibrio termico a quel punto si continua riducendo la temperatura.

Il processo termina quando la percentuale delle transizioni di stato accettate è minore di 1.

Poiché l'articolo è molto confuso su cosa si intenda per *raggiungimento dell'equilibrio termico* abbiamo determinato un appropriato schema di raffreddamento in base al problema specifico definendolo come l'aver testato ogni possibile transizione esattamente 10 volte. I risultati empirici ottenuti ci confortano su tale scelta, i tempi biblici impiegati un po' meno. Probabilmente si potrebbe prendere in considerazione anche un numero minore. La temperatura viene ridotta, ad ogni stadio, moltiplicandola per una costante chiamata *cooling ratio*.

Una scelta implementativa doverosa da commentare è la seguente. Dovendo testare esattamente 10 volte ogni possibile transizione riguardante ciascuna componente μ_i del vettore soluzione è necessario tenere memoria del numero di transizioni effettuate e quali componenti erano interessate.

Viene quindi simulata un'estrazione senza ripescaggio da un urna contenente 10 indici i per ogni componente mu_i.

4. UN'IMPLEMENTAZIONE PARALLELA dell'MFA

Una volta verificata la lentezza di questi algoritmi abbiamo deciso, quantomeno per il Mean Field Annealing sincrono per il quale questa operazione era piuttosto semplice, di crearne una versione parallela.

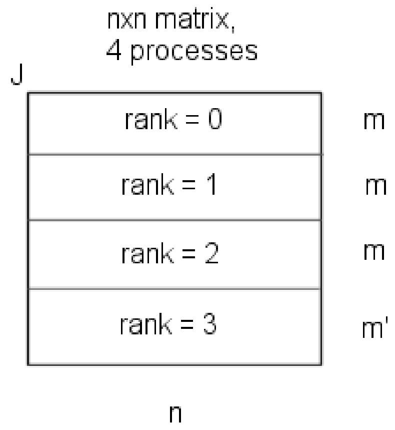


Figura 2. Esempio di decomposizione del dominio. Nella figura la matrice dei coefficienti $n \times n$ viene divisa fra quattro processori. Quindi $m = \frac{n}{4}$ e $m' = m + n \ mod \ m$.

Supponiamo di avere K processori numerati da 0 a $K - 1$, tale numero viene chiamato *rango* della macchina.

Il processo in esecuzione sulla macchina di rango 0 (quella dalla quale il software è stato lanciato) si preoccupa di estrarre dal file di input la lista degli edge, di convertirla in una lista di edgel e di creare la matrice di interazione $J_{n \times n}$. Tale matrice viene divisa in K sottomatrici di n colonne e $\frac{n}{K}$ righe[8]. Nel caso in cui n non sia perfettamente divisibile per il numero delle macchine l'ultima matrice creata avrà $\frac{n}{K} + (n \ mod \ \frac{n}{K})$ righe. Si può vedere un esempio di tale suddivisione in figura 2

Questa fase viene detta *decomposizione del dominio*, le sottomatrici così costruite vengono poi distribuite a tutti i processi coinvolti.

A questo punto ogni processo possiede una sezione della matrice. Le righe della matrice J iniziale assegnate al processo di rango i sono quelle che vanno dalla $\left(i * \frac{n}{K}\right)$ alla $\left((i+1) * \frac{n}{K} - 1\right)$.

Ciascuno può, dunque, calcolare un frammento del vettore soluzione come indicato in 3.3. Ad ogni ciclo i processi si scambiano le informazioni computate e ciascuno ricostruisce il vettore soluzione calcolato per quell'iterazione mettendo assieme le parti ricevute dagli altri e la sua parte.

Su quel vettore (che a questo punto è uguale per tutti) ciascuno riprende ad eseguire l'algoritmo. Si noti che gli scambi di informazioni fra processi saranno frequenti ma la quantità di dati che viene passata è molto piccola. Al solito ogni 6 iterazioni i processi

[8]Chiaramente viene utilizzata la divisione euclidea per calcolare il numero di righe.

eseguono il test sulla condizione di uscita[9] e decidono se terminare.

Nel caso in cui la condizione di uscita risulti vera il processo di rango 0 si occupa di creare il file di output.

5. RISULTATI

In questa sezione si possono vedere i risultati che abbiamo ottenuto mentre cercavamo di individuare i parametri corretti per gli algoritmi e di testare qualitativamente e quantitativamente la nostra implementazione.

Ottenere dei risultati quantitativamente soddisfacenti è stato piuttosto faticoso: dopo due mesi di incessante lavoro abbiamo finalmente delle versioni che hanno tempi di esecuzione nell'ordine di minuti invece che nell'ordine di ore.

