FUNZIONI DI TAGLIA PER IMMAGINI SVG

Introduzione

Il ventunesimo secolo che si è aperto da pochi anni sarà senza dubbio il secolo dell'informazione, intendendo con questo termine qualsiasi forma di comunicazione o contenuto espressivo che abbia un modo per essere conservata. In particolare sarà il campo dell'informazione digitale (o informatica) a dover stare al passo con le sempre più ampie e d'altro canto specifiche richieste della vita contemporanea. La quantità di informazioni, e nello specifico delle informazioni digitali, è destinata ad aumentare in modo esponenziale, perciò è compito della ricerca scientifica e tecnologica trovare degli strumenti che ne permettano una facile fruibilità, in modo che esse non vadano sprecate. In questa ottica si vorrebbe sviluppare un metodo di indicizzazione dei prodotti digitali, in particolare di quelli presenti sul web, che superi l'approccio delle parole chiave utilizzato dai più comuni motori di ricerca. L'idea, che porterebbe al superamento delle barriere linguistiche e delle limitazioni dovute all'utilizzo del mezzo espressivo verbale, è quella di associare ad ogni tipo di prodotto digitale (testo, immagine, file audio...) una sua descrizione semantica o analogica effettuata tramite delle immagini semplici ed evocative: le *keypics*.

All'interno di questo ambizioso progetto, ma anche come questione a sé stante, assume una notevole importanza il problema del confronto e riconoscimento delle immagini. In particolare, all'interno del Dipartimento di Matematica dell'Università di Bologna, si stanno studiando dei metodi, descritti genericamente come *Teoria della Taglia*, che potrebbero essere parti-

colarmente adatti a questo scopo: il riconoscimento della forma degli oggetti, nello specifico delle immagini. L'obiettivo di questa tesi è iniziare lo studio della teoria della taglia su degli oggetti che si presterebbero particolarmente ad essere utilizzati come keypics, ovvero delle immagini di tipo vettoriale non molto elaborate: più o meno il parallelo digitale dei cosiddetti schizzi. A tale scopo si è individuata una prima classe ristretta di immagini di tipo vettoriale e ed è stato implementato un software per sperimentare su di essa le potenzialità della teoria della taglia.

Nel fare questo ci siamo basati principalmente sui risultati teorici che in questi anni sono stati ottenuti per la teoria della taglia, che abbiamo esposto sinteticamente nel primo capitolo. In particolare nella prima parte ci siamo soffermati sulla descrizione delle cosiddette *Funzioni di Taglia*, che rappresentano lo strumento principale della teoria della taglia, cercando di metterne in evidenza i punti essenziali. Nella seconda parte, prendendo spunto da alcuni risultati sulla rappresentazione discretizzata di tali funzioni abbiamo approfondito il caso particolare del calcolo delle funzioni di taglia sui Grafi.
Nel secondo capitolo, invece, abbiamo dato una breve panoramica sulla grafica vettoriale, cercando di introdurre tutti i concetti che poi sarebbero risultati utili nella descrizione del software implementato. Nello specifico abbiamo presentato il particolare formato vettoriale che abbiamo deciso di utilizzare come standard nello sviluppo del programma, l'SVG, riservando particolare attenzione alle motivazioni che ci hanno portato a questa scelta.
La parte centrale della tesi è costituita dal terzo capitolo in cui abbiamo presentato l'algoritmo utilizzato per calcolare le funzioni di taglia di una particolare classe di immagini vettoriali. In questa parte ci siamo basati molto sul contributo di D'Amico (cfr. [7]), il quale ha a sua volta sviluppato un software analogo, ma destinato ad immagini di tipo bitmap. Abbiamo cercato di dare una descrizione molto precisa dell'algoritmo, sorvolando invece sui dettagli implementativi. Laddove ci è sembrato che questi fossero particolarmente importanti, li abbiamo inseriti all'interno delle appendici che

si trovano alla fine di questa tesi. In esse abbiamo talvolta presentato dei pezzi del codice sorgente del software, soprattutto di quelle parti che risultano di particolare importanza all'interno dell'algoritmo.
In ultimo, nel quarto capitolo, abbiamo descritto e discusso un esperimento che abbiamo effettuato con il nostro software su di un particolare insieme di immagini campione. Lo scopo di questo esperimento, e di questa tesi in generale, è stato quello di inziare a muovere i primi passi verso quello che è il vero obiettivo per il futuro, ovvero lo studio della teoria della taglia sulle immagini vettoriali. Benchè i risultati siano ancora ridotti, siamo convinti che la strada intrapresa possa essere il giusto approccio al problema: per ciò nel corso dell'intera tesi abbiamo sempre cercato di evidenziare i limiti dell'attuale impostazione, dando però dei suggerimenti su come proseguire gli studi.

Indice

Capitolo 1

La teoria della taglia

1.1 Introduzione alla teoria della taglia

L'argomento di questa tesi si inquadra nel vasto campo della visione artificiale: in particolare riguarda il tema del riconoscimento della forma (*pattern recognition*). In letteratura sono presenti diversi approcci a questo genere di problemi, di tipo statistico, analitico (per mezzo di vari tipi di trasformate), geometrico-topologico e altro ancora. In questa tesi ci occuperemo della cosiddetta Teoria della Taglia, uno strumento matematico in via di sperimentazione che affronta i problemi relativi alla forma e al suo riconoscimento scegliendo come punto di vista quello geometrico-topologico.
L'obiettivo della teoria della taglia è cercare di stabilire quando due forme sono simili, dove i criteri di somiglianza posso essere i più vari. Gli strumenti utilizzati a questo scopo sono le *Distanze Naturali di Taglia* e le *Funzioni di Taglia*. Questi due concetti danno alla Teoria della Taglia una caratteristica molto importante: la modularità. Ovvero gli strumenti utilizzati per descrivere e confrontare le forme dipendono dalla scelta arbitraria di una funzione (detta *funzione misurante*) le cui caratteristiche si possono individuare a seconda delle esigenze del problema. È la scelta di questa funzione che stabilisce l'ambito dei criteri di somiglianza. Ciò si ha perchè entrambi gli strumenti della teoria della taglia ereditano dalla funzione misurante even-

tuali sue invarianze per determinati tipi trasformazioni (affini, proiettive, o altro). Per esempio se una funzione misurante ϕ è invariante per isometrie, sia le distanze naturali di taglia sia le funzioni di taglia risulteranno esse stesse invarante per isometrie.
Tale fatto è molto importante; infatti, continuando l'esempio, è abbastanza chiaro che la percezione umana riesce a riconoscere quando due figure sono costituite dalla stessa forma, disposta nello spazio in modo differente. Lo stesso avviene per i descrittori della forma della teoria della taglia se la funzione misurante scelta è invariante per isometrie: la trasformazione isometrica "non inganna" la teoria della taglia.

1.1.1 Distanze naturali di taglia

Come è stato detto nel paragrafo precedente la teoria della taglia affronta i problemi legati alla forma da un punto di vista geometrico-topologico, per cui il modo naturale che ha per rappresentare le forme è quello ci considerarle spazi topologici. In particolare in questo capitolo M sarà sempre uno spazio topologico compatto (che tipicamente si può immaginare immerso in uno spazio euclideo $\mathbb{R}^k$, anche se questa ipotesi può in certi casi essere trascurata). Data una funzione continua ϕ, definita su M e a valori in $\mathbb{R}$, la coppia (M, ϕ) si dirà *coppia di taglia*, e la funzione ϕ sarà la funzione misurante. Indicheremo con *Size* l'insieme delle coppie di taglia.

Definizione 1.1. Se (M, ϕ) e (N, ψ) sono due coppie di taglia, indichiamo con $H(M, N)$ l'insieme degli omeomorfismo da M ad N. Per ogni omeomerfismo $f \in H(M, N)$ definiamo la funzione $\Theta : H(M, N) \longrightarrow \mathbb{R}$ nel seguente modo:

$$\Theta(f) := \max_{P \in M} |\phi(P) - \psi(f(P))|$$

La funzione Θ si dice *Misura Naturale di Taglia* su $H(M, N)$ relativa alle funzioni misuranti ϕ e ψ.

La misura naturale di taglia dà un'indicazione di quanto f modifica i

valori delle due funzioni misuranti. A partire da essa di può dare la seguente definizione:

Definizione 1.2. Si indica con $\Sigma : \mathit{Size} \times \mathit{Size} \longrightarrow \mathbb{R} \cup \{+\infty\}$ la funzione definita da:

$$\Sigma\left((M,\phi),(N,\psi)\right) := \begin{cases} \inf_{f \in H(M,N)} \Theta(f) \text{ , se } H(M,N) \neq \emptyset \\ +\infty \text{ , altrimenti} \end{cases}$$

Si verifica che la funzione Σ è una pseudometrica su *Size*, nel senso che date due coppie $(M,\phi),(N,\psi)$ tali che $\Sigma\left((M,\phi),(N)\right) = 0$ non necessariamente $(M,\phi) = (N,\psi)$, dove due coppie di taglia sono uguali se $M = N$ e le due funzioni coincidono.

Definizione 1.3. La metrica $\sigma : \mathit{Size}/_{\approx} \times \mathit{Size}/_{\approx} \longrightarrow \mathbb{R} \cup \{\infty\}$ indotta dalla pseudometrica Σ sullo spazio quoziente $\mathit{Size}/_{\approx}$ si dice *Distanza Naturale di Taglia*, dove $\approx$ è la relazione di equivalenza definita nel seguente modo: $(M,\phi) \approx (N,\psi) \Leftrightarrow \Sigma\left((M,\phi),(N,\psi)\right) = 0$. La classe di equivalenza di una coppia di taglia (M,ϕ) si indica $[(M,\phi)]$.

Definizione 1.4. Si dice che $\bar{f} \in H(M,N)$ è un omeomorfismo ottimale se $\Theta(\bar{f}) = \sigma\left([(M,\phi)],[(N,\psi)]\right)$.

È importante osservare che un omeomorfismo ottimale in generale non esiste, neanche se assumiamo forti ipotesi di regolarità per M, N, ϕ e ψ. Ma, assumendo l'esistenza di un tale omeomorfismo e una sufficiente regolarità per le due funzioni misuranti, si può dimostrare un importante risultato che lega i punti critici di ϕ e ψ con la distanza naturale di taglia. Ovvero:

Teorema 1.1. *Siano (M,ϕ) e (N,ψ) due coppie di taglia e supponiamo che esista un omeomorfismo ottimale, allora la distanza naturale di taglia tra $[(M,\phi)]$ e $[(N,\psi)]$ è data dalla distanza euclidea tra un valore critco della ϕ e un valore critico della ψ.*

1.1.2 Funzioni di taglia

Le distanze naturali di taglia sono uno strumento molto efficace per stabilire la somiglianza tra due spazi topologici (e quindi tra le due forme che essi rappresentano), ma hanno il difetto di essere difficilmente calcolabili, in quanto è necessario tener conto di tutti gli omeomerfismi tra i due spazi. Il concetto di funzione di taglia, che verrà ora introdotto, ha invece il pregio di essere più facilmente calcolabile. Inoltre come vedremo nell'ultimo risultato di questo paragrafo, oltre che essere uno strumento molto interessante di per sé, il calcolo delle funzioni di taglia ci permette di ricavare importanti informazioni sulle distanze naturali di taglia.

Definizione 1.5. Consideriamo una coppia di taglia (M, ϕ). Sia $x \in \mathbb{R}$ un numero reale qualsiasi. Si indica con $M_{\phi \leq x}$ l'insieme $\{P \in M : \phi(P) \leq x\}$. Dati due punti P e Q appartenenti ad M, si dice che essi sono $\langle \phi \leq x \rangle$ −connessi se e solo se $P = Q$ oppure esiste un sottoinsieme connesso di $M_{\phi \leq x}$ che contiene entrambi i punti.

Osservazione 1.1. Per come è data la definzione, si verifica rapidamente che, fissato $y \in \mathbb{R}$, la relazione di $\langle \phi \leq y \rangle$ −connessione è una relazione di equivalenza tra i punti di M.

Definizione 1.6. La funzione di Taglia $\ell_{(M,\phi)} : \mathbb{R}^2 \longrightarrow \mathbb{N} \cup \{\infty\}$ associata alla Coppia di Taglia (M, ϕ) è quella funzione che associa ad ogni punto del piano di coordinate (x, y) il numero delle classi di equivalenza nelle quali l'insieme $M_{\phi \leq x}$ viene diviso dalla relazione di $\langle \phi \leq y \rangle$ −connessione.

Si intende che la relazione di equivalenza è definita su tutto M, e poi viene ristretta ad $M_{\phi \leq x}$. Dunque due punti P e Q in $M_{\phi \leq x}$ (ovvero $\phi(P), \phi(Q) \leq x$) sono $\langle \phi \leq y \rangle$ −connessi anche se l'insieme connesso che li contiene entrambi è un sottoinsieme di $M_{\phi \leq y}$.

Osservazione 1.2. Se si indica con $\Delta = \{(x, y) \in \mathbb{R}^2 : x < y\}$ (ovvero il semipiano aperto superiore individuato dalla retta di equazione $x = y$), e

si assume che la restrizione della funzione di taglia a Δ abbia in un punto (x, y) valore finito, allora tale valore assume un'altra interessante interpretazione. Ovvero esso rappresenta il numero di componenti connesse di $M_{\phi \leq y}$ che hanno almeno un punto in $M_{\phi \leq x}$.

Vediamo di seguito alcune immediate proprietà delle funzioni di taglia:

Proposizione 1.2. *Sia (M, ϕ) una coppia di taglia e $\ell_{(M,\phi)}$ la corrispondente funzione di taglia, allora:*

1. *$\ell_{(M,\phi)}(x, y)$ è non decrescente in x e non crescente in y*
2. *se M è anche localmente connesso allora, su Δ, $\ell_{(M,\phi)}(x, y)$ assume sempre valori finiti*
3. *$\ell_{(M,\phi)}(x, y) = 0$, se $x < \min_{P \in M} \phi(P)$*
4. *$\ell_{(M,\phi)}(x, y) = +\infty$ per ogni x, y per i quali esiste un punto non isolato $P \in M$ tale che $y < \phi(P) < x$*
5. *per ogni $y \geq \max_{P \in M} \phi(P)$ si ha che $\ell_{(M,\phi)}(x, y)$ è pari al numero di componenti connesse M_α di M per le quali $x \geq \min_{P \in M_\alpha} \phi(P)$. In particolare se anche $x \geq \max_{P \in M} \phi(P)$ allora $\ell_{(M,\phi)}(x, y)$ è pari al numero di componenti connesse di M.*

Questi risultati mettono in evidenza come la parte rilevante di una funzione di taglia sia la sua restrizione all'insieme

$$\Delta \cap \left\{ (x, y) \ : \ y \leq \max_{P \in M} \phi(P), x \geq \min_{P \in M} \phi(P) \right\}$$

I prossimi due teoremi costituiscono i principali risultati che mettono in relazione le funzioni di taglia con le distanze naturali di taglia.

Teorema 1.3. *Sia $\epsilon > 0$. Se $\sigma\left([(M, \phi)], [(N, \psi)]\right) < \epsilon$, allora per ogni $h \geq \epsilon$ e per ogni $x, y \in \mathbb{R}$ si verificano le seguenti disuguaglianze:*

$$\ell_{(M,\phi)}(x - h, y + h) \leq \ell_{(N,\psi)}(x, y) \leq \ell_{(M,\phi)}(x + h, y - h)$$

Teorema 1.4. *Supponiamo che esistano $x_1, y_1, x_2, y_2 \in \mathbb{R}$ tali che valga la disuguaglianza $\ell_{(M,\phi)}(x_1, y_1) < \ell_{(N,\psi)}(x_2, y_2)$. Allora si verifica che:*

$$\sigma\left([(M,\phi)], [(N,\psi)]\right) \geq \min\{x_1 - x_2, y_2 - y_1\}$$

Si deve osservare che l'ultimo risultato permette di individuare un limite inferiore per la distanza naturale di taglia grazie alla conoscenza delle funzioni di taglia.

1.1.3 Calcolo delle funzioni di taglia

In questo paragrafo verranno dati dei risultati che ci permettono, sotto determinate ipotesi, di calcolare la funzione di taglia $\ell_{(M,\phi)}(x, y)$ di una coppia di taglia (M, ϕ). La tecnica di calcolo consiste nell'approssimare l'insieme M con un suo sottoinsieme finito P di punti e allo stesso modo approssimare la funzione di taglia $\ell_{(M,\phi)}(x, y)$ con un'altra funzione $\ell_{approx}(x, y)$ più facile da calcolare. Infine grazie ad un risultato che ci dà una correlazione tra la funzione di taglia e la sua approssimazione si può dedurre dalla seconda i valori assunti dalla prima.
Si assume da questo momento che lo spazio topologico M sia un sottoinsieme compatto e localmente connesso di uno spazio euclideo $\mathbb{R}^k$ e che la funzione di taglia ϕ sia la restrizione di una funzione continua a valori reali $\bar{\phi}$ definita su tutto $\mathbb{R}^k$. Si indicherà con $\omega(\delta)$ il *modulo di continuità* della funzione $\bar{\phi}$, ovvero

$$\omega(\delta) = \sup\left\{\left|\bar{\phi}(P) - \bar{\phi}(Q)\right| \,:\, P, Q \in \mathbb{R}^k, \|P - Q\| < \delta\right\}$$

Definizione 1.7. Sia $P = \{P_1, \ldots, P_n\}$ un insieme finito di punti di $\mathbb{R}^k$. Indichiamo con B_δ l'insieme delle $n+1$ palle aperte $B(P_i, \delta)$ di raggio δ e centro i punti $P_i \in P$
B_δ si dice un δ-ricoprimento di M se e solo se

(a) $M \subset \bigcup_{i=0}^{n} B(P_i, \delta)$

(b) per ogni indice $0 \leq i \leq n$, l'insieme $M \cap B(P_i, \delta)$ è un sottospazio non vuoto e connesso di M

Infine sui punti di P definiamo la relazione $\approx_\delta$ definita da: $P_i \approx_\delta P_j$ se e solo se $(B(P_i,\delta) \cup B(P_j,\delta)) \cap M$ è un sottospazio connesso di M

Osservazione 1.3. Si verifica che $\approx_\delta$ è una relazione di equivalenza sui punti di P.

D'ora in poi B_δ sarà un δ-ricoprimento di M e $P = \{P_0, \ldots, P_n\}$ sarà l'insieme dei suoi centri. Riprendendo la notazione già introdotta in precedenza per M, dati $x, y \in \mathbb{R}$ indichiamo con $P_{\bar\phi \le x}$ l'insieme dei punti di P sui quali la funzione $\bar\phi$ assume valori minori di x, e con $\approx_{\bar\phi \le y}$ la relazione d'equivalenza sui punti di P per la quale $P_r \approx_{\bar\phi \le y} P_s$ se e solo se $P_r = P_s$ oppure esiste una successione finita di punti $\left\{P_{t(i)}\right\}_{i=0,\ldots,h}$ in $P_{\bar\phi \le y}$ tale che $P_{t(0)} = P_r$, $P_{t(h)} = P_s$ e per ogni indice $0 \le i \le h-1$ si ha $P_{t(i)} \approx_\delta P_{t(i+1)}$. Questa relazione di equivalenza si dice $\langle \bar\phi \le y \rangle$-*equivalenza relativa a* B_δ.
Ora l'insieme P rappresenta l'approssimazione di M; per cui si definisce il valore della funzione $\ell_{approx}(x,y)$ come il numero di classi di equivalenza in cui $P_{\bar\phi \le x}$ è diviso dalla $\langle \bar\phi \le y \rangle$-equivalenza. Quello che è interessante è il legame tra la $\ell_{(M,\phi)}$ e la ℓ_{approx} dati dai due seguenti risultati:

Teorema 1.5. *Per ogni* $x, y \in \mathbb{R}$ *e per ogni* $\bar\omega \ge \omega(\delta)$ *per cui* $x + \bar\omega \le y - \bar\omega$, *si hanno le seguenti disuguaglianze:*

$$\ell_{approx}(x - \bar\omega, y + \bar\omega) \le \ell_{(M,\phi)}(x,y) \le \ell_{approx}(x + \bar\omega, y - \bar\omega)$$

$$\ell_{(M,\phi)}(x - \bar\omega, y + \bar\omega) \le \ell_{approx}(x,y) \le \ell_{(M,\phi)}(x + \bar\omega, y - \bar\omega)$$

Corollario 1.6. *Siano* $\bar x, \bar y, b, c \in \mathbb{R}$*con* $b, c \ge 0$*; sia poi* $\bar\omega \ge \omega(\delta)$, *tale che* $\bar x + \bar\omega \le \bar y - \bar\omega$. *Se la funzione* ℓ_{approx} *ha un valore costante* v *nei punti* $(\bar x + \bar\omega, \bar y - \bar\omega)$ *e* $(\bar x - b - \bar\omega, y + c + \bar\omega)$, *allora anche la funzione di taglia* $\ell_{(M,\phi)}(x,y) = v$ *nel rettangolo*

$$R = \left\{(x,y) \in \mathbb{R}^2 \,:\, \bar x - b \le x \le \bar x, \bar y \le y \le \bar y + c\right\}$$

Dal corollario segue un immediato metodo di calcolo di $\ell_{(M,\phi)}$. Scelto un arbitrario $\bar\omega \ge \omega(\delta)$, si calcola la funzione ℓ_{approx} nei punti dell'insieme

$$S_{\bar\omega} = \left\{(x,y) \in \mathbb{R}^2 : x = i\bar\omega, y = j\bar\omega \text{ , dove } i, j \in \mathbb{Z}, i \le j\right\}$$

Ogni qualvolta la funzione ℓ_{approx} assume lo stesso valore v in due punti differenti $(i\bar{\omega}, j\bar{\omega})$ e $((i-p)\,\bar{\omega}, (j+q)\,\bar{\omega})$ di $S_{\bar{\omega}}$ dove $p, q \geq 2$, allora nel rettangolo definito dai vertici $((i-1)\,\bar{\omega}, (j+1)\,\bar{\omega})$ e $((i-p+1)\,\bar{\omega}, (j+q-1)\,\bar{\omega})$ anche la funzione di taglia $\ell_{(M,\phi)}$ vale v.
Si deve osservare che grazie a questo procedimento il calcolo della funzione approssimante ℓ_{approx} ci permette di conoscere l'esatto valore della funzione di taglia in determinate zone del piano. D'altra parte utilizzando la definizione 1.6 sappiamo che $\ell_{(M,\phi)}$ assume solo valori interi, inoltre per le proprietà elencate in 1.2 si conosce anche l'andamento crescente e decrescente lungo le direzioni delle ascisse e ordinate. Per cui l'incertezza data dall'approssimazione il più delle volte consiste solo nella posizione dei punti di discontinuità della funzione di taglia stessa.

1.1.4 Rappresentazione delle funzioni di taglia

In questa sezione vedremo come la conoscenza dei punti di discontinuità di una funzione di taglia ci permette di rappresentarla in maniera compatta come una serie formale di punti del piano. In pratica tutte le informazioni importanti di una funzione di taglia sono contenute nei suoi punti di discontinuità.
Come detto in 1.2 la funzione $\ell_{(M,\phi)}$ è non decrescente nella variabile x e non crescente nella variabile y. Inoltre, essendo per definizione $\ell_{(M,\phi)}$ a valori interi, se assume più di un valore (non è costante) non può essere continua. Perciò per ogni $\bar{x}$ fissato la funzione $\ell_{(M,\phi)}(\bar{x}, \cdot)$ della sola variabile y data dalla restrizione della $\ell_{(M,\phi)}$ alla retta di equazione $x = \bar{x}$ avrà dei ben determinati punti in cui salta da un valore più alto ad uno più basso. Lo stesso varrà per la $\ell_{(M,\phi)}(\cdot, \bar{y})$, ma con salti verso valori maggiori. Si può dimostrare il seguente risultato:

Proposizione 1.7. *(i) se $\bar{x} < \bar{y}$ è un punto di discontinuità per $\ell_{(M,\phi)}(\cdot, \bar{y})$, allora $\bar{x}$ è un punto di discontinuità anche di $\ell_{(M,\phi)}(\cdot, y)$ per ogni y tale che $\bar{x} < y < \bar{y}$*

(i) se $\bar{y} > \bar{x}$ è un punto di discontinuità per $\ell_{(M,\phi)}(\bar{x}, \cdot)$, allora $\bar{y}$ è un punto di discontinuità anche di $\ell_{(M,\phi)}(x, \cdot)$ per ogni x tale che $\bar{x} < x < \bar{y}$

Si può inoltre dimostrare che nel semipiano aperto Δ la funzione $\ell_{(M,\phi)}$ non ha punti di discontinuità di tipo differente da questo. Perciò i punti di discontinuità di una funzione di taglia hanno una disposizione geometrica molto particolare: individuano dei triangoli rettangoli del semipiano Δ con cateti paralleli agli assi e ipotenusa contenuta nella retta $x = y$. Chiariamo meglio questo punto.

Definizione 1.8. Si definiscono per ogni punto $P = (x, y)$ del piano i valori

$$\begin{aligned}\mu_{\alpha,\beta}(P) &= \ell_{(M,\phi)}(x+\alpha, y-\beta) - \ell_{(M,\phi)}(x+\alpha, y+\beta) - \\ &\quad - \ell_{(M,\phi)}(x-\alpha, y-\beta) + \ell_{(M,\phi)}(x-\alpha, y+\beta) \\ \mu(P) &= \inf\{\mu_{\alpha,\beta}(P) : \alpha, \beta > 0, x+\alpha < y-\beta\}\end{aligned}$$

Se $\mu(P) > 0$ (e si verifica che è sempre un numero intero non negativo) il punto P si dice *punto angolare* e il valore $\mu(P)$ è la sua *molteplicità*.

Definizione 1.9. Allo stesso modo per ogni $\bar{x} \in \mathbb{R}$ si individua la retta r di equazione $r : x = \bar{x}$. Per ognuna di queste rette si pone per definizione:

$$\mu(r) = \inf_{\alpha>0\ ,\ \bar{x}+\alpha<y} \left(\ell_{(M,\phi)}(\bar{x}+\alpha, y) - \ell_{(M,\phi)}(\bar{x}-\alpha, y)\right)$$

Se $\mu(r) > 0$ (e ancora si verifica che può assumere solo valori interi non negativi) allora la retta r si dice *retta angolare* e il numero $\mu(r)$ è la sua molteplicità.

Osservazione 1.4. Usando le proprietà di monotonia delle funzioni di taglia di può dimostrare che

$$\begin{aligned}\mu(P) &= \lim_{\epsilon\to 0} \mu_{\epsilon,\epsilon}(P) = \\ &= \lim_{\epsilon\to 0} \big(\ell_{(M,\phi)}(x+\epsilon, y-\epsilon) - \ell_{(M,\phi)}(x-\epsilon, y-\epsilon) - \\ &\quad - \ell_{(M,\phi)}(x+\epsilon, y+\epsilon) + \ell_{(M,\phi)}(x-\epsilon, y+\epsilon)\big)\end{aligned}$$

per ogni punto $P = (x, y) \in \Delta$ e

$$\mu(r) = \lim_{\epsilon \to 0} \left[\ell_{(M,\phi)} \left(\bar{x} + \epsilon, \frac{1}{\epsilon} \right) - \ell_{(M,\phi)} \left(\bar{x} - \epsilon, \frac{1}{\epsilon} \right) \right]$$

per ogni retta r di equazione $x = \bar{x}$.

In pratica i punti e le rette angolari (supponiamo di averne uno per tipo: $P = (\bar{x}, \bar{y})$ e $r : x = \tilde{x}$) permettono di conoscere tutti i punti di discontinuità della funzione di taglia, in quanto essi sono tutti contenuti nei segmenti $T_P^{(1)}, T_P^{(2)}$ o nelle semirette R_r da essi univocamente determinati secondo le relazioni:

$$\begin{cases} T_P^{(1)} = \{(x, y) \in \mathbb{R}^2 : y = \bar{y}, \bar{x} \leq x \leq \bar{y}\} \\ T_P^{(2)} = \{(x, y) \in \mathbb{R}^2 : x = \bar{x}, \bar{x} \leq y \leq \bar{y}\} \\ R_r = \{(x, y) \in \mathbb{R}^2 : x = \tilde{x}, y \geq \tilde{x}\} \end{cases}$$

Più semplicemente si possono considerare i due segmenti $T_P^{(i)}$ come i cateti di un triangolo rettangolo T_P avente l'angolo retto in P, e l'ipotenusa contenuta nella retta di equazione $x = y$. Inoltre si può interpretare la semiretta R_r come il cateto di un triangolo di area infinita T_r che si trova mandando $\bar{y}$ all'infinito in un triangolo del tipo T_P.
A questo punto possiamo enunciare il seguente teorema di rappresentazione:

Teorema 1.8. *Se si indica con Δ^* l'insieme*

$$\Delta^* = \Delta \cup \{(k, \infty) \ : k \in \mathbb{R}\}$$

allora per ogni $(\bar{x}, \bar{y}) \in \Delta$ si ha:

$$\ell_{(M,\phi)}(\bar{x}, \bar{y}) = \sum_{\substack{(x,y) \in \Delta^* \\ x \leq \bar{x}, y \geq \bar{y}}} \mu((x, y))$$

In maniera del tutto equivalente si può esprimere il teorema di rappresentazione nel seguente modo:

Teorema 1.9. *Sia $\ell_{(M,\phi)}$ una funzione di taglia, allora esiste una successione eventualmente finita di punti non necessariamente distinti $\{Q_n\}_{n \in \mathbb{N}}$ con*

$$Q_i = (x_i, y_i) \in \{(x, y) \in \mathbb{R} \times (\mathbb{R} \cup \{\infty\}) \ : x < y\} = \Delta^*$$

tale che, considerati gli insiemi

$$T_i = T_{Q_i} = \left\{(x, y) \in \mathbb{R}^2 \,:\, x_i \leq x < y < y_i\right\}$$

si ha

$$\ell_{(M,\phi)}(x, y) = \sum_{n\in\mathbb{N}} \chi_{T_n}(x, y) \text{ per ogni } x < y$$

a meno di un insieme di misura nulla, dove χ_{T_n} è la funzione caratteristica del triangolo T_n. Inoltre la successione $\{Q_n\}_{n\in\mathbb{N}}$ è unica a meno di permutazioni.

Si completa questo paragrafo con un risultato che permette di individuare le discontinuità di una funzione di taglia sotto delle ipotesi di regolarità per M e ϕ:

Teorema 1.10. *Supponiamo che M sia una varietà differenziale di classe C^2 e che la funzione misurante ϕ sia di classe C^1. Si ha che, se (x, y) è un punto di discontinuità della funzione di taglia $\ell_{(M,\phi)}$ ed $x < y$, allora o x o y o entrambi sono valori critici di ϕ.*

Nelle applicazioni molto spesso lo spazio M e la funzione ϕ soddisfano, almeno parzialmente, le ipotesi del teorema, per cui questo risultato ci dà un criterio molto importante da considerare quando si passa dallo spazio topologico M alla sua approssimazione P: si devono conservare i punti nei quali la funzione ha un valore critico.

1.2 Funzioni di taglia di grafi

Nella sezione precedente è stato individuato come metodo per calcolare la funzione di taglia di uno spazio topologico M la sua approssimazione con un sottoinsieme finito di suoi punti P che in qualche modo conservi la struttura dello spazio in sé (il concetto di δ-ricoprimento) e sul quale la restrizione della funzione misurante ϕ mantenga alcune proprietà importanti (come i valori critici). La struttura matematica che meglio si presta a descrivere l'insieme P è quella di grafo.

1.2.1 Grafi di taglia

In questa sezione verranno date le definizione di base sui grafi e introdotta la teoria della taglia nel caso particolare in cui lo spazio topologico M è proprio un grafo.

