Contents

Introduzione

Un problema classico della ricerca operativa è quello di determinare la più piccola superficie che contiene una famiglia assegnata di figure geometriche. Questo ed altri problemi affini, vengono indicati in letteratura con il nome di *floorplanning design*, *placement problem* o anche come *il problema del calzolaio*.

Consideriamo la seguente particolare istanza: data una famiglia di poligoni convessi (detti pezzi) determinare il più piccolo rettangolo di base fissata in modo che tutti i poligoni siano contenuti nel rettangolo senza intersecarsi. (vedi fig. 1.) Chiamiamo questa istanza col nome di *floorplanning design a base fissata* (da qui in avanti FDB). Com'è facile immaginare numerose sono le applicazioni pratiche che ha questo problema, per esempio progettazione di circuiti VLSI, industria tessile, taglio di laminati, costruzione di gommoni, ecc.

L'idea che ci ha guidato verso la soluzione presentata in questo lavoro è stata quella di utilizzare un metodo classico della teoria dell'ottimizzazione: **trovare una funzione e minimizzarla**. Supponiamo di avere una famiglia di poligoni convessi che disponiamo, in un qualunque modo, su di un rettangolo; la domanda che ci poniamo è: "riusciamo ad avere una misura di quanto questi poligoni si intersecano?" E ancora: "riusciamo ad avere una misura di quanto questi poligoni escono fuori dal rettangolo?"

Vedremo che è possibile rispondere a queste domande costruendo una particolare funzione a valori reali Ψ. Vedremo, inoltre, come sia possibile partendo da Ψ costruire due modelli: il primo è chiamato **PLMT** (da PLaceMenT) e formalizza il seguente problema: *dato un rettangolo e data una famiglia di poligoni convessi, posizionare "al meglio" i poligoni sul rettangolo.* Quando diciamo "al meglio" intendiamo dire che i poligoni vanno posizionati in modo da non intersecarsi tra di loro e da non uscire fuori dal rettangolo, se è possibile. Il secondo è chiamato **FDB** e formalizza il seguente problema:*data una famiglia di poligoni convessi, determinare il più piccolo rettangolo di base b tale che nessun poligono esca dal rettangolo e che nessuna coppia di poligoni si intersechi.* Poiché la funzione Ψ è il cuore dei due modelli, è naturale studiarne le proprietà principali. Quella di più facile dimostrazione è la non convessità; questo fatto lascia

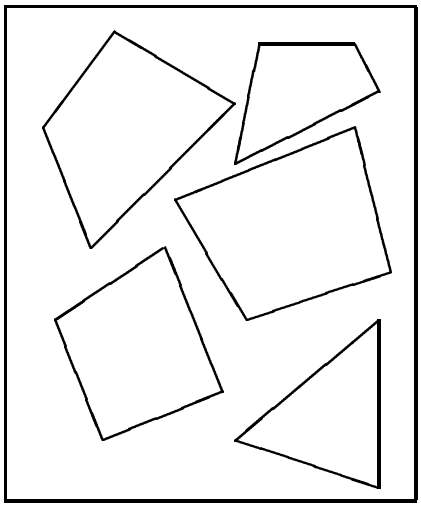


Figura 1: Una soluzione non ottima per FDB.

prevedere l'uso di algoritmi d'ottimizzazione non convessa per la risoluzione di **PLMT** e di **FDB**.

Facciamo notare che la formulazione del modello per PLMT è fatta sul continuo; in letteratura si trovano anche modelli discreti (vedi per esempio [3, 2]) e euristiche (vedi per esempio [1, 4]).

Studieremo altre due importanti proprietà di Ψ: la continuità e la differenziabilità. La prima si dimostra usando nozioni classiche riguardanti la continuità di funzioni tra spazi topologici. Per quanto riguarda la seconda la situazione è molto più critica, infatti con un semplice esempio faremo vedere che Ψ non è ovunque differenziabile: questo implica che al concetto di "gradiente" va sostituito il concetto di "direzione di decrescita".

Presenteremo anche gli algoritmi utilizzati per calcolare il valore e una direzione di decrescita per Ψ in un punto. Una cosa che va notata è che gli algoritmi proposti non sono *iterativi*, cioè non tendono ad approssimare la soluzione, bensì sono *diretti* nel senso che la determinano in un numero di iterazioni noto a priori. Questa qualità si apprezza meglio se si considera che Ψ è un integrale doppio.