In modo particolare, questo è il risultato di un esecuzione tipica su un Duron 1Ghz equipaggiato di 512mb di RAM, che ha richiesto circa 10 minuti:

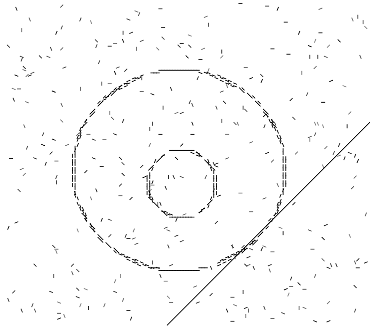
```
$ time ./loglin.opt.exe inout/Paolina.pgm
reading image...
Logical Linear Detection...: 100%
real    3m7.990s
user    0m0.010s
sys     0m0.010s

$ time ./camleye.opt.exe -n 2 inout/Paolina-edges.ps
reading edges... converting 23191 edges into edgels...
done, 8069 edgels created
removing edgels with less than 2 edge... done, 5184 remaining edgels
creating interaction matrix... done
Mean Field Annealing...
iteration: 26976  Annealing...: 100%
done
getting figure and writing out... done, figure has 333 edgels
real    7m0.435s
user    0m0.020s
sys     0m0.010s
```

Per effettuare i test abbiamo utilizzato due immagini di riferimento. La prima fornita dall'archivio di immagini di test del Caltech Vision Group, utilizzata anche da Iverson in [6] (un particolare di Paolina Borghese, opera del Canova scolpita tra il 1805 ed il 1808). L'originale di Paolina è al seguente indirizzo `http://www.vision.caltech.edu/html-files/archive.html` sebbene sia qui riprodotto per comodità. La seconda immagine è un'immagine di sintesi creata appositamente per testare la separazione della figura dallo sfondo: si tratta di una linea e due cerchi con del rumore aggiunto point-wise con probabilità 0.01.

[9]Si noti che il test, essendo eseguito su vettori eguali, fornirà su ogni macchina la stessa risposta. Dunque quando termina un processo terminano anche gli altri, risolvendo banalmente il problema della terminazione distribuita.

Figura 3. Antonio Canova, *Paolina Borgese*, 1805-1808



(a) (b)

Figura 4. (a) Immagine di sintesi per i test (b) Con il rumore aggiunto

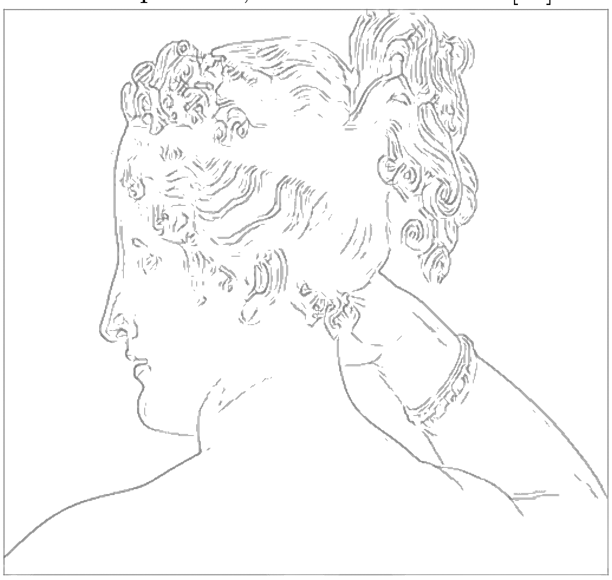
5.1. Riconoscimento contorni

Queste immagini sono a supporto dell'utilizzo di due soli componenti nell'operatore tangenziale; mostrano i risultati ottenuti su Paolina con un numero di componenti che va da uno a otto. Si può notare come un solo componente mantenga troppo rumore, mentre con quattro componenti alcuni contorni significativi vengano a mancare. Con due componenti tutti i contorni significativi vengono trovati, sebbene permanga del rumore (che comunque eliminerà il discriminatore, spesso in pre-processing).

1 componente

2 componenti, come nell'articolo [4]

4 components, come nell'implementazione [5]



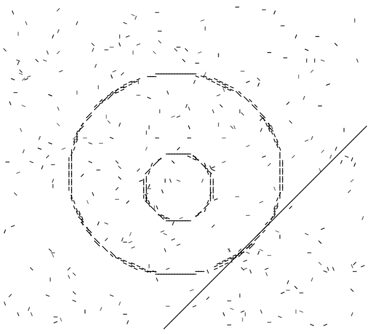
8 components

5.2. Separazione della figura dallo sfondo

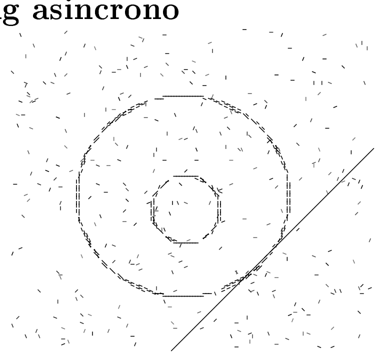
L'immagine di sintesi utilizzata per il test ha 740 edges di cui 417 originali e 323 di rumore. Il pre-processing ottiene 495 edgels.

I risultati indicano i parametri utilizzati, il numero di iterazioni dopo cui il metodo utilizzato converge, gli edgels e corrispettivi edges finali. Sono presentati in forma grafica per valuare il risultato qualitativamente e tabellare per una visione d'insieme.