Definizione 1.10. Un grafo G è dato dai seguenti elementi:

1. un insieme finito $V(G) = \{v_1, \ldots, v_n\}$ i cui elementi si dicono *vertici* o *nodi*

2. un particolare sottoinsieme del prodotto cartesiano di $V(G)$ per se stesso $E(G) = \{e_1, \ldots, e_m\} \subset V(G) \times V(G)$ i cui elementi si dicono *spigoli* o *archi*

Gli spigoli si possono dunque indicare come $e = (v, w)$, con $v, w \in V(G)$, noi adottiamo la convenzione di escludere i casi in cui $v = w$ (tali particolari spigoli si dicono *cappi*). Si dice che il grafo G non è orientato se dato $(v, w) \in E(G)$ allora anche $(w, v) \in E(G)$. Se questa proprietà non è necessariamente verificata allora il grafo si dice orientato. In generale quando si parla di grafo si intende un grafo non orientato. Nel caso di grafo orientato si predilige la nomenclatura "nodi" e "archi" per gli elementi di $V(G)$ ed $E(G)$.

In generale si può dare la definizione di grafo anche con infiniti vertici, ma per gli scopi di questa tesi non è necessario introdurre questo caso che comporterebbe definizioni e notazioni più complicate. Inolte la definizione data di grafo non ammette il caso in cui tra due vertici ci sia più di uno spigolo. Con un aggiustamento della definizione si può includere anche questo caso, e si hanno i *multigrafi*, ma come per i grafi eventualmente infiniti tralasciamo la generalizzazione che non ci è utile. Dunque se G è un grafo orientato con $(v, w) \in E(G)$ allora necessariamente $(w, v) \notin E(G)$.
Si può considerare un grafo come un particolare complesso simpliciale monodimensionale (o analogamente un CW-complesso monodimensionale), associando un punto ad ogni vertice (0-simplesso) e ad ogni spigolo $e = (v, w)$ un segmento (un 1-simplesso o più generalmente uno spazio topologico omeomorfo

a $\overline{D^1} \approx [0,1]$) che unisce i punti associati a v e w. In questo modo l'unione di tutte le parti (0- e 1- simplessi) quozientata in modo opportuno, risulta naturalmente uno spazio topologico.
Nella prossima definizione introduciamo gran parte della nomenclatura legata ai grafi.

Definizione 1.11. Sia G un grafo non necessariamente orientato, $v, w \in V(G)$ due suoi vertici e $V \subset V(G)$ un particolare sottoinsieme di $V(G)$. Si dice che:

- v è adiacente a w se esiste $e = (v, w) \in E(G)$ e si scrive $v \rightharpoonup w$;
- v è connesso a w se esiste $\{v_{n(1)}, \dots, v_{n(t)}\}$ successione di vertici distinti tali che $v_{n(1)} = v$, $v_{n(t)} = w$ e $v_{n(i)} \rightharpoonup v_{n(i+1)}$ per ogni indice $1 \leq i \leq t-1$. La successione di spigoli $\{e_i\} = \{(v_{n(i)}, v_{n(i+1)})\}_{1 \leq i \leq t-1}$ si dice *cammino* in G.
- Si dice *ciclo* un cammino $\{e_1, \dots, e_{n(t)}\}$ tale che $e_{n(1)} = (u, v_i)$ e $e_{n(t)} = (v_j, v)$, con $u = v$.
- Se G non è orientato si dice *degree* di $u \in V(G)$ il numero di archi $e = (v, w) \in E(G)$ tale che $v = u$ oppure $w = u$;
- se invece G è orientato si dice *outdegree* di $u \in V(G)$ il numero degli archi $e = (v, w) \in E(G)$ tali che $v = u$ e *indegree* di $u \in V(G)$ il numero degli archi $e = (v, w) \in E(G)$ tali che $w = u$.
- V si dice *componente connessa* di G se tutti i vertici sono connessi tra loro e per ogni $u \in V(G) - V$ si ha che u non è connesso ad alcun vertice di V;
- G si dice connesso se esiste una sola componente connessa in G.
- Il sottografo indotto da G su V è quel grafo G' in cui $V(G') = V$ e $E(G') = \{(v', w') \in E(G) : v', w' \in V\}$.
- Se G è connesso e non contiene cicli si dice *albero*;

- se, invece, non è connesso e non contiene cicli si dice *foresta*. Perciò le componenti connesse di una foresta sono alberi;

- si dice *arborescenza* un particolare albero orientato tale che tutti i nodi hanno indegree pari ad 1, tranne un unico nodo (detto *radice*) che ha indegree pari a 0. In una arborescenza esistono dei nodi che hanno outdegree pari a 0, tali elementi si dicono *foglie*.

- se G è una arborescenza e v non è una foglia, allora i vertici w tali che $(v, w) \in E(G)$ si dicono figli di v, e v si dice padre dei w.

A questo punto, grazie all'osservazione precedente, si può passare a definire la funzione di taglia di un grafo; in particolare si vede come il concetto di connessione tra due vertici appena definito assume il ruolo che in precedenza aveva la connessione topologica (in quanto ne è equivalente). Questo spiega perchè si è dovuto introdurre la struttura di grafo per descrivere l'insieme P che approssima M, quando invece considerandolo come semplice insieme non sarebbe stato possibile trovare un analogo.

Definizione 1.12. Si dice *grafo di taglia* una coppia $G = (G, \phi)$ in cui G è un grafo e $\phi : V(G) \longrightarrow \mathbb{R}$ è una funzione a valori reali definita sui vertici del grafo. Se $x \in \mathbb{R}$ si indica con $G_{\phi \leq x}$ il sottografo indotto da G sul sottoinsieme dei vertici v tali che $\phi(v) \leq x$. Ancora si definisce che due vertici v e w di G sono $\langle \phi \leq y \rangle$-connessi se stanno nella stessa componente connessa del sottografo $G_{\phi \leq y}$. La relazione di $\langle \phi \leq y \rangle$-connessione è naturalmente una relazione di equivalenza. Si indica con ℓ_G la funzione di taglia del grafo di taglia G, e il suo valore in un punto (x, y) è data dal numero di componenti connesse in cui $G_{\phi \leq x}$ è diviso dalla relazione di $\langle \phi \leq y \rangle$-connessione.

Osservazione 1.5. A questo punto si osserva che, con una esatta corrispondenza nella terminologia tra spazi topologici e grafi, per ogni $x < y$ il valore assunto dalla funzione $\ell_G(x, y)$ nel punto (x, y) è pari al numero di componenti connesse del sottografo $G_{\phi \leq y}$ che contengono almeno un vertice u con $\phi(u) \leq x$.

Il risultato espresso nel teorema 1.9 si estende nel caso dei grafi e diventa:

Teorema 1.11. *Sia G un grafo di taglia, allora esiste una successione finita di punti non necessariamente distinti $\{Q_i\}_{1\leq i\leq n}$ dove*

$$Q_i \in \{(x,y) \in \mathbb{R} \times (\mathbb{R} \cup \{\infty\}) \ : x < y\} = \Delta^*$$

per cui vale

$$\ell_{(M,\phi)}(x,y) = \sum_{i=0}^{m} \chi_{T_i}(x,y)$$

su tutto l'insieme Δ dove χ_{T_i} è la funzione caratteristica del tringolo T_i con

$$T_i = T_{Q_i} = \left\{(x,y) \in \mathbb{R}^2 \ : \ x_i \leq x < y < y_i\right\}$$

Anche in questo caso la successione $\{Q_i\}_{1\leq i\leq n}$ è unica a meno di permutazioni.

1.2.2 Calcolo della funzioni di taglia di un grafo

Come è stato più volte detto l'approssimazione di uno spazio topologico M mediante un grafo G è utile perchè il calcolo della funzione di taglia del secondo (più semplice) ci permette di calcolare quella del primo. Quello che ci serve è quindi un metodo per calcolare la funzione di taglia di un grafo. In questo paragrafo ci rifaremo in gran parte a [7].
La tecnica che si utilizzerà consiste nell'operare sul grafo di taglia G delle trasformazioni che hanno lo scopo di semplificarlo lasciandone però inalterata la funzione di taglia. Alla fine di questo procedimento il grafo che si ottiene è tale da permettere un facile computo della sua funzione di taglia, per costruzione uguale a quella del grafo di partenza. Vediamo ora le definizioni di queste trasformazioni.
Nel seguito di questo paragrafo, se $G = (G, \phi)$ indica un grafo di taglia con insieme dei vertici totalmente ordinato mediante indici $V = \{v_1, \ldots, v_n\}$, possiamo supporre un'altra relazione d'ordine totale $>_G$ sui vertici data da: $v_i >_G vj$ se e solo se $\phi(v_i) > \phi(v_j)$ oppure nel caso $\phi(v_i) = \phi(v_j)$ se e solo

se $i > j$. Data questa relazione d'ordine e un grafo non orientato, possiamo sempre dargli un'orientazione supponendo che l'insieme degli archi $E(G)$ diventi

$$\vec{E(G)} = \{e = (v, w) \in E(G) : v >_G w\}$$

D'ora in poi quindi dato un grafo di taglia si suppone che sia data l'orientazione appena descritta, e con $E(G)$ si indicherà l'insieme degli archi dopo che è stata data l'orientazione.

Definizione 1.13. Sia $G = (G, \phi)$ un grafo di taglia. Supponiamo che v_1, v_2, v_3 siano tre vertici distinti di G tali che v_1 e v_3 sono adiacenti a v_2 e che $\phi(v_2) < \phi(v_1) \leq \phi(v_3)$. Si consideri allora il nuovo grafo H dato da:

(1V) $V(H) = V(G)$

(1E) $E(H) = (E(G) - \{e = (v_1, v_2)\}) \cup \{e' = (v_3, v_1)\}$

dove l'arco e' si aggiunge se non è già presente in $E(G)$. Si dice che il grafo di taglia $H = (H, \phi)$ si ottiene da G mediante una Δ^*-*mossa di tipo 1.*

Definizione 1.14. Sia $G = (G, \phi)$ un grafo di taglia. Supponiamo che v_1, v_2 siano due vertici distinti e adiacenti di G con $\phi(v_1) = \phi(v_2)$ e $v_2 >_G v_1$. Allora si consideri il nuovo grafo H dato da:

(2V) $V(H) = V(G) - \{v_1\}$

(2E) $E(H) = E(G) - (\{e = (v_2, v_1)\} \cup E_{v_1}) \cup \tilde{E}_{v_1}$

dove gli insiemi E_{v_1} e $\tilde{E}_{v_1}$ sono dati da

$$E_{v_1} = \{(u, v) \in E(G) : u = v_1, v \neq v_2 \text{ o } v = v_1, u \neq v_2\}$$
$$\tilde{E}_{v_1} = \{(u, v) : u = v_2 \text{ e } (v_1, v) \in E_{v_1} \text{ oppure } v = v_2 \text{ e } (u, v_1) \in E_{v_1}\}$$

e gli elementi di $\tilde{E}_{v_1}$ si aggiungono solo se non sono già presenti in $E(G)$. Si dice che il grafo di taglia $H = (H, \phi)$ si ottiene da G mediante una Δ^*-*mossa di tipo 2.*

Definizione 1.15. Sia $G = (G, \phi)$ un grafo di taglia. Supponiamo che v_1, v_2 siano due vertici distinti e adiacenti di G con $\phi(v_1) > \phi(v_2)$. Si assume che v_2 sia l'unico vertice con questa proprietà, relativamente a v_1. Allora si consideri il nuovo grafo H dato da:

(3V) $V(H) = V(G) - \{v_1\}$

(3E) $E(H) = E(G) - (\{e = (v_2, v_1)\} \cup E_{v_1}) \cup \tilde{E}_{v_1}$

dove gli insiemi E_{v_1} ed $\tilde{E}_{v_1}$ sono definiti come sopra, salvo che in questo caso particolare non possono esserci in E_{v_1} archi della forma (v_1, u). Ancora gli archi di $\tilde{E}_{v_1}$ si aggiungono solo se non sono già presenti in $E(G)$. Si dice che il grafo di taglia $H = (H, \phi)$ si ottiene da G mediante una Δ^*-*mossa di tipo 3*.

Definizione 1.16. Sia G un grafo di taglia. Ogni grafo di taglia che si ottiene applicando a G una successione finita $\{\Delta_1^*, \ldots, \Delta_m^*\}$ di Δ^*-mosse si dice una Δ^*-*riduzione* di G. Una Δ^*-riduzione di G alla quale non è possibile applicare più nessuna Δ^*-mossa si dice Δ^*-*riduzione totale* di G. Un grafo di taglia H si dice *totalmente* Δ^*-*ridotto* se nessuna Δ^*-riduzione è applicabile ad H.

Lo scopo delle Δ^*-riduzioni è quello di semplificare i grafi, infatti provocano una diminuzione del numero di vertici ed archi di un grafo. Inoltre grazie ai risultati che esporremo di seguito si vede che l'operazione di Δ^*-riduzione è un procedimento consistente, anzi utile, al calcolo delle funzioni di taglia.

Proposizione 1.12 (invarianza)**.** *Sia H una Δ^*-riduzione di un grafo di taglia G, allora la funzione di taglia dei due grafi è uguale su tutto $\mathbb{R}^2$.*

Proposizione 1.13 (esistenza)**.** *Per ogni grafo di taglia G una Δ^*-riduzione totale H esiste.*

E quello che più conta la dimostrazione di questo risultato è costruttiva e dà un algoritmo per trovarne una, che verrà descritto nei dettagli nel capitolo terzo. Diamo la seguente definizione preliminare:

Definizione 1.17. Siano G_1 e G_2 due grafi, una funzione $f : V(G_1) \longrightarrow V(G_2)$ si dice *isomorfismo di grafi*, e si scrive $f : G_1 \longleftrightarrow G_2$, se è biunivoca e $(v, w) \in E(G_1) \Leftrightarrow (f(v), f(w)) \in E(G_2)$. Se esiste un isomorfismo $f : G_1 \longrightarrow G_2$, i due grafi si dicono *isomorfi*

Proposizione 1.14 (unicità)**.** *Siano H_1 e H_2 due diverse Δ^*-riduzioni totali di un grafo di taglia G, allora esiste un isomorfismo di grafi $f : H_1 \longrightarrow H_2$ tale che $\phi(v) = \phi(f(v))$ per ogni vertice $v \in V(H_1)$.*

Lemma 1.15. *Sia H un grafo di taglia totalmente Δ^*-ridotto e connesso (o rispettivamente non connesso). Allora si hanno i seguenti risultati:*

- *H è un albero (foresta);*
- *$\exists! v \in V(H)$ tale che $\phi(w) < \phi(v)$ per ogni $w \in V(H) - \{v\}$ (per ogni componente connessa H_α di H, $\exists! v_\alpha \in V(H_\alpha)$ tale che $\phi(w) < \phi(v_\alpha)$ per ogni $w \in V(H_\alpha) - \{v_\alpha\}$);*
- *sia v come nel punto precedente, allora per ogni $w \in V(H)$ connesso con v e da esso diverso esiste ed è unica una successione finita di vertici $\{v_1, \dots v_n\}$ tali che $v_1 = v$, $v_n = w$, per ogni indice $1 \leq i \leq n-1$ l'arco $(v_i, v_{i+1}) \in E(H)$ e $\phi(v_i) > \phi(v_{i+1})$ ovvero $(v_i, v_{i+1}) \in \vec{E}(H)$;*
- *ogni $v \in H$ ha outdegree o nullo o maggiore di 1.*

Questo significa che la Δ^*-riduzione traforma i grafi di taglia in arborescenze (eventualmente in una foresta le cui componenti connesse sono arborescenze) la cui radice è il vertice con ϕ massima e nelle quali non esistono nodi "interlocutori" (cioè o sono foglie o hanno più di un figlio).

Proposizione 1.16 (calcolo)**.** *Sia H un grafo di taglia totalmente Δ^*-ridotto e H_α le sue componenti connesse, allora per ogni $(x, y) \in \Delta$ si ha:*

$$\ell_H(x, y) = \sum_\alpha \ell_{H_\alpha}(x, y)$$

Si deve osservare che in realtà questo risultato vale per un grafo qualsiasi.
Se poi H è un grafo di taglia connesso e totalmente Δ^*-ridotto (quindi un

albero) e v è la foglia di H con valore massimo della ϕ, si indichi con H' il sottografo indotto su H da $V(H) - \{v\}$ e con $H' = (H', \phi')$ il conseguente grafo di taglia dove $\phi' = \phi|_{V(H')}$. Allora si ha:

i) se v è anche la radice di H allora

$$\ell_H(x, y) = \chi_{T^{\infty}_{\phi(v)}}(x, y)$$

ii) se v non è una radice e quindi esiste un ben determinato $w \in V(H)$ tale che $(w, v) \in \vec{E(G)}$ allora

$$\ell_H(x, y) = \chi_{T^{\phi(w)}_{\phi(v)}}(x, y) + \ell_{H'}(x, y)$$

dove $\chi_A(x, y)$ è la funzione caratteristica dell'insieme A e T^b_a è la regione triangolare

$$T^b_a = \{(x, y) \,:\, a \leq x < y < b\}$$

Ovvero, ed è questo il risultato che volevamo esporre in questo capitolo, la catena di proposizioni ci permette di trovare un algoritmo per calcolare la funzione di taglia di un grafo:

1. si calcola H, una Δ^*-riduzione totale del grafo di taglia G (*esistenza*)
2. la funzione di taglia della Δ^*-riduzione totale H è la stessa di quella del grafo di taglia G quale che sia la riduzione (*invarianza e unicità*)
3. si utilizza l'algoritmo dell'ultima proposizione per calcolare la funzione di taglia della riduzione H e si ha così quella di G (*calcolo*)

Capitolo 2

Grafica vettoriale, SVG e Inkscape

2.1 La grafica vettoriale

La *computer graphics* è quella parte dell'informatica che si occupa della produzione delle immagini digitali. Un'immagine digitale è una rappresentazione di un'immagine reale o virtuale effettuata con i mezzi e i modi possibili del mondo digitale. Al momento la computer graphics utilizza essenzialmente due tecniche per rappresentare un'immagine: la grafica *raster* e quella *vettoriale*.
La prima rappresenta una immagine mediante una sua approssimazione discreta: sia nel dominio spaziale (con una operazione detta *campionatura* che produce una matrice di pixel) sia in quello dell'intensità della luce (operazione di *quantizzazione* del colore che si realizza nel numero finito di possibili sfumature cromatiche di ogni pixel).
Al contrario la grafica vettoriale descrive l'immagine da rappresentare mediante delle primitive matematiche (per esempio punti, linee, curve), cioè mediante delle vere e proprie equazioni, alle quali sono attribuiti colori e sfumature. Lo spazio e il colore dell'immagine rappresentata sono dunque descritti in modo continuo, in esatta opposizione alla discretizzazione della

grafica raster.
Le due tipologie di grafica digitale hanno ciascuna i loro pregi e difetti, quindi, a seconda delle esigenze, i vari settori di produzione ed elaborazione di immagini prediligono l'utilizzo dell'una o dell'altra. Il pregio maggiore della grafica vettoriale consiste nel fatto che è *scalabile*, ovvero indipendente dalla risoluzione. Essendo l'immagine descritta mediante primitive matematiche, discretizzate solo al momento della visualizzazione sullo schermo, nessuna elaborazione dell'immagine la fa perdere in qualità[1]. In secondo luogo le immagini vettoriali essendo rappresentate mediante primitive matematiche, e quindi secondo una qualche logica, permettono di organizzare le informazioni contenute nell'immagine stessa in modo più compatto, occupando così meno spazio fisico in memoria rispetto ad un'immagine raster. I pregi delle immagini di tipo vettoriale si possono quindi riassumere nel seguente modo:

- si prestano ad essere rappresentate in diverse risoluzioni

- occupano poca memoria

D'altra parte il problema principale della grafica vettoriale consiste nel fatto che non è adatta a rappresentare immagini molto elaborate, come fotografie digitali. Per queste invece l'ideale è utilizzare tecnologie raster con alta risoluzione che permettono una resa dell'immagine di qualità davvero elevata. Infine, avendo a che fare con primitive matematiche non sempre semplici (polinomi di Bézier, curve Spline, ecc.), un altro difetto delle tecnologie vettoriali consiste in una non immediata *usability* per quanto riguarda produzione ed elaborazione di immagini da parte di utenti non esperti.
In conclusione possiamo affermare che la grafica vettoriale è l'deale per immagini che siamo visivamente semplici (composte da pochi elementi grafici) e

[1]Tipicamente la qualità di un'immagine dipende dalla sua risoluzione che è rappresentabile con il numero di pixel per ogni pollice (ppi) dell'immagine stessa. Uno zoom che raddoppia l'immagine le fa perdere il quadruplo di risoluzione (perchè si ragiona su quadrati). In pratica la perdita di qualità produce un'immagine cosiddetta *pixellata*.

che abbiamo la necessità-possibilità di essere visualizzate in varie risoluzioni o dimensioni (come le immagini di un sito internet la cui risoluzione può dipendere dal browser, dallo schermo dell'utente, ecc.).

Negli ultimi anni è stato sviluppato all'interno dell'ARCES di Bologna un software in grado di calcolare le funzioni di taglia di immagini raster, allo scopo di testare con esperimenti reali l'effettiva utilizzabilità della teoria della taglia come ''tool'' per riconoscere le forme. Citiamo per esempio [5] in cui si è utilizzato il software per paragonare Marchi di Fabbrica (ma altri esperimenti si sono effettuati con Targhe arabe, Firme, Melanomi, ecc.). L'obiettivo di questa tesi è sviluppare e descrivere un software analogo, ma che operi su immagini di tipo vettoriale. Questa scelta è motivata sia da un'esigenza naturale di testare le potenzialità della teoria della taglia nel più ampio ambito possibile; sia dalla considerazione che la maggior parte degli esperimenti pratici effettuati con il software per immagini raster è stata fatta su data-base di immagini con caratteristiche tali da risultare più adatte ad essere rappresentate con immagini vettoriali (Marchi di Fabbrica, Targhe arabe). Inoltre uno dei possibili sviluppi della teoria della taglia applicata ad immagini vettoriali è quello del riconoscimento delle cosiddette *keypics*, campo di sviluppo ampio ed interessante descritto in [3].

2.2 Lo standard SVG

Nell'attuale panorama informatico si ritrova una vasta disponibilità di software di produzione e elaborazione di immagini vettoriali e di conseguenza anche un'ampia varietà di formati e standard. In vista di un utilizzo il più ampio possibile del software, ma soprattutto della teoria della taglia nell'ambito della grafica vettoriale, si è deciso di utilizzare lo standard SVG che verrà brevemente presentato in questo parte.
Scalable Vector Graphics (abbreviato in SVG), è una tecnologia open source

sviluppata per essere in grado di visualizzare oggetti di grafica vettoriale. In particolare si tratta formato immagine vettoriale (*.svg*) che si pone l'obiettivo di descrivere, soprattutto in ambito web ma non solo, figure bidimensionali statiche e animate e che è realizzato mediante un linguaggio di alto livello derivato dall'XML. SVG è anche diventato una raccomandazione (standard) del World Wide Web Consortium nel settembre 2001 e, grazie alla sua licenza libera e alla sua predispozione ad essere accessibile da qualsiasi tipo di piattaforma (Linux, Mac e Windows), ha ormai raggiunto una vasta diffusione. In quanto derivazione del linguaggio XML, un file SVG è anche un testo che in quanto tale può essere visualizzato e modificato da un qualsiasi editor testuale. Queste ultime due caratteristiche (la vasta diffusione e la facile accessibilità alle informazioni contenute nelle immagini) sono state alla base della scelta di SVG come formato standard per il nostro software.

Vediamo ora le sue caratteristiche principali. SVG permette di utilizzare tre tipi di oggetti grafici:

- forme vettoriali

- oggetti grafici di tipo raster

- testo

Questi oggetti grafici possono essere raggruppati, soggetti a trasformazioni, dotati di uno stile, inclusi in oggetti già sottoposti al rendering (cioè alle elaborazioni necessarie alla visualizzazione su schermo). Inoltre il formato permette una vasta gamma di operazioni per elaborare le immagini, quali clipping, trasformazioni annidate, effetti di filtro, ecc.
Nei file SVG le informazioni da disegnare sono espresse tramite oggetti. Un oggetto è una unità grafica dell'immagine costituita da uno o più elementi: primitive grafiche ed eventualmente da altri sotto-oggetti annidati uno dentro l'altro. La struttura logica e grafica dell'immagine viene così stabilita dall'albero di oggetti da cui è descritta. Questa caratteristica di modularità del

formato SVG permette all'utente la più ampia libertà di utilizzi e rappresentazioni. Ciò su cui ci soffermeremo, perciò, non è la vasta varietà di tecniche che SVG permette, bensì gli oggetti di base da cui si parte nella costruzione delle immagini: le primitive grafiche. In particolare tralasceremo i metodi di inserimento di immagini raster e la gestione del testo, per concentrarci sulle forme vettoriali che saranno il vero oggetto su cui testare il nostro software.

2.2.1 Le primitive grafiche

Come è stato spiegato, le primitive grafiche da cui sono costituite le immagini in formato SVG sono descritte tramite delle equazioni. L'immagine, da un punto di vista matematico, è dunque una porzione rettangolare del piano su cui è assegnato un sistema di coordinate cartesiane. È la definizione di un sistema di riferimento globale che dà alle forme geometriche, definite in assoluto e separatamente l'una dall'altra, una posizione relativa, in quanto le "immerge" nello stesso spazio. Va notato che, come è convenzione in molti software grafici, il sistema di riferimento è definito con l'origine in alto a sinistra nella finestra dell'immagine (cioé lo spazio che essa occupa) ed ha l'asse verticale rivolto verso il basso, cioè opposto al comune modo di rappresentare gli assi cartesiani.
Le primitive grafiche vettoriale che SVG mette a disposizione sono di due tipi:

1. forme

2. cammini

Le forme sono varie tipi di enti geometrici del piano ciascuno descritto tramite una diversa primitiva grafica e la scelta di alcuni parametri detti *proprietà* o *attributi*. Le proprietà sono sia di tipo geometrico, ovvero quelle che definiscono intrinsecamente le caratteristiche spaziali dellla forma (come il raggio per un cerchio), sia di tipo grafico (per esempio il colore, lo spessore delle

linee, l'eventuale riempimento delle forme chiuse, ecc.). Le forme che SVG mette a disposizione sono:

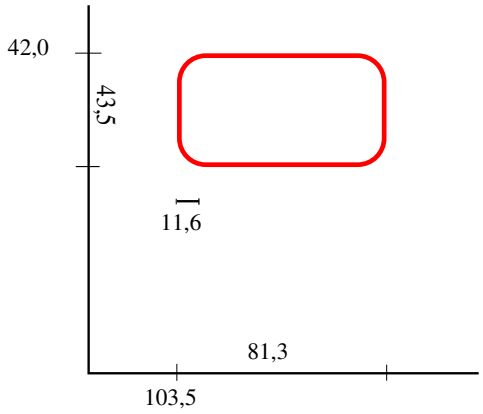
- rettangoli
- cerchi
- ellissi
- segmenti
- linee spezzate poligonali
- poligoni

I cammini, invece, sono dati da successioni di curve del piano di vario tipo (segmenti, curve parametriche di Bezier, archi di cerchio ed ellisse) che si concatenano una di seguito all'altra, cioè la successiva inizia esattamente nel punto in cui finisce la precedente. Anche le curve, come le forme, sono descritte tramite la specificazione di proprietà sia geometriche che grafiche.
Ci soffermiamo ora solo brevemente sugli attributi grafici delle primitive, in quanto sono una parte importante dello standard SVG (anche per capire meglio alcuni esempi che verranno fatti in seguito), ma non sono nell'interesse principale della nostra analisi. Ciò che ci interessa è, infatti, soprattutto il modo di rappresentare le primitive geometriche in modo da sapere come considerarle matematicamente quando dovremo calcolare le funzioni di taglia. I principali descrittori delle proprietà grafiche degli oggetti che vengono disegnati sono i parametri `fill` e `stroke` che indicano l'utilizzo del colore nella visualizzazione delle forme. Il parametro `fill` serve per gestire il riempimento delle forme chiuse con un colore a scelta (omogeneo o eventualmente con vari tipi di gradienti cromatici). Al contrario l'attributo `stroke` serve per gestire il modo di rappresentare le linee, nel caso di cammini o del bordo di forme chiuse. Indica per esempio lo spessore delle linee, il loro stile (tratteggiato, continuo, puntinato, ecc.), il modo di rappresentare i vertici iniziali e finali delle curve e molto altro.

Diamo ora la descrizione delle forme. Per ciascuna di esse verranno indicati e spiegati gli attributi che la descrivono (indicati, come anche in precedenza i due parametri `fill` e `stroke`, in `courier`), poi a titolo di esempio si mostreranno uno o più estratti di file in formato SVG che contengono la descrizione della forma (anche essi in `courier`), con a fianco la loro visualizzazione, cioè il risultato finale dell'immagine[2].

Rettangoli. I parametri che descrivono un rettangolo sono le coordinate del vertice in alto a sinistra (`x`, `y`), la larghezza (`width`) e l'altezza (`height`). Ci sono poi due parametri che indicano l'eventuale smussamento dei vertici per mezzo di archi di ellisse (`rx`, `ry` che rappresentano la lunghezza dei semiassi).

```
<rect
  id="rect1"
   width="81.3"
   height="43.5"
   rx="11.6"
   ry="11.6"
   x="103.5"
   y="42.0"
   style="fill:none;
   stroke:red;stroke-width:2;"
   />
```



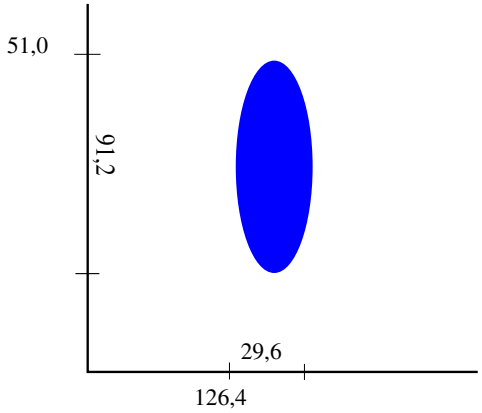
Esempio di rettangolo smussato.

Ellissi. I parametri che decrivono un'ellisse sono le coordinate del suo centro (`cx` e `cy`) e la lunghezza del semiasse orizzontale e di quello verticale (`rx` e `ry`). Ovviamente nel caso in cui i due semiassi abbiano lunghezza

[2]La parte di immagine rappresentata dalle informazioni contenute nell'estratto dei file sarà colorata, mentre in nero saranno delle parti aggiuntive inserite per rendere più comprensibile il rapporto tra testo e immagine, come per esempio il sistema di riferimento.

uguale il risultato grafico è un cerchio, ma memorizzato nella struttura di un'ellisse.

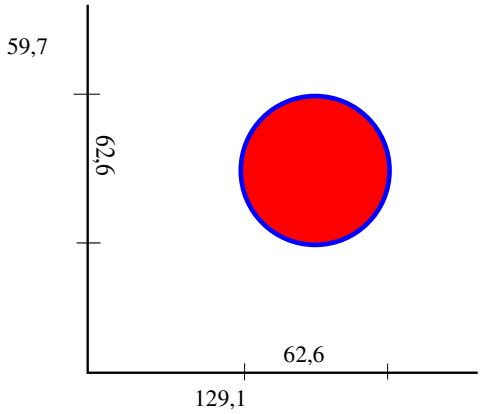
```
<ellipse
   cx="141.2"
   cy="96.6"
   rx="14.8"
   ry="45.6"
   style="fill:blue;
   stroke:none;" />
```



Esempio di ellisse.