Una problematica molto delicato, che per ora rimane aperta, è la determinazione di un algoritmo che risolva bene e in poco tempo FDB. Noi ne presenteremo due: il primo si basa su un metodo di minimizzazione non convessa

per funzioni reali, il secondo è un euristica che sfrutta anche i risultati teorici trovati. Verranno anche presentati i risultati di alcune prove d'esecuzione, mediante le quali cercheremo di stimare l'errore relativo che viene commesso quando risolviamo FDB con gli algoritmi presentati.

Nell'ultimo capitolo sono mostrati alcuni possibili sviluppi degli argomenti che presenteremo: estensione del modello per FDB al caso di poligoni non convessi, espressione di particolare tipi di vincoli e miglioramenti implementativi.

Questo lavoro è concluso da un'appendice contenente definizione e proprietà di alcuni algoritmi noti in letteratura, che riportiamo per comodità.

Capitolo 1

Survey

Come già accennato, esistono molti settori industriali (microelettronica, tessile, lavorazione delle pelli, ecc.) che richiedono la determinazione del più piccolo rettangolo che contiene una famiglia assegnata di rettangoli o, più in generale, di poligoni. Quindi esistono già in letteratura dei lavori che affrontano e cercano di risolvere il problema del "compattamento".

Facciamo presente che la maggiorparte degli algoritmi che vedremo sono usati nella progettazione dei circuiti VLSI; nella loro presentazione venivano quindi trattati anche problemi non strettamente legati alla compattazione, per esempio la minimizzazione della lunghezza delle linee di interconnessione tra i vari blocchi, che noi tralasceremo per semplicità.

1.1 Compattamento mediante grafo dei vincoli

Questa tecnica consiste di due passi:

- Costruzione di due *grafi dei vincoli* che indicano la posizione relativa dei blocchi rispettivamente per l'asse delle x e l'asse delle y;
- Utilizzo di un algoritmo dei cammini massimi per il compattamento prima orizzontale e poi verticale.

Generalmente si assume di avere già una disposizione iniziale dei blocchi (nel caso dei circuiti VLSI questa deriva da dei vincoli di progetto) e quindi di poter esprimere per ogni coppia (i, j) di blocchi una coppia di disuguaglianze:

$$x_i + k_{ij} \leq x_j,$$
$$y_i + h_{ij} \leq y_j.$$

La prima dice che il blocco i-esimo è più a sinistra del blocco j-esimo e la loro distanza lungo l'asse delle x deve essere superiore a k_{ij} unità. In modo analogo si spiega la seconda disequazione.

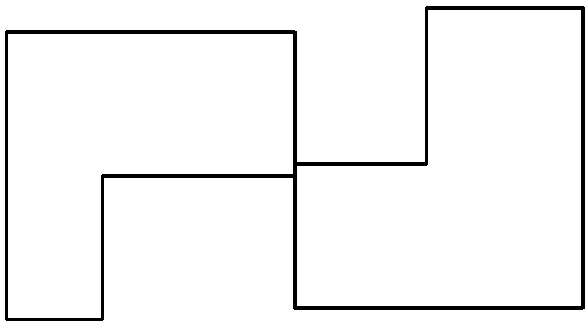
Quindi compattare il layout di partenza, per esempio lungo l'asse delle x, equivale a risolvere un sistema di disuguaglianze del tipo appena visto. Consideriamo il seguente grafo dei vincoli così costruito: per ogni disuguaglianza $x_i + k_{ij} \leq x_j$ esiste un arco diretto dal nodo i al nodo j ed etichettato con k_{ij}. Inoltre il grafo dei vincoli deve contenere un nodo x_0 che rappresenta il bordo sinistro verso il quale vengono schiacciati tutti i pezzi. A questo punto per risolvere il sistema di disequazioni e quindi trovare una disposizione compatta dei blocchi basta assegnare ad ogni x_i il valore del cammino più lungo che parte da x_0 e raggiunge x_i, ad ogni y_i il valore del cammino più lungo che parte da y_0 e raggiunge y_i.

1.2 Compattamento bidimensionale

L'algoritmo presentato precedentemente ha un grosso limite dovuto al fatto che i blocchi si possono spostare lungo una sola direzione alla volta. Nella figura 1.1 viene mostrato un esempio che evidenzia questa limitazione, infatti il blocco 2 non può essere spostato completamente verso il bordo sinistro perché uno dei suoi vertici tocca il blocco 1. Per questo motivo sono stati sviluppati algoritmi che cercano di compattare spostando i blocchi contemporaneamente su entrambe le direzioni.