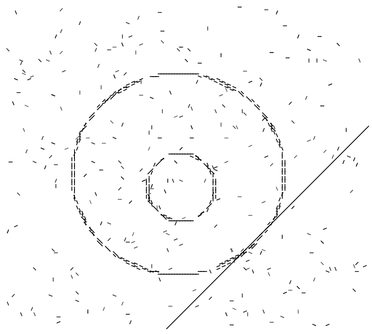
5.2.1. Mean Field Annealing asincrono



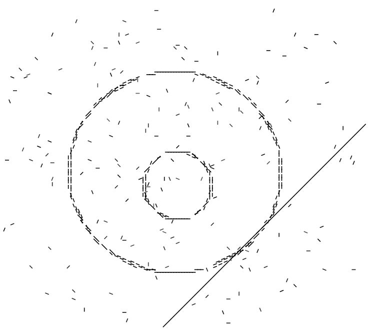
$k = 1.5, \alpha = 0.0055$
$i = 12282$



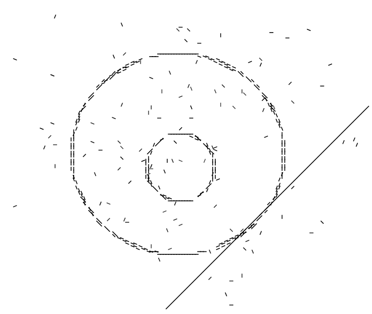
$k = 1.5, \alpha = 0.001$
$i = 12132$



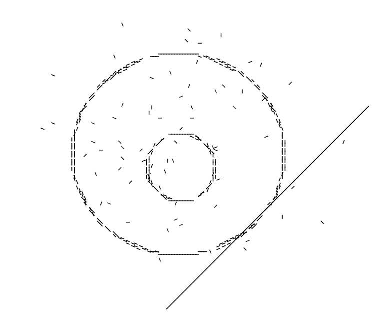
$k = 1.5, \alpha = 0.01$
$i = 11460$



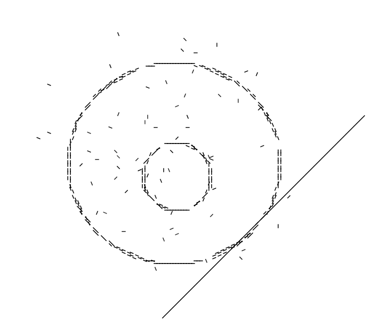
$k = 1.5, \alpha = 0.03$
$i = 11526$



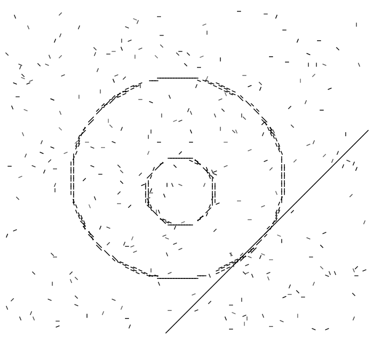
$k = 1.5, \alpha = 0.05$
$i = 13062$



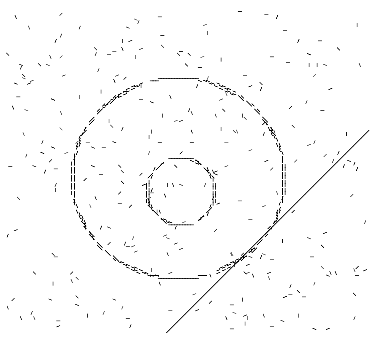
$k = 1.5, \alpha = 0.07$
$i = 11112$



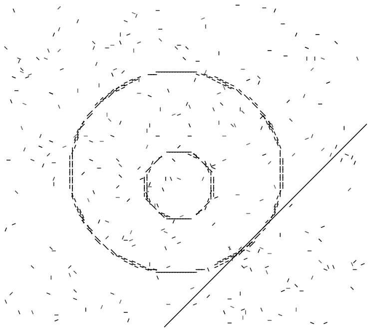
$k = 1.5, \alpha = 0.075$
$i = 12270$



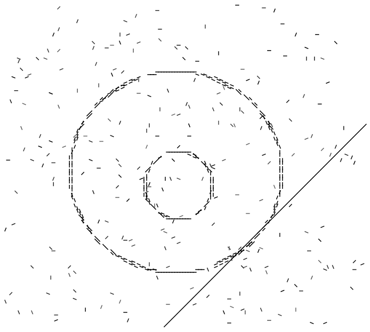
$k = 1, \alpha = 0.0055$
$i = 12504$



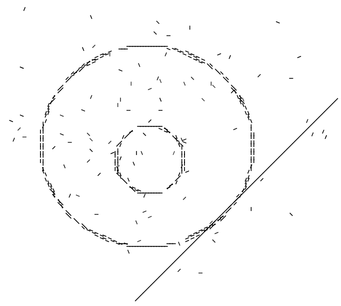