Cerchi. I parametri che descrivono un cerchio sono le coordinate del suo centro (`cx` e `cy`) e la lunghezza del raggio (`r`).

```
<circle
   cx="160.4"
   cy="91.0"
   r="31.3"
   style="fill:red;
   stroke:blue;
   stroke-width=2;" />
```

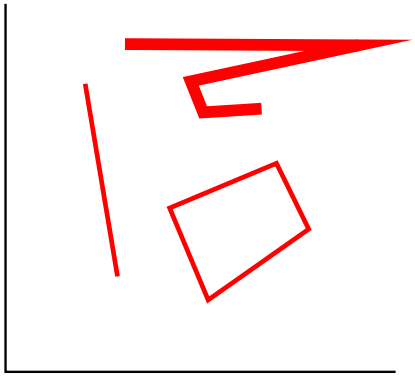


Esempio di cerchio con fill e stroke.

Segmenti, Linee Spezzate, Poligoni. I parametri che descrivono un segmento sono le coordinate dei suoi due estremi (`x1`, `y1` e `x2`, `y2`). Le linee spezzate sono descritte tramite le coordinate della successione di punti che

vengono uniti da segmenti, si ha dunque il parametro `points` costituito da una successione di coppie di valori (le coordinate dei punti). Se la spezzata è costituita da soli due punti, allora risulta un segmento, ma memorizzato come una spezzata. Esattamente allo stesso modo vengono descritti i poligoni, solo che in fase di visualizzazione viene disegnato anche il segmento che unisce l'ultimo punto al primo.

```
<g style="stroke:red;
   stroke-width=2;" >
  <line
    x1="51.0" y1="63.4"
    x2="59.7" y2="189.1" />
  <polyline
    points 61.3,64.0
      201.5,65.2 102.7,95.0
      112.0,122.1 138.4,124.0
      style="stroke-width=4;" />
  <polygon
    points 100.9,141.3
      141.0,138.1 115.4,199.0 />
</g>
```



Esempio di segmento, spezzata e poligono.

Passiamo ora ad esaminare i cammini. Il tag che serve a disegnare i cammini è `<path>`. Al suo interno si ha la descrizione geometrica del cammino che è contenuta interamente nel parametro `d`. Tale parametro ha come possibili valori una successione di *comandi* che permettono di descrivere l'aspetto geometrico del cammino: posizione nello spazio, elementi costituitivi, eventuale chiusura del cammino stesso. Ciascun comando a sua volta necessita di alcuni parametri per specificarlo. Diamo ora una presentazione dei possibili comandi, la loro espressione nei file SVG, i parametri che li completano e il loro significato grafico, cioé l'informazione in essi contenuta.
La prima informazione necessaria è il punto di partenza del cammino, che permette di posizionarlo all'interno dell'immagine. I comandi che consentono

di scegliere tale punto si dicono comandi *moveto*, sono `M` e `m` e hanno l'effetto di spostare un ideale cursore in un particolare punto dello spazio occupato dall'immagine, senza disegnare nulla. A tal proposito si deve fare una considerazione che sarà utile anche in seguito. I file in formato SVG sono fatti, come si è visto, con una struttura ad albero, la cui visita (per esempio al momento della visualizzazione) viene fatta in modo sequenziale, rispetto all'elenco strutturato di elementi grafici descritti nel file. Per questo motivo SVG supporta una serie di *comandi assoluti* espressi con lettere maiuscole (come `M`) a cui è associato un corrispettivo *comando relativo*, che si indica con lettera minuscola (come `m`). Vediamo un esempio per spiegare cosa si intende con assoluto e relativo: se con `M` $100.0, 250.0$ si sposta il cursore nel punto di coordinate $x = 100.0$, $y = 250.0$ in qualsiasi "momento" del file, usando successivamente il comando `m` $10.0, 30.0$ il cursore si sposta nel punto di coordinate $x = 100.0 + 10.0 = 110.0$, $y = 250.0 + 30.0 = 280.0$. Dunque mentre i parametri dei comandi assoluti sono coordinate, quelli dei comandi relativi rappresentano un vettore, e quindi possono assumere nelle due componenti anche valori negativi[3]. Come si vedrà questo doppia categoria di comandi è data per tutti quelli destinati a descrivere i cammini. Per semplificare si descriverà in seguito solo il significato dei comandi assoluti, in quanto quello dei relativi si ricava abbastanza facilmente.

Una volta scelto il punto di partenza con un comando moveto, ogni comando che disegna una parte del cammino, ha anche l'effetto di spostare il cursore nell'ultimo punto della porzione di curva data da quel comando. Dunque ogni comando che disegna una curva sottindende come primo suo punto, l'ultimo del comando precedente. È proprio questo che comporta l'accodamento una all'altra delle singole curve e quindi e la costruzione del cammino.

[3]Se un comando di tipo relativo non ha nessun comando assoluto che lo precede, viene interpretato come un comando assoluto o, equivalentemente, come se il cursore fosse posto nel punto di coordinate $(0, 0)$.

Se i comandi moveto servono per iniziare i cammini e si trovano sempre come primo elemento del parametro `d`[4], all'ultimo posto si può trovare un comando *closepath* dato dalla presenza o meno di `Z` o `z`. In questo caso particolare il comando assoluto e quello relativo hanno lo stesso effetto cioé disegnano un segmento dal punto corrente (l'ultimo del cammino) al primo del cammino stesso, generando così un cammino chiuso. Anche il cursore viene spostato in corrispondenza del punto di partenza del cammino. Si osservi che l'effetto grafico di un cammino chiuso (con il comando "closepath") è differente da quello di uno aperto (senza comando "closepath"), anche se il primo e l'ultimo punto coincidono. Infatti nel primo caso si considera il cammino senza punti estremi, mentre nel secondo caso questi ultimi sono presenti (e coincidenti) e possono, grazie ai parametri grafici, essere evidenziati.
Dopo la scelta del punto origine del cammino e prima di stabilire se il cammino è chiuso o meno, si ha la successione dei comandi che indicano il tipo di curva scelta (tra quelli elencati in precedenza): vediamo ora brevemente quali sono e che informazioni contengono.

Lineto. I comandi *lineto* si esprimono tramite i comandi `L`, `V` o `H` (o i relativi `l`, `v`, `h`) e hanno lo scopo di disegnare dei tratti rettilinei, cioè dei segmenti. Il comando `L` ha come parametri una coppia di coordinate (x, y) e disegna un segmento dal punto in cui è il cursore al punto di coordinate (x, y). I comandi `V` e `H` (*vertical lineto* e *horizontal lineto*) sono invece seguiti da un solo parametro, un valore numerico (r), e si utilizzano per disegnare segmenti rispettivamente verticali ed orizzontali (per cui serve solo la coordinata - rispettivamente - sull'asse verticale o orizzontale, data dal numero r. Vediamo ora un esempio in cui si chiarisce anche il concetto di accodamento delle curve. Data una successione di comandi di tipo `L` x_i, y_i per esempio:

```
<path
    style="stroke:blue;stroke-width=2;"
```

[4]Se non compare un comando moveto, viene considerato come punto iniziale la corrente posizione del cursore, che se non è precisata corrisponde all'origine delle coordinate.

```
d="M 100.0,100.0
      L 150.0,100.0 L 150.0,50.0
      L 100.0,50.0 L 50.0,50.0
      L 50.0,100.0 L 50.0,150.0
      L 100.0,150.0 L 150.0,150.0 />
```

essa produce, confrontando l'immagine, una sorta di spirale retta (tratto continuo) e non la stella di segmenti uscenti dal punto 100.0, 100.0 (tratteggiata).

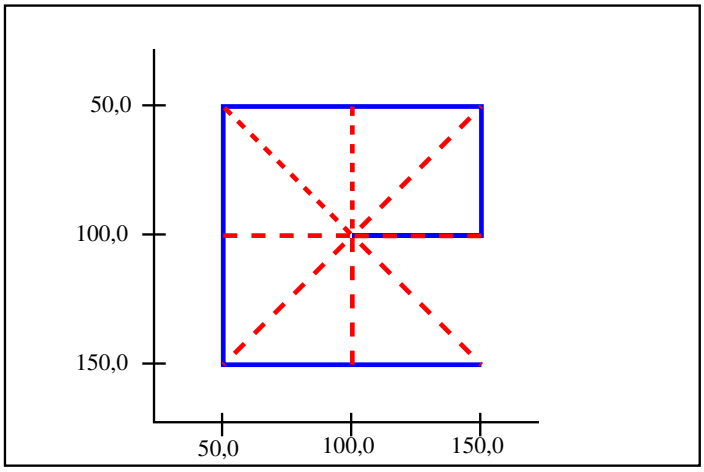


Figura 2.1: Incodamento di curve nel caso di segmenti.

Lo stesso risultato si sarebbe ottenuto con

```
<path
    style="stroke:blue;stroke-width=2;"
    d="M 100.0,100.0 H 150.0 V 50.0
                     H 50.0 V 150.0 H 150.0 />
```

oppure ancora con

```
<path
    style="stroke:blue;stroke-width=2;"
    d="M 100.0,100.0 h 50.0 v -50.0
                     h -100.0 v 100.0 h 100.0 />
```

I successivi tipi di comando generano invece delle effettive curve, cioè cammini non rettilinei, sono chiamati comandi *curveto* e sono di tre tipi: cubiche di Bézier, quadratiche di Bézier, archi di ellisse. Ricordiamo brevemente che una curva parametrica piana di Bézier di grado n si descrive per mezzo di $n+1$ punti $P_0 = (x_0, y_0), \ldots, P_n = (x_n, y_n)$ detti punti di controllo ed è data dall'equazione vettoriale

$$P(t) = \sum_{i=0}^{n} \binom{n}{i} t^i (1-t)^{n-i} P_i = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = \sum_{i=0}^{n} \binom{n}{i} t^i (1-t)^{n-i} \begin{pmatrix} x_i \\ y_i \end{pmatrix}$$

con t che varia nell'intervallo $[0, 1]$ per cui valgono P_0 se $t = 0$, P_n se $t = 1$, ovvero sono particolari curve da P_0 a P_n. Le quadratiche e cubiche di Bézier sono in modo naturale curve di Bézier di grado rispettivamente 2 e 3. Per una descrizione approfondita delle curve di Bézier rimandiamo a [16].

Cubiche. I comandi che permettono di disegnare delle cubiche di Bézier sono `C` e `S` (e i relativi `c` e `s`). Il primo (`C`) deve essere seguito da tre coppie di coordinate (`C` $x_1, y_1\ x_2, y_2\ x, y$) e disegna una cubica parametrica di Bézier dal punto in cui è attualmente il cursore al punto P di coordinate (x, y) utilizzando come punti P_1 e P_2 quelli di coordinate (x_1, y_1) e (x_2, y_2). Il comando S serve per costruire delle successioni di curve cubiche di Bézier che nel loro insieme risultino di classe C^1. Per ottenere questo effetto si deve imporre (supponendo che P_i siano i punti di controllo della curva precedente e P'_i quelli della curva successiva):

- $P_3 = P'_0$
- $\frac{1}{2}(P_2 + P'_1) = P_3 = P'_0$ ovvero il penultimo punto di controllo delle curva precedente P_2 e il secondo della curva successiva P'_1 sono simmetrici rispetto a $P_3 = P'_0$

Dunque il comando `S` è seguito da due sole coppie di valori `S` $x_2, y_2\ x, y$ che rappresentano le coordinate dei punti P'_2 e P'_3, in quanto quelle di P'_1 si calcolano grazie ai punti della curva precedente. Se il comando `S` viene usato senza che prima sia stato utilizzato `C` allora si intende che $P'_1 = P'_0$.

Quadratiche. In modo analogo a quanto appena detto, le curve di Bézier quadratiche vengono descritte con i comandi `Q` e `T` (o con i comandi relativi `q` e `t`). Il comando `Q` deve essere seguito da due coppie di valori (`Q` $x_1, y_2\ x, y$) che rappresentano le coordinate di P_1 e P_2 dove P_0 è dato dalla posizione corrente del cursore, e disegna la curva piana parametrica di Bézier di grado 2. Il comando `T` serve per creare successioni di curve che risultino C^1 cioé, con una notazione simile al caso cubico, se $P_2 = P_0'$ e $(P_1 + P_1')/2 = P_2 = P_0'$. Dunque necessita di una sola coppia di valori (`T` x, y) che indicano le coordinate del punto P_2' dove quelle di P_0' e P_1' si ricavano da quelle della curva precedente. Ancora in analogia se `T` viene utilizzato non in seguito ad un comando `Q` allora si suppone $P_1' = P_0'$.

Archi di ellisse. Il comando `A` (o il relativo `a`) pemette di costruire una curva che unisce due punti e che corrisponde ad una porzione di ellisse con assi maggiore e minore paralleli agli assi coordinati, cioé un arco di ellisse. Si osserva preliminarmente che dati due punti $P_0 = (x_0, y_0)$ e $P = (x, y)$ ci sono due diverse ellissi che passano per quei due punti e quindi quattro diversi archi ellittici che li uniscono, per cui il comando `A` è seguito da una coppia di valori `x,y` che danno il punto di arrivo (dove quello di partenza è dato dalla posizione corrente del cursore) e da due altri parametri binari che permettono di indentificare quale dei quattro possibili archi si è scelto[5].

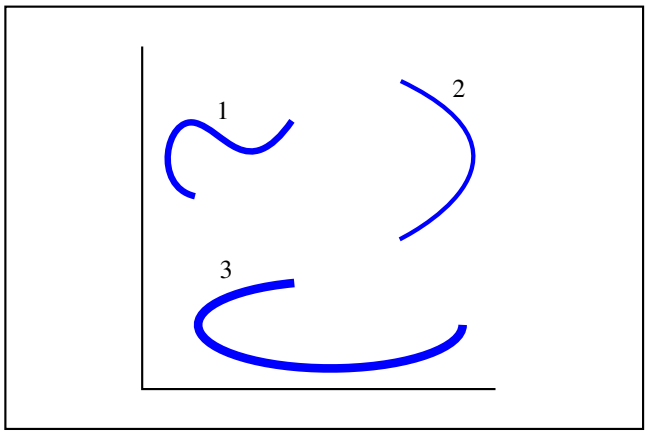


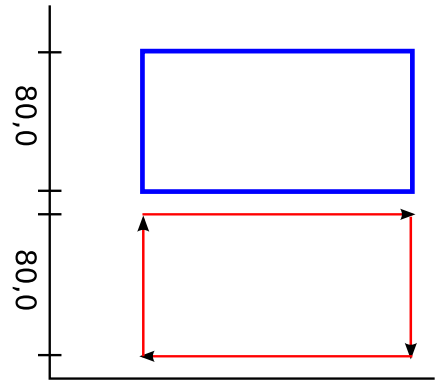
Figura 2.2: Esempi di cammini di diverso tipo.

Nella figura precedente ci sono tre esempi di curve: un cammino costituito da due cubiche di Bézier rese tangenti con il comando `S` (curva 1), una quadratica di Bézier (curva 2) ed un arco di ellisse (curva 3).

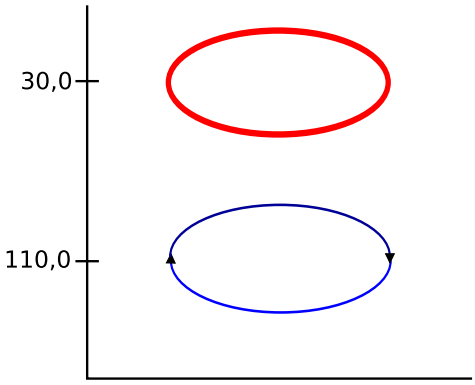
Come ultima considerazione sulle curve, si osserva che tutti i tipi di forme che sono state presentate in precedenza possono essere riprodotte con un opportuno cammino costituito dalle curve che abbiamo appena descritto. Per esempio un rettangolo con vertice in $100.0, 100.0$ e con altezza e larghezza rispettivamente $30, 150$ si può rappresentare con un cammino

```
<path
    style="stroke:blue;stroke-width=2;"
    d="M 100.0,100.0 h 150.0 v 30.0
                     h -150.0 v -30.0 Z />
```

O ancora una ellisse si può rappresentare come due archi di ellisse, che uniscono gli estremi dell'asse maggiore P_1 e P_2, uno simmetrico all'altro rispetto all'asse maggiore stesso.



Rappresentazione di un rettangolo con una poligonale.



Rappresentazione di un'ellisse tramite due archi ellittici.

[5] Il fatto che ci siano esattamente quattro archi di ellisse per due punti e il modo in cui tali archi vengono identificati dai due parametri binari è un dettaglio tecnico che non approfondiamo, anche perchè allo stato attuale del software da noi sviluppato non si sono prese in considerazione curve con archi di ellisse.

Questa considerazione è molto interessante perché permette di ridurre tutte le primitive grafiche disponibili in SVG ad una unica primitiva geometrica: i cammini; e si può andare anche oltre tenendo conto che sia le curve quadratiche e cubiche di Bézier, sia ovviamente i segmenti, sia gli archi ellittici, possono essere ricondotti anch'essi ad un'unica struttura matematica, le curve B-Spline. Avere un unico oggetto matematico con cui rappresentare tutte le primitive grafiche è dal punto di vista dell'approccio al "calcolo su SVG" (cioè qualsiasi tipo di analisi quantitativa sulle immagini in formato SVG) molto interessante, perchè permette di studiare un unico metodo con cui effettuare tale calcolo, e non doverlo differenziare a seconda delle diverse primitive[6]. Tale è, a mio parere, la direzione in cui dovrebbe volgersi un eventuale ulteriore sviluppo del programma di calcolo delle funzioni di taglia su immagini SVG, visto che al momento, per semplificare l'analisi del problema, è stato implementato in modo da gestire come unica forma di primitive i rettangoli.

2.3 Inkscape

Come è stato detto all'interno del precedente paragrafo, la scelta di SVG come formato delle immagini vettoriali è dovuta da un lato alla vasta diffusione di tale formato oramai utilizzabile su tutte le piattaforme, dall'altro (ed è su questo aspetto che ci vogliamo ora soffermare) perchè consente una notevole facilità di accesso alle informazioni contenute nell'immagine. Questo si ha perchè un file in formato SVG è in realtà un particolare file scritto in XML, dunque leggibile da qualsiasi browser o editor testuale come un vero e proprio file di testo. È possibile dunque sia visualizzare l'immagine con opportuni software, sia leggere il file come se fosse la descrizione di ciò che l'immagine contiene. E il termine *descrizione* non è casuale perchè, se è vero che anche un file in formato BMP (bitmap) può essere "letto" da un qualsiasi

[6]Inoltre in un lavoro di B. Di Fabio ([9]) si può trovare già una interessante trattazione teorica su particolari funzioni di taglia di curve Spline.

editor di testo che supporti il formato ASCII, la differenza sta nel modo in cui i due diversi formati presentano le informazioni contenute nell'immagine. Una bitmap, specie se di una immagine non in bianco e nero, è praticamente illeggibile: una matrice di numeri da cui sicuramente non si riesce a distinguere le forme dell'immagine che rappresenta. Al contrario, con una minima conoscenza della sintassi SVG, un file in tale formato è perfettamente leggibile e permette di avere una idea delle componenti grafiche dell'immagine senza la necessità di visualizzarla. Il formato-testo: è questa una delle caratteristiche fondanti di SVG e, a nostro parere, uno dei suoi principali pregi. Ma inevitabilmente, oltre che pregi, dal suo essere un testo derivano anche alcuni difetti. Infatti la presentazione dell'immagine come testo, pur seguendo ovviamente una determinata sintassi, permette comunque ai file una notevole libertà di "stile", libertà che viene ampliata dalla duttilità propria dell'XML. Si intende con libertà il fatto che il formato SVG, pur essendo uno standard, non stabilisce per una gran parte delle sue funzionalità una forma standard per esprimerle. Per fare un paragone, esattamente come in tutte le lingue, per esprimere un concetto si possono usare un numero grande a piacere di modi diversi, magari utilizzando perifrasi complicate, purchè si rispetti la grammatica e la sintassi. Allo stesso modo (come anche abbiamo già evidenziato nel caso delle forme rappresentabili come cammini), in un file SVG si può esprimere lo stesso concetto grafico in modi anche molto differenti l'uno dall'altro. Tale caratteristica, se da un lato può sembrare un pregio, è risultata dal nostro punto di vista un notevole difetto, in quanto non permette un approccio standard quando si ha a che fare con una immagine. Con questo si intende che ogni diversa piattaforma di disegno, ogni utente, può presentare come prodotto finale un file con uno "stile" anche molto diverso: quindi pensare un software che calcoli la funzione di taglia di una qualsiasi immagine SVG è un'impresa davvero ardua. Il primo ostacolo è in particolare dovuto al programma con cui si è prodotta l'immagine, ma è arginabile scegliendo un particolare software e utilizzando solo immagini create con quello. Il secondo è invece dovuto all'arbitrarietà con cui un utente può de-

cidere di fare il "disegno", ma questo fattore, legato alla creatività umana, non è in alcun modo affrontabile, almeno a questo livello di analisi, e quindi non ce ne siamo occupati. In pratica ci siamo comportati come se ci fosse solo un numero finito di modi possili per esprimere le informazioni grafiche. La scelta di questi "modi" è stata fatta in modo da risolvere contemporaneamente anche il primo problema: si è scelto un particolare software di disegno vettoriale che utilizzasse SVG e si sono considerati solo gli "stili" da esso utilizzati. In particolare è stato scelto il software Inkscape, che presentiamo ora brevemente.

Inkscape è un software libero per il disegno vettoriale, utilizzabile su ogni tipo di piattaforma (Linux, Mac e Windows), e progettato come strumento grafico non professionale che supporti lo standard SVG. Nasce nel 2003 come "fork" di Sodipodi, un altro software di grafica vettoriale, proprio perchè i suoi sviluppatori avevano obiettivi diversi, cioè la completa implementazione dello standard SVG, mentre nel progetto Sodipodi l'intento era creare un software generico per la grafica vettoriale, anche a scapito di SVG.
Una delle priorità di Inkscapa è l'*usability* ovvero un interfaccia che permetta all'utente un utilizzo del software il più facilitato possibile. In particolare quindi l'interfaccia non presenta un numero molto elevato di finestre di dialogo, il maggior numero di funzioni possono essere eseguite anche da tastiera, e ci sono a disposizione dei brevi ma utili tutorial che permettono l'apprendimento di tutte le potenzialità basilari del software. Questo ha comportato il fatto che Inkscape non risulta un sofware di progettazione e modellazione bidimensionale di alto livello, al contrario è uno strumento non professionale, ideale per un utilizzo semplificato, cioé per la creazione di immagini non troppo elaborate (adatto quindi ai nostri scopi). Per meglio comprendere il livello di elaborazione grafica raggiungibile con Inkscape si consiglia di scorrere alcune delle immagini contenute nella Open Clip Art Library[7], progetto gestito dagli sviluppatori di Sodipodi e Inkscape per la creazione e messa a disposizione su web di una vasta collezione di disegni vettoriali in formato

[7] www.openclipart.org

SVG (clip art) scaricabili e utilizzabili da tutti (open source), che sono stati disegnati per la maggior parte con Inkscape.
Come detto, il progetto Inkscape prevede la completa implementazione dello standard SVG, ma al momento tale obiettivo non è stato completamente raggiunto: alcuni tipi di effetti di filtro, animazioni e caratteri infatti non sono ancora supportati. D'altra parte però, tralasciando queste eccezioni, Inkscape supporta praticamente tutte le potenzialità di SVG (per la creazione e manipolazione di oggetti, la selezione degli stili, la gestione dei cammini e del testo, ecc.) Mette inoltre a disposizione dell'utente anche alcune possibilità in più: per esempio in parallelo al tool per disegnare i cammini prevede una funzione *pencil* per tracciare curve a mano libera (che vengono gestite con curve di Bézier che le approssimano); oppure, oltre alle forme geometriche che abbiamo presentato in precedenza, vi sono anche, come primitive grafiche supportate, i poligoni regolari (o le stelle regolari) e le spirali. A questo punto non riteniamo significativo fare un elenco dettagliato delle funzione e delle possibilità di Inkscape, perchè si tratterebbe in gran parte di ripetere la parte descrittiva su SVG; né tanto meno ci sembra utile dare delle indicazioni sull'utilizzo del software, che si possono facilmente trovare in vari tutorial che il programma stesso mette a disposizione. Quello che è interessante, soprattutto per i nostri scopi, è analizzare come l'utilizzo di Inkscape incide nello "stile" dei file immagine: ovvero non come sono le immagini, bensì come è fatto il testo che le descrive.

La principale considerazione da fare è che Inkscape offre due diversi *sottoformati* per il salvataggio delle immagini. Con sottoformati si intendono due diversi "stili", che però il software ha al suo interno standardizzato: essi sono *inkscape-svg* e *plain-svg*. Questa libertà di scelta è possibile proprio per la libertà che SVG lascia alla formattazione testuale dei suoi file. La differenza tra i due "stili" sta nella diversa ampiezza di possibilità offerte. In pratica lo stile plain-svg è più elementare, supporta solo alcune funzioni e alcuni tipi di primitive e, in caso di salvataggio con questo stile di immagini non supportate, il software avvisa che c'e' la possibilità di perdita di informazione. Al

contrario lo stile inkscape-svg è quello naturale per il salvataggio e supporta tutte le opzioni del programma. Scendendo più nel dettaglio le differenze tra i due stili si ritrovano soprattutto al livello della scelta di stili grafici (a cui però non siamo troppo interessati), ma anche nella gestione di alcune primitive. Queste sono le caratteristiche stilistiche che abbiamo analizzato in modo più approfondito, perchè sono quelle con cui ci siamo dovuti confrontare nelle scrittura del programma per il calcolo delle funzioni di taglia. Anche in questo caso elencare precisamente tutti i punti in cui i due stili si discostano non è molto utile, per ciò presentiamo solo alcuni esempi che ci sembrano più significativi e che potrebbere essere interessanti per ulteriori sviluppi del nostro software.

La tendenza di entrambi gli stili è quella di sfoltire il numero delle primitive grafiche utilizzate (si ricorda infatti come molte primitive siano riconducibili ad esempio ai cammini), anche se in modo non sistematico. Per esempio entrambi, pur essendo presente a livello di interfaccia una funzione per il disegno delle ellissi (o eventualmente solo di un arco ellittico ma in modo meno semplice) utilizzano per la loro descrizione a livello del file SVG solo il tag `<path>` con i comandi `A` e mai quello `<ellipse>`. E in modo contrario, per la descrizione dei rettangoli, entrambi utilizzano il tag `<rect>` e non la descrizione tramite cammino o poligono.

Caso interessante è quello della gestione dei cammini ed è naturale perchè si tratta, tra tutte le primitive geometriche a disposizione, di quelle descritte tramite gli strumenti matematici più sofisticati (o meno immediati): le curve di Bézier. Il software permette un gran numero di funzionalità per la modellazione di tali curve: la funzione di disegno è per default impostata in modo abbastanza intuitivo, ma poco duttile; sono però presenti e facilmente utilizzabili ad un secondo stadio (cioé per chi conosce la struttura matematica sottostante) praticamente tutti gli strumenti possibili per modificare le curve (spostamento, aggiunta, duplicazione, accoppiamento dei punti di controllo, passaggio da segmenti a curve, imposizione di tangenza o meno ecc.). Tutte queste funzionalità sono poi supportate e diversificate dallo stile inkscape-

svg; al contrario lo stile plain-svg opera al momento del salvataggio una notevole semplificazione delle strutture. Quello che avviene è che tutte le curve di Bézier vengono automaticamente trasformate in cubiche (anche se sono di grado 2): si perde così parte della struttura matematica che era stata data all'immagine, ma questo non ha nessun effetto nella visualizzazione che risulta esattamente uguale.

Si potrebbero far anche altri esempi, ma riteniamo sufficienti quelli presentati, anche perchè si tratterebbe di entrare in uno specifico poco interessante. Ciononostante abbiamo ritenuto utile dare abbastanza importanza alla questione dello stile di salvataggio all'interno dei file SVG per capire il tipo di problemi che ci siamo trovati ad affrontare. L'approccio che abbiamo usato in ogni momento, visto che si trattava della prima esplorazione della teoria della taglia in ambito vettoriale, è stato quello della semplificazione. Ogni volta che ci si poneva davanti una scelta, tenendo comunque presente la prospettiva di un software in grado di gestire la più ampia varietà di immagini vettoriali, abbiamo cercato di scegliere la strada più semplice, perdendo magari in generalità, ma permettendoci di arrivare ad uno sviluppo completo del software e quindi alle prime sperimentazioni reali. In tale prospettiva ci è sembrato naturale operare come prima semplificazione dei possibili stili di immagini da prendere in considerazione, la scelta di utilizzare solo immagini in stile plain-svg. Successivamente sono state necessarie semplificazioni anche molto più drastiche, ma di queste parleremo nel prossimo capitolo in cui verrà presentato nel dettaglio l'algoritmo di calcolo delle funzioni di taglia delle immagini vettoriali e la sua implementazione.

Capitolo 3

Algoritmo e implementazione del Sizetool

Come è stato anticipato, lo scopo di questa tesi è la scrittura di un software (che abbiamo chiamato *Sizetool*) per il calcolo delle funzioni di taglia di immagini vettoriali. In questo capitolo verrà descritto l'algoritmo che è stato utilizzato a questo scopo e esporremo i dettagli relativi all'implementazione dell'algoritmo stesso (che abbiamo effettuato in linguaggio ANSI C). Abbiamo voluto dare particolare attenzione alle strutture dati e ai passaggi implementativi più delicati o importanti, il tutto cercando di evidenziare le possibili prospettive per sviluppare e ampliare il programma.
Abbiamo esposto nel primo capitolo di questa tesi che una delle caratteristiche principali della teoria della taglia è la modularità. Per questo motivo nell'affrontare l'implementazione del programma ci siamo posti come principio quello di conservare la modularità che è propria del problema. In questa prospettiva l'algoritmo studiato ed implementato si può dividere in tre parti totalmente indipendenti:

A) input

B) calcolo della funzione misurante

C) calcolo della funzione di taglia

La parte A dell'algoritmo consiste nella lettura del file in formato SVG e nella memorizzazione delle informazioni geometriche contenute nell'immagine in opportune strutture dati. A partire da queste strutture dati, la parte B opera il calcolo della funzione misurante sull'immagine e la rappresenta, discretizzandola, mediante un grafo. La parte C è stata per la maggior parte ripresa dal lavoro di D'Amico [7]: parte dalla funzione misurante sotto forma di grafo di taglia, tramite un algoritmo di riduzione ne ottiene una Δ^*-riduzione totale e infine di questa calcola la funzione di taglia.
Dicendo che la struttura è modulare si intende che le tre parti dell'algoritmo sono totalmente indipendenti una dall'altra e si può lavorare su di esse (per esempio per ampliarle o generalizzarle) come se fossero un tutt'uno, senza considerare la loro relazione con il resto del programma. Proprio per questa caratteristica nel seguito di questo capitolo analizzeremo una per una separatamente le diverse parti dell'algoritmo.
Infine, per rendere la teoria della taglia utilizzabile a livello sperimentale come strumento di riconoscimento della forma, è necessario studiare degli indicatori che, date due funzioni di taglia, le confrontino quantificando la loro somiglianza o meno. Questo aspetto è analizzato nell'ultima parte dell'algoritmo (D), anche essa indipendente dalle precedenti, e costituisce una questione teorica dai risvolti molto interessanti. In particolare vedremo che il modo migliore di confrontare due funzioni di taglia, sfruttando la rappresentazione mediante successioni di punti e rette angolari, è quello di definire delle opportune distanze sullo spazio delle successioni e utilizzare come criterio di confronto il valore della distanza. L'analisi di questa parte, l'algoritmo utilizzato e la sua implementazione verrà esposta alla fine del capitolo.
A questa suddivisione dell'algoritmo in quattro parti tra loro indipendenti abbiamo fatto corrispondere l'organizzazione del codice del software in quattro librerie di funzioni, più una utilizzata solamente per definire i tipi di dato necessari in tutto il programma. In particolare quindi il programma è gestito da una funzione `main` che richiama al suo interno varie routine affrontando in successione le quattro parti dell'algoritmo. Le funzioni relative alla

parte A sono definite nella libreria `utilities.h`; quelle della parte B nella `mis_fun.h`; quelle della parte C nella `sizetool.h` e infine quelle della parte D nella `distances.h`. La libreria con la definizione dei tipi di dati è stata chiamata `definition.h`.