Un'euristica di un certo interesse si ispira ad un procedimento detto *raffinamento di zona* usato per la purificazione di lingotti di cristallo. Data una disposizione iniziale dei blocchi cerchiamo di eliminare le impurità, che nel nostro caso corrispondono agli spazzi vuoti. Partendo da un lato del layout iniziale e procedendo per righe, i blocchi vengono selezionati a uno a uno e spostati nella riga sottostante cercando di trovare una posizione che aumenti il tasso di compattazione. Ovviamente sono permessi sia spostamenti orizzontali che verticali.

Il fatto che ad ogni iterazione viene preso in considerazione solo un piccolo gruppo di pezzi, rende l'algoritmo computazionalmente efficace. Le soluzioni che si ottengono sono comunque lontane dall'ottimo, ciò è dovuto al fatto che il problema che si tenta di risolvere è NP-completo.



1 2

Figura 1.1: C'è bisogno di spostamenti bidimensionali.

1.3 Compattamento con algoritmi paralleli

Anche questo metodo prevede di avere in input una disposizione iniziale dei blocchi la quale è, per esempio, indotta da dei vincoli di progettazione (vedi anche il paragrafo 1.1). Partendo dalla disposizione iniziale è possibile costruire un albero detto di *partizionamento* che descrive i vincoli spaziali dei blocchi. Più esattamente ogni nodo dell'albero rappresenta una regione rettangolare ρ contenente blocchi. La struttura di un nodo è la seguente:

- coppia di coordinate rappresentanti il punto del piano dove disporre il rettangolo ρ;
- un sottoinsieme di moduli;
- un'etichetta che può appartenere a $\{h, v, t\}$ e indica se i rettangoli figli sono ottenuti tramite un taglio orizzontale (h), verticale (v) oppure se ρ non ha figli.

L'idea di funzionamento dell'algoritmo è quella di partire dai nodi terminali dell'albero di partizionamento e trovare per questi una disposizione ottima che non violi i vincoli spaziali. Risolte tutte le foglie si aggregano le soluzioni in esse contenute per costruire le soluzioni dei nodi padre e così via fino alla radice.

Per ottenere buone soluzioni finali è necessario che i nodi foglia abbiano il maggior numero possibile di blocchi. Dall'altro canto però l'aumento del numero di blocchi nei nodi foglia aumento notevolmente il tempo di calcolo,

ricordiamo infatti che i nodi foglia vengono risolti all'ottimo. Per ovviare a questo inconveniente sono stati adottati modelli di calcolo parallelo che sfruttano due particolari caratteristiche dell'approccio:

- il calcolo della soluzione associata a due nodi foglia distinti può essere fatto in parallelo;
- esistono algoritmi paralleli per la dislocazione ottima dei rettangoli all'interno dei nodi foglia.

In definitiva possiamo dire che all'aumentare dei processori aumenta la precisione della soluzione determinata e diminuisce il tempo di calcolo.

1.4 Compattamento mediante espressioni Polish normalizzate

Questa tecnica suppone che i blocchi non abbiano le loro dimensioni fissate rigidamente, ma che sia possibile indicare un intervallo dei loro possibili *rapporti d'aspetto* (rapporto tra base ed altezza dei blocchi); questo permette di costruire layout molto compatti perché ogni blocco può assumere tutte le forme consentite dal proprio rapporto d'aspetto.

Un modulo generico viene indicato con una terna (A, r, s), $r \leq s$; questa sta ad indicare che, posto w la lunghezza della base e con h la lunghezza dell'altezza, dobbiamo avere:

- $wh = A$;
- $r \leq \frac{h}{w} \leq s$ se il blocco ha l'orientamento fisso;
- $r \leq \frac{h}{w} \leq s \vee \frac{1}{s} \leq \frac{h}{w} \leq \frac{1}{r}$ se il blocco ha orientamento libero.

Fissato anche il rapporto d'aspetto del layout finale (p, q), $p \leq q$, definiamo una *soluzione ammissibile* come un rettangolo R con rapporto d'aspetto compreso tra p e q. R è diviso da linee orizzontali e verticali in n (numero dei blocchi) rettangoli disgiunti. L'i-esimo rettangolo deve essere sufficientemente grande da contenere l'i-esimo blocco. Il *costo* di una soluzione ammissibile è dato dall'area del rettangolo R.