$k = 0.75, \alpha = 0.0055$
$i = 13092$



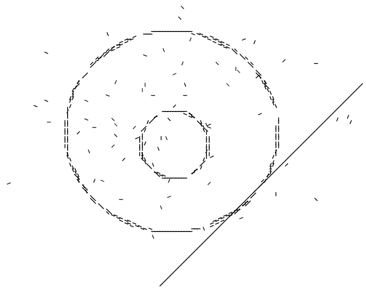
$k = 0.5, \alpha = 0.0055$
$i = 12600$



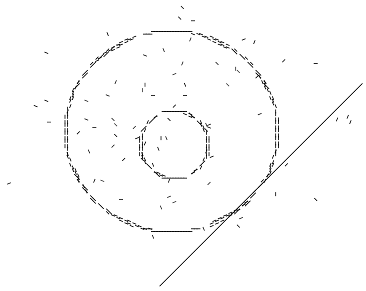
$k = 0.25, \alpha = 0.0055$
$i = 13014$



$k = 0.5, \alpha = 0.04$
$i = 13188$



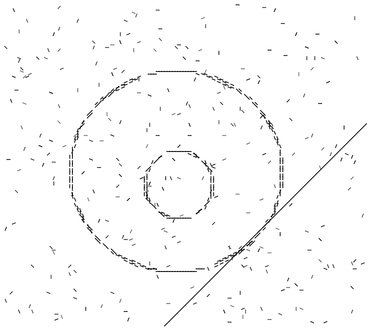
$k = 0.5, \alpha = 0.05$
$i = 12978$



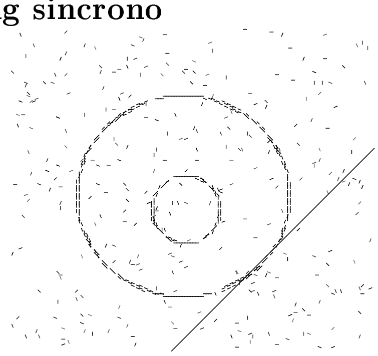
$k = 0.6, \alpha = 0.055$
$i = 12438$

iterazioni	k	α	edgels finali	edges finali
12282	1,5	0,0055	465	710
12132	1,5	0,001	487	732
11460	1,5	0,01	441	686
11526	1,5	0,03	349	594
13062	1,5	0,05	288	532
11112	1,5	0,07	242	486
12270	1,5	0,075	235	478
12504	1	0,0055	458	703
13092	0,75	0,0055	454	699
12600	0,5	0,0055	447	692
13014	0,25	0,0055	430	675
13188	0,5	0,04	266	510
12978	0,5	0,05	236	475
12438	0,6	0,055	232	473