3.1 Parte A

La parte A dell'algoritmo prende in input una immagine in formato SVG e ottiene come output il salvataggio delle informazioni in essa contenute in opportune strutture dati. Come abbiamo cercato di mettere in evidenza, questa è probabilmente la parte più complicata dell'intero algoritmo, a causa dell'ampia libertà di stile concessa ai file in formato SVG. Partendo da questa caratteristica dell'input, è stato necessario restringere notevolmente i gradi di libertà concessi alle immagini. In secondo luogo, dal momento che lo scopo dell'implementazione del nostro programma era quello di poter iniziare a fare le prime sperimentazione della teoria della taglia su immagini vettoriali, si è deciso di ridurre notevolmente le primitive grafiche utilizzabili. In questo modo si è ottenuto di semplificare proprio la parte dell'algoritmo relativa al recupero delle informazioni (per l'appunto la parte A), lasciando così più spazio alle restanti parti dell'algoritmo, meno delicate ma che rappresentano la vera essenza della teoria. La modularità del problema ci ha permesso di effettuare questa scelta. Infatti abbiamo potuto creare un programma che pur essendo molto lontano dall'essere esaustivo, si può considerare completo, cioè in grado di svolgere dall'inizio alla fine il compito per cui è stato scritto: il calcolo della funzione di taglia di una immagine. Raggiunto questo risultato e verificata l'effettiva funzionalità del programma, allora ha senso affrontare la fase implementativa di estensione dei possibili input, cosa che può essere fatta in modo totalmente indipendente dal resto del software. In pratica, allo scopo di avere la massima semplificazione possibile e contemporaneamente di non perdere però in generalità (almeno potenzialmente), la scelta che abbiamo effettuato è quella di considerare come input immagini

costituite solamente da un tipo di primitiva grafica, nello specifico i rettangoli. Ovvero le immagini che abbiamo utilizzato contengono solamente il tag `<rect>` in cui sono sempre esplicitati tutti e quattro i parametri geometrici fondamentali (`x`, `y`, `width` e `height`). Si è inoltre sempre supposto che non ci fosse smussamento ellittico degli angoli (ovvero `rx` e `ry` entrambi nulli).

La scelta di considerare solo rettangoli ha comportato un'altra semplificazione del problema del reperimento delle informazioni e della loro rappresentazione, ovvero il problema della scelta delle strutture dati da utilizzare: cerchiamo ora di esporla. Il modo più semplice di memorizzare i dati contenuti nelle immagini è quello di creare un vettore di una particolare struttura dati che permetta di memorizzare al suo interno qualsiasi tipo di primitiva grafica. Questa è stata la nostra scelta che è stata messa in pratica nel seguente modo:

1. Si è definito per ogni primitiva grafica uno opportuno tipo di dato che permettesse di contenere tutte le informazioni geometriche della primitiva stessa (per esempio nel caso dei rettangoli è stata definita la struttura `rect_t` che contiene come campi le coordinate del vertice in alto a sinistra `x` e `y`, l'altezza `height` e la larghezza `width` del rettangolo e i parametri relativi allo smussamento `rx` e `ry`). La strutturazione dei tipi di dati (ovvero la scelta dei campi delle strutture) in teoria può essere fatta abbastanza liberamente, per cui si è scelto di ricalcare il più possibile l'impostazione che propone SVG, in modo di non creare inutile confusione sul significato delle variabili (per esempio nel caso del rettangolo si poteva decidere di utilizzare invece di `height` e `width` le coordinate del vertice in basso a destra).

2. Poi si è definito un tipo unione `forma_t` che può contenere una qualsiasi delle strutture che descrivono le primitive descritte nel punto precedente.

3. Infine si è definito il tipo di dato `element_t` che è una struttura i cui

campi sono una variabile detta `elemento` di tipo `forma_t` che conterrà le informazioni relative alla primitiva e una variabile di tipo carattere chiamata `forma` che esplicita il tipo di primitiva memorizzata scegliendo quindi quale tipo di dato assegnare alla variabile `elemento` (per esempio per scegliere il tipo `rect_t` si deve assegnare il valore r alla variabile `forma`).

4. Si è creato un vettore `disegno` le cui componenti sono di tipo `element_t` e conterranno ciascuno le informazioni relative ad una diversa primitiva.

Così abbiamo impostato il software in modo che memorizzi l'immagine in un vettore di strutture dati che ricalcano abbastanza fedelmente la struttura delle primitive grafiche di SVG; d'altra parte, per la sua organizzazione in oggetti e sotto-oggetti, un file SVG non ha una struttura lineare (non è un vettore di primitive), bensì una naturale struttura ad albero. Uno dei compiti della parte A dell'algoritmo è la trasformazione (o meglio la conversione) dell'albero in un vettore. Ancora una volta a causa della libertà di stile del formato questo problema è molto complesso, a meno di effettuare delle opportune semplificazioni. Anche in questo caso, infatti, la risoluzione completa del problema, benchè sia un obiettivo che ci dobbiamo porre, ha poco a che fare con la teoria della taglia applicata alle immagini vettoriali ed è solo un ostacolo di natura "formale".
La scelta di prendere come primitive grafiche ammissibili solo rettangoli ha notevolmente semplificato questo problema. Avendo, infatti, imposto che all'interno dei tag `<rect>` fossero sempre specificati tutti e quattro i parametri fondamentali per descrivere la geometria del rettangolo (vertice in alto a sinistra, altezza e larghezza), la lettura di tali dati è indipendente dalla posizione all'interno del file SVG. Al contrario, se per esempio si fosse permesso di sottintendere le coordinate del vertice, queste sarebbero state date dalla posizione corrente del cursore che è invece un dato che dipende dalla struttura del file e dalla posizione in cui il tag è inserito all'interno del testo.

D'altra parte se con questi accorgimenti si è riusciti a risolvere (o meglio

momentaneamente aggirare) il problema relativo alle informazioni geometriche, questo non avviene con quelle grafiche. Infatti la definizione di uno stile (colore del riempimento, spessore delle linee, ecc.) all'interno di un oggetto definisce a cascata gli stessi attributi per tutti i sotto oggetti in cui non è indicato diversamente. Anche i parametri grafici quindi sono fortemente dipendenti dalla struttura ad albero. Ancora una volta la soluzione che abbiamo proposto all'interno del nostro sofware è stata quella di semplificare, imponendo che l'unico colore utilizzato fosse il nero, e che in una immagine i rettangoli fossero o tutti riempiti di nero, oppure tutti non riempiti e con contorni neri di spessore fissato[1].

A questo punto può essere interessante osservare che uno dei possibili sviluppi della teoria della taglia su immagini vettoriali consiste nello studio di come il colore potrebbe essere considerato uno dei parametri di differenziazione tra due immagini, insieme alla forma, alle dimensioni eccetera. In particolare si potrebbero provare a sperimentare delle particolari funzioni misuranti che dipendono non da caratteristiche geometriche, ma solo da parametri cromatici, o più generalmente grafici. Implicitamente infatti non considerere queste variabili che sono parte integrante delle immagini è come imporre di considerare solo funzioni invarianti per cambiamento di colore.

Passiamo ora a spiegare algoritmo che si è utilizzato per la ricerca delle informazioni all'interno del testo di un file SVG e come questo è stato implementato. L'algoritmo è stato impostato per effettuare una ricerca iterata per tag. Ovvero per ogni tipo di tag (per esempio `<rect>`) che il formato SVG prevede, si effettua un ciclo di lettura dell'intero file che va a cercare tutte le ricorrenze della stringa nel testo e, una volta trovata, inizia la lettura dei dati contenuti all'interno[2].

[1] Una più specifica trattazione sulla differenziazione tra rettangoli "pieni" e rettangoli "vuoti" verrà fatta nella sezione successiva dove si spiega la parte B dell'algoritmo.

[2] Ricordiamo che SVG è scritto in XML, il quale pur essendo molto rigido riguardo all'apertura e chiusura dei tag, permette due opzioni, ovvero il tag contratto che con-

Si osserva che, effettuando questo tipo di ricerca che potrebbe essere definito sequenziale (cioè elemento per elemento e seguendo l'ordine ma non la struttura data dal testo), si perdono completamente le informazioni sulla struttura ad albero dell'immagine. D'altra parte l'ipotesi di avere solo rettangoli ci permette di non avere problemi di perdita di informazione. Si è deciso di effettuare comunque la ricerca in questo modo, anche se presenta delle lacune, perchè permette un facile ampliamento del software volto ad aumentare il numero di primitive grafiche ammesse come input. Infatti avendo impostato un ciclo sui diversi tag, che al momento prevede una sola iterazione (relativa a `<rect>`), è sufficiente inserite le opportune funzioni di lettura dati per altri tag e poi estendere il ciclo anche ad essi, inserendoli come casi ammissibili.
L'algoritmo opera dunque in questo modo[3]:

1. Sceglie un tag

2. Finchè il file non è finito cerca la prima ricorrenza del tag scelto

3. Se trova il tag

    - legge e memorizza i dati relativi alla primitiva grafica
    - sposta il cursore di lettura alla fine del tag (dopo i caratteri `/>`) in modo da poter proseguire nella ricerca dell'eventuale tag successivo

tiene al suo interno le informazioni come parametri (per esempio`<rect width=''10'' height=''20'' />`) e quello standard caratterizzato da una apertura (per esempio `<rect>`, seguita dai parametri espressi come campi di una struttura e infine con una chiusura (come `</rect>`). Nell'implementazione della ricerca del tag, si è perciò sempre imposto la ricerca della stringa `<rect` senza la parentesi uncinata chiusa, ritrovando in questo modo tutte e due le possibili presentazioni del tag. Poi in realtà l'input che abbiamo utilizzato prevedeva solo il tipo contratto di tag, ma con una semplice modifica del codice si può introdurre la libertà di utilizzare entrambi.

[3]I dettagli relativi all'implementazione e alcune parti del codice sono inseriti alla fine di questa tesi nell'Appendice A.

4. Se invece arriva alla fine del file, sceglie un altro tag e ricomincia la lettura

Benchè siano previsti solo rettangoli, la fase di lettura e memorizzazione dei dati è stata impostata in modo che sia gestita da una funzione unica (che abbiamo chiamato `assegna_el`) ma estendibile ad ogni tipo di primitiva. Ovvero l'idea è che essa prenda in input il tipo di tag che si sta leggendo, e sulla base di questo operi una successione di ricerche (all'interno del tag stesso) dei comandi che introducono i parametri descrittivi della primitiva, in modo da individuare precisamente dove andare a leggere i dati (mediante delle funzioni di tipo *scanf*) e che significato dare loro (ovvero in che campo delle strutture dati memorizzarle). Perció, dato che ogni struttura dati ha parametri diversi, la cosa più naturale sarà quella di implementare per ogni singolo tag una diversa parte della funzione, che sia alternativa alle altre (mediante un controllo di tipo `switch`), e che scelga i parametri da cercare.

Prima di passare all'analisi delle parti successive dell'algoritmo, riteniamo utile evidenziare quali sono le direzioni in cui muoversi per sviluppare e perfezionare il software.
In un primo momento sarebbe utile, per superare l'attuale condizione in cui le sole primitive ammesse sono i rettangoli, implementare le funzioni di lettura dei parametri e memorizzazione dei dati anche per altri tipi di primitive grafiche. Per fare questo si dovrebbe per prima cosa creare delle nuove strutture dati apposite per gestire tali primitive (per esempio i poligoni, le ellissi, ecc.), poi implementare le nuove parti della funzione `assegna_el` per poter leggere e memorizzare i dati della primitiva sulla struttura apposita, infine prevedere che l'algoritmo di ricerca dei tag operi anche su quelli aggiunti.
In un secondo momento sarebbe molto importante affinare l'algoritmo di ricerca dei tag (o come è spiegato meglio nell'appendice A in generale delle stringhe). Per fare questo si può sicuramente cercare di appoggiarsi su algoritmi più stabili implementati da altri autori, in quanto lo stato dell'arte sulla ricerca di stringhe è piuttosto avanzato.

Infine, e questo è il compito più difficile, si dovrebbe affrontare la questione di come non perdere informazioni nella riduzione dell'albero di oggetti a vettore di strutture dati. Una prima soluzione, non completa, è quella di convertire, al momento della lettura, tutti i comandi di tipo relativo in comandi assoluti e, in caso di parametri sottointesi, essere in grado di ricavarne il valore in modo che, al momento della memorizzazione nel vettore di strutture dati, né si perdano informazioni né queste siano soggette ad errori. Al momento non ci sentiamo di dare una direzione in cui trovare una possibile soluzione a questo problema, ma possiamo con sicurezza affermare che non si tratta solo di un problema tecnico (cioè magari dovuto all'inadeguatezza degli algoritmi di lettura e ricerca del file), bensì di qualcosa di molto più profondo che ha a che fare da un lato con l'ampia liberà di stili concessa da SVG (di cui abbiamo già ampiamente parlato) e dall'altro con gli inevitabili limiti che si incontrano quando si cerca di ingabbiare, in rigide strutture dati, concetti che sono il frutto della creatività, e che quindi sono quanto di più lontano dagli schemi della razionalità.

3.2 Parte B

La parte B dell'algoritmo riceve in input dalla precedente, sotto forma di un vettore di strutture dati, le informazioni geometriche relative alle primitive da cui è costituita l'immagine; a partire da queste calcola il valore che assume la funzione misurante scelta nei diversi punti dell'immmagine e, dopo un procedimento di discetizzazione, fornisce come output un grafo di taglia che rappresenta la funzione misurante stessa in opportuni punti dell'immagine. In particolare, come si è cercato di spiegare nel primo capitolo di questa tesi, il grafo di taglia contiene tutti quei punti dell'immagine sui quali la funzione misurante assume un valore critico (di massimo o di minimo).
In questa parte dell'algoritmo si trova maggiormente il contenuto originale della tesi. Questo perchè nella prima parte, come abbiamo visto, si sono riscontrati per lo più problemi non legati alla teoria della taglia, ma alla

"forma" di rappresentazione delle informazioni, e infatti non ci siamo concentrati molto nella risoluzione di tali problemi preferendo assumere ipotesi restrittive per aggirarli. Al contrario la terza e la quarta parte dell'algoritmo sono già stati ampliamente affrontati da D'Amico ([7]) e abbiamo potuto in gran parte utilizzare i suoi risultati con un semplice lavoro di adattamento. La novità di questa tesi sta nell'applicazione della teoria della taglia alle immagini vettoriali, perciò è proprio il passaggio da tali immagini alle funzioni misuranti rappresentate come grafi di taglia che costituisce il passaggio nuovo e cruciale per l'esito dei nostri studi. Una volta, infatti, calcolato il grafo di taglia, è sufficiente utilizzare l'algoritmo di D'Amico per implementare la Δ^*-riduzione totale e calcolare la funzione di taglia.

Il vero problema di questa parte è quindi trovare un modo di rappresentare, tramite strutture matematiche, una immagine bidimensionale in modo da poterci calcolare la funzione misurante, verificare in che punti assume valori critici e trovarne una discretizzazione efficiente. In particolare si deve tenere conto che la rappresentazione dell'immagine è solo un passaggio intermedio per permettere il calcolo della funzione misurante. Infatti una volta che si conoscono i valori che assume, la sua trasformazione in grafo di taglia le fa perdere qualsiasi riferimento geometrico-spaziale globale, ovvero i nodi non avranno una posizione o delle coordinate, bensì conterranno solo informazioni di tipo locale (a quali altri nodi sono connessi e il valore della funzione di taglia). Questo è importante perchè ci deve far capire che la rappresentazione dell'immagine può anche essere non del tutto rispettosa dell'originale, ma ciò che conta è che permetta un facile calcolo della funzione misurante. Si può anzi dire di più: un approccio sicuramente interessante è quello di modificare il modo in cui rappresentarla, a seconda della funzione misurante scelta. E questo è stato il metodo di lavoro che abbiamo utilizzato: prima scegliere la (o le) funzioni misuranti, poi decidere come rappresentare l'immagine per poterci fare i calcoli sopra. Come ulteriore considerazione, poichè abbiamo a che fare con una immagine vettoriale e quindi rappresentata da oggetti continui, il calcolo della funzione misurante implica già implicitamente una sua

discretizzazione, in quanto non è ovviamente possibile avere una struttura dati digitale che contenga i valori di una funzione su un insieme non discreto. Ricapitolando:

1. scelta della funzione misurante
2. scelta (di conseguenza) del modo di rappresentare l'immagine
3. calcolo e contemporanea discretizzazione della funzione misurante in un grafo di taglia

Partiamo dunque dalla scelta delle funzioni misuranti che abbiamo effettuato. Benchè abbiamo elaborato, almeno teoricamente, un metodo di rappresetazione abbastanza generale per una immagine vettoriale su cui è definita una funzione misurante (metodo che esporremo alla fine di questa parte come possibile sviluppo dei nostri studi), si è deciso di iniziare con delle funzioni che fossero abbastanza semplici, ma non per questo poco significative. Questo per far sì di poter ottenere con facilità dei risultati e poi lavorare su di questi. In particolare in questa fase del lavoro non ci interessa elaborare un software che effettivamente sia un buon riconoscitore di forma, ma siamo ancora allo stadio preliminare in cui si vuole testare su opportuni database di immagini gli algoritmi implementati con varie funzioni misuranti, per capire che tipo di informazioni sulla geometria dell'immagine ci permettono di ricavare. Ovvero lo scopo, fissato che si vuole utilizzare le funzioni di taglia come descrittori delle immagini, è capire cosa queste "mostrano" delle immagini stesse, a seconda delle funzioni misuranti scelte.
Le funzioni misuranti che abbiamo utilizzato, raggruppabili in due famiglie, hanno una caratteristica che permette una notevole semplificazione della loro rappresentazione. Tali funzioni infatti non sono propriamente definite sull'immagine (quindi su uno spazio 2-dimensionale), ma su opportune rette del piano in cui l'immagine è contenuta, sono perciò funzioni di una sola variabile. Si vede facilmente come la discretizzazione di una funzione reale di una sola variabile reale si può rappresentare per mezzo di una struttura dati a lista ordinata: a ogni punto della funzione discretizzata (supponiamo infatti

che si sappia già quali punti si devono rappresentare) si fa corrispondere un nodo della lista con due valori numerici che indicano la sua ascissa (x) e la sua ordinata (y), inoltre i nodi vengono ordinati in modo crescente rispetto alla x.
La scelta di funzioni di questo tipo sembra farci uscire dalla teoria della taglia come è stata esposta nel primo capitolo. Se si osserva bene, però, il fatto che il dominio della funzione non sia l'immagine stessa ma un qualsiasi altro spazio topologico, non è rilevante. La funzione misurante, infatti, serve solo in un primo momento per ottenere delle informazioni numeriche relative all'immagine. Successivamente il legame con il dominio svanisce, poichè sono solo i valori critici della funzione misurante e non i punti in cui questi vengono assunti, ciò che veramente serve per il calcolo delle funzioni di taglia. Questo fatto è davvero cruciale per la teoria della taglia, e, a nostro parere risulta uno dei suoi punti di forza. Per cui, ciò che si può supporre in modo più generale, è che la funzione misurante sia una applicazione f da uno spazio topologico X a valori in $\mathbb{R}$ che dipende in qualche modo dall'immagine. Lo scopo della teoria della taglia è studiare come questa dipendenza si manifesta e fino a che livello si può quantificare con il punto vista delle funzioni di taglia.
Descriviamo ora le due famiglie di funzioni misuranti utilizzate (entrambe dipendono dalla scelta di una particolare retta del piano).

Funzione SALTI. Si considera una retta, appartenente al piano in cui è contenuta l'immagine, che supponiamo di equazione parametrica

$$r : \begin{cases} x = at + b \\ y = ct + d \end{cases}$$

Si prende come variabile indipendente della funzione il parametro reale t e per ogni suo valore si contano i salti bianco-nero-bianco che si hanno lungo

la retta perpendicolare, ovvero di equazione parametrica (in u)

$$r_{\perp}(t) : \begin{cases} x = -cu + (at + b) \\ y = au + (ct + d) \end{cases}$$

Per esempio, se si considera un'immagine costituita da un segmento L che unisce i punti P e Q di coordinate $P = (x_1, y_1)$ e $Q = (x_2, y_2)$, allora la funzione misurante relativa a questa immagine e alla retta di equazione $y = 0$ vale identicamente 1 nell'intervallo $[x_1, x_2]$, mentre altrove è nulla. Dal momento che le immagini sono costituite dall'unione di un numero finito di forme geometriche continue, si ricava che la funzione misurante che si ottiene è costante a tratti con un numero finito di punti di discontinuità, ovvero non è altro che una combinazione lineare finita a coefficienti interi di funzioni caratteristiche di intervalli. Per questo motivo si può trovare una rappresentazione discreta della funzione che in realtà permette di conoscere il suo valore esatto in ogni punto (eccetto su un insieme discreto e finito di punti, quindi di misura nulla). Tale rappresentazione avviene tramite la definizione di una lista ordinata (finita) di N nodi (che qui indichiamo con $K(i)$), in cui a ciascun nodo sono associati due valori numerici ($K(i).x$ e $K(i).val$) che rappresentano rispettivamente la variabile indipendente della funzione misurante (t) e il valore che la funzione misurante assume nell'intervallo aperto $(K(i).x, K(i+1).x)$ per ogni $1 \leq i \leq N-1$. Supponendo un primo nodo sottointeso ($K(0)$ con $K(0).x = -\infty$) e con valore nullo ($K(0).val = 0$), si ha il valore della funzione misurante in ogni punto reale tranne che in quelli di discontinuità $K(i).x$. E in realtà anche in tali punti si può sempre ricavare, infatti si ha $f_{SALTI}(K(i)) = \max\{K(i-1).val, K(i).val\}$.

Funzione FILL. Questa funzione è molto simile alla precedente, ma il suo calcolo non è significativo se si considerano rettangoli non pieni di colore, per cui il calcolo di questa funzione è stato effettuato solo per immagini costituite esclusivamente da rettangoli 'pieni' (da cui *FILL*). Ricordiamo infatti che tutte le primitive geometriche utilizzate da SVG in cui è previsto l'attributo `fill`, sono dotate di una estensione 'reale' (nel senso di misurabile) e si può

dunque calcolare quanto sono larghe, lunghe o per esempio la loro area[4]. Per la definizione di questa funzione f_{FILL} si considera sempre una retta del piano r data dall'equazione parametrica precedente, e, come variabile indipendente, il parametro t. Il valore della funzione in un punto t è pari alla 'quantità di nero intercettata' dalla retta perpendicolare $r_\perp(t)$: ovvero, se I indica l'insieme dei punti colorati di nero, la somma delle lunghezze dei segmenti che costituiscono l'insieme $r_\perp(t) \cap I$ (un segmento degenere ha lunghezza nulla).

Per esempio (si confronti il disegno) se si considera un'immagine che consiste di un solo rettangolo dato da `x`= 0, `y`= 100, `width`= 400, `height`= 100 e come retta $r : x + y = 0$. Allora la funzione misurante è nulla fino al punto di ascissa $t = -\frac{100}{\sqrt{2}}$ poi sale linearmente con coefficiente angolare pari a 2, fino ad arrivare al punto $t = 0$ in cui la funzione assume il valore $100\sqrt{2}$. La funzione ha questo valore costante fino al punto di ascissa $t = \frac{300}{\sqrt{2}}$ poi descresce linearmente con coefficiente angolare -2 fino al punto di ascissa $t = \frac{400}{\sqrt{2}}$ dove vale 0 e rimane in seguito constantemente nulla.

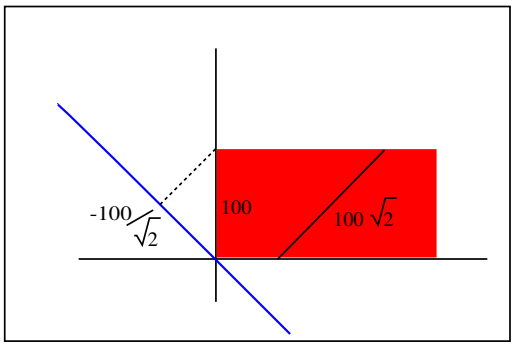


Figura 3.1: Calcolo della funzione fill.

[4]Questo a priori non si può fare per le curve, che non hanno l'opzione `fill` perchè si tratta di primitive monodimensionali e quindi teoricamente prive di spessore. In realtà però SVG con il parametro `stroke` attribuisce a queste curve uno spessore quantificabile (misurato in millimetri o pixel) quindi si potrebbe estendere questa funzione anche al caso di entità geometriche teoricamente non dotate di spessore. Non è stato fatto perchè anche solo nel semplice caso dei rettangoli "vuoti", si generavano delle funzioni molto più irregolari o complesse da calcolare.

Se invece si prende come retta l'asse $x = 0$ allora molto più semplicemente la funzione misurante vale 400 nell'intervallo $[-100, 0]$ ed è nulla altrove (si suppone che l'asse verticale sia orientato verso il basso come convenzione spesso usata in ambito grafico).
Dai due esempi si osserva che, nel caso particolare in cui si scelga come retta r uno degli assi cartesiani, la funzione di taglia è come in precedenza una combinazione lineare (non più intera) di funzioni caratteristiche di intervalli, mentre nel caso generico si ottiene una funzione continua e lineare a tratti. Entrambi questi due tipi di funzioni si possono rappresentare facilmente con una struttura di lista ordinata: la prima come già spiegato, la seconda con una successione di nodi $K(i)$ dotati di due valori $K(i).x$ e $K(i).val$ ma ipotizzando sempre che il valore della funzione in un punto t sia dato dall'interpolazione lineare tra i valori dei nodi con x immediatamente precedente e successivo, ovvero trovato i tale che $K(i).x \leq t \leq K(i+1).x$ si ha:

$$f_{FILL}(t) = K(i).val + \frac{K(i+1).val - K(i).val}{K(i+1).x - K(i).x}\,(t - K(i).x)$$

Ovviamente i due casi in cui si sceglie come rette gli assi, costituiscono dei casi limite, in cui la funzione lineare avrebbe in alcuni tratti coefficiente angolare infinito, per cui se da un punto di vista delle strutture dati non c'è da fare alcuna distinzione, in realtà nel calcolo che si effettua e quindi anche nella rappresentazione si deve tenere presente quando si ha a che fare con questi due casi[5].

Dopo aver scelto le funzioni misuranti, seguendo l'elenco numerato che abbiamo inserito all'inizio di questa sezione (per schematizzare il modo di procedere per risolvere il problema relativo alla parte B dell'algoritmo), si dovrebbe trattare del modo in cui rappresentare l'immagine. D'altra parte

[5]Inoltre si deve osservare che noi abbiamo imposto che i rettangoli delle immagini siano sempre solo paralleli agli assi, ma se si decidesse di implementare anche eventuali inclinazioni dei lati o rotazioni delle figure (entrambi possibili in SVG), si dovrebbe tenere conto delle possibilità di avere in una stessa funzione misurante tratti degeneri e non, quale che sia la retta r scelta.

ad una più attenta analisi si osserva che quello che ci interessa veramente non è rappresentare l'immagine, bensì rappresentare la funzione misurante (e quindi il suo dominio prima di tutto). Infatti una volta trovato un modo di gestire l'immagine quello che si fa è calcolare su di essa la funzione di taglia e discretizzarla trasformandola in un grafo di taglia: in questo modo si perdono alcuni dettagli del dominio, e si considerano solo proprietà locali, come la connessione tra due nodi o meno. Rappresentare l'immagine è solo un modo di rappresentare il dominio della funzione misurante. Ma nel nostro caso, come è stato spiegato, dal momento che il dominio non è l'immagine, ma l'asse reale, abbiamo una facile rappresentazione della funzione come lista ordinata. Ed abbiamo anche visto, in entrambi i casi delle due famiglie di funzioni misuranti, come la rappresentazione sia contemporaneamente un modo per descrive tutti i punti della funzione, lasciando inalterata la struttura dati, ma modificando l'interpretazione, e anche una via per trovare la discretizzazione. Nel caso delle funzioni FILL questo è ovvio, nell'altro è sufficiente supporre che i nodi non rappresentino la funzione caratteristica di un intervallo (come abbiamo detto in precedenza), ma solo uno qualsiasi dei punti di quell'intervallo, insieme con il valore assunto dalla funzione.
In pratica, nel caso particolare delle due famiglie di funzioni misuranti, i due punti finali vengono eseguiti implicitamente con la sola definizione dell'opportuna struttura dati per le funzioni stesse. L'unico problema è quello di calcolare i loro valori e inserirli nella struttura dati, ed è quello che viene fatto dalla principale funzione della libreria `mis_fun.h` che abbiamo chiamato `rect2fun` e di cui ora spieghiamo brevemente l'algoritmo.
La funzione prende in input il vettore di strutture dati che contengono le informazioni sui rettangoli e una lista ordinata di punti "vuota". Alla fine della funzione la lista rappresenta la funzione misurante scelta (nel caso di entrambe le famiglie). L'algoritmo opera in modo iterativo, prende in esame uno per volta i rettangoli del vettore e sulla base di questi modifica la lista inserendo nuovi nodi e/o modificando i valori dei nodi già presenti. Affinchè anche il primo passaggio della funzione fosse implementato in questo modo,

la lista cosiddetta vuota, che si passa come input alla funzione e che viene man mano aggiornata, è in realtà data da due nodi con *val* pari a 0 e che rappresentano in un certo senso i due estremi della funzione: hanno come valore della x degli opportuni x_{min} e x_{max} tali che tutti i nodi che verranno inseriti saranno intermedi tra i due. Vediamo come avviene il passo generico n-esimo della funzione. Si assume che sia data una certa lista di nodi $K(i)$ con $0 \leq i \leq N(n)$ (al primo passaggio $N(1) = 1$) e l'n-esimo rettangolo della lista $R = R(n)$. Per prima cosa si calcolano le coordinate x^- e x^+ che individuano l'intervallo corrispondente alla proiezione ortogonale di R sulla retta r. Per esempio nel semplice caso in cui la retta è l'asse delle ascisse, tali punti sono $R.x$ e $R.x + R.width$. Successivamente si individuano i nodi $K(i^-)$ e $K(i^+)$ tali che

$$K(i^-).x \leq x^- < K(i^- + 1).x \text{ e } K(i^+ - 1).x < x^+ \leq K(i^+).x$$

Se le disuguaglianze appena scritte sono strette, si aggiungono degli opportuni nodi K_n^- e K_n^+ in modo che la lista diventi (usiamo '$\succ$' come simbolo di nodo successivo):

$$K(0) \succ \ldots \succ K(i^-) \succ K_n^- \succ K(i^- + 1) \succ \ldots$$
$$\ldots \succ K(i^+ - 1) \succ K_n^+ \succ K(i^+) \succ \ldots K(N(n))$$

dove $K_n^-.x = x^-$ e $K_n^+ = x^+$. Se invece le disuguaglianze non sono strette e dunque esistono già dei nodi con coordinata x pari a x^- o x^+, questi sono proprio $K(i^-)$ e/o $K(i^+)$. Si pone perciò $K_n^- := K(i^-)$ e/o $K_n^+ := K(i^+)$.
Fin qui gli algoritmi per le due famiglie di funzioni sono identici. Il passo successivo consiste nell'eventuale (ri)assegnazione di $K_n^-.val$ e/o $K_n^+.val$ e nella modifica di tutti i nodi ad essi intermedi (se ce ne sono). Questa parte dell'algoritmo invece si differenzia a seconda della famiglia di funzioni misuranti che stiamo considerando. L'assegnamento del valore dei nodi K_n^- e K_n^+ se sono stati aggiunti e la modifica dei valori dei nodi compresi tra questi due verrà trattata nell'appendice B. Questo perchè è vero che da un punto di vista algoritmico la risoluzione del problema è immediata, ovvero si assegna:

- $K_n^-.val := a$
- $K_n^+.val := b$
- $\forall\ K(j)$ intermedio tra questi due $K(j).val = K(j).val + d$

dove a seconda dei casi si deve calcolare il valore delle costanti a, b e d. D'altra parte, però, il calcolo di questo costanti è spesso non immediato, e comunque rappresenta un tecnicismo che non ha senso proporre in questo momento (si tratta per lo più di alcune considerazioni trigonometriche). Una volta assegnati i nuovi valori, l'iterazione è finita e si passa alla successiva avendo come input un nuovo rettangolo $R = R(n+1)$ e la lista appena modificata data dai nodi, che si indicano genericamente ancora $K(i)$, ma con $0 \leq i \leq N(n+1)$.