Una sequenza $\alpha_1, \ldots, \alpha_{2n-1}$ di $2n-1$ elementi dell'alfabeto $\{1, 2, \ldots, n, *, +\}$ è una *sequenza Polish normalizzata* se e solo se valgono le seguenti tre relazioni:

- ogni $i \in \{1, \ldots, n\}$ appare esattamente una volta nella sequenza;

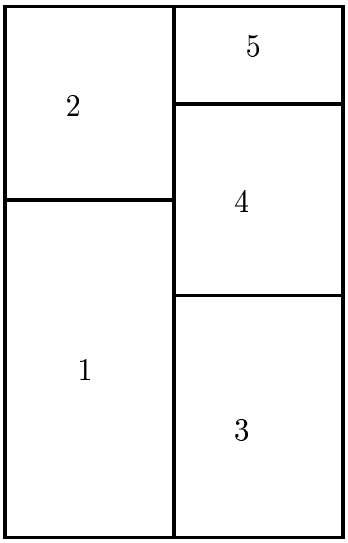


Figura 1.2: Il layout corrispondente all'espressione "1 2 + 3 4 + 5 + *".

- per ogni k, $1 \leq k \leq 2n-1$, deve essere che il numero di *operatori* ($\{*, +\}$) in $\{\alpha_1, \ldots, \alpha_k\}$ è minore del numero di *operandi* ($\{1, \ldots, n\}$) in $\{\alpha_1, \ldots, \alpha_k\}$.
- nell'espressione non possono apparire due operandi uguali consecutivi.

Si può dimostrare che tra le soluzioni ammissibili e le espressioni Polish normalizzate lunghe $2n-1$ esiste una corrispondenza 1–1. La figura 1.2 mostra un esempio di espressione Polish normalizzata e il layout corrispondente.

Le seguenti tre mosse definiscono nel modo ovvio un *intorno* per una espressione Polish normalizzata:

- scambia due operandi adiacenti;
- complementa le catene di operatori;
- scambia gli elementi di una coppia (operatore, operando) adiacenti.

Con questa definizione siamo ora in grado di costruire un *simulated annealing* (vedi appendice). I risultati ottenuti sono molto incoraggianti, infatti nel caso in cui tutti i blocchi sono ad aspetto variabile l'area sprecata si aggira intorno al 0.5%–4%.

1.5 Compattamento di figure generiche

Finora abbiamo visto il problema del compattamento di rettangoli. Generalizzando possiamo voler compattare un insieme di figure geometriche che non sono necessariamente dei rettangoli. Anche per questo problema in letteratura si trova una metodologia di risoluzione basata sul *simulated annealing*. Facendo riferimento alla descrizione del simulated annealing presente in appendice, descriviamo solo le caratteristiche principali dipendenti dal problema in esame.

Supponiamo di dover compattare queste figure geometriche all'interno di una striscia la cui ampiezza è fissata, l'estremità sinistra è fissa e l'estremità destra è libera di muoversi in modo da poter allungare o accorciare la striscia stessa.

La funzione da minimizzare è composta da tre addendi:

- l'area delle intersezioni tra i pezzi;
- la superficie che si guadagna impaccando i pezzi verso l'estremità sinistra della striscia;
- l'area delle parti di pezzi fuoriuscenti dalla striscia.

L'intorno di una soluzione è calcolabile mediante:

- spostamenti orizzontali e verticali dei pezzi;
- scambio di pezzi;
- rotazioni simmetriche dei pezzi;
- avvicinamento dell'estremo destro a quello sinistro.

Il metodo fornisce generalmente soluzioni migliori di altri metodi euristici deterministici; il miglioramento è stimabile intorno al 3%. Il tempo di calcolo è alto (25–30 minuti su un DPS8) e a volte capita che la soluzione finale presenta dei pezzi sovrapposti.

Capitolo 2

Il modello

In questo capitolo definiamo i modelli **PLMT** e **FDB** appena accennati nell'introduzione. Nel paragrafo 2.1 diamo una definizione preliminare che formalizza, in modo sufficiente per i nostri fini, l'idea di "pezzo". Con questa definizione siamo in grado, nel paragrafo 2.2, di definire un modello per **PLMT**. Partendo da questo definiamo anche un modello per **FDB** e ne stabiliamo alcune caratteristiche.

2.1 Funzioni indicatrici

Diamo prima un'idea di cosa vogliamo modellare per poi fornire la definizione esatta. Consideriamo un rettangolo con i lati paralleli agli assi del piano cartesiano; com'è facile intuire, un poligono vogliamo poterlo spostare all'interno del rettangolo a nostro piacimento. Più esattamente vogliamo poter specificare una coppia di coordinate che identificano il punto del rettangolo dove piazzare il baricentro del pezzo e un parametro che ne indica l'orientamento.