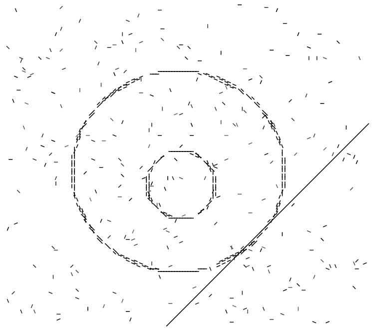
5.2.2. Mean Field Annealing sincrono



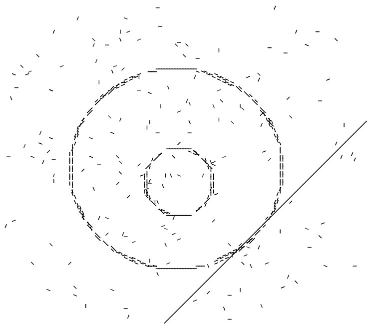
$k = 1.5, \alpha = 0.0055$
$i = 11910$



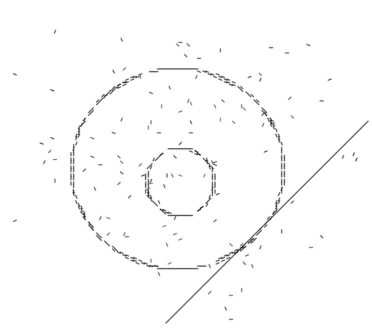
$k = 1.5, \alpha = 0.001$
$i = 11856$



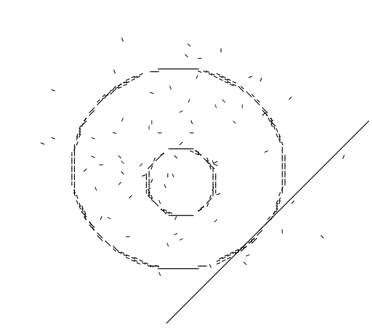
$k = 1.5, \alpha = 0.01$
$i = 10890$



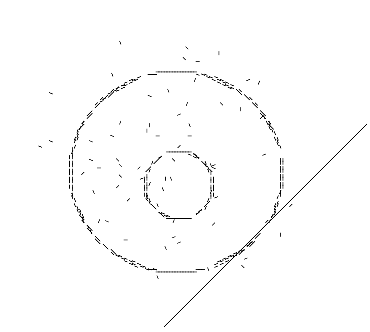
$k = 1.5, \alpha = 0.03$
$i = 10950$



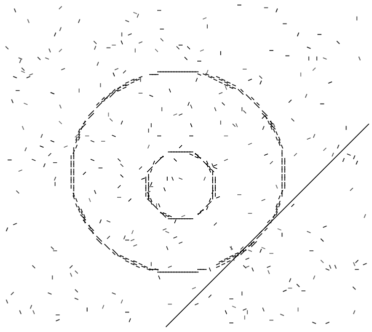
$k = 1.5, \alpha = 0.05$
$i = 12366$



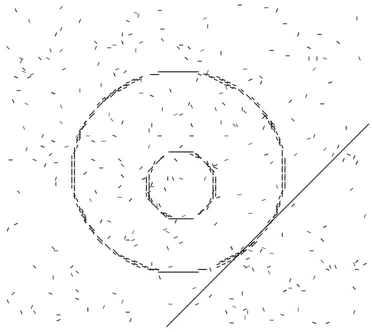
$k = 1.5, \alpha = 0.07$
$i = 10584$



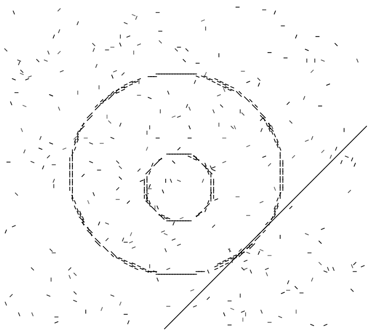
$k = 1.5, \alpha = 0.075$
$i = 11868$



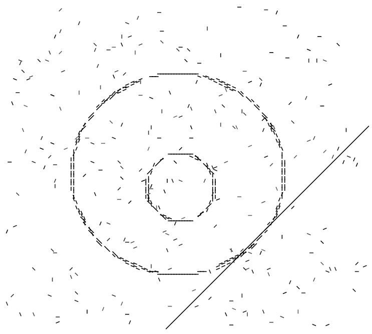
$k = 1, \alpha = 0.0055$
$i = 11994$



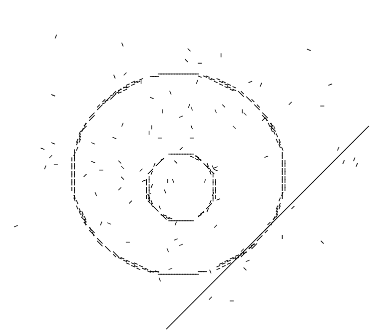
$k = 0.75, \alpha = 0.0055$
$i = 12666$



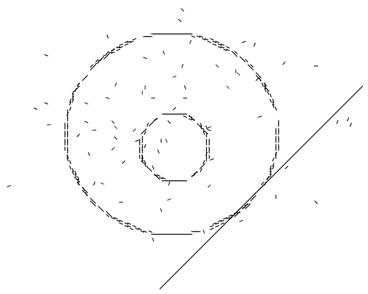
$k = 0.5, \alpha = 0.0055$
$i = 12084$



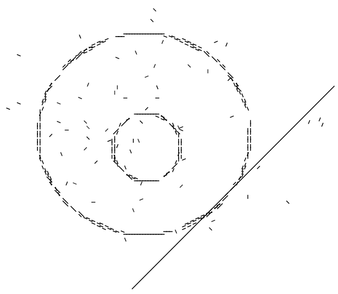
$k = 0.25, \alpha = 0.0055$
$i = 12378$



$k = 0.5, \alpha = 0.04$
$i = 12588$



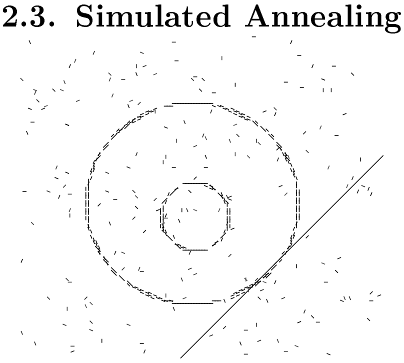
$k = 0.5, \alpha = 0.05$
$i = 12120$



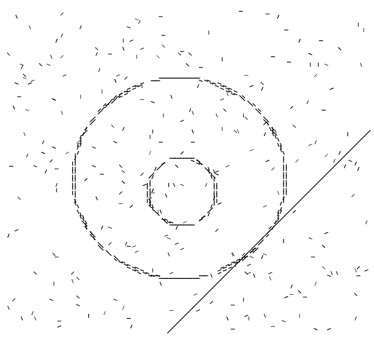
$k = 0.6, \alpha = 0.055$
$i = 11934$

iterazioni	k	α	edgels	edges
11856	1,5	0,001	487	732
10890	1,5	0,01	441	686
10950	1,5	0,03	349	594
12366	1,5	0,05	288	532
10584	1,5	0,07	242	486
11868	1,5	0,075	235	478
11994	1	0,0055	458	703
12666	0,75	0,0055	454	699
12084	0,5	0,0055	447	692
12378	0,25	0,0055	430	675
12588	0,5	0,04	266	510
12120	0,5	0,05	236	475
11934	0,6	0,055	231	470