Chiudiamo ora l'esposizione della parte B dell'algoritmo, esponendo, come possibile prossimo obiettivo per l'ampliamento di questa parte, un metodo di rappresentazione delle immagini che abbiamo studiato teoricamente, ma non sperimentato. L'unica assunzione che si fa sull'immagine è che sia costituita esclusivamente da cammini (esclusi gli archi di ellisse) e da forme non riempite (escluse le ellissi), il cui contorno è rappresentabile quindi anch'esso come un cammino. Come è stato detto nel capitolo 2 tutti questi diversi tipi di primitive si possono sempre rappresentare come successioni di cammini le cui componenti sono tutte curve parametriche di grado variabile tra 1 e 3. Ovviamente ogni cammino si può trasformare a sua volta in una sottosuccessione di cammini semplici (cioè costituiti da una sola curva), semplicemente considerando ogni curva come un cammino a sé stante. Non è dunque riduttivo, nelle nostre ipotesi, supporre che l'immagine sia costituita da una successione di curve c_i con $1 \leq i \leq n$ parametriche di equazione:

$$c_i : \begin{cases} x = x_i(t) = a_i^1 t^3 + b_i^1 t^2 + c_i^1 t + d_i^1 \\ y = y_i(t) = a_i^2 t^3 + b_i^2 t^2 + c_i^2 t + d_i^2 \end{cases}$$

dove eventualmente alcuni dei coefficienti $a_i^j, b_i^j, c_i^j, d_i^j$ possono essere nulli e $t \in [0,1]$ per ogni curva. A questo punto l'idea è di costruire un particolare tipo di grafo in cui i nodi sono dati dai punti di inizio e fine di ogni curva, e gli spigoli sono in corrispondenza con le curve stesse e contengono l'informazione relativa alla loro equazione. Vediamo il modo in cui costruire il grafo per capire meglio come è fatto. Per prima cosa si costruiscono due liste di elementi. La prima V è quella dei vertici, in cui inseriamo un nodo per ogni punto di inizio e/o fine di una curva. La seconda E è la lista degli spigoli, in cui inseriamo un elemento per ogni curva. I dati contenuti in ogni nodo v sono le coordinate del punto P rappresentato e il puntatore ad ogni spigolo che rappresenta una curva $c(i)$ con estremo iniziale o finale il punto P. Mentre i dati contenuti in ogni spigolo $e = e(i)$ sono i coefficienti delle equazioni parametriche che definiscono la curva $(a_i^j, b_i^j, c_i^j, d_i^j)$ e il puntatore al punto di inizio e fine della curva stessa, dove se il puntatore è unico si ha che la curva parte ed arriva nello stesso punto. Questa struttura costituisce di per sè già un grafo, ed in realtà, sfruttando la parametrizzazione, si può pensare che ogni spigolo sia uscente dal vertice che rappresenta il punto $P = (x_i(0), y_i(0))$ e diretto nel vertice che rappresenta $Q = (x_i(1), y_i(1))$. In questo modo il grafo diventa un grafo orientato. Ma è ancora molto lontano dal rappresentare l'immagine di partenza. Quello che si deve fare è inserire i punti di intersezione tra i vari archi. Per farlo si considera uno per volta gli archi della lista E e si verifica se si intersecano con quelli precedentemente considerati. In caso di intersezione, si aggiungono all'insieme V tanti punti quante sono le intersezioni fra la curva e la parte già aggiornata del grafo (dei suoi spigoli) e ogni spigolo che viene diviso in due da una intersezione viene cancellato dalla lista E aggiungendo però due nuovi elementi, uno per ciascuna delle due parti un cui è diviso. Operando ogni volta delle riparametrizzazioni in modo che tutte curve inserite siano con $t \in [0,1]$ (e questo comporta la modifica dei coefficienti) e aggiornando per ogni nodo e spigolo i vettori di puntatori, rispettivamente, agli archi entranti e uscenti e ai punti di inizio e fine si ottiene una rappresentazione standard, mediante

un grafo, di tutti i punti dell'immagine, la quale rispecchia anche abbastanza fedelmente la struttura globale dell'immagine stessa.
A partire da questo grafo si può calcolare una qualsiasi funzione di taglia f in ogni punto, basta infatti percorrere ogni arco della lista E e calcolare il valore della funzione lungo la curva che esso rappresenta, grazie alla conoscenza dei coefficienti:

$$f(P) = f(x, y) = f(x_i(\bar{t}), y_i(\bar{t}))$$

Il passaggio finale per ottenere anche una rappresentazione discreta della funzione misurante è come nel passaggio precedente, considerare uno per volta ogni spigolo e cercare su di esso tutti i punti critici, inserirli nella lista dei vertici V e dividere tutti gli archi in due per ogni punto critico trovato (la parte precedente e quella successiva), riparametrizzando e modificando i coefficienti e modificando opportunamente E. A questo punto è sufficiente che ogni nodo del grafo abbia un campo in cui assegnare il valore della funzione misurante nel punto rappresentato dal nodo in questione e si ha la rappresentazione della funzione misurante. A questo livello, ripetiamo ancora una volta, di tutta la struttura precedente non interessa più la descrizione dell'immagine (per esempio i coefficienti delle curve parametriche o le coordinate dei punti), ma solo le connessioni tra i nodi e il valore in essi della f. Si ha cioè il grafo di taglia, ovvero l'output desiderato.
Come abbiamo detto questo algoritmo è stato studiato solo teoricamente e non sono stati approfonditi i dettagli tecnici per implementarlo. In particolare non ci si è soffermati sul metodo in cui trovare i punti di intesezione tra curve oppure i punti critici della funzione f. L'idea di fondo è che si è cercato di utilizzare per la rappresentazione dell'immagine delle strutture che fossero simboliche e che permettessero di ricostruire con facilità le proprietà analitiche e geometriche delle primitive matematiche in modo da consentire di fare dei veri e proprio calcoli su di esse. In particolare, e ci sembra un notevole pregio, supponendo le curve di grado sufficientemente basso (al più 3), si avrebbe che i calcoli possono essere fatti in modo algebrico, utilizzando delle formule chiuse sia per trovare i punti di intesezione che i punti di

critici della f (se si assume anche questa di tipo polinomiale e grado basso). In questo modo si eviterebbe una generazione-propagazione di errori dovuta agli algoritmi di calcolo analitico, aumentando così la solidità dell'algoritmo. A nostro parere questo ci sembra l'approccio giusto da avere in futuro per effettuare una valida ed efficace implementazione di un numero maggiore di funzioni di taglia e di primitive accettabili a livello di immagine input.

3.3 Parte C

La parte C dell'algoritmo prende in input un grafo di taglia, che rappresenta la discretizzazione di una funzione misurante (d'ora in poi f), e ne calcola la funzione di taglia. Si è divisa questa procedura in due parti: la prima trasforma il grafo (d'ora in poi G) in una sua Δ^*-riduzione totale, rappresentata da un albero (o se non connesso da una foresta)[6]; la seconda calcola la funzione di taglia dell'albero. Questo tipo di approccio al problema si basa sui risultati teorici che abbiamo esposto nel primo capitolo: possiamo modificare il grafo di partenza con una successione di trasformazioni (dette Δ^*-riduzioni) che lo semplificano, senza modificare la funzione di taglia associata. Diamo ora una spiegazione di come opera la procedura. Ci siamo essenzialmente basati sull'algoritmo esposto da D'Amico in [7], ma nella descrizione che egli ha dato abbiamo riscontrato una lacuna (su di un caso particolare) e quindi vi abbiamo inserito una modifica che in seguito esporremo. Vediamo intanto l'algoritmo di D'Amico.

[6]In tutta l'esposizione dell'algoritmo, a meno di diversa indicazione, è indifferente che il grafo risulti connesso o meno, per cui per semplicità supporremo sempre i grafi connessi. Parleremo quindi di alberi e non foreste, anche se il termine potrebbe essere, a seconda dei casi, improprio.

3.3.1 La Δ^*-riduzione

Innanzitutto ricordiamo che l'algoritmo di riduzione genera una struttura che è un po' di più di un albero: una arborescenza (d'ora in poi H). Ovvero l'albero che si trova ha un modo naturale di individuare una radice e, dati due nodi adiacenti, di orientare lo spigolo tra i due, individuando così un padre e un figlio. Il criterio è semplicemente dato dal valore assunto dalla f sul nodo: la radice è il nodo in cui si ha il punto di massimo globale, e un arco va sempre da un valore maggiore ad un valore minore. Data questa proprietà del risultato che si vuole ottenere, preliminarmente all'algoritmo, si riordinano i nodi del grafo di taglia in modo crescente rispetto alla f, ottenendo così che l'insieme $V(G)$ sia totalmente ordinato dalla relazione $\succ$ definita da $v \succ w \Leftrightarrow f(v) < f(w)$. Nel caso in cui due nodi abbiamo lo stesso valore della f, si conserva l'ordine presente nella lista dei nodi che rappresenta $V(G)$. Da questo riordinamento dei nodi (che si osserva essere semplicemente opposto - e quindi equivalente - alla relazione d'ordine $>_G$ definita nel primo capitolo) si ottiene una naturale orientazione del grafo, in cui uno spigolo tra due nodi v e w, con $v \succ w$, è necessariamente (w, v).
Dopo questo passo preliminare l'algoritmo considera in modo sequenziale i nodi di G e, utilizzando di ognuno solamente le adiacenze che ha con i suoi precedenti, riesce ad individuare le possibili Δ^*-mosse di riduzione e a costruire così l'albero ridotto. Quando prende in considerazione il nodo n-esimo di G, stabilisce se, a causa delle mosse di riduzione, esso svanisce e quindi non compare nell'albero H o, al contrario, se ne costituisce un nodo: in tal caso lo aggiunge alla lista $V(H)$ e aggiorna opportunamente le sue adiacenze. In particolare, se il nodo non costituisce un nodo di H, vuol dire che svanisce per effetto di una qualche Δ^*-mossa. D'Amico rappresenta questo venir meno con la definizione di classi disgiunte di nodi di G che, tramite la scelta di un loro opportuno rappresentante, individuano un particolare nodo dell'albero H. I punti centrali dell'algoritmo sono la costruzione di tali classi e l'assegnamento degli archi di adiacenza ogni volta che si aggiunge un nodo ad H. Vediamo ora i dettagli.

Supponiamo come sopra che si stia considerando l'$(n+1)$-esimo nodo $g = g_{n+1}$ di V e che i precedenti nodi $V_n = \{g_1, \ldots, g_n\}$ di V siano divisi in classi disgiunte $c_1, \ldots, c_{k(n)}$. Indichiamo con C_n tale famiglia di classi. Si suppone inoltre che ad ogni classe c_i corrisponda un particolare rappresentante $h = h(c_i)$ che è un nodo dell'albero H che stiamo, passaggio dopo passaggio costruendo. Infine si deve assumere che H al momento sia un grafo H_n con insieme dei vertici $V(H_n)$ e degli archi $E(H_n)$. Si considera l'insieme E_{n+1} di tutti gli archi uscenti da g; per ciascuno di essi si considera il nodo di arrivo v e si identifica la classe a cui appartiene (unica perchè sono disgiunte). Ogni classe h_i trovata in questo modo si considera "segnata" in modo da non venir contata più volte. Dopo aver considerato tutti gli archi $e \in E_{n+1}$ si contano le classi trovate, siano esse $c^g_1, \ldots, c^g_{\bar{c}(n+1)}$ dove $\bar{c}(n+1)$ è il loro numero; allora si hanno i seguenti casi possibili:

$\bar{c}(n+1) = 0$ *nessuna classe.* Allora il nodo g diventa una classe a sé stante $c_{k(n)+1}$, di cui è l'unico elemento e rappresentante. A $V(H)$ si aggiunge un nodo $h = g$ da cui non escono archi, per cui $E(H)$ rimane invariato.

$\bar{c}(n+1) = 1$ *una sola classe.* Allora il nodo g diventa un elemento di questa classe, che mantiene il suo rappresentante. Non si aggiungono dunque né nodi né archi ad H.

$\bar{c}(n+1) > 1$ *più di una classe.* Allora si costruisce una classe c' costituita da g e dall'unione di tutte le classi c^g_i con $1 \leq i \leq \bar{c}(n+1)$, avente come rappresentante proprio g. Si aggiunge un nodo $h = g$ a $V(H_n)$ e, per ogni classe trovata si aggiunge ad $E(H_n)$ un arco da h ad $h(c^g_i)$, ovvero da g al rappresentate di c_i.

In formule si ha

$$\text{- } \bar{c}(n+1) = 0 \Rightarrow \begin{cases} C_{n+1} = C_n \cup \{g\} \\ h(\{g\}) = g \\ V(H_{n+1}) = V(H_n) \cup \{g\} \\ E(H_{n+1}) = E(H_n) \end{cases}$$

$$\text{- } \bar{c}(n+1) = 1 \Rightarrow \begin{cases} C_{n+1} = (C_n - c_1^g) \cup c' \\ c' = c_1^g \cup \{g\} \text{ dove } h(c') = h(c) \\ V(H_{n+1}) = V(H_n) \\ E(H_{n+1}) = E(H_n) \end{cases}$$

$$\text{- } \bar{c}(n+1) > 1 \Rightarrow \begin{cases} C_{n+1} = C_n - \left(c_1^g \cup \ldots \cup c_{\bar{c}(n+1)}^g\right) \cup c' \\ c' = \{g\} \cup c_1^g \cup \ldots \cup c_{\bar{c}(n+1)}^g \text{ dove } h(c') = g \\ V(H_{n+1}) = V(H_n) \cup \{g\} \\ E(H_{n+1}) = E(H_n) \cup (g, h(c_1)) \cup \ldots \cup (g, h(c_{\bar{c}(n+1)})) \end{cases}$$

Alla fine dell'algoritmo si trova un albero H_N che è proprio l'albero H cercato.

Cerchiamo ora di capire perchè facciamo agire l'algoritmo in questo modo, tornando alla definizione delle Δ^*-mosse di riduzione. Consideriamo quella di tipo 3: se si hanno due nodi v_1 e v_2 con delle particolari caratteristiche che dipendono dal valore della funzione di taglia (v_2 unico figlio di v_1), si cancella v_1 e si trasformano gli archi uscenti da questo in archi uscenti da v_2 (se non ci sono già), in cui l'altro estremo rimane invariato. Definiamo dunque una relazione $\wr$ sui nodi di G in cui $v \wr w$ se soddisfano (nell'ordine che si preferisce) le ipotesi della Δ^*-mossa di tipo 3. Supponiamo inoltre, in modo naturale, che $v \wr v$ per ogni $v \in V(G)$. Infine si suppone che, se applichiamo due di queste mosse coinvolgendo entrambe uno stesso nodo u, allora gli altri due nodi coinvolti sono in $\wr$-relazione con u e tra loro. Ovvero assumiamo delle ipotesi secondo le quali la $\wr$-relazione è una relazione di equivalenza in V. Ora quando nell'algoritmo di D'Amico consideriamo le classi disgiunte di nodi di V, per come queste sono costruite (vedi il caso $\bar{c}(n+1) = 1$), si possono in

realtà considerare come classi disgiunte di $\wr$-classi di equivalenza di nodi di V. Inoltre, se supponiamo per il momento che nella riduzione del grafo G non si effettui mai una Δ^*-mossa di tipo 2 (è sufficiente supporre che non ci siano due nodi in cui la f assume lo stesso valore), allora, poichè le uniche cancellazioni di nodi avvengono per le Δ^*-mosse di tipo 3, si ha che ad ogni $\wr$-classe di equivalenza corrisponde un unico nodo di H, e quindi si possono interpretare, ad ogni passo, le classi $c_i \in C_n$ come classi disgiunte di nodi di H. Nel seguito della spiegazione dunque diremo impropriamente che gli elementi di tali classi sono elementi di H, intendendo con questa espressione le $\wr$-classi di equivalenza. Quello che abbiamo appena mostrato è che il caso $\bar{c}(n+1) = 1$ dell'algoritmo implementa le Δ^*-mosse di tipo 3 e ci permette di individuare i nodi di G che diventano nodi di H.
Se invece si analizzano gli altri due casi previsti in precedenza (in pratica $\bar{c}(n+1) \neq 1$) si ricava che ad ogni passaggio dell'algoritmo c'è una corrispondenza biunivoca tra le classi di nodi di H_n e le sue componenti connesse. Ovvero gli altri due casi ci dicono che tali classi individuano proprio le componenti connesse in cui è diviso il grafo H_n. Diamo la dimostrazione per induzione sulle iterazioni dell'algoritmo. Il caso base ($n = 0$) prevede che, quando si inserisce il primo nodo $g = g_1$ di G, si abbia necessariamente $\bar{c}(1) = 0$, dunque da un lato g diventa il primo nodo di H, da cui $V(H_1) = \{g\}$ e $E(H_1) = \emptyset$; dall'altro si crea la prima classe $c_1^g = \{g\}$ in modo che $C_1 = \{c_1^g\}$. La corrispondenza binunivoca tra le classi disgiunte e le componenti connesse è ovvia. Nel caso induttivo si suppone che prima di effettuare il passo $n+1$ ci sia tale corrispondenza biunivoca. Allora, quando di considera il nuovo nodo $g = g_{n+1}$ di G, si ha: se $\bar{c}(n+1) = 1$ non si aggiungono né nodi né classi quindi la corrispondenza rimane; se $\bar{c}(n+1) = 0$, allora si aggiunge un nodo non connesso con gli altri (una componente connessa in più) a cui corrisponde biunivocamente l'aggiunta in C_{n+1} della classe $c' = \{g\}$; infine se $\bar{c}(n+1) > 1$ allora si costruisce una unica classe fatta dalle classi c_i^g e dall'elemento g, ma in parallelo si aggiunge il nodo g ad H_n e lo si connette ad uno dei nodi contenuti in ciascuna di quelle classi, creando

così anche una unica componente connessa. Dunque per il procedimento induttivo, ad ogni passo dell'algoritmo c'è una corrispondenza biunivoca tra le classi c_i e le componenti connesse di H_n.
La dimostrazione ci permette di dire molto di più: il rappresentante $h = h(c_1)$ di ogni classe disgiunta è il nodo di tale classe con valore maggiore della funzione misurante. Infatti ogni volta che si aggiunge un nodo $g = g_{n+1}$, su di questo la f ha valore maggiore che sui precedenti. Se siamo nel caso $\bar{c}(n+1) = 0$ si costituisce una nuova classe di cui g è l'unico elemento e rappresentante, quindi la condizione da verificare è vuota. Nel caso $\bar{c}(n+1) > 1$ invece, come abbiamo visto, la nuova classe c', costituita dall'unione delle classi c_i^g e da g stesso, ha proprio g come rappresentante e quindi $f(g') < f(g)$ per ogni $g' \in c'$.
Questo risultato è molto importante perchè, per come è fatto l'algoritmo, implica che ad ogni passo le classi c_i (che sono componenti connesse di H_n) siano arborescenze. La dimostrazione segue ancora una volta per induzione. Nel caso base, dopo aver inserito $g = g_1$, si ha che H_1 è costituito da un solo nodo (g appunto) e quindi è ovviamente una arborescenza. Se poi si suppone che, prima di aggiungere l'elemento $g = g_{n+1}$, le classi $c_i \in C_n$ siano aroborescenze, allora, per quanto dimostrato, il loro rappresentante è la loro radice. Nel caso in cui $\bar{c}(n+1) \leq 1$ le arborescenze restano tali, creandone eventualmente una nuova se $\bar{c}(n+1) = 0$. Mentre nel caso $\bar{c}(n+1) > 1$, si connette il nuovo nodo g, che ha il valore massimo della f, alle radici di alcune arborescenze, generando appunto una unica nuova arborescenza che ha proprio g per radice. Ancora una volta la tesi segue dal procedimento induttivo.
Quello che è interessante, oltre al risultato di ottenere una particolare struttura (che poi ci faciliterà nel calcolo della funzione di taglia), è che in questa dimostrazione si nasconde il modo in cui l'algoritmo implementa le Δ^*-mosse di tipo 1. Tutto sta nella scelta dell'individuazione delle classi c_i^g e del loro rappresentante. Infatti ad ogni arco uscente da g si fa corrispondere una classe c_i e a tale classe a sua volta si associa l'arco da g alla sua radice. In

pratica si fa 'risalire' la destinazine degli archi uscenti da g fino ai nodi con valore massimo della f, cioè proprio quello che fanno le Δ^* mosse di tipo 1. Dunque a questo punto abbiamo visto come l'algoritmo implementa le Δ^*-mosse di tipo 1 e 3, supponendo che non si dovessero mai applicare quelle di tipo 2. Togliendo questa ipotesi, queste mosse sono implementate nell'algoritmo di D'Amico mediante una modifica al caso $\bar{c}(n+1) > 1$. Ovvero, dopo aver verificato che le classi c_i^g sono più di una e aver creato il nuovo nodo g, al momento dell'assegnazione dei nuovi archi (g, h_i), con $h_i = h(c_i^g)$, si verifica se il valore della funzione misurante in h_i è uguale o strettamente minore di $f(g)$; nel secondo caso l'algoritmo opera come è stato detto, se invece $f(g) = f(h_i)$ allora si cancella il nodo h_i e si trasformano tutti gli archi da esso uscenti in archi che partono da g. Si vede facilemente come in questo modo si implementino le Δ^*-mosse di tipo 2.

È a questo punto che abbiamo ritenuto indispensabile introdurre la modifica all'algoritmo descritto da D'Amico. Per spiegare la necessità della modifica presentiamo brevemente un esempio. Supponiamo che G sia un grafo di taglia molto semplice, dato da tre nodi v, v_1, v_2, con $f(v) = 0$ e $f(v_1) = f(v_2) = 1$. Inoltre supponiamo che gli archi di G siano $e = (v_2, v)$ ed $e' = (v_2, v_1)$. Allora si vede facilemente che la Δ^*-riduzione totale di G è il grafo di taglia H costituito dal solo nodo v. Se si applica l'algoritmo di D'Amico però non si trova questo risultato (in figura si trovano i due diversi risultati).

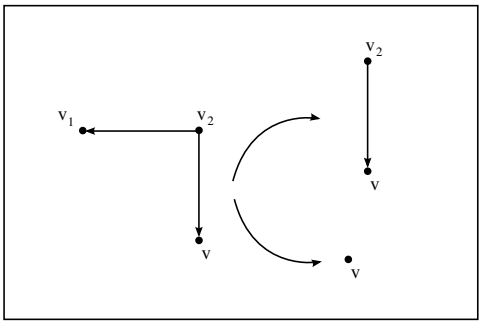


Figura 3.2: L'errore nell'algoritmo di D'Amico.

Infatti al momento dell'inserimento del terzo nodo v_2 si ha che $C_2 = \{\{v\}, \{v_1\}\}$, mentre H_2 è costituito dai soli due nodi sconnessi v e v_1. Gli archi uscenti da v_2 sono appunto e ed e' ed individuano le due classi disgiunte $\{v\}$ e $\{v_1\}$ che hanno gli elementi stessi per rappresentanti. Poichè $\bar{c}(3) = 2 > 1$ si aggiunge ad H il nodo v_2, poi si costruisce l'arco e, infine, poichè $f(v_1) = f(v_2)$ allora si cancella v_1 da H. A questo punto, non avendo v_1 archi uscenti e non essendoci altri nodi in G, l'algoritmo termina. Il risultato è un grafo con due nodi v e v_2 connessi dall'arco e, che è diverso dalla Δ^*-riduzione totale di G. In particolare questa differenza non è significativa da un punto di vista del risultato, perchè le funzioni di taglia dei due grafi sono uguali, ma produce dei problemi al momento della seconda parte dell'algoritmo, cioè appunto quello del calcolo della funzione di taglia, in quanto ci si ritrova con un grafo diverso da quello atteso.

In generale i casi in cui l'algoritmo di D'Amico dà problemi sono quelli in cui, dato il nodo $g = g_{n+1}$, si ritrova un numero m di classi c_i, con rappresentante h_i che non ha figli in H_n e su cui la f assume lo stesso valore che in g, ed una sola altra classe c con rappresentante h tale che $f(h) < f(g)$. La soluzione che abbiamo trovato è la seguente: al momento in cui si vanno ad individuare le classi c_i, se sul rappresentante h_i la f vale come su g e h_i non ha figli in H_n allora non si conta come classe, ovvero non si incrementa il valore di $\bar{c}(n+1)$ e si inseriscono g ed h_i nella stessa classe, mantenendo h_i come rappresentante. A questo punto, dopo aver calcolato il valore di $\bar{c}(n+1)$, questi particolari h_i, che non hanno individuato una classe c_i^g, non vengono trattati e d'altra parte, essendo nella stessa classe di g, si comportano esattamente come se fossero tutti coincidenti in esso.

Con questa modifica l'algoritmo che abbiamo ottenuto produce in tutti i casi come output un particolare grafo di taglia, ovvero una foresta di arborescenze e permette dunque il calcolo della funzione di taglia mediante la seconda parte dell'algoritmo che ora analizzeremo. Prima però ci sembra importante ricordare che, a livello implementativo, lo sviluppo di questa parte comporta la risoluzione di alcuni problemi davvero interessanti, soprattutto legati

alla gestione insiemistica delle classi disgiunte, per i quali rimandiamo a [2]. Nell'appendice C descriviamo un esempio di Δ^* riduzione di una funzione misurante con questo algoritmo.

3.3.2 Il calcolo della funzione di taglia

Una volta ottenuta la foresta di arborescenze, il calcolo della funzione di taglia è molto semplice e oltretutto ha la particolarità che si può fare separatamente sulle diverse componenti connesse. Ovvero se H_α con α che varia in un opportuno insieme di indici sono le arborescenze di H allora in ogni punto (x, y) del semipiano Δ si ha:

$$\ell_H(x,y) = \ell_{\bigcup_\alpha H_\alpha}(x,y) = \sum_\alpha \ell_{H_\alpha}(x,y)$$

Dunque nel seguito dell'esposizione faremo l'ipotesi, non riduttiva, che H sia connesso e quindi una arborescenza; inoltre, d'ora in poi, in questa parte lo indicheremo con T (dall'inglese *tree*), anche per coerenza con la notazione usata nel nostro software. L'arborescenza trovata, infine, ha anche un'altra caratteristica importante, ovvero non ha nodi cosiddetti "interlocutori" cioè nodi che hanno un solo figlio. Un nodo di T o è una foglia e quindi non ha figli, oppure concorre alla "ramificazione" dell'albero avendo almeno due figli.
L'algoritmo procede in modo iterativo: se si suppone che all'inizio si abbia $T = T_0$, ad ogni passo generico n-esimo si parte da T_n arborescenza e, dopo aver effettuato dei calcoli che determinano parte della funzione di taglia ℓ_H, si crea una nuova arborescenza T_{n+1} che ha un numero di nodi minore della precedente (anch'essa priva di nodi interlocutori). L'algoritmo termina quando non ci sono più nodi: in quel momento la funzione trovata è esattamente $\ell_T = \ell_H = \ell_G$ e quindi abbiamo raggiunto lo scopo, calcolare la funzione di taglia del grafo di taglia che discretizza l'immagine di partenza. Seguendo l'esposizione che abbiamo fatto nel primo capitolo, la funzione di taglia viene calcolata e rappresentata come successione (lista ordinata) di punti-linee an-

golari[7]. Ogni passaggio dell'algoritmo comporta l'inserimento di un nuovo punto angolare l_{n+1} nella successione $L = (l_i)_{1 \leq i \leq n}$ che rappresenta la funzione ℓ_T.
Vediamo ora nel dettaglio l'algoritmo di calcolo. Ad ogni passo si considera il nodo v_n di T_n che risulta essere la foglia su cui la f è massima. Il fatto che T_n sia una arborescenza senza nodi interlocutori, comporta che localmente T_n vicino a v_n può assumere solo un numero limitato di configurazioni distinte: v_n non ha un padre e dunque è anche la radice di T_n, che non ha altri nodi; v_n ha il padre e un solo fratello (su cui necessariamente la f assume valore minore); v_n ha il padre e più di un fratello (su cui necessariamente la f assume valori minori). Vediamo come si comporta l'algoritmo a seconda dei casi:

1. v_n è radice di T_n. Allora si aggiunge ad L la retta angolare rappresentata da $l_{n+1} = (f(v_n), \infty)$. Si cancella il nodo v_n e quindi, non essendocene altri, l'algoritmo termina.

2. il padre di v_n ha solo un altro figlio w, con $f(w) \leq f(v_n)$. Sia v il padre di v_n (necessariamente $f(v_n) < f(v)$). Allora si aggiunge ad L il punto angolare rappresentato da $l_{n+1} = (f(v_n), f(v))$. Si ottiene T_{n+1} cancellando il nodo v_n, dunque si ha che v ha un solo figlio w. Per evitare questo si cancella anche il nodo v e quindi w diventa figlio del padre di v, se esiste, o in caso contrario costituisce l'unico nodo di T_{n+1}[8].

3. il padre di v_n ha più di un figlio. Indichiamo con w il padre di v_n. Allora si aggiunge ad L il punto angolare rappresentato da $l_{n+1} =$

[7] D'ora in poi per non appesantire il discorso useremo sempre solo l'espressione *punti angolari*, o addirituttura quando sarà chiaro il contesto, solo il termine *punti*, sottintendendo il fatto che potrebbero essere eventualmente linee angolari.

[8] Dal momento che l'albero in partenza non ha nodi interlocutori, se v ha un padre allora ha anche dei fratelli, perciò quando lo si cancella è come se al suo posto si mettesse w, che avrà dunque un padre e dei fratelli, non risultando un nodo interlocutorio.

$(f(v_n), f(w))$. Si cancella il nodo v_n e quello che rimane è l'arborescenza T_{n+1}.

Il motivo per cui l'algoritmo opera in questo modo segue facilmente dalla definizione di funzione di taglia e dalla sua rappresentazione nel semipiano Δ mediante una successione di punti-linee angolari. Partiamo infatti dalla configurazione generica della foglia v_n al passo n dell'algoritmo. Per la particolarità di T_n di essere una arborescenza si vede che è sufficiente considerare solo la configurazione locale della foglia, ovvero il padre e i fratelli (si confronti la figura). Sia $\bar{x} = f(v_n)$ e $\bar{y} = f(w)$ dove w è il padre di v_n. Allora è chiaro che quale che sia il valore della funzione di taglia in un punto (x, y) con $x < \bar{x}$ e $\bar{x} \leq y < \bar{y}$ se si passa ad un altro punto (x', y) con $x' > \bar{x}$ il numero di componenti connesse di $(T_n)_{f \leq y}$ aumenta di uno perchè si prende in considerazione anche il vertice v_n. E allo stesso modo se si considera (x, y) con $y < \bar{y}$ e $\bar{x} \leq x < \bar{y}$, passando poi ad un altro punto (x, y') con $y' \geq \bar{y}$ il numero di componenti connesse cala del numero di fratelli di v perchè potendo includere anche il padre w si connette v_n ai suoi fratelli "minori". In tutto questo, non abbiamo detto altro che $(f(v), f(w))$ è un punto angolare (confronta la figura), per questo lo si aggiunge alla lista.