Uno strumento matematico che ci consente di formalizzare agevolmente la nostra idea è la funzione indicatrice, o meglio una sua generalizzazione, che ha il seguente tipo:

$$f_i : \mathbf{R^2} \times [0, 2\pi] \times \mathbf{R^2} \to \{1, 0\}$$

con i che varia tra 1 e $NumPezzi$ (il numero di pezzi che vogliamo trattare).

La definizione di f_i dovrà apparire come:

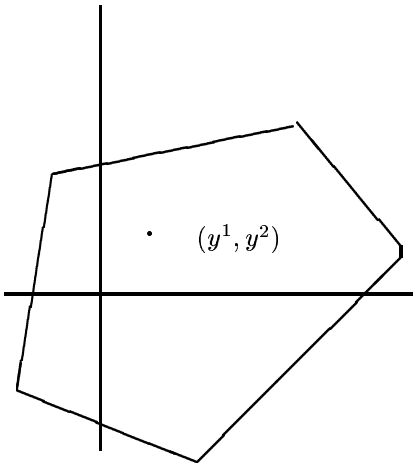


Figura 2.1:

$$f_i(x_{i1}, x_{i2}, x_{i3}, y_1, y_2) = \begin{cases} 1 & \text{se } (y_1, y_2) \text{ appartiene al poligono } i\text{-esimo} \\ & \text{con } \underline{\text{centro}} \text{ di coordinate } (x_{i1}, x_{i2}) \text{ e} \\ & \underline{\text{orientamento}}\ x_{i3} \\ 0 & \text{altrimenti} \end{cases}$$

Quando sarà necessario useremo $\overline{x}_i$ al posto di (x_{i1}, x_{i2}, x_{i3}) e $\overline{y}$ al posto di (y_1, y_2).

Siano $\{Pol_i\}_{i=1,\ldots,NumPezzi}$ una famiglia di sottoinsiemi di $\mathbf{R^2}$ tale che

$$(0,0) \in Pol_i :$$

Pol_i rappresenta l'i-esimo pezzo che vogliamo piazzare.

Sia $(y^1, y^2) \in Pol_i$ un generico punto (vedi fig.2.1); se lo vogliamo ruotare di un angolo x_3 e lo vogliamo traslare con un vettore (x_1, x_2) otteniamo il nuovo punto

$$(y_1, y_2) = (y^1, y^2)R(x_3) + (x_1, x_2),$$

dove $R(x_3)$ rappresenta la matrice di rotazione:

$$R(x_3) = \begin{pmatrix} \cos x_3 & -\sin x_3 \\ \sin x_3 & \cos x_3 \end{pmatrix}.$$

Da questa considerazione otteniamo che le f_i si definiscono come:

$$f_i\ (x_{i1}, x_{i2}, x_{i3}, y_1, y_2\ ) = \begin{cases} 1 & \text{se } [(y_1, y_2) - (x_{i1}, x_{i2})]R^{-1}(x_{i3}) \in Pol_i \\ 0 & \text{altrimenti} \end{cases}$$

dove

$$R^{-1}(x_{i3}) = \begin{pmatrix} \cos x_{i3} & \sin x_{i3} \\ -\sin x_{i3} & \cos x_{i3} \end{pmatrix}.$$

Per comodità di notazione diamo anche un nome al poligono i-esimo spostato di un vettore (x_{i1}, x_{i2}) e ruotato di un angolo x_{i3}:

Definizione 2.1

$$Pol_i(x_{i1}, x_{i2}, x_{i3}) = \{(y_1, y_2) \in \mathbf{R^2} \mid f_i(x_{i1}, x_{i2}, x_{i3}, y_1, y_2) = 1\}.$$

2.2 Definizione del modello

Possiamo ora definire i modelli per PLMT e FDB. Usando una notazione alla object oriented, indichiamo con

$$S' = [0, S'.base] \times [0, S'.altezza]$$

il rettangolo di base $S'.base$ e con vertice sud-ovest in $(0, 0)$ che dovrà contenere i pezzi e con

$$S = [-d, S'.base + d] \times [-d, S'.altezza + d]$$

un rettangolo contenente S'. Il valore di $d > 0$ andrà scelto in modo tale che comunque piazziamo il baricentro di un pezzo p in S', p è contenuto completamente in S (vedi Fig. 2.2):

$$\forall i = 1, \ldots, NumPezzi,\ \forall (x_{i1}, x_{i2}) \in S',\ \forall x_{i3} \in [0, 2\pi] :\ Pol_i(\overline{x}_i) \subset S.$$

Siano ($S.minx$,$S.miny$) (rispettivamente ($S'.minx$,$S'.miny$)) le coordinate del vertice inferiore sinistro di S (rispettivamente S') e ($S.maxx$,$S.max$ y) (rispettivamente ($S'.maxx$,$S'.maxy$)) le coordinate del vertice superiore destro di S (rispettivamente S').