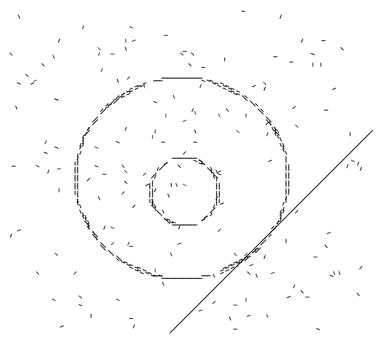
5.2.3. Simulated Annealing



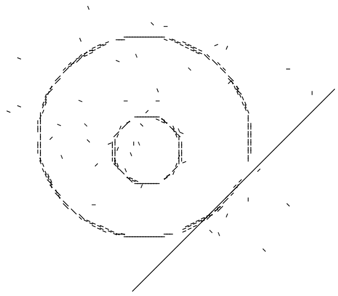
$\alpha = 0.0055$
$i = 509953$



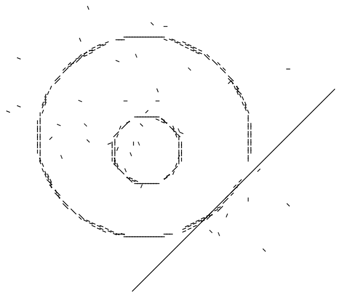
$\alpha = 0.001$
$i = 623826$



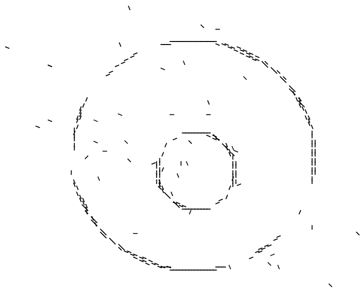
$\alpha = 0.01$
$i = 455492$



$\alpha = 0.03$
$i = 381227$



$\alpha = 0.033$
$i = 396080$



$\alpha = 0.04$
$i = 376276$

iterazioni	α	edgels finali	edges finali
509953	0,0055	375	620
623826	0,001	439	684
455492	0,01	335	580
381227	0,03	212	451
396080	0,033	203	441
376276	0.04	164	300

5.2.4. Immagine non di sintesi con MFA asincrono

Confrontando questi risultati con i precedenti è evidente come il parametro α necessiti di aggiustamenti relativi all'immagine considerata e, più precisamente, al numero di edgels della stessa. La ragione è da ricercarsi nella normalizzazione della matrice dei coefficienti di interazione che utilizza il parametro α moltiplicato per il numero degli edgels.

Immagine originale

MFA sincrono, $k = 1.5$, $\alpha = 0.0055$

MFA asincrono, $k = 1.5$, $\alpha = 0.0055$

MFA asincrono, $k = 1.5$, $\alpha = 0.002$

MFA asincrono, $k = 1.5$, $\alpha = 0.001$

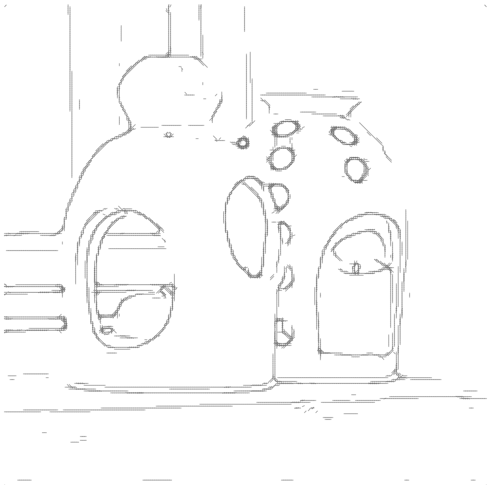
5.2.5. Andamento degli spin con le iterazioni

L'analisi dell'evoluzione temporale delle componenti del vettore μ degli spin, mostra come realmente queste si avvicinino ai loro valori limite $(+1, -1)$ confermandoci il buon funzionamento degli algoritmi utilizzati. I grafici relativi sono in fig. 5 e fig. 6.

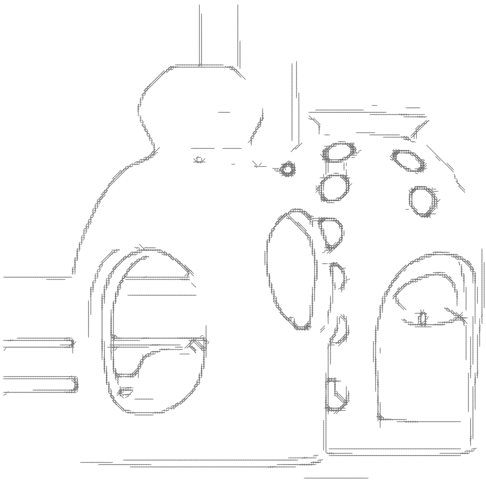
5.2.6. Altri casi reali

Questi risultati sono stati ottenuti con il Mean Field Annealing asincrono variando il solo parametro α.

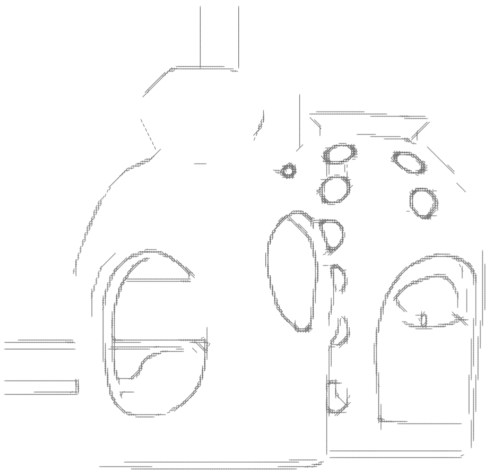
Immagine originale



Contorni (6727 edges, 1836 edgels)



$\alpha = 0.0005$



$\alpha = 0.0010$

Il seguente risultato è stato ottenuto con un thresholding degli edgel di almeno due edge (originariamente 4953 edgels):

Immagine originale

Contorni (12628 edges, 2895 edgels)

$\alpha = 0.0001$

$\alpha = 0.0010$

6. MANUALE PER L'UTENTE

Entrambi i programmi possono essere compilati sia per Windows che per Linux ed hanno le stesse modalità di utilizzo.