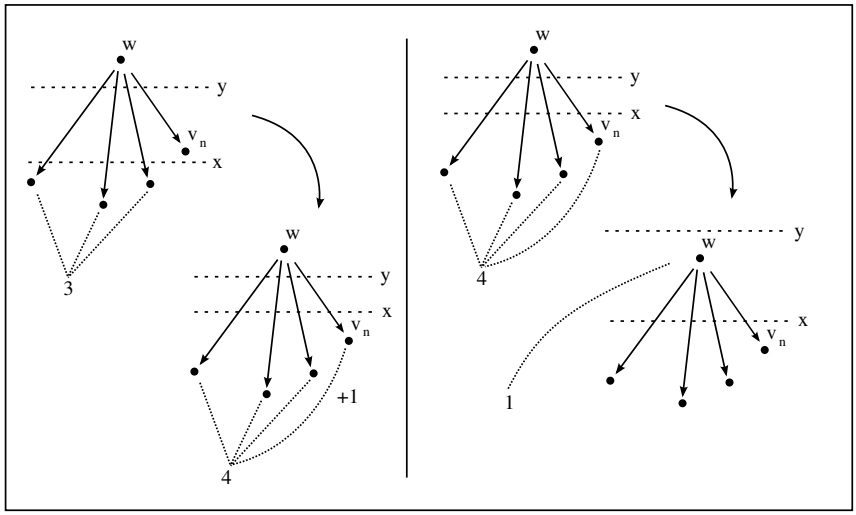


Figura 3.3: La struttura di grafo che induce un punto angolare.

L'algoritmo poi elimina il nodo v_n e riprende. L'opzione particolare in cui v_n ha un solo fratello viene distinta solo perchè poi si deve effettuare una Δ^*-mossa di tipo 3 prima di riprendere l'algoritmo. Vediamo invece il caso in cui v_n sia radice. Allora preso il punto (x, y), se $x < \bar{x} = f(v)$ allora quale che sia y il valore della funzione non cambia. E lo stesso vale se $\bar{x} < x$. Al contrario se si suppone $y \geq \bar{x}$, il passaggio da un $x < \bar{x}$ ad un $x' \geq \bar{x}$ ovviamente il numero di componenti connesse cresce. Ovvero si ha una retta angolare $(f(v), \infty)$. Quello che abbiamo visto è che ad ogni passaggio la funzione di taglia dell'arborescenza T_n in un punto (x, y) si può scrivere in modo ricorsivo come:

$$\ell_{T_n}(x, y) = \chi_{T_{(f(v_n), f(w))}}(x, y) + \ell_{T_{n+1}}(x, y)$$

dove χ_{T_P} segue la notazione del primo capitolo, con $P \in \Delta^*$ e se v_n non ha padre w allora si intende $f(w) = \infty$. Poichè l'algoritmo fa calare in modo progressivo il numero di nodi di T_n fino ad annullarlo la ricorsione del calcolo ha termine.

Concludiamo questa parte solo facendo notare che questo algoritmo è scritto e pensato per ricevere in input arborescenze senza nodi interlocutori; d'altra parte con il suo procedere si possono creare delle situazioni in cui si generano tali nodi interlocutori e, per eliminarli, è necessario operare una Δ^*-mossa di tipo 3 (nel caso di v_n con un solo fratello). È possibile dunque che a livello implementativo si imponga, prima di ogni iterazione, il controllo nodo per nodo con l'eventuale esecuzione di questa mossa di riduzione. Se così si facesse, la presenza di nodi interlocutori in un qualsiasi punto dell'arborescenza data in input a questa parte dell'algoritmo, non darebbe problemi in quanto essi verrebbero eliminati al primo passaggio. Questa potrebbe essere la spiegazione del motivo per cui, benchè l'algoritmo di D'Amico dia la possibilità di ottenere delle arborescenza con nodi interlocutori, il software da lui implementato per le immagini bitmap non ha dato problemi di calcolo e quindi si possono considerare validi i risultati ottenuti nei suoi esperimenti. Da parte nostra abbiamo preferito che il nostro algoritmo rispettasse la

descrizione che ne abbiamo dato, implementando nella sua seconda parte la Δ^*-riduzione di tipo 3, ma mirata solo ai casi previsti (quelli dei due fratelli), e non effettuando ad ogni iterazione un controllo nodo per nodo. Ovviamente questa scelta, o l'opposta, non ha conseguenze sui risultati degli esperimenti effettuati, così come non ne ha avuti nel casi di D'Amico, ma ci è sembrato che fosse più coerente con l'impostazione che avevamo dato all'intero algoritmo.

3.4 Parte D

Fino a questo punto abbiamo esposto in che modo, data in input una immagine in formato SVG, riusciamo a calcolarne la funzione di taglia. Il nostro scopo, come abbiamo detto, è utilizzare tali funzioni come strumento di confronto tra le immagini: dobbiamo dunque trovare un indicatore che ci dica quanto due funzioni di taglia sono differenti. Lo strumento matematico più naturale per dare questo tipo di informazioni è costituito dalle distanze ed in questo capitolo analizzeremo quelle che abbiamo studiato ed utilizzato. Prima però diamo una descrizione di come abbiamo impostato il software per poter effettuare delle sperimentazioni, cioè, una volta calcolate le funzioni di taglia, come abbiamo effettuato i confronti e come abbiamo cercato di rendere leggibili i dati ricavati.

Si suppone al momento che abbiamo indentificato una opportuna distanza d[9]. Supponiamo pure che sia stato individuato un insieme di immagini da testare con il nostro software[10], siano esse date dai file `img1.svg`, ..., `imgN.svg`. Il software è organizzato in due cicli sulle immagini.

[9]Tutta la spiegazione che segue potrebbe essere generalizzata al caso con n distanze, complicando la notazione, ma permettendo più possibili combinazioni di normalizzazioni, medie, ecc., e dunque una elaborazione più complessa ma probabilmente più significativa dei risultati. Non avendo implementato nel nostro software che una sola distanza, abbiamo ritenuto di non presentare una esposizione inutilmente più complicata e qundi abbiamo preferito la notazione con una sola distanza.

[10]Sulla scelta dell'insieme di test ci soffermeremo nel prossimo capitolo.

Il primo ciclo prende in considerazione una per una tutte le immagini e ne calcola la funzione di taglia utilizzando l'algoritmo descritto. Ciascuna funzione viene salvata in un opportuno file formato SZF creato da noi. Si tratta di un file di testo in cui sono scritte in colonna le coppie di coordinate che individuano i punti angolari della funzione di taglia. Se ne può vedere un esempio nell'appendice D. Se l'immagine è `imgK.svg` il file viene chiamato per comodità `imgK.szf` e viene salvato in una cartella separata (`size_funs`). Alla fine del primo ciclo si sono calcolate e memorizzate le funzioni di taglia di ogni immagine. In particolare se si è deciso di utilizzare un numero m di funzioni misuranti $\phi_1, \ldots, \phi_m$, ad ogni immagine corrisponderanno m funzioni di taglia, divise ciascuna in una cartella diversa per evitare confusione. Usiamo qui la notazione $\ell_K^{(i)}$ per indicare la funzione di taglia dell'immagine `imgK.svg` relativa alla funzione misurante ϕ_i.
Il secondo ciclo è triplo ovvero:

- *ciclo in* $1 \leq K \leq N$. Prende in considerazione una per una tutte le immagini. Ad ogni ciclo l'immagine considerata è la cosiddetta immagine *modello*.

- *ciclo in* $1 \leq i \leq m$. Prende in considerazione una per una tutte le funzioni di taglia relative alle diverse funzioni misuranti.

- ciclo in $1 \leq J \leq N$. Prende di nuovo in considerazione una per una tutte le immagini. Ad ogni ciclo l'immagine considerata è la cosiddetta immagine *test* (che può coincidere con l'immagine modello).

Dentro quest'ultimo ciclo si calcola $d_{KJ}^{(i)} = d(\ell_K^{(i)}, \ell_J^{(i)})$ ovvero la distanza tra le funzioni di taglia del modello K e del test J, entrambe relative alla funzione misurante ϕ_i. In pratica si ottiene una matrice $d = d_{KJ}^{(i)}$ tridimensionale $(N \times m \times N)$ che contiene tutte le informazioni che ci interessano: questa matrice è il risultato del nostro esperimento.
A questo punto si possono implementare i modi più vari per dare una maggiore leggibilità ai risultati. Noi ne abbiamo scelti in particolare due: la distanza *normalizzata* e la distanza *media*. La prima, per ogni "faccia" (cioè

fissato i), cerca il valore massimo (max_i) contenuto in essa. Questo rappresenta il diametro del nostro insieme di funzioni di taglia. Con questo valore si normalizzano le componenti della matrice-faccia, ovvero tutti i valori vengono divisi per max_i. A questo punto per ogni K e J fissato si sommano in i gli m valori delle distanza tra le funzioni di taglia $\ell_K^{(i)}$ e $\ell_J^{(i)}$. Si ottiene così una nuova matrice d_{KJ}^{norm}. Questa è la matrice delle distanze normalizzate. Un altro modo è calcolare per ogni K e J fissati la distanza media sulle diverse funzioni misuranti, si genera così la matrice delle distanze medie $d_{KJ}^{media} = \frac{1}{m}\left(d_{KJ}^{(1)} + \ldots + d_{KJ}^{(m)}\right)$.
A questo punto, quale che sia la matrice calcolata, che indichiamo genericamente con d_{KJ}, per renderla più leggibile la si divide in righe. Per ogni K fissato (ovvero si fissa un modello, da cui il nome dato alle immagini del ciclo in K) si considera solo il vettore riga $v_J^K = d_{KJ}$ e lo si ordina in modo crescente. Fatto questo, sempre per ogni K fissato, si creano due file di output. Il primo è `imgK.res` che è un file di testo in cui compaiono in colonna uno sotto l'altro i nomi di tutte le immagini `imgJ.svg`, elencati per l'appunto in ordine crescente rispetto al vettore v_J^K delle distanze, con a fianco l'indicazione della loro distanza dal modello. Questo file è molto utile per vedere l'omogeneità o meno dell'insieme di test che abbiamo considerato, in relazione alle caratteristiche dell'immagine che abbiamo scelto da modello. Ovvero, supponiamo per esempio di aver scelto la matrice delle distanze normalizzate d_{KJ}^{norm}. Se la maggior parte dei valori è abbastanza prossima a m vuol dire che le immagini dell'insieme sono nel complesso diverse da quella presa a modello, e che questa quindi non rappresenta un "buon modello" dell'insieme, in quanto si differenzia dalla gran parte degli elementi. Al contrario se si ritrovano molti valori bassi, significa che l'immagine ne ha molte altre a lei prossime e quindi rappresenta un "buon modello" per l'insieme.
Il secondo file di output serve invece per cercare di capire quali caratteristiche grafiche sono significative se lette dal punto di vista delle fuzioni di taglia. Infatti, sempre a partire dal vettore riordinato in modo cresente v_J^K, si crea un file `imgK.RES.svg` che è una immagine in SVG. Tale immagine è imposta-

ta su due colonne: nella prima compare N volte l'immagine `imgK.svg`, nella seconda, una sotto l'altra, ci sono le immagini `imgJ.svg` incolonnate secondo l'ordine dato dal vettore v_J^K. Dunque ad ogni riga si ha il confronto tra il modello e il test, e, man mano che si scende, le due immagini diventano sempre più diverse (se lette attraverso le funzioni di taglia), in modo che si può cercare di distinguere cosa, a livello grafico, causa questo allontanamento.
Con questo terminiamo la spiegazione di come viene impostato un esperimento con il nostro software e di come i risultati che si ottengono vengono elaborati. L'impostazione è ancora ad un primissimo livello, probabilmente un po' superficiale, ma è un primo passo per cercare di ricavarne delle informazioni interessanti sulle funzioni di taglia e il loro utilizzo sulle immagini vettoriali (cioè appunto lo scopo della nostra tesi). Descriveremo nel prossimo capitolo un esperimento che abbiamo effettuato, esibendo i risultati ottenuti e cercando di darne una interpretazione. Passiamo però prima ad analizzare la questione delle distanze tra funzioni di taglia che abbiamo lasciato in sospeso.

L'idea del problema è trovare una distanza sull'insieme delle funzioni di taglia. Ovviamente non è per nulla banale, dato un insieme di oggetti qualsiasi, trovare su di esso la necessaria struttura per definire su di esso una distanza che abbia un qualche significato. Il pregio delle funzioni di taglia è che sono un oggetto matematico molto duttile, che si presta cioè a molte possibili elaborazioni. Per esempio si possono considerare le funzioni di taglia come un particolare sottoinsieme di $L^\infty(\Delta)$, che è uno spazio di Banach, su cui quindi si può indurre una distanza a partire dalla norma. Un caso più generale, supponendo che la distanza tra due funzioni possa anche essere infinita, ovvero che si consideri una ∞-metrica o metrica estesa, è quello di calcolare la distanza come differenza della norma delle due funzioni in $L^p(\Delta)$ per ogni $p \in [1, \infty]$.
Una trattazione molto approfondita delle possibili distanze da applicare all'insieme delle funzioni di taglia si ritrova sempre in [7]. In particolare è interessante la definizione di una classe molto generale di distanze (in realtà

pseudo-quasi metriche estese) da cui si possono dedurre tutti gli esempi che faremo[11]. Per i dettagli teorici rimandiamo quindi alla pubblicazione citata, mentre ci soffermiamo sulle sole distanze da noi considerate ed implementate. Il punto di partenza della nostra analisi è la rappresentazione delle funzioni di taglia come successioni di punti del piano. Da quanto spiegato nel primo capitolo, sappiamo infatti che, dato un grafo di taglia G (che ricaviamo dalla discretizzazione dell'immagine), possiamo scrivere la sua funzione di taglia come

$$\ell_G = \sum_{i=1}^{m} \chi_{T_{Q_i}}$$

dove i T_{Q_i} sono degli opportuni triangoli del piano (eventualmente di area infinita) identificati da dei punti $Q_i \in \Delta^*$, dove

$$\Delta^* = \{(x, y) \in \mathbb{R} \times (\mathbb{R} \cup \{+\infty\} \ : \ x < y\}$$

Sappiamo inoltre che le successioni di questi punti sono sempre finite e la rappresentazione tramite esse è unica a meno di permutazioni. Il problema del calcolo della distanza si può quindi spostare da quello tra le funzioni di taglia a quello tra le successioni di punti che le rappresentano. In questo modo si semplifica notevolmente il calcolo, anche dal punto di vista computazionale, in quanto si ha a che fare solo con insiemi finiti di punti. Una particolare attenzione si deve fare alla molteplicità, ovvero al numero di volte in cui uno stesso punto compare nella funzione misurante. Come abbiamo sottolineato in precedenza, la successione di punti angolari che rappresenta una funzione di taglia non è data da punti necessariamente tutti distinti. Ovvero la successione $((0, \infty), (0, 1))$ è diversa dalla successione $((0, \infty), (0, 1), (0, 1))$, anche se hanno la stessa immagine (o sostegno) in $\mathbb{R}^2$, in quanto esse rappresentano funzioni diverse. Dunque il passaggio da distanza tra le successioni di punti con molteplicità a quella tra gli insiemi di punti che sono sostegno delle successioni stesse, comporta una semplificazione del problema. D'altra parte,

[11] Anche le distanze naturali di taglia definite nel primo capitolo rientrano in questa classe.

come vedremo nel prossimo capitolo analizzando i risultati di un esperimento che abbiamo effettuato, implica anche una perdita di informazione: una distanza tra insiemi induce non una distanza, ma una pseudodistanza tra e funzioni di taglia (cfr. il capitolo 4).
Ora, tralasciando questo problema, ci si può soffermare sulle sole distanze tra sottoinsiemi di $\mathbb{R}^2$. La nostra prima scelta è stata per la distanza di Hausdorff. Vediamo come è definita:

Definizione 3.1. Siano A e B due sottoinsiemi compatti di $\mathbb{R}^n$. Sia δ una qualsiasi metrica su $\mathbb{R}^n$. Si pone per definizione che la distanza di Hausdorff $d_\delta(A, B)$ tra gli insiemi A e B, relativa alla metrica δ è data da:

$$d_\delta(a, B) := \max\left\{\sup_{a\in A}\left(\inf_{b\in B}\delta(a,b)\right), \sup_{b\in B}\left(\inf_{a\in A}\delta(b,a)\right)\right\}$$

Si osserva che, poichè A e B sono compatti, gli inf e i sup della definizione possono essere sostituiti da dei minimi e massimi. Ovvero esistono due particolari punti $\bar{a}$ e $\bar{b}$ che verificano $\delta(\bar{a}, \bar{b}) = d_\delta(A, B)$.

L'implementazione del calcolo di questa misura, poichè possiamo considerare la definizione con i *min* e *max* al posto degli estremi superiore e inferiore, è stata abbastanza semplice. Si è definita una opportuna struttura dati per memorizzare i punti angolari delle funzioni di taglia e quest'ultime sono state memorizzate come liste di nodi definiti con tali strutture. Utilizzando le liste la distinzione fra punti con uguali coordinate è risultata immediata, associando a tali punti tanti nodi quanta è la loro molteplicità. In seguito il calcolo è stato fatto esattamente seguendo la definizione, ovvero con un doppio ciclo: quello più esterno sui nodi della prima funzione di taglia (sia essa ℓ_1 ed essi P_i) e quello interno sui nodi della seconda (ℓ_2 con punti Q_j). Nel ciclo più interno, dove si considera un P_i fissato, vengono calcolate sequenzialmente le distanze $d_{ij} := \delta(P_i, Q_j)$ per ogni j, dove si è scelto che la distanza δ fosse la metrica euclidea del piano. A questo punto, in parallelo al calcolo delle distanze d_{ij}, poichè queste vengono calcolate sequenzialmente, si identifica con dei confronti il loro minimo ($d_i = \min_j d_{ij}$). Allo stesso

modo, nel ciclo esterno, si confrontano sequenzialmente i d_i e se ne cerca il massimo. Alla fine dei due cicli si è trovato $d^{1,2} = \max_i d_i = d_\delta^{1,2}(\ell_1, \ell_2)$. Rifacendo lo stesso procedimento, ma con le due successioni invertite di posizione, si trova $d^{2,1} = d_\delta^{2,1}(\ell_2, \ell_1)$. Il massimo tra questi due valori è la distanza di Hausdorff cercata. L'unica complicazione di questo algoritmo è la possibilità, che si ha nel caso in cui uno solo tra P_i e Q_j sia una retta angolare, che la distanza tra i due punti $\delta(P_i, Q_j)$ sia infinita. Questo perchè nel caso in cui entrambi sono rette angolari si è deciso di porre per definizione $\delta\left((x_1, \infty), (x_2, \infty)\right) = |x_1 - x_2|$, che risulta quindi finita. La possibilità di avere distanza infinita tra due punti delle nostre successioni, rende la nostra distanza una pseudo-metrica e ci impone a livello implementativo di creare una opportuna struttura dati, per rappresentare i reali estesi (cioè i reali eventualmente infiniti). Quella che abbiamo creato si chiama `ext_real_t` e possiede due campi: il primo `is_infty` è un intero di tipo booleano (0 il numero reale è finito, 1 il numero reale è infinito); il secondo `real` è una variabile di tipo *double*, che si utilizza solo se la prima è pari a 0, e in tal caso è il valore reale finito da rappresentare (ovviamente troncato secondo la precisione dei *double*). Fatto questo, tutti i confronti tra numeri per individuare i minini e i massimi si sono dovuti implementare tenendo conto della possibilità che le distanze potessere essere infinite. Nonostante questa possibilità, vedremo nel prossimo capitolo, analizzando l'esperimento da noi effettuato, che in realtà per i particolari tipi di funzioni misuranti scelte, le distanza tra qualsiasi coppia di immagini testate risulterà finita. Non per questo, però, è stato possibile non implementare l'opzione che le distanze risultassero infinite, caratteristica comunque generale e utile per eventuali sviluppi futuri del software.

Prima di passare al prossimo capitolo, descriviamo brevemente un altro tipo di distanza: la distanza di Matching. Questa distanza è ampiamente utilizzata in letteratura, e sembra particolarmente adatta per il confronto delle fuzioni di taglia. Non è stata implementata, ma, a nostro parere, è una di

quelle estensioni da prevedere per il nostro software in quanto contribuirebbe a rendere gli esperimenti da effettuare più significativi. Infatti vogliamo sottolineare come, al di là di tutto il discorso teorico sulle funzioni di taglia, il valore effettivo a livello sperimentale della teoria della taglia, dipende in gran parte dall'individuazione di una metrica che renda "visibili" le indicazioni di somiglianza e dissimiglianza a cui esse sono sensibili.

Per meglio capire la distanza di matching è necessario osservare preliminarmente come si manifesta, a livello delle funzioni di taglia, la presenza del cosiddetto 'rumore' in una immagine qualsiasi. In generale si definiscono *rumore*, tutte le piccole oscillazioni intorno ad un valore, ipoteticamente supposto corretto, assunto da una qualsiasi variabile: nel nostro caso dalle primitive grafiche che descrivono l'immagine e dai valori della funzione misurante. Queste piccole variazioni provocano un moltiplicarsi di valori critici alternati (massimi e minimi relativi) molto vicini come valori. A livello delle funzioni di taglia, tale effetto si manifesta tramite tutta una serie di punti angolari che vengono a crearsi vicino alla diagonale (cioè punti in cui la x e la y sono molto vicine). Dunque, ragionando a posteriori, in generale si interpretano i punti vicino alla diagonale come un effetto dovuto al rumore, e quindi si attribuisce loro un "peso", almeno a livello interpretativo, minore. In pratica un punto vicino alla diagonale si può in qualche modo trascurare[12]. Vediamo come questo ci torna utile per la distanza di matching. Per prima cosa supponiamo che siano date due funzioni di taglia ℓ_1 e ℓ_2 con un uguale numero di punti N. Siano P_i i punti della prima e Q_j quelli della seconda. La distanza di matching si basa sul concetto di accoppiamento. Un accoppiamento (in inglese appunto *matching*) è una applicazione iniettiva e suriettiva f_α tra i P_i e i Q_j. In pratica si trovano N coppie di punti $(P_i, Q_j) = (P_i, f_\alpha(P_i))$ dove nessun punto delle due funzioni di taglia rimane escluso o compare due volte. Assegnata la funzione f_α, le si può associare il valore reale (o eventualmente

[12]Il fatto che i punti vicini alla diagonale siano legati al rumore si ricava come conseguenza del *Matching Stability Theorem* in [8].

infinito)

$$F(f_\alpha) := \left(\sum_i \left[\delta \left(P_i, f_\alpha(P_i) \right) \right]^p \right)^{\frac{1}{p}}$$

dove δ è ancora una qualsiasi metrica su $\mathbb{R}^2$ e $p \in [1, \infty)$. Si definisce la distanza di matching tra le due funzioni ℓ_1 e ℓ_2 come

$$d_M(\ell_1, \ell_2) := \inf_\alpha F(f_\alpha)$$

dove l'indice α varia in modo da considerare tutti i possibili accoppiamenti tra i P_i e i Q_j.

Ovviamente se le due funzioni non hanno lo stesso numero di punti, la definizione che abbiamo dato è mal posta, perchè non ha senso parlare di accoppiamento. Ma è in questo momento che ci viene utile la nostra interpretazione dei punti vicini alla diagonale come effetto dovuto al rumore. Supponiamo, non è riduttivo, che la seconda funzione abbia un numero di punti maggiore (N') rispetto a quello della prima (N). Allora si considerano tutte le possibili funzioni f_α iniettive e suriettive su un qualsiasi sottoinsieme di cardinalità N dei punti della seconda. Rimangono in questo modo "esclusi" $M = N' - N > 0$ punti di ℓ_2 che indichiamo come il sottoinsieme dei Q_j tali che $j \in I_\alpha$. Se si indica con $\delta_\Delta(Q_i)$ la distanza δ tra il punto Q_i e la retta di equazione $x = y$, allora generalizzando quanto abbiamo definito prima, si può porre, per ogni f_α:

$$F(f_\alpha) := \left(\sum_i \left[\delta \left(P_i, f_\alpha(P_i) \right) \right]^p + \sum_{j \in I_\alpha} \left[\delta_\Delta(Q_j) \right]^p \right)^{\frac{1}{p}}$$

E quindi si può in questo modo definire la distanza di matching per qualsiasi coppia di funzioni di taglia, come

$$d_M(\ell_1, \ell_2) := \inf_\alpha F(f_\alpha)$$

dove il parametro α varia in modo da considerare tutti i possibili accoppiamenti (nel senso dato sopra) tra i punti angolari delle due funzioni.

Si osserva che, essendo gli accoppiamenti possibili un numero finito, in realtà

nella definizione si può sostituire l'inf con il min. Questo in particolare significa che che esisterà un particolare accoppiamento ottimale: sia esso $f = f_{\bar{\alpha}}$ e sia $I_{\bar{\alpha}}$ l'insieme di indici che dà i punti della seconda funzione non accoppiati. Allora necessariamente, essendo f l'accoppiamento ottimale, la parte "non accoppiata" della distanza, ovvero $\sum_{I_{\bar{\alpha}}} [\delta_\Delta(Q_i)]^p$ non deve essere molto grande, ovvero i punti Q_i con $i \in I_{\bar{\alpha}}$ devono essere abbastanza vicini alla diagonale. È in quest'ottica che è interessante analizzare la distanza di matching, in quanto, l'intepretazione dei punti vicini alla diagonale come effetto dovuto alla presenza di rumore, ci dice che tale distanza è molto resistente, ovvero anche in caso di immagini molto disturbate (o perturbate) il risultato che si ottiene è abbastanza simile a quello che si otterrebbe con l'immagine intatta. Questo a nostro parere è uno dei motivi più importanti per procedere allo sviluppo di un algoritmo che implementi nel nostro software la distanza di matching. Inoltre, per concludere, osserviamo che, operando gli accoppiamenti tra i punti, questi vengono considerati con la loro molteplicità. Dunque questo comporta che la distanza di matching risulta effettivamente una distanza tra funzioni di taglia e non, come nel caso della distanza di Hausdorff, una pseudodistanza. Abbiamo così un ulteriore incentivo ad implementarla.

Capitolo 4

Esperimenti e risultati

4.1 Esperimenti

In questo capitolo vorremmo esporre un esperimento che abbiamo effettuato con il nostro software discutendo poi i risultati che ne abbiamo ricavato. In generale nell'approccio ad un esperimento con il nostro software, il primo problema è la scelta del campione su cui effettuarlo. Inoltre, soprattutto a questo livello in cui il programma è ancora in fase di studio, è cruciale individuare un insieme campione che risulti significativo, ovvero che permetta al programma di evidenziare le sue potenzialità e di mettere in mostra quello a cui può servire. Come abbiamo cercato di dire più volte, infatti, non è ancora il momento di utilizzarlo per operare degli effettivi recuperi di immagini da un data-base, per ora è molto più utile ed interessante studiare cosa esso è in grado di dirci sulla forma di una immagine. Oltre alle immagini da inserire nell'insieme da sottoporre al test, c'è da fare un'altra scelta da cui dipende pesantemente l'esito dell'esperimento: si deve decidere quali funzioni misuranti utilizzare. In un qualsiasi esperimento sono questi due i dati di input che diamo al software, e quindi da questi dipenderà se i risultati che troveremo saranno più o meno significativi. In particolare è importante osservare come questi due dati sono intimamente correlati, e non ha senso pensare all'uno se non in relazione alla scelta che si farà per l'altro. Ed è a partire da questa

considerazione che abbiamo stabilito come effettuare la scelta del date-base di immagini e delle funzioni misuranti. Infatti, da un lato, per i motivi implementativi che sono stati descritti nel capitolo precedente, abbiamo deciso di utilizzare immagini costituite solamente da rettangoli; dall'altro, sempre per semplificare il problema, le funzioni di taglia che abbiamo utilizzato appartengono alle due famiglie descritte in precedenza. In realtà inizialmente abbiamo utilizzato solo quelle della famiglia $SALTI$, poi da alcune considerazioni sui risultati che abbiamo ottenuto e che mostreremo in seguito, abbiamo deciso di testare anche le funzioni della famiglia $FILL$. Per far sì che queste due scelte trovassero un accordo, dato che le abbiamo imposte, prima di pensare agli esperimenti, per motivi essenzialmente pratici e di semplificazione del problema, siamo intervenuti sulla possibile disposizione dei rettangoli nelle immagini. Dunque abbiamo fatto in modo che fossero presenti, tramite la scelta della posizione reciproca dei rettangoli, delle caratteristiche grafiche a cui le funzioni misuranti fossero sensibili. Per prima cosa quindi, utilizzando la funzione salti, abbiamo dovuto considerare solo rettangoli vuoti. Successivamente abbiamo fatto la seguente osservazione preliminare, basata su un esempio pratico molto semplice: si consideri una immagine con due soli rettangoli vuoti ($R1$ e $R2$) e come funzione misurante quella $SALTI$ relativa all'asse delle ascisse. Solo due funzioni di taglia sono ottenibili da questa immagine, e il risultato dipende solo dalla posizione reciproca dei due rettangoli. Se essi sono uno a fianco dell'altro (ovvero $(R1.x + R1.width) < R2.x$) la funzione di taglia si rappresenta con due punti angolari di coordinate $(0, 2)$ e una retta angolare $(0, \infty)$; se al contrario le loro proiezioni sull'asse delle x hanno intersezione non vuota, allora la funzione di taglia che si ottiene si rappresenta con la stessa retta angolare, ma un solo punto angolare di coordinate $(0, 4)$.

Quello che si deduce da questo esempio è l'idea di cosa, a priori prima di effettuare l'esperimento, ci sembra sia sensibile per l'occhio delle funzioni di taglia, una volta scelta la funzione $SALTI$. Esse distinguono se due rettangoli, relativamente alla proiezione lungo un asse, sono sovrapposti o meno.

Combinando le informazioni che si possono ricavare considerando più assi di proiezione, tali funzioni ci danno informazioni sulle eventuali intersezioni dei rettangoli.

Per questo motivo si è deciso di creare tre macro classi di immagini. Nella prima imponiamo che non ci siano intersezioni tra nessuno dei rettangoli. Nella seconda invece che tutti i rettangoli si intersechino tra di loro. La terza classe vorrebbe essere in qualche modo intermedia, ovvero prevede entrambe le caratteristiche precedenti, decidendo se creare intersezioni o meno in modo casuale. In totale sono state create 150 immagini, 50 per classe. L'idea di base che abbiamo utilizzato è quella che, una volta identificate le caratteristiche generali che le immagini dovevano soddisfare, i parametri geometrici dei rettangoli (e anche il loro numero all'interno di ogni immagine) fossero decisi casualmente. Allo stesso tempo, poichè l'attenzione delle funzioni misuranti scelte cade sulle intersezioni e i rettangoli hanno una disposizione geometrica unica nello spazio delle nostre immagini (lati paralleli agli assi), si è deciso che era sufficiente utilizzare le funzioni $SALTI$ relative a solo tre rette: gli assi coordinati e la bisettrice del secondo e quarto quadrante. Vediamo ora i dettagli relativi alla costruzione delle immagini.