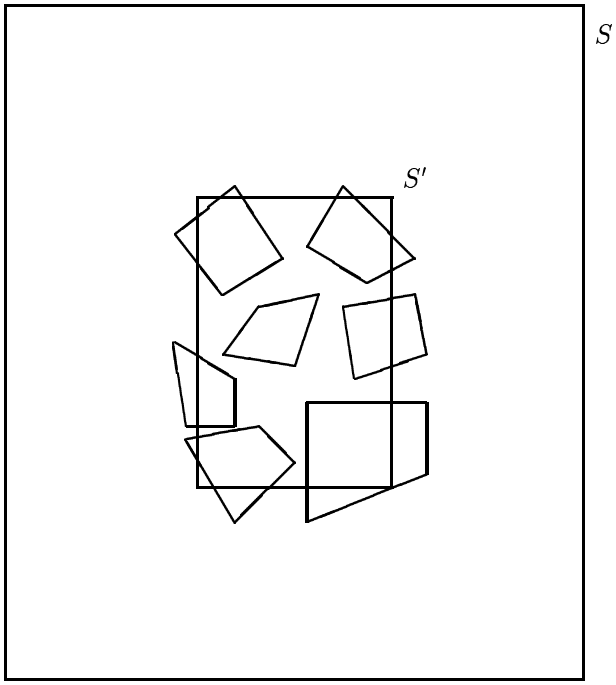


Figura 2.2: Un esempio di S' e S

Definiamo, inoltre, una particolare funzione indicatrice f_0 che ha lo scopo di rappresentare $S \setminus S'$:

$$f_0(x_{01}, x_{02}, x_{03}, y_1, y_2) = \begin{cases} 1 & \text{se } (y_1, y_2) \in S \setminus S', \\ 0 & \text{altrimenti.} \end{cases}$$

Notiamo che a destra dell'uguale mancano le x_{01}, x_{02}, x_{03}; questo implica che il valore ritornato dalla f_0 non vi dipende. Abbiamo deciso di far dipendere, in modo fittizio, la f_0 anche da x_{01}, x_{02}, x_{03} per avere una notazione più uniforme.

La proposizione e la definizione che seguono ci permetteranno di comprendere meglio la definizione del modello.

Proposizione 2.1 *L'area del poligono i-esimo è pari a:*

$$\begin{aligned} \int_S f_i(x_{i1}, x_{i2}, x_{i3}, y_1, y_2) dy_1 dy_2 \quad &= \quad \int_S Pol_i(x_{i1}, x_{i2}, x_{i3}) \\ & \quad \forall i = 1, \ldots, NumPezzi \\ & \quad \forall (x_{i1}, x_{i2}) \in S' \\ & \quad \forall x_{i3} \in [0, 2\pi[\end{aligned}$$

Definizione 2.2 (Funzione obiettivo Ψ)

$$\Psi(\overline{x}_1, \ldots, \overline{x}_{NumPezzi}) = \int_S \left(\sum_{i=0}^{NumPezzi} f_i(\overline{x}_i, \overline{y}) \right)^2 d\overline{y}.$$

Ψ è la funzione obiettivo del modello:

Definizione 2.3 (PLMT(PLaceMenT))

$$\begin{aligned} min \quad & \Psi(\overline{x}_1, \ldots, \overline{x}_{NumPezzi}) \\ t.c. \quad & \overline{x}_i \in \mathbf{R^3} \\ & x_{i1} \in [0, S'.base] \\ & x_{i2} \in [0, S'.altezza] \\ & x_{i3} \in [0, 2\pi] \\ & \forall\ i = 1, \ldots, NumPezzi \end{aligned}$$

Con il modello PLMT, come suggerisce il nome, risolviamo il problema del posizionamento dei pezzi all'interno di un rettangolo di dimensione fissata; con la proposizione seguente spieghiamo come:

Proposizione 2.2 *Dati*

$$Z = min\left\{\Psi(\overline{x}_1, \ldots, \overline{x}_{NumPezzi}) \mid \overline{x}_i \in S' \times [0, 2\pi], i = 1, \ldots, NumPezzi\right\},$$

e

$$A = \sum_{i=0}^{NumPezzi} \int_S Pol_i.$$

allora si ha che:

1. *Se $Z = A$ allora possiamo disporre i pezzi su S' senza sovrapposizioni e nella soluzione ottima $(\overline{x}_1, \ldots, \overline{x}_{NumPezzi})$ c'è scritto come mettere i pezzi. In questo caso diciamo che esiste una disposizione ammissibile.*
2. *Se $Z > A$ allora o c'è stata una sovrapposizione di almeno due pezzi, oppure almeno un poligono esce fuori da S'. In questo caso diciamo che non esiste una disposizione ammissibile.*

Dimostrazione

L'idea che sta dietro al modello è che la funzione Ψ tende a diminuire quando allontaniamo due pezzi che sono sovrapposti, infatti Ψ può essere riscritta come:

$$\begin{aligned}
\Psi(\overline{x}_1, \ldots, \overline{x}_{NumPezzi}) &= \int_S \left(\sum_{i=0}^{NumPezzi} f_i(\overline{x}_i, \overline{y}) \right)^2 d\overline{y} = \\
&= \int_S \sum_{i=0}^{NumPezzi} \left(f_i(\overline{x}_i, \overline{y})\right)^2 d\overline{y} + \\
&+ 2 \int_S \sum_{i=0}^{NumPezzi-1} \sum_{j=i+1}^{NumPezzi} f_i(\overline{x}_i, \overline{y}) f_j(\overline{x}_j, \overline{y}) d\overline{y}
\end{aligned}$$

e posto

$$\Psi_0 = \int_S \sum_{i=0}^{NumPezzi} \left(f_i(\overline{x}_i, \overline{y})\right)^2 d\overline{y}$$

$$\Psi_{ij}(\overline{x}_i, \overline{x}_j) = \int_S f_i(\overline{x}_i, \overline{y}) f_j(\overline{x}_j, \overline{y}) d\overline{y}$$

si ha che Ψ_0 è una *quantità* costante; Ψ_{ij} misura l'area dell' intersezione dei pezzi i-esimo e j-esimo e quindi diminuisce quando allontaniamo i due pezzi. Da queste considerazioni si ricava che la funzione

$$\Psi = \Psi_0 + 2 \sum_{i=0}^{NumPezzi-1} \sum_{j=i+1}^{NumPezzi} \Psi_{ij} \tag{2.1}$$

ha minimo pari a $\Psi_0 = A$ quando esiste una disposizione ammissibile per i pezzi, altrimenti ha minimo pari a $\Psi_0 + v > A$ dove v è una misura di quanto i pezzi si intersecano e/o escono fuori S'.

□

Facciamo osservare che il problema di minimo PLMT è equivalente al seguente problema d'esistenza:

Definizione 2.4 ($\exists PLMT$)

$$\begin{aligned} \textit{Determinare} \quad & \overline{x}_i \in \mathbf{R^3},\ i = 1, \ldots, NumPezzi \\ \textit{tale che} \quad & \Psi(\overline{x}_1, \ldots, \overline{x}_{NumPezzi}) = A \\ & x_{i1} \in [0, S'.base] \\ & x_{i2} \in [0, S'.altezza] \\ & x_{i3} \in [0, 2\pi] \\ & \forall i = 1, \ldots, NumPezzi \end{aligned}$$

Notiamo che l'introduzione della f_0 ci permette di esprimere in una forma molto elegante il vincolo che *ogni pezzo deve stare in* S':

$$\forall i\ Pol_i(\overline{x}_i) \subset S'.$$

Introduciamo ora il modello per FDB:

Definizione 2.5 (FDB)

$$\begin{aligned} \min \quad & S'.altezza \\ t.c. \quad & \int_S \left(\sum_{i=0}^{NumPezzi} f_i(\overline{x}_i, \overline{y}) \right)^2 d\overline{y} = A \end{aligned}$$

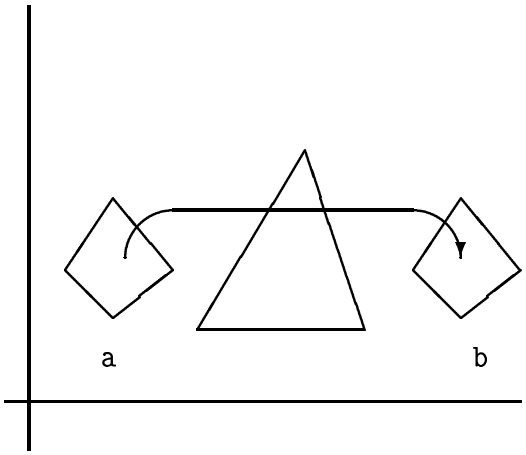


Figura 2.3: Esempio di riferimento

$$x_{i1} \in [0, S'.base] \ \forall i = 1, \ldots, NumPezzi$$
$$x_{i2} \in [0, S'.altezza] \ \forall i = 1, \ldots, NumPezzi$$
$$x_{i3} \in [0, 2\pi] \ \forall i = 1, \ldots, NumPezzi$$
$$S'.altezza > 0$$
$$S = [-d, S'.base + d] \times [-d, S'.altezza + d]$$