6.1. Loglin: implementazione funzionale di un edge detector

L'utilizzo di *loglin*, ovvero del riconoscitore di *edge*, è molto semplice. Nel qual caso si tenti di eseguire il programma senza parametri o con parametri errati viene visualizzato sullo schermo il messaggio che segue:

```
Linear Logic Edge Detector
Usage: loglin.exe [options] file
  file           (pgm/pbm/ppm) file containing ascii or binary image
```

```
Options:
  -help           display this help message
  -ndirs          number of directions (only even values) (default = 16)
  -t              threshold value (default = 0.01)
```

Da una rapida lettura se ne evince che il programma necessita di un parametro obbligatorio: il nome del file su cui si vuole effettuare il riconoscimento degli edge. I formati di immagine letti sono *pgm, pbm, ppm*[10] e possono essere memorizzati indifferentemente in versione binaria o ascii.

Chiaramente, essendo le immagini *ppm* a colori, prima di eseguire la edge detection viene eseguita una trasformazione in toni di grigio tramite la seguente formula (standard ITU YCbCr):

$$grey = \begin{pmatrix} 0.2125 & 0.7154 & 0.0721 \end{pmatrix} \begin{pmatrix} red \\ green \\ blue \end{pmatrix}$$

Il programma *loglin* può essere altresì eseguito con dei parametri opzionali:

- Il parametro *-help*, come nella maggior parte del software, visualizza un messaggio di aiuto per l'utente,
- il parametro *-ndirs* specifica il numero di direzioni in cui viene discretizzato l'angolo degli edge; viene supposto essere pari,
- il parametro *-t* seguito da un numero floating point viene utilizzato come threshold.

6.2. Camleye: implementazione funzionale di un figure ground discriminator

L'utilizzo di *camleye*[11], ovvero del figure ground discriminator, è un po' più complesso. Nel qual caso si tenti di eseguire il programma senza parametri o con parametri errati viene visualizzato sullo schermo il messaggio che segue:

```
Caml's eye
Usage: camleye.exe [options] file
  file            ps file containing edges

Options:
  -help           display this help message
  -k              k value (default = 1.5)
  -alpha          alpha value (default = 0.0055)
  -n              minimum number of edge in an edgel (default = 1)
  -s              save edge map
  -syn            MFA's synchronous version
  -sa             Simulated Annealing algorithm
```

[10] La scelta di tali formati è dipesa dalla semplicità con la quale possono essere letti.

[11] Il nome *Caml's eye* ovvero *occhio del cammello* è stato determinato dal fatto che il codice è stato scritto in Objective Caml e risolve un problema di visione (artificiale). Ci piace pensare che l'output del nostro programma corrisponda a come il nostro cammello riconosca le figure all'interno di un immagine. Giochi di parole di questo tipo sono all'ordine del giorno per il software sviluppato con tale linguaggio.

Il programma necessita di un parametro obbligatorio il nome del file postscript contenente una edge map creata tramite il programma loglin. Nel qual caso il file non esista o gli venga dato un ps arbitrario viene restituito un errore.

Il programma *loglin* può essere altresì eseguito con dei parametri opzionali:

- Il parametro *-help*, come nella maggior parte del software, visualizza un messaggio di aiuto per l'utente,
- il parametro *-k* seguito da un numero floating point setta il valore della variabile k alla quale si fa riferimento nell'articolo [1],
- il parametro *-alpha* seguito da un numero floating point setta il valore della variabile α alla quale si fa riferimento nell'articolo [1], nel caso del Simulated Annealing corrispondealla variabile λ,
- il parametro *-n* seguito da un intero serve a scartare tutti gli edgel che sono composti da meno di n edge (è una forma di thresholding),
- il parametro *-s* serve per salvare la edge map risultante dall'operazione di thresholding appena descritta,
- il parametro *-syn* serve per utilizzare la versione sincrona nel qual caso si utilizzi l'MFA,
- il parametro *-sa* serve per utilizzare l'algoritmo Simulated Annealing in luogo dell'MFA.

7. RINGRAZIAMENTI

Il progetto qui presentato è stato sviluppato a Mestre (Venezia), Treviso e Belluno in un periodo di circa due mesi a cavallo tra il 2003 ed il 2004 (17 novembre 2003 - 25 gennaio 2004). Sono state utilizzate risorse del Dipartimento di Scienze dell'Informazione e risorse personali degli autori. In modo particolare l'acquisto di un elaboratore portatile di ultima generazione da parte di uno dei componenti del gruppo ci ha permesso di ottenere risultati molto più in fretta.

Ringraziamo inoltre le seguenti persone: Giacomo Colle, per il reperimento del materiale fotografico, Pietro Perona, per l'originale di Paolina Borghese, Antonino Salibra e Marta Simeoni, per l'accesso all'aula tesisti, Alvise Spanò, per i preziosi consigli su OCaml, Lucia Gianmaria, per l'ospitalità.

A. Appendice: Scelta del linguaggio

Abbiamo deciso di sviluppare questo progetto facendo uso di un linguaggio a paradigma funzionale quale OCaml (Objective Caml versione 3.0.7), un dialetto del più noto ML, in quanto meccanismi quali *type inference* e *pattern matching* consentono di diminuire sensibilmente i tempi di debugging e di utilizzare agevolmente anche strutture dati complesse.