Le immagini della prima classe sono state chimate `intersezioniK.svg` dove $1 \leq K \leq 50$, vediamo come le abbiamo create, considerando l'algoritmo per la generica immagine k-esima. Dopo aver deciso il numero di rettangoli da inserire, questi sono creati sequenzialmente, uno di seguito all'altro. Ogni volta che se ne è creato uno si impone che il successivo lo intersechi. In questo modo tutti i rettangoli hanno intersezione non vuota almeno con un altro. L'idea che abbiamo imposto è che, sulla base della posizione del rettangolo corrente rispetto al precedente, si decida come intersecarlo con il successivo. Dato un generico rettangolo, i modi che abbiamo previsto per intersecarlo con il successivo sono 10 (da 0 a 9), lungo i quatto lati, sui quattro angoli e lungo le due direzioni orizzontale e verticale. Per capire meglio si confronti

l'immagine.

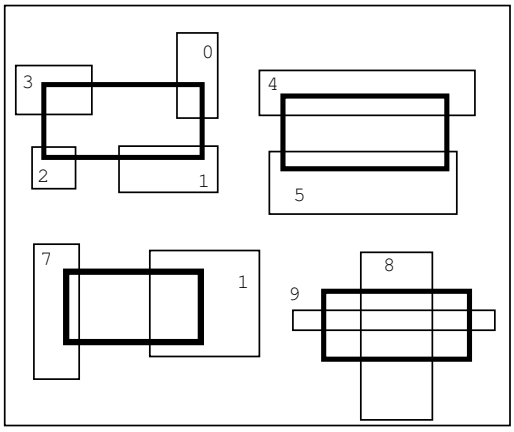


Figura 4.1: Possibili configurazioni di intersezioni tra rettangoli.

Quando si crea il primo rettangolo tutti i 10 possibili parametri sono accettabili. Al contrario, nei successivi, essendo già parte del rettangolo corrente occupato dall'intersezione con il precedente, non tutte le posizioni sono ammissibili. L'unica difficoltà dunque è consistita nel prevedere i vari casi.

Le immagini della seconda classe sono state chiamate `distantiK.svg`, dove $1 \leq K \leq 50$. Anche queste sono state create casualmente, ma con un algoritmo differente che impedisse le intersezioni. In particolare l'algoritmo che abbiamo creato utilizza un asse coordinato come supporto. Per cui, per evitare che pesasse sulle immagini un vincolo dovuto ad una potenziale asimmetria dell'algoritmo, abbiamo creato 25 immagini utilizzando come supporto l'asse delle x e 25 usando quello delle y. Diamo la spiegazione nel primo caso, in quanto l'altro è esattamente analogo, solo con i ruoli invertiti. Il controllo delle intersezioni è stato effettuato mediante un lista di valori che contiene le proiezioni sulla asse delle x dei rettangoli creati. Ogni volta che si crea un rettangolo nuovo R, se la sua proiezione sull'asse delle ordinate non interseca nessuna delle precedenti, allora si lascia totale libertà alla scelta dell'ordinata dei vertici di R. Al contrario, se la proiezione interseca almeno

una proiezione, si percorre la lista dei precedenti, e per ogni rettangolo per il quale si ha intersezione lungo la x si impone la non intersezione lungo la y. Ovvero si vanno a scegliere i vertici di R in quegli intervalli non coperti dagli altri rettangoli.

Le immagini della terza classe sono state chiamate `contiguiK.svg`, dove ancora $1 \leq K \leq 50$. Si è usato praticamente nella sua totalità l'algoritmo utilizzato per creare il primo insieme di immagini, con una sola piccola modifica, che rende questa classe forse la più interessante da studiare. Come abbiamo detto in precedenza, le funzioni di taglia che abbiamo scelto concentrano la loro attenzione sulle intesezioni, per cui l'idea di questa classe è creare delle immagini in cui i rettangoli tra di loro abbiamo tre possibili relazioni:

- se si intersecano, i rettangoli hanno intersezioni minime (rispetto alle loro dimensioni)
- se non si intersecano, la distanza tra i rettangoli è piccola
- hanno alcuni porzioni di spigoli sovrapposti, per cui c'è intersezione, ma solo tra i bordi

La scelta di quale tra queste caratteristiche scegliere tra un rettangolo e il suo successivo è ancora stata fatta in modo casuale. Ci è sembrato utile implementare una classe di questo tipo anche per verificare l'eventuale resistenza del nostro software al rumore (dove in questo caso si intende una eventuale piccola variazione dei dati relativi a rettangoli abbastanza vicini, tale da poter modificare la loro diposizione relativa: intersezione, sovrapposizione di parte dei bordi, non intersezione).

Infine, come abbiamo anticipato, visto il particolare tipo di dati che abbiamo dato in inuput al software, i risultati che si ottengono a livello di funzioni di taglia (e non ancora di confronto tramite le distanze) presentano uno scarso range di variazione. Essenzialmente solo valori interi positivi e

non molto elevati (inferiori a 10). Il motivo di questo verrà affrontato nel prossimo paragrafo di presentazione e discussione dei risultati che abbiamo ottenuto. Abbiamo deciso, quindi, di utilizzare le funzioni di tipo $FILL$, che ci hanno permesso di ottenere dei risultati decisamente più variabili. L'idea che abbiamo avuto è di utilizzare queste diverse funzioni sullo stesso insieme testato precedentemente con le funzioni $SALTI$. Per fare questo si è dovuto passare da rettangoli vuoti a rettangoli pieni, cosa che in alcuni casi ha comportato un pò di più che la semplice modifica dei valori da assegnare alle funzioni misuranti. Infatti nel caso delle immagini `distantiK.svg`, l'unica modifica all'algoritmo di calcolo dei valori è stato nel valore d da aggiungere ai nodi contenuti nella proiezione del nuovo rettangolo sulla retta r (confronta l'appendice B). Al contrario, nel caso delle immagini `intersezioniK.svg`, non abbiamo potuto utilizzare questo procedimento. Infatti se due rettangoli si intesecano, a livello di quantità di nero la parte contenuta nell'intersezione deve essere contata una sola volta e quindi si devono fare dei controlli aggiuntivi. La soluzione che abbiamo trovato consiste nel ricondurre il problema al caso precedente (dei rettangoli senza intersezione, a parte il bordo) traformando ogni rettangolo che si inserisce in n rettangoli distinti che sono individuati dall'intersezione stessa, in modo che uno solo cosituisse l'intersezione e quindi si tenessero solo gli altri $n-1$. In questo caso è più semplice spiegarlo con una immagine che a parole.

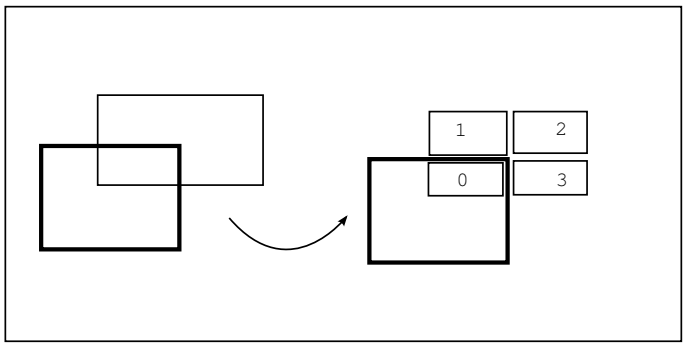


Figura 4.2: Eliminazione dell'intersezione tra rettangoli.

Il rettangolo più spesso è il precedente, quello sottile (entrambi vuoti per maggiore leggibilità) è quello che si aggiunge. Esso viene diviso in quattro rettangoli (da 0 a 3) di cui si considerano solo 1, 2 e 3.

4.2 Risultati

Ora presenteremo alcuni aspetti dei risultati trovati e cercheremo di darne una interpretazione. La prima osservazione da fare consiste nel fatto che ogni esperimento, se condotto come lo abbiamo descritto nel paragrafo precedente, produce come output un numero di $2*D*N$ file di output (dove N è il numero di immagini dell'insieme test e D è il numero di modi in cui si rielaborano le distanze trovate), in ciascuno dei quali sono contenuti N valori numerici (distanze). Per fare un esempio, nel caso del nostro esperimento abbiamo $N = 150$ e $D = 2$, per cui otteniamo come risultato 90.000 valori numerici, divisi in 600 file di output. In altre parole non è facile gestire tutta questa quantità di dati, per il semplice fatto che non è facile neanche da visionare. Al momento quindi l'analisi che si può fare è per lo più qualitativa, basata su particolari dati che saltano agli occhi o su distribuzioni abbastanza regolari dei risultati.

La prima considerazione importante che si deve fare è di carattere qualitativo, sulla scelta dell'insieme di test. Se si vanno a considerare i file relativi alla distanza normalizzata, ci si accorge che nella grande maggioranza dei casi (80%) la maggior parte delle immagini (il 66%) sta ad una distanza normalizzata minore di 1.2. Ovvero due terzi del campione sono abbastanza vicini tra loro (1.2 significa che in media sulle tre funzioni misuranti utilizzate distano 0.4). Questo ci sembra una prima indicazione sulla buona scelta dell'insieme campione. Ovvero le immagini che abbiamo scelto sono abbastanza omogenee tra di loro. Inoltre si è osservato che non c'e' un gruppo in particolare (tra i 3) che si distacca dagli altri: più o meno le immagini che si allontanano dalla media del campione appartengono in egual numero alle 3 classi. D'altra

parte però, le immagini di tipo `distantiK.svg` benchè abbiamo i 2/3 delle altre immagini sufficientemente vicini (< 1.2), mediamente presentano una distribuzione delle distanze più elevata, ovvero sono maggiori in numero le immagini che hanno distanza nell'intervallo $[0.6, 1.2]$ di quelle che distano meno di 0.6. Questo comportamento si può interpretare come una sensibilità effettiva (mentre in precedenza l'avevamo solo ipotizzata) alla presenza di intersezioni tra i rettangoli da parte delle funzioni misuranti utilizzate.
Una possibile conferma di questo aspetto si può trovare in un'altra considerazione sulla distribuzione dei risultati. Se si considerano i file con le distanze medie dalle immagini di tipo `distantiK.svg`, ci si accorge che nella maggior parte dei casi, la maggioranza delle immagini dello stesso tipo sta nella prima metà delle classifiche. Questo dà una indicazione di come tendenzialmente le immagini di tipo `distantiK.svg` siano vicine tra loro, ma è meno esplicito sul rapporto con quelle delle altri classi. A tal proposito possiamo portare due esempi contraddittori. Nel primo caso si considerano le distanza medie da `contigui3.svg`, nel secondo da `contigui4.svg`. Il primo sembra farci vedere come in effetti le immagini `distantiK.svg` distino un po' di più dalle altre: infatti si ritrova che da un lato tale immagine è un buon modello in quanto ben 127 immagini stanno sotto la soglia di distanza di 1.2, e d'altra parte in parallelo ben 25 (la metà) immagini che non presentano intersezioni sono posizionate negli ultimo 40 posti. Dunque verrebbe da pensare che sono proprio le immagini di tipo `distantiK.svg` a discostarsi dallo standard costituito dall'insieme di test. D'altra parte, nell'altro esempio abbiamo che sempre la metà (25) di queste sono nei primi 54 posti (il primo terzo) e in parallelo anche questa immagine costituisce un buon campione rappresentativo. Quindi questo caso spingerebbe a pensare che non si ha un discostamento effettivamente rilevante. Ma ancora si potrebbe supporre che è proprio la presenza di un numero elevato di tali immagini nei primi posti della classifica a renderlo un buon rappresentante, perchè se così non fosse, se la maggior parte delle immagini `distantiK.svg` fossero nell'ultima parte, queste contribuirebbero ad aumentare la distanza dall'insieme campione. A partire da

questa interpretazione quindi, una immagine come la `contigui4.svg` è da un punto di vista statistico un buon campione, ma da un punto di vista grafico un caso particolare, in quanto la sua vicinanza a molte immagini senza intersezioni potrebbe essere non una informazione sulla omogeneità di queste all'interno dell'insieme, bensì una indicazione di somiglianza del modello a queste che si discostano dalla media.

Approfittiamo di questo esempio per analizzare il rapporto tra le due diverse eleborazioni delle distanze che abbiamo utilizzato (la distanza normalizzata e quella media). Abbiamo infatti utilizzato i dati che si ricavano da entrambe le distanze, ritenendo che fossero paragonabili. A priori però non si può dare questa conclusione. Osservando i file di output che si generano nel nostro esperimento, si vede che mediamente l'andamento delle due diverse distanze è abbastanza paragonabile. Globalmente infatti rispecchiano lo stesso andamento, le uniche differenze si hanno localmente e non sono mai eccessive. Questo ci permette di dire, come del resto ci sembrava a priori naturale, che le due diverse eleborazioni possono essere usate in parallelo, interscambiate e confrontate almeno sui grandi numeri. Poi, come abbiamo cercato di spiegare nell'ultima sezione del capitolo terzo, ciascuna può essere utilizzata nel dettaglio per ricavare informazioni più specifiche.

Il dato più significativo che si ricava però dall'analisi dei risultati riguarda proprio il tipo di esperimento che abbiamo efettuato e gli strumenti utilizzati. Se si osserva infatti l'elenco delle distanze normalizzate dall'immagine `contigui28.svg` si ritrovano che ben tre immagini hanno da questa distanza pari a 0. Andando a confrontare le funzioni di taglia di queste immagini, relative alle varie funzioni misuranti, ci si rende conto che questa particolarità dipende da due fattori. In primo luogo la distanza di Hausdorff in effetti è una distanza per i sottoinsiemi del piano, ma nel caso delle funzioni di taglia, in cui conta anche la molteplicità con cui compare un punto, non soddisfa la proprietà di separabilità. Cioè si ha che se $d(\ell_1, \ell_e) = 0)$ allora non

necessariamente $\ell_1 = \ell_2$, ovvero d costituisce una pseudodistanza. D'altra parte, questo è il caso delle immagini `contigui28.svg`, `contigui33.svg` e `distanti50.svg`, si verifica che immagini diverse possono avere tutte le funzioni di taglia relative alle diverse funzioni misuranti uguali tra loro. Questo problema, come avevamo anticipato nella sezione precedente, dipende essenzialmente dalla ristretta possibilità di variazione data ai valori assunti dai punti angolari delle funzioni di taglia. La causa di questa degenerazione va dunque ricercata nella scelta delle funzioni misuranti. Si verifica infatti, analizzando le varie funzioni di taglia generate sulle immagini, che queste hanno punti angolari con coordinate sempre intere e comprese tra 0 e 10. Questo problema dipende dal fatto che si è imposto che nelle immagini da testare il numero di rettangoli fosse limitato (al più 7). Infatti, con questa ipotesi, risulta chiaro che la funzione misurante $SALTI$, che già per costruzione assume solo valori interi pari, avrà un massimo limitato da $2k$ dove k è il numero di rettangoli dell'immagine. Data quindi una possibilità così ristretta di variazione delle coordinate dei punti e al contrario un numero così elevato (almeno confrontato col precedente) di immagini, è molto probabile che vi siano alcune di queste con le funzioni di taglia uguali. In particolare si osserva che le due immagini `contigui28.svg`e `contigui33.svg` sono molto simili anche graficamente, mentre hanno un aspetto che si differenzia dalla `distanti50.svg`.

Da tutto questo si ricava che la scelta della misura di Hausdorff non è delle più efficaci, soprattutto nel caso in cui le funzioni di taglia abbiano un intervallo di variazione ristretto; in secondo luogo si vede che il tipo di funzione misurante che abbiamo scelto, benchè sia significativa non è abbastanza "distintiva", ovvero non riesce a distinguere abbastanza le immagini. Per risolvere il secondo problema abbiamo deciso di implementare anche l'altra famiglia di funzioni misuranti ($FILL$), in modo da permettere una maggiore variazione dei risultati. Infatti si osserva che questa scelta comporta un notevole abbassamento della probabilità che due diverse immagini abbiamo associate funzioni di taglia uguali oppure funzioni di taglia differenti ma rap-

presentate da successioni con supporto uguale (come accade invece nel caso della `contigui28.svg` e `distanti32.svg`). Non inseriamo, per evitare di appesantire troppo l'esposizione, anche la descrizione dei risultati di questo secondo esperimento, lasciandolo quindi da approfondire in un eventuale lavoro futuro.

Ci concentriamo invece su un altro aspetto interessante, ovvero l'interpretazione dei risultati di tipo grafico. La prima considerazione da fare riguarda la non esistenza di una "ground truth". Ovvero, come abbiamo più volte detto, dato il tipo di immagini che abbiamo scelto, molto astratte e quasi del tutto prive di un significato[1], non ha molto senso cercare di stabilire quali di esse sono simili o dissimili. In particolare quindi non ha senso ancora dare una valutazione sui risultati che il software ci può dare. Dato un esperimento, possiamo solo cercare di ricavarne il maggior numero di informazioni possibili, relativamente a cosa esso mette in evidenza delle caratteristiche grafiche, o al contrario quali non vengono tenute in considerazione.
È a questo scopo che abbiamo previsto che si producessero anche degli output "grafici", cioè i file in cui la classifica delle immagini ordinate in modo crescente rispetto alla distanza da un modello è integrata dal confronto tra le due immagini di cui si ha la distanza. In questo modo, scorrendo verso i valori più alti di distanza, si può cercare di individuare quali sono le caratteristiche grafiche che, sotto la lente delle funzioni misuranti che abbiamo scelto, vengono evidenziate dalla teoria della taglia.
Con questo tipo di approccio siamo riusciti ad individuare essenzialmente una sola caratteristica grafica che influisce in modo sostanziale sulla distanza tra le immagini. Osservando, sulla maggior parte dei modelli, le 10 immagini più vicine abbiamo constatato che in queste viene generalmente conservato il numero di rettangoli. A nostro parere la capacità di riconoscere il numero di rettangoli dell'immagine sta nella ridotta possibilità di variazione dei punti

[1]Intendiamo con significato il rimando ad un oggetto concreto o ideale, che l'immagine rappresenterebbe. Ovvero il lato semantico dell'immagine stessa.

angolari. Da questo discende una relazione molto stretta tra il numero di rettangoli e le coordinate dei punti angolari, in quanto sono legati ai valori critici della funzione misurante. Si è inoltre osservata una maggiore tendenza alla conservazione del numero di rettangoli quando il modello scelto appartiene alla classe `intersezioniK.svg`. Non ci sembra però che questo dato sia abbastanza marcato da indurci a trarne alcune conclusioni sulla particolarità di tale gruppo di immagini rispetto alle altre.

Infine, una ultima considerazione che si può fare, ma andrebbe verificata nel dettaglio con un esperimento mirato, riguarda le dimensioni dei rettangoli utilizzati. In particolare (come si può verificare per esempio nel caso dell'immagine `contigui33.svg`) qualora il modello contenga un rettangolo di piccole dimensioni (confrontate con quelle degli altri), tale caratteristica si conserva nelle immagini ad esso più vicine. D'altra parte da un punto di vista matematico non è del tutto chiaro perchè questo particolare venga evidenziato, per cui rimandiamo ad un esperimento ad hoc, da effettuare in un prossimo futuro, per poter trarre delle conclusioni più solide.
Come prospettiva futura descriviamo un altro interessante esperimento che si potrebbe effettuare e che riteniamo potrebbe essere il vero momento di verifica della funzionalità di questo software: un test mirato alla verifica della effettiva capacità di riconoscimento del nostro programma. Si tratterebbe di creare un insieme di immagini costituite solo da rettangoli (pieni e/o vuoti) ma dotate di un aspetto semantico, ovvero delle vere e proprie costruzioni di rettangoli che, utilizzati come parti componibili, visti come un tutt'uno rappresentino degli oggetti reali. Qualora le classifiche create dal software rispettassero anche dei criteri di somiglianza delle immagini, questo sarebbe un ottimo incentivo a continuare nella nostra ricerca e nello sviluppo del software.

Per concludere questa discussione dei risultati che abbiamo trovato effettuando l'esperimento, utilizziamo un software di elaborazione dati utilizzato

diffusamente nell'ambito degli studi genetici. Si tratta di una applicazione open-source costituita da un elaboratore di calcolo *Phylip* e da un visualizzatore *Phylodraw*[2]. Questo software viene utilizzato per ricavare il livello di imparentamento di vasti insiemi di specie viventi sulla base del loro codice genetico: costruisce la matrice della distanza tra ogni coppia di specie e poi, sulla base di questa, crea una specie di albero che evidenzia in modo qualitativo quali sono i rapporti di parentela genetica tra di loro. Noi abbiamo utilizzato il programma fornendogli direttamente come input la matrice delle distanze tra le diverse immagini e ottenendo così l'albero di parentela. Sistemando opportunamente i vari parametri di visualizzazione abbiamo ottenuto i due alberi che si vedono in figura nella pagina successiva. Il primo, essendo molto ramificato, mette in evidenza come non ci sia una raggruppamento in poche famiglie numerose di immagini tra loro strettamente imparentate: ovvero ci dà una ulteriore conferma alla nostra ipotesi che l'insieme test che abbiamo scelto è abbastanza omogeneo. D'altra parte nella seconda, si vede che (come era in parte prevedibile) in realtà le tre classi di immagini che abbiamo creato per l'esperimento, si differenziano abbastanza tra di loro (le `distantiK` si trovano per lo più nei nodi tra 50 e 90, le `contiguiK` invece tra 13 e 49, infine quelle di tipo `intersezioniK` tra 106 a 149), anche se solo le prime si concentrano praticamente in un solo ramo dell'albero. E infatti avevamo notato che si distinguevano maggiormente come classe a sè stante. Sottolineiamo che abbiamo utilizzato questo software perchè ci sembra che sia uno strumento interessante per fare delle analisi qualitative dei nostri dati, ma per essere veramente utile, riteniamo che debba essere affiancato da una elaborazione che ci permetta di avere delle informazioni maggiormente quantitative per dargli un riscontro più solido.

[2]Si veda [22] e [23].

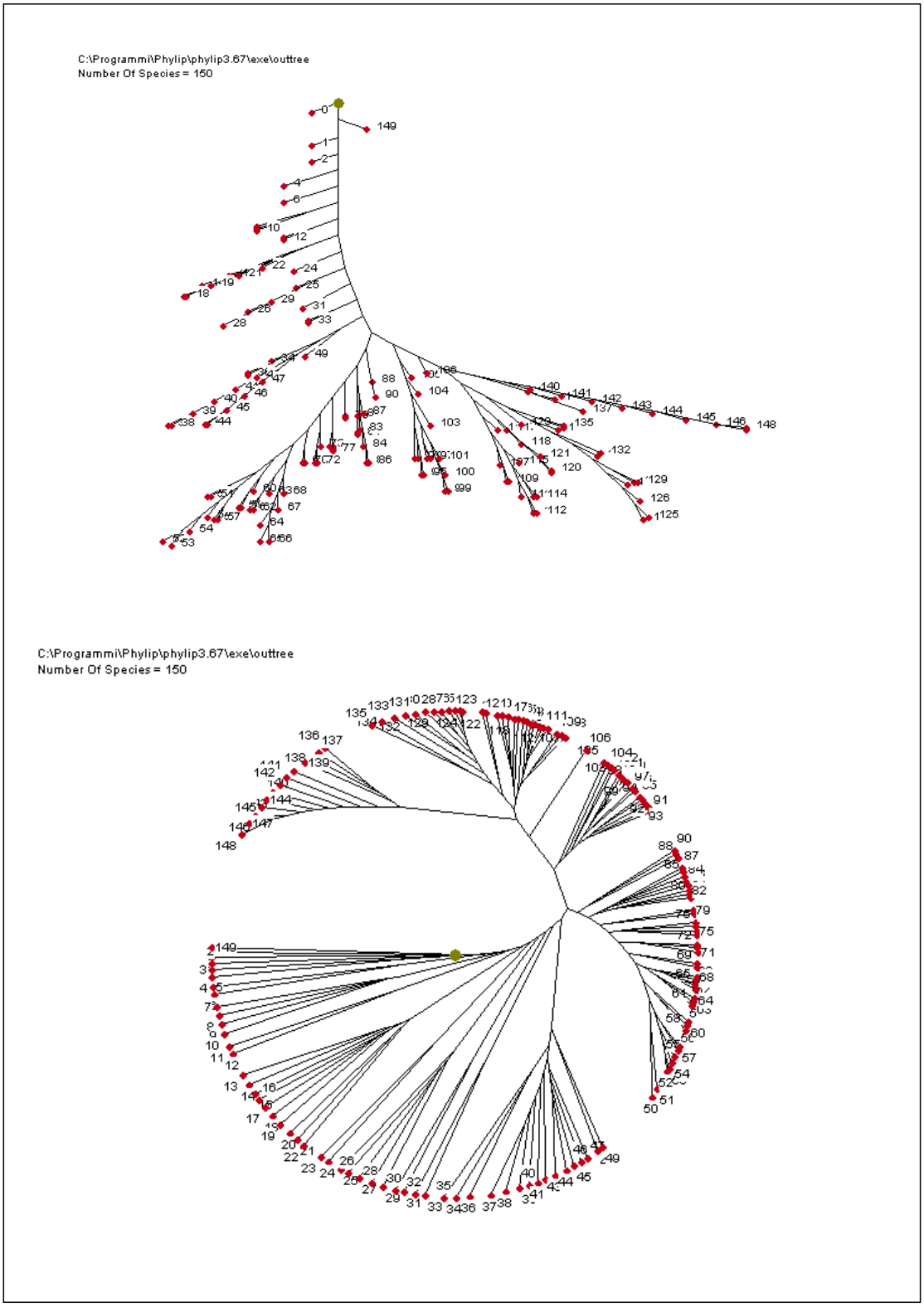


Figura 4.3: Albero dell'imparentamento "genetico" delle immagini.

Conclusioni

Il continuo aumento della produzione di materiale digitale di tutti i tipi e, in parallelo, il fatto che lo sviluppo di internet permette a qualsiasi utente la possibilità di mettere i propri contenuti a disposizione degli altri, mettono di fronte alla necessità di trovare metodi sempre più efficaci ed efficienti per recuperare e gestire l'informazione. L'idea di associare ad ogni tipo di prodotto digitale, ed in particolare alle immagini, una o più *keypics* (cioè immagini-chiave sull'analogo delle parole-chiave) sembra aprire un approccio innovativo e fruttifero a questo tipo di problemi. La rappresentazione di concetti per mezzo di immagini stilizzate costituisce un mezzo espressivo per certi versi meno preciso e potente delle parole-chiave, ma che, proprio per questo, si presta ad un tipo di indicizzazione dei contenuti digitali, e conseguentemente di ricerca, più specificamente analogico (tramite concetti più "sfumati"). Apparentemente quindi sembra ricalcare maggiormente quelli che sono i processi del pensiero, libero e non vincolato dal veicolo linguistico. Questa prospettiva incrementa ancora di più il nostro interesse, già molto diffuso a livello accademico, per il riconoscimento e il recupero di immagini; tanto più quando i metodi da noi utilizzati in questo campo (la Teoria della Taglia e più nello specifico le Funzioni di Taglia) sembrano particolarmente adatti al tipo di immagini che costituirebbero le keypics. Si è aperto così un ampio capitolo di studio tecnico per riuscire a stabilire quale sia il formato più idoneo per la costruzione di tali immagini, che noi al momento abbiamo individuato in SVG, e di conseguenza una analisi dei metodi di acquisizione dell'informazione grafica contenuta in tale formato. In questa tesi abbiamo

evidenziato come quest'ultimo passaggio rappresenta un problema molto ampio e di non facile soluzione, per motivi tecnici, ma soprattutto teorici. D'altra parte abbiamo anche cercato di mostrare come, sotto determinate ipotesi più o meno restrittive, si riescono a trovare delle soluzioni efficienti per poter gestire l'informazione contenuta nelle immagini, in modo da potersi dedicare al secondo nucleo concettuale del problema: lo studio effettivo dei metodi di recupero e confronto. Le sperimentazioni che abbiamo effettuato in questa direzione sono ancora ad uno stadio elementare, ma già danno la possibilità di organizzare dei test più significativi, come abbiamo cercato di suggerire in questa tesi.

Infine l'approccio di questo tipo al problema, che potremmo sintetizzare con la coppia Teoria della Taglia più Grafica Vettoriale, benchè ancora elementare, mette già in chiara evidenza i suoi pregi intrinseci (non tutti approfonditi in questa tesi). Tra i principali citiamo la predisposizione a metodi più geometrici (e quindi appunto analogici) e la concentrazione, e maggiore organizzazione logica, dell'informazione grafica con il conseguente abbattimento del tempo di calcolo. Il risultato maggiore che speriamo di aver ottenuto con questa tesi è l'aver contribuito ad incentivare la ricerca teorica e tecnico-algoritmica in questo particolare campo: non abbiamo avuto infatti la presunzione di dare delle risposte definitive, ma al contrario la volontà di lasciare aperte importanti questioni e porre il maggior numero di domande, in quanto queste costituiscono il vero motore per avere un effettivo avanzamento della conoscenza.

Appendice A

Dall'immagine al testo

A.1 Strutture dati per le primitive grafiche

Inseriamo in questa appendice la parte di codice[1] che contiene la definizione delle strutture dati utilizzate per memorizzare le informazioni contenute all'interno delle primitive grafiche. Come abbiamo spiegato nel terzo capitolo abbiamo creato una struttura potenzialmente in grado di contenere i dati relativi a qualsiasi primitiva, anche se al momento è stata predisposta solo la sottostruttura per gestire i rettangoli. Per rendere il codice più esplicativo abbiamo inserito anche una sottostruttura per un altro tipo di primitiva (i segmenti), nonostante nel resto del software non vengano mai utilizzati.

```
// per prima cosa l'elenco delle strutture dati
//per contenere i vari tipi di elemento grafico
typedef struct{
  int name;
  double x, y;
  double width, height;
    double rx, ry;
// si potrebbe aggiungere anche altro legato allo style
} rect_t;
```

[1] Dal momento che il software è in continua evoluzione, lasciamo all'interno del codice che inseriamo alcune delle parti di commento, infatti ci sembra che da un lato possano renderlo più comprensibile, dall'altro danno indicazioni sulla direzione e sui modi in cui il codice stesso si sta evolvendo.

```
//provvisoria, prevede solo i mono-segmenti
typedef struct{
        double x1, y1;
        double x2, y2;
} line_t;

typedef union{
        rect_t rect;
        line_t line;
//aggiungere gli altri possibili tipi di elemento grafico
} forma_t;

typedef struct{
        char forma;
        forma_t elemento;
} element_t;
```

A.2 L'algoritmo di ricerca delle stringhe

Come abbiamo spiegato la parte A dell'algoritmo opera ciclicamente sui tag che sono inseriti come ammissibili in lettura. Il tutto è implementato all'interno di un'unica funzione `input` la quale apre un file SVG, che contiene una immagine, e memorizza all'interno di un vettore di tipo `element_t` il contenuto delle primitive grafiche. Dopo aver scelto un tag, ricerca ciclicamente all'interno del file la stringa associata, fino a che non arriva alla fine. Ogni volta che trova una ricorrenza della stringa vuol dire che ha individuato una primitiva e quindi ne momorizza i dati nell'apposita struttura dati.
A livello implementativo, la difficoltà maggiore che abbiamo riscontrato consiste nel risolvere il problema della ricerca delle stringhe. Questo problema lo si incontra sia nel momento in cui si devono trovare i diversi tag all'interno del testo sia, una volta trovati, per trovare le stringhe date dai comandi che definiscono i parametri geometrici della primitiva. Descriviamo quindi ora l'algoritmo che abbiamo utilizzato per la ricerca delle stringhe.
Tale algoritmo è stato implementato nella funzione `trova_stringa`. Questa routine ha lo scopo di ricercare una determinata stringa passata come parametro di input, sia all'interno di un file di testo sia all'interno di una

stringa più lunga, e restituisce come output 1 se la stringa è stata trovata 0 altrimenti. Si distingue se effettuare la ricerca su un file (di cui si passa l'indirizzo come parametro di input) o all'interno di una stringa (anche di questa si passa l'indirizzo) tramite un parametro di input (`type`) di tipo carattere che assume rispettivamente i soli valori *f* o *s*. Una volta trovato il tag all'interno del file (se `type`=*f*) il cursore di lettura è posizionato nel carattere immediatamente successivo. Per avere un output simile nel caso della ricerca su stringa si è definito un parametro di output `p` di tipo puntatore ad un intero che permette, alla fine, di conoscere l'indice, all'interno della stringa maggiore, dell'ultimo carattere della stringa cercata.
Il fatto di dover differenziare due diversi tipi di ricerca, ovvero su file o su stringa è stato necessario per risolvere ancora una volta un problema legato alla libertà di stile di SVG. Infatti quando si è trovato all'interno del file il tag desiderato, non è possibile prevedere a priori l'ordine in cui i parametri geometrici che lo descrivono sono presentati. E d'altra parte se si pensasse (per esempio per trovare la larghezza del rettangolo) di effettuare sul file la ricerca della stringa `width` con il medesimo algoritmo, poichè il cursore di lettura non può ritornare indietro, si rischia di perdere tutte le informazioni scritte prima di tale stringa. La soluzione che si è trovata è stata quindi quella di copiare, una volta trovato il tag, tutto il suo contenuto (cioè fino a quando non si trova il carattere `>`) su una apposita stringa ausiliare che abbiamo chiamato `sorgente`. Su questa, utilizzando gli indici, non solo si può andara avanti e indietro a piacimento con un ideale cursore di lettura, ma è anche immediato che, ogni volta che la si legge, si parte dal primo carattere succesivo al particolare tag cercato. Dunque la ricerca su stringa, benchè avvenga esattamente con lo stesso principio di quella su file, è molto più semplice da gestire e, proprio per questo particolare, si è potuto implementare un controllo in più, non previsto nella ricerca su file. Tale controllo verifica che, qualora si trovi la stringa desiderata, non si abbia che questa sia contenuta all'interno di una parola: per esempio se si deve cercare la stringa `x` non si accetta come valida la parte finale della stringa `rx`. Tale controllo, come mostra l'esempio presentato, è molto importante perchè evita la confusione tra i diversi parametri. In pratica l'algoritmo non ricerca la stringa come semplice successione di caratteri, ma come qualcosa in più: la ricerca come un comando, che quindi ha senso solo se è dato da una parola a sé stante. Per il particolare tipo di stringhe che si ricercano, nel nostro caso, sul testo (per esempio `<rect`) il fatto di non prevedere questo tipo di controllo non costituisce un problema, dato che la presenza della parentesi uncinata

all'inizio della stringa impedisce di trovarla all'interno di parole più lunghe. Analizzate queste caratteristiche preliminari della funzione, mostriamo ora come agisce l'algoritmo di ricerca delle stringhe. Per comodità supponiamo che la ricerca sia su stringa (è il caso più completo e semplifica la notazione).

```
Input: Stringa = stringa da ricercare (n = lunghezza di Stringa);
       Sorgente = stringa all'interno della quale cercare Stringa
       *p = puntatore ad un intero (parametro di output)
01 per i = 0 -> lunghezza di Sorgente
02 se Sorgente[i] == stringa[0]
03     controlla se Sorgente[k] e' all'inizio di parola
04     se e' all'inizio di parola
04         per j = 0 -> n
05             controlla se Sorgente[i+j] == stringa[j]
06         se tutti i caratteri sono uguali esce
           dalla funzione e restituisce 1
07     altrimenti ricomincia la ricerca da Sorgente[i+1]
08 se arrivo alla fine di Sorgente esce dalla funzione e restituisce 0
Output: 1 o 0 se ha torvato o meno la stringa
        se output = 1
           *p = puntatore all'intero k+n,
           permette di trovare il carattere Stringa[k+n]
           cioe' l'ultimo della stringa
```

Il controllo della riga 03 si effettua verificando se il carattere `Sorgente[k-1]` è o meno una lettera dell'alfabeto (minuscola o maiuscola), ovvero verificando se in codifica ASCII è compreso tra 65 e 90 o tra 97 e 122.
Come si vede c'e' tutta una serie di controlli inclusi uno dentro l'altro: per implementarla si sono definite due variabili di controllo (`flag2, flag3`)[2]. Queste, in combinazione con il possibile valore dell'indice j alla fine del procedimento di verifica della riga 05 (se $j = n$ ho trovato `Stringa`, se $j > n$ non l'ho trovata), permettono di stabilire che parametro di output generare. Il puntatore `p` serve per poter continuare la ricerca all'interno della stringa. In particolare ogni qual volta si è trovato uno dei comandi che introducono

[2]La variabile `flag1` è definita nella parte di ricerca su file, è utilizzata per controllare se si è arrivati alla fine del file.

gli attributi grafici di una primitiva, esso permette di conoscere la posizione in cui leggere il numero che rappresenta l'attributo.

Includiamo per completezza l'intero codice di questa funzione (anche la parte relativa alla ricerca su file):

```
// funz che trova la prima ricorrenza della stringa richiesta
//    si potrebbe semplificare ???
int trova_stringa(FILE *immagine, char sorgente[],
                          char stringa[], char type, int *p )
{
    int i, j;   char c;
    int flag1 = 1;    int flag2 = 0;   int flag3;

    switch (type){
    case 'f':
       while( flag1 != EOF && flag2 == 0)
       {
            flag1 = fscanf(immagine,"%c",&c);
            // per tenere conto che il file e' finito e non si puo'
            // continuare la ricerca ->  flag2 = -1
            if(flag1 == EOF)
                flag2 = -1;
            else
                if(c == stringa[0])
                     for(i=1; i <= strlen(stringa)-1 && flag1 != EOF; i++)
                     {
                         flag1 = fscanf(immagine,"%c",&c);
                         if( flag1 == EOF)
                             flag2 = - 1;
                         else
                             if(c != stringa[i])
                                  i = strlen(stringa);
                     }  // alla fine si ha un i++
                     if( i < strlen(stringa)+1  )
                         flag2 = 1;
      }
      break;
```

```
    case 's':
      j = strlen(stringa) +1;
      //fa si che la condizione j < strlen si verifichi solo quando
      // si e' trovata la parola cercata (cioe' si e' passati dentro
      // la condizione flag3 != 0 e non anche quando non ci si e'
      // passati, nel qual caso j non e' assegnato
      for(i=0; i <= strlen(sorgente)-1; i++)
      {
          if(sorgente[i] == stringa[0])
          {
             flag3 = 1;
             if(i>0)  // vedo se la stringa e' in mezzo a 1 parola
                if( (sorgente[i-1] >= 65) && (sorgente[i-1] <= 90) ||
                     (sorgente[i-1] >= 97) && (sorgente[i-1] <= 122) )
                {
                      flag3 = 0;
                      j = strlen(stringa) + 1;
                }
             if(flag3 != 0)
                  for(j=1; j <= strlen(stringa)-1 &&
                             i+j <= strlen(sorgente)-1; j++)
                     if(sorgente[i+j] != stringa[j])
                         j = strlen(stringa);
          }       // alla fine del ciclo si ha j++
          if( j < strlen(stringa)+1 )
          {
              flag2 = 1;
              *p = i+j;
              i = strlen(sorgente); //alla fine del ciclo e' incrementato!
          }
      }
      if(i == strlen(sorgente) )
         flag2 = -1;
      break;
    }
    return((flag2 + 1)/2);
}
```

Appendice B

L'algoritmo di calcolo delle funzioni $SALTI$ e $FILL$

Inseriamo per prima cosa la parte di codice che contiene la definizione delle strutture dati con cui abbiamo rappresentato le due famiglie di funzioni misuranti presentate nel terzo capitolo. Come si vede, e come abbiamo già specificato tali strutture costituiscono una lista dinamica:

```
// e' il nodo per la lista dinamica
typedef struct mf_node_s {
        double mf_x;
        int mf_val;
        struct mf_node_s *mf_next;
} mf_node_t;

// e' la struttura per la lista dinamica che rappresenta
// la funzione misurante (Measuring Function)
typedef struct {
        int mf_nodes;
        mf_node_t *pt;
} mf_list_t;
```

L'algoritmo di calcolo delle funzioni misuranti è stato dettagliatemente descritto nel terzo capitolo, tralasciando solo alcuni dettagli tecnici. Riprendiamo qui l'esposizione e cerchiamo di esporre anche tali dettagli. Supponendo di avere un vettore di rettangoli $R = (R(i))_{1 \leq i \leq n}$ e una lista di nodi K inizialmente vuota (ovvero con $K(0) \succ K(1)$ con $K(0).x = x_{min}$,

$K(1).x = x_{max}$ e $K(0).val = K(1).val = 0.0$). L'algoritmo ad ogni passo prende in considerazione un rettangolo $R(i)$ e modifica la lista aggiungendo dei nodi e/o modificandone i valori. Nel dettaglio

```
01 per ogni i = 0 -> n
02   considero il rettangolo R(i) e calcolo (x-) e (x+)
03   trovo nella lista i due nodi K(i-) e K(i+)
04   se K(i-).x < (x-) creo un nodo K-
      altrimenti considero K(i-) := K-
05   se (x+) < K(i+).x creo un nodo K+
      altrimenti considero K(i+) := K+
06   modifico i nodi compresi tra K- e K+
```

Abbiamo già spiegato che x^- e x^+ sono gli estremi dell'intervallo costituito dalla proiezione ortogonale del rettangolo $R(i)$ sulla retta r e che i nodi $K(i^-)$ e $K(i^+)$ sono gli unici che verificano

$$\ldots K(i^-).x \leq x^- \leq K(i^- + 1).x \leq \ldots \leq K(i^+ - 1).x \leq x^+ \leq K(i^+).x \ldots$$

All'interno del nostro software questa parte di algoritmo (contenuta nella funzione `rect2fun`) è gestita da 3 diverse procedure: `leftnode` e `rightnode` implementano la riga 03, il resto è contenuto nella `new_node`. Come abbiamo già spiegato, le questioni tecniche sono costituite proprio da questa parte, che ora esponiamo nel dettaglio. Consideriamo due casi separati: il primo è quello delle funzioni di tipo $SALTI$ e delle funzioni di tipo $FILL$, quest'ultime nel solo caso in cui la retta r è una degli assi coordinati; il secondo è il caso delle funzioni $FILL$ quando r non è uno dei due assi coordinati (e per noi sarà la retta di equazione $x + y = 0$). Nel primo caso si ha:

```
01 (K-).val = K(i-).val + d
02 per ogni nodo K(i) compreso tra K- e K+ (esclusi)
03    K(i).val = K(i).val + d
04 K(+).val = K((i+)-1).val - d
```

dove, se considero le funzioni $SALTI$, allora $d = 2$, mentre, nel caso delle funzioni $FILL$, si ha che $d = R(i).height$ se $r : y = 0$ e $d = R(i).width$ se $r : x = 0$. Più complicato è il secondo caso, in quanto la funzione rappresentata è più complessa. Essa non è più una semplice combinazione di funzioni

caratteristiche, bensì è lineare a tratti. Il primo problema è quello di calcolare la proiezione t di un punto generico (x, y) sulla retta $r : x + y = 0$. Questa è data da $t = \rho(x, y)\cos\left(\alpha(x, y) + \frac{\pi}{4}\right)$ dove $\rho(x, y) = (x^2 + y^2)^{\frac{1}{2}}$ e $\alpha(x, y) = \arctan\left(\frac{y}{x}\right)$ (se $x = 0$, si ha naturalmente che $\arctan\left(\frac{y}{x}\right) = sgn(y)\frac{\pi}{4}$). E questo ci permette di calcolare i valori x^- e x^+ e quindi di individuare i nodi K^- e K^+. A questo punto se pongo:

$$\begin{cases} m' = \min\{R(i).width, R(i).height\} \\ M' = \max\{R(i).width, R(i).height\} \\ m = \frac{m'}{\sqrt{2}}\ ,\ M = \frac{M'}{\sqrt{2}} \end{cases}$$

si ha, seguendo la traccia dell'altro algoritmo:

```
01 se K(i-) == K(-) allora (K-).val = K(i-).val
02   altrimenti (K-).val = K(i-).val + q(-) *( (x-) - K(i-).x )
03 per ogni nodo K(i) tale che K(-).x < K(i).x < (x-) + m
04   K(i).val = K(i-1).val + 2 * ( K(i).x - (x-) )
05 se non esiste un nodo K((i-)') tale che K((i-)').x == (x-) + m
     allora creo un nodo K(-)' tale che K(-)'.x = x(-) + m
06 altrimenti pongo K(-)' = K((i-)')
07 se non esiste un nodo K((i+)') tale che K((i+)').x == (x-) + M - m
     allora creo un nodo K(+)' tale che K(+)'.x = x(-) + M - m
08 altrimenti pongo K(+)' = K((i+)')
09 per ogni nodo K(i) tra K(-)' e K(+)' compresi
     K(i).val = 2 m
10 per ogni nodo K(i) tra K(+)' e K(+) esclusi
     K(i).val = K(i).val + 2 * ( (x+) - K(i).x )
11 se K(i+) == K(+) allora (K+).val = K(i+).val
12   altrimenti (K+).val = K(i(+)-1).val + q(+) *( (x+) - K((i+)-1).x )
```

dove i termini q^+e q^- sono dati da

$$\begin{cases} q^- = \frac{K(i^-+1).val - K(i^-).val}{K(i^-+1).x - K(i^-).x} \\ q^+ = \frac{K(i^+).val - K(i^+-1).val}{K(i^+).x - K(i^+).x} \end{cases}$$

utilizzando come valore di $K(i^+ - 1).val$ quello precedente alla modifica della riga 10. E con questo l'esposizione dell'algoritmo è completata.

Appendice C

Esempio di calcolo di una funzione di taglia

Nel terzo capitolo abbiamo esposto, seguendo la traccia data da D'Amico, l'algoritmo per ridurre un grafo di taglia e calcolarne la funzione di taglia. In questa appendice ci sembra utile presentare un esempio di calcolo di una funzione di taglia, presentando così l'algoritmo passo passo su un caso concreto. Consideriamo l'immagine (utilizzata anche nell'esperimento che abbiamo eseguito) `intersezioni26.svg` e la funzione misurante di tipo $SALTI$ relativa all'asse delle ordinate.

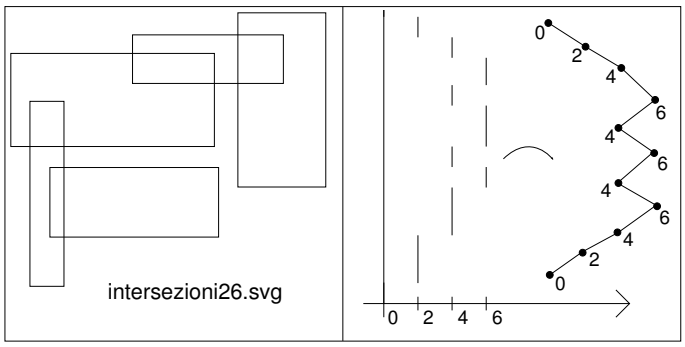


Figura C.1: Funzione misurante $SALTI$ associata a `intersezioni26.svg`.

Si verifica dunque che la funzione misurante si può rappresentare, con la

notazione della lista dinamica che abbiamo introdotto, come una successione di punti di cui indichiamo solo il parametro *val*:

$$0 \succ 2 \succ 4 \succ 6 \succ 4 \succ 6 \succ 4 \succ 2 \succ 0 \succ 0$$

Se riordiniamo i punti della lista in ordine crescente rispetto a *val* otteniamo un grafo di questo tipo:

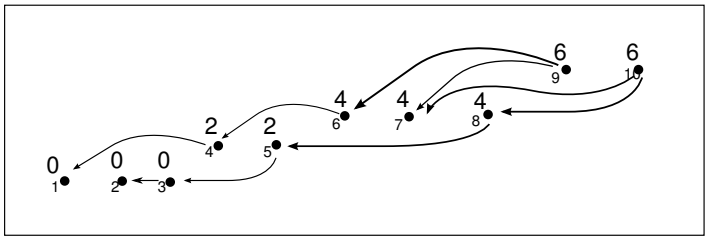


Figura C.2: Riduzione del grafo di taglia - 1.

in cui i nodi sono indicati con il numero piccolo in basso a sinistra e il loro valore è dato dal numero grande in alto a sinistra; le frecce costituiscono gli archi orientati. Procediamo ora con la riduzione (si veda in figura sotto):

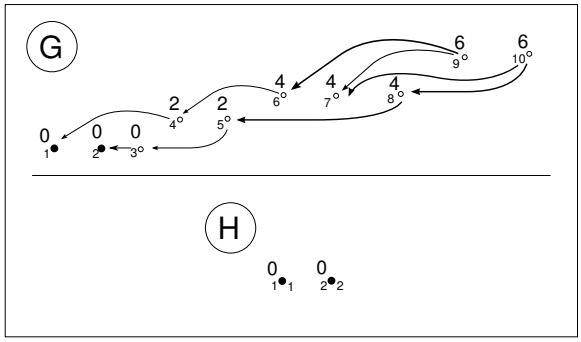


Figura C.3: Riduzione del grafo di taglia - 2.

indicheremo con un pallino pieno i nodi di G già presi in considerazione e con un pallino vuoto quelli ancora da processare. I primi due nodi che si considerano (l'1 e il 2) entrambi non hanno archi uscenti per cui diventano

un nodo del grafo H (che avrà le stesse notazioni grafiche di G, con in più un numero piccolo in basso a destra dei nodi per indicare il loro "nome" in H) e in parallelo costituiscono due classi di elementi separate. Indicheremo le classi tra parentesi graffe, il primo elemento sarà il rappresentante della classe. Tramite il suo numero si associa ad esso un nodo di G che corrisponde ad un nodo di H. A questo punto le classi sono $\{1\}$ e $\{2\}$.
Quando si va ad inserire il terzo nodo, esso ha un solo arco uscente verso la classe $\{2\}$, dunque non diventa un nodo di H ed entra a far parte di tale classe (che conserva il suo rappresentante). Lo stesso accade quando si aggiungono i nodi $4, 5, 6, 8$. Mentre quando si aggiunge il nodo 7 esso si comporta come l'1 e il 2, diventa un nodo di H e costituisce una classe a sé stante. Dunque nel momento prima di inserire il nodo 9 la situazione è:

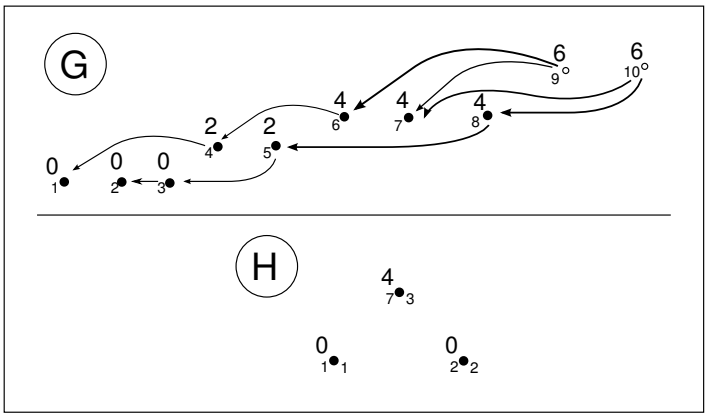


Figura C.4: Riduzione del grafo di taglia - 3.

dove le classi sono $\{1, 4, 6\}$, $\{2, 3, 5, 8\}$ e $\{7\}$. Quando si inserisce il nodo 9 questo ha due archi uscenti, verso i nodi 6 e7 di G. Dunque le classi individuate sono quelle di 1 e 7. Il nuovo nodo diventa un nodo di H da cui escono due archi verso i nodi rappresentati dalle due classi (si veda l'immagine nella prossima pagina), dove le classi diventano $\{9, 1, 4, 6, 7\}$ e $\{2, 3, 5, 8\}$. Infine si passa a considerare l'ultimo nodo, il 10. Questo ha due archi uscenti verso i nodi 7 e 8 questi individuano rispettivamente le due classi di elementi

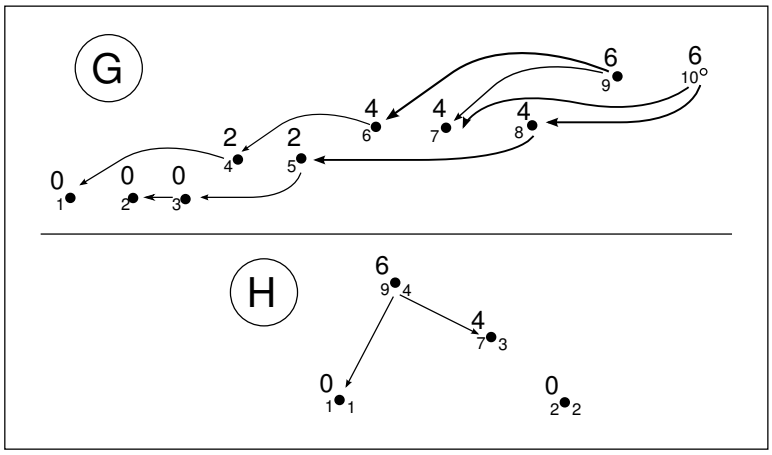


Figura C.5: Riduzione del grafo di taglia - 4.

elencate sopra, rappresentate dai nodi 4 e 2 di H. Dunque si aggiunge tale nodo (che diventa il 5) ad H, lo si connette ai due elencati sopra e si uniscono le classi di elementi in un unico insieme. Quando si connette il nuovo nodo a 4 si osserva che essi hanno lo stesso valore (6), per cui si cancella il nodo 4 e si trasferiscono gli archi da lui uscenti in archi che partono da 5. Il risultato è l'albero H, che costituisce una Δ^*-riduzione totale di G, dato da:

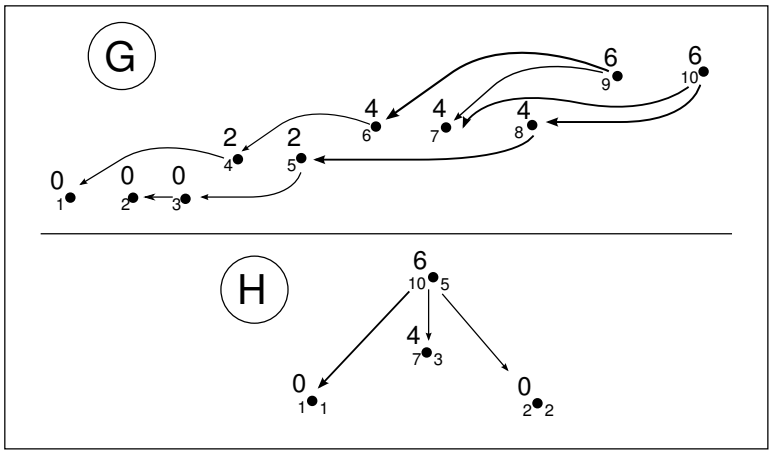


Figura C.6: Riduzione del grafo di taglia - 5.

A questo punto si può passare alla seconda parte dell'algoritmo, quella

del calcolo vero e proprio delle funzione di taglia. Si va a cercare la foglia con valore più alto, che nel nostro caso è il nodo 3. Non è una radice per cui si crea un punto angolare della funzione di taglia con coordinate il valore di 3 e il valore di suo padre: $\ell_G = ((4, 6))$. Si passa poi ad eliminare il nodo 3 in modo che l'albero diventa:

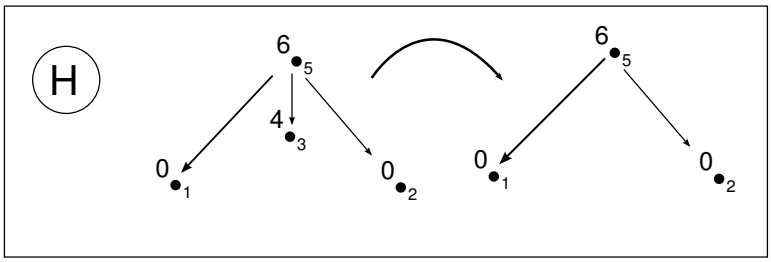


Figura C.7: Calcolo della funzione di taglia - 1.

Si ripete il procedimento, questa volta la foglia con valore massimo è data indifferentemente dai nodi 1 o 2. Supponiamo di scegliere l'1. Anche questo ha un padre per cui si aggiunge un altro punto angolare alla funzione di taglia che diventa $\ell_G = ((0, 6), (4, 6))$. Si passa a cancellare la foglia, in questo modo però il nodo padre rimane con un solo figlio, per cui si applica una Δ^*-mossa di tipo 2 e l'albero diventa:

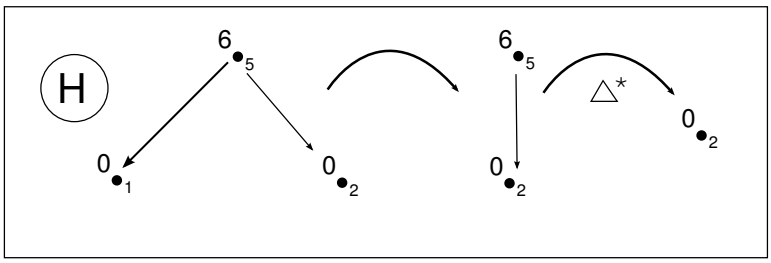


Figura C.8: Calcolo della funzione di taglia - 2.

Infine con l'ultima iterazione si prende in considerazione il nodo 2 che è anche radice per cui si aggiunge alla funzione di taglia una retta angolare, in

modo che diventa: $\ell_G = ((0, \infty), (0, 6)\,(4, 6))$. Non essendoci più nodi in H questa è la funzione di taglia.

Appendice D

La struttura del Sizetool

In questa appendice descriviamo più nel dettaglio la struttura generale del software che abbiamo scritto, in modo da facilitare la comprensione del codice a chi volesse studiarlo e/o modificarlo. L'input dell'algoritmo sarebbe dato da un insieme di immagini. Per renderlo il più possibile variabile abbiamo pensato di creare una cartella (che abbiamo chiamato `test`) al cui interno mettere tutte le immagini da elaborare. Al programma viene passato in input un file formato testo (`test_set.txt`) al cui interno sono scritti in colonna i nomi dei file (senza estensione) che contengono le immagini. Il programma legge uno per uno i nomi e li memorizza in un apposito vettore di stringhe (`test_list`) che permetterà in ogni momento di ricostruire su che immagine si sta lavorando. Successivamente inizia l'algoritmo vero e proprio:

```
Input: test_list = vettore dei nomi delle immagini;
       sia N la lunghezza di test_list
01 per j = 0 -> N
02    apri il file \test\test_list[j].svg (= immagine j)
03    memorizza le informazioni grafiche contenute
04    per k = 0 -> 2 (3 e' il numero delle funzioni misuranti)
05       calcola la funzione misurante k dell'immagine j
06       calcola la relativa funzione di taglia dell'immagine j
07       memorizza la funzione di taglia su un file
```

A questo punto dell'algoritmo sono state calcolate tutte le funzioni di taglia, relative ad ogni funzione misurante, di ciascuna immagine. Queste funzioni di taglia vengono memorizzate in un file `test_list[j].szf` (formato che abbiamo creato appositamente) e vengono salvate in cartelle separate per

funzione misurante. Per esempio relativamente alle tre funzioni misuranti della famiglia *SALTI* abbiamo creato altrettante cartelle `oriz`, `vert` e `diag`. Il formato che abbiamo creato è un file di testo al cui interno sono scritte in colonna le coppie di coordinate dei punti angolari della funzione di taglia. Per esempio la funzione di taglia calcolata nell'appendice C viene memorizzata come:

```
0.000000 infinito
0.000000 6.000000
4.000000 6.000000
```

Calcolate le funzioni di taglia, l'algoritmo può passare alla seconda parte, quella di elaborazione dei risultati, che serve per effettuare degli esperimenti. Viene creata una matrice `hausd_dist` di elementi di tipo `ext_real_t` (la struttura dati per gestire i numeri reali estesi) di dimensione $N \times 5 \times N$ dove il 5 dipende dal fatto che abbiamo 3 funzioni di taglia per immagine e che vogliamo creare 2 elaborazioni delle distanze (quella normalizzata e quella media).

```
01 per j = 0 -> N
02    per k = 0 -> 2
03       considero la funzione di taglia k dell'immagine j
04       per i = 0 -> N
05          considero la funzione di taglia k dell'immagine i
06          calcolo la distanza tra le due hausd_dist[j][k][i]
```

A questo punto si capisce perchè stato necessario memorizzare su dei file tutte le funzioni di taglia. Infatti è il modo più comodo per avere reperibili in ogni momento i valori delle funzioni, che ci servono quando dobbiamo calcolare la matrice delle distanze. Dopo questo calcolo l'algoritmo prevede la parte di gestione dell'output. Per ogni immagine j fissata, identifica la matrice di distanze `hausd_dist[j][k][i]` e la trasforma nei due vettori delle distanze medie e normalizzate che memorizza rispettivamente in `hausd_dist[j][3][i]` e `hausd_dist[j][4][i]` e poi opera come è stato dettagliatamente descritto nella parte D del quarto capitolo, riordinando in modo crescente i vettori e creando i file in cui scrivere le "classifiche". E con questo si chiude l'algoritmo.

Bibliografia

[1] C. Berge, *Theorie des graphes et ses applications*, Dunod, Parigi, 1958.

[2] A. Bertossi, *Algoritmi e srtutture dati*, UTET, Torino, 2000.

[3] A. Cerri, M. Ferri, P. Frosini, D. Giorgi, *Keypics: free-hand drawn iconic keywords*, Interational Journal of Shape Modelling, **13**, pp. 1-13, 2007

[4] P. G. Ciarlet, *Introduzione all'analisi numerica matriciale e all'ottimizzazione*, Masson, Milano, 1989.

[5] A. Cerri, M. Ferri, D. Giorgi, *Retrieval of trademark images by means of size fnuctions*, Graphical Models, **68**, pp. 451-471, 2006

[6] M. D'Amico *A new optimal algorithm for computing size function of shapes*, CVPRIP Algorithms III, Proc. International Conference on Computer Vision, Pattern Recognition and Image Processing, pp 367-370, Atlantic City, 2000.

[7] M. D'Amico, *Aspetti computazionali delle Funzioni di Taglia*, Tesi di Dottorato, Università di Bologna, Bologna, 2002.

[8] M. D'Amico, P. Frosini, C. Landi, *Natural pseudo-distance and optimal matching between reduced size functions*, preprint.

[9] B. Di Fabio, *Calcolo e visualizzazione delle funzioni di taglia di curve spline*, Tesi di Laurea, Università di Bologna, Bologna, 2003.

[10] P. Frosini, *Connection between Size Functions and critical points*, Math. Meth. Appl. Sci., **19**, pp. 555-569, 1996.

[11] P. Frosini, *Discrete computation of size functions*, J. Combin. Inform. System Sci., **17** (3-4), pp. 232-250, 1992.

[12] P. Frosini, C. Landi, *Size Functions and formal series*, Appl. Algebra Engrg. Comm. Comput., **12**, pp. 327-349, 2001.

[13] P. Frosini, C. Landi *Size theory as topological tool for computer vision*, Pattern Rec. and Image Analysis, **9** (4), pp. 596-603, 1999.

[14] L. Muracchini, *Introduzione alla teoria dei grafi*, Boringhieri, Torino, 1967.

[15] A. Paoluzzi *Informatica grafica: metodi, algoritmi, programmi per il disegno automatico col calcolatore*, NIS, Roma, 1987

[16] D.F.Rogers, *Mathematical Elements for Computer Graphics*, McGraw-Hill, 1989.

[17] R. Sedgewick, *Algorithms in C*, Addison-Weslat, 1990.

[18] P. Serafini, *Ottimizzazione*, Zanichelli, Bologna, 2004.

[19] http://www.w3.org/Graphics/SVG

[20] http://www.svgunict.altervista.org/index.html

[21] http://xml.html.it/guide/leggi/54/guida-svg

[22] http://evolution.genetics.washington.edu/phylip.html

[23] http://pearl.cs.pusan.ac.kr/phylodraw/