Inoltre la disarmante semplicità con cui si possono definire funzioni di ordine superiore e il polimorfismo[12] di cui OCaml è dotato sono strumenti, a nostro avviso, indispensabili per poter progettare in modo veloce e chiaro un software così elaborato. La velocità e l'efficienza del garbage collector di OCaml ci hanno, inoltre, permesso di ottenere (per quanto possibile) una buona velocità di esecuzione.

Per il passaggio dei dati fra i computer nell'implementazione parallela del Mean Field Annealing abbiamo utilizzato MPI (Message Passing Interface) versione 1.2.5, una libreria per la programmazione parallela in stile MIMD (Multiple Instruction, Multiple Data). Esistono dei bindings per OCaml (scritti da Leroy) per un largo sottoinsieme delle funzioni di MPI.

REFERENCES

1. L. Hérault, R. Horaud, *Figure Ground Discrimination: A Combinatorial Optimization Approach*, IEEE PAMI, Vol 15, No 9, pp. 899-914, September 1993.
2. A. Amir , M. Lindenbaum, *Ground from Figure Discrimination*, Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, p.521, June 23-25, 1998
3. P. Parent and S.W.Zucker. *Trace inference, Curvature Consistency and Curve Detection.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 11(8):823-839, August 1989.
4. Lee A. Iverson, Steven W. Zucker: *Logical/Linear Operators for Image Curves.* IEEE Transactions on Pattern Analysis and Machine Intelligence 17(10): 982-996 (1995)
5. L.A. Iverson, *Logical/Linear Operators for Image Curves implementation.* `ftp://ftp.cim.mcgill.ca/pub/people/leei/loglin.tar.gz`
6. L.A. Iverson, *Toward Discrete Geometric Models for Early Vision*, Ph.D. Dissertation, McGill University, Montreal, 1993.

[12] Che favorisce in modo incredibile il riuso di codice.

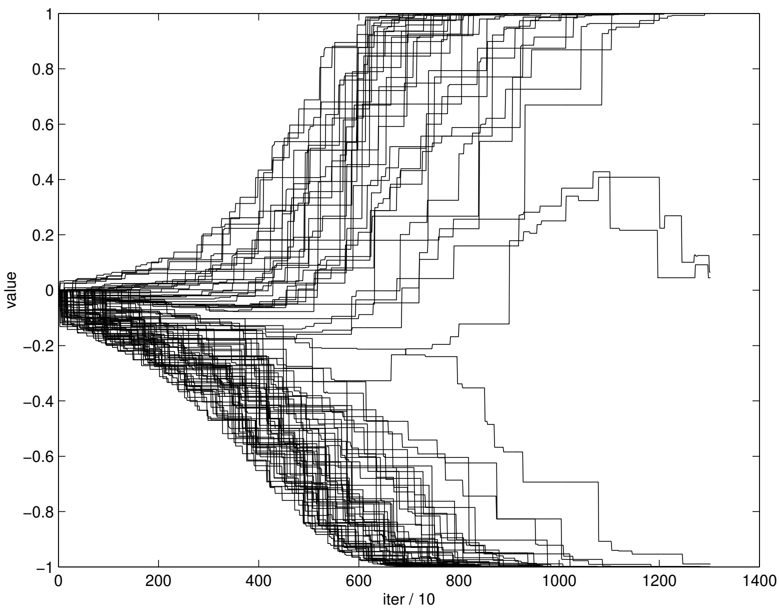


Figura 5. Andamento temporale degli spins nella versione asincrona

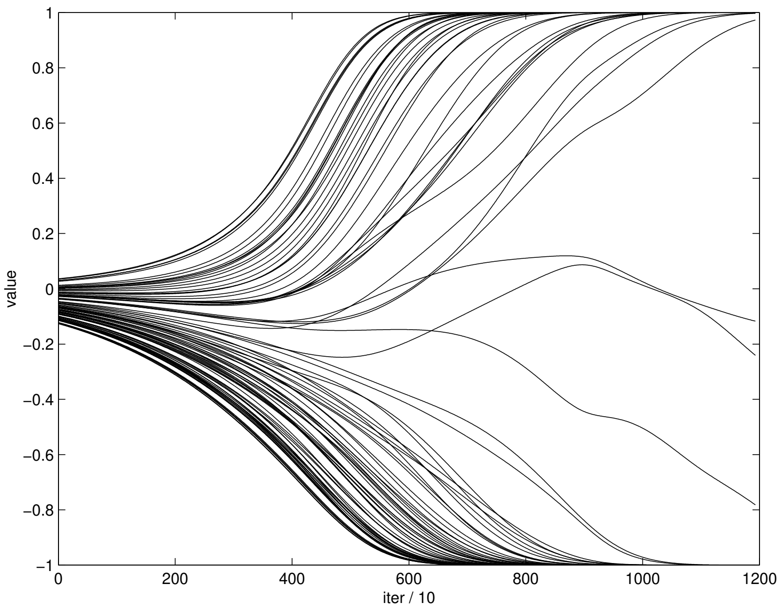


Figura 6. Andamento temporale degli spins nella versione sincrona