Con questo modello abbiamo formalizzato l'idea di trovare l'altezza più piccola per S' per la quale esiste una disposizione ammissibile.

Mostriamo con un semplice controesempio che Ψ non è convessa. Con riferimento alla fig.2.3 abbiamo che se spostiamo il pezzo più piccolo dal punto a al punto b, la funzione ha un andamento come in fig. 2.4.

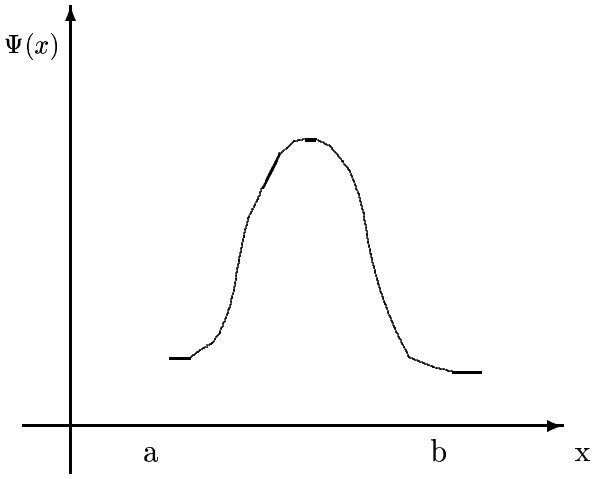


Figura 2.4: Andamento di Ψ da a ad b.

Capitolo 3

Continuità e differenziabilità di Ψ

In questo capitolo discuteremo della continuità e della differenziabilità di Ψ. Nel paragrafo 3.1 dimostreremo che Ψ è continua: a tale fine utilizzeremo alcuni risultati noti di Analisi Matematica sulla continuità di funzioni tra spazi topologici. Nel paragrafo 3.2 studieremo alcune proprietà riguardanti la differenziabilità di Ψ; in particolare, con alcuni controesempi, mostreremo che Ψ non è differenziabile con continuità. Nel paragrafo 3.3 mostreremo un'euristica per la determinazione di una direzione di decrescita e ne daremo una possibile interpretazione geometrica.

3.1 Continuità

Allo scopo di dimostrare la continuità di Ψ è utile premettere la seguente definizione:

Definizione 3.1 (Banda) *Sia $\mathcal{F}$ l'immagine di un'applicazione*

$$f : [a, b] \to R^n$$

continua e tale che

$$f(a) = f(b),$$

ovvero una curva chiusa. Diciamo che

$$\overline{\mathcal{F}} = \{x \in R^n | \ \exists y \in \overline{\mathcal{F}} \ ||x - y|| < \alpha\}$$

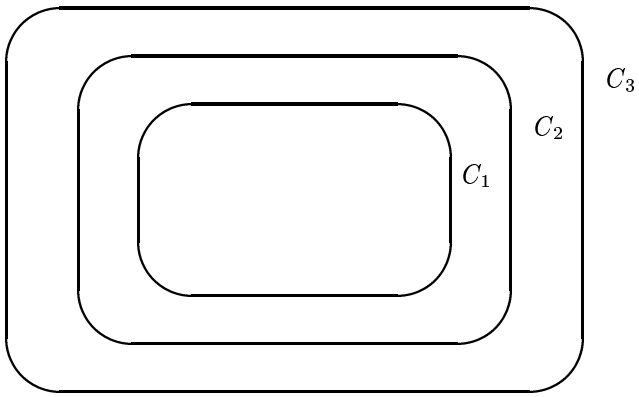


Figura 3.1: Esempio di una banda.

è una banda di ampiezza α, *con* $\alpha > 0$. *Diciamo anche che* $\overline{\mathcal{F}}$ *ha per* supporto $\mathcal{F}$.

La figura 3.1 mostra un esempio grafico di cos'è una banda.

L'idea che sta dietro la definizione di banda è quella di definire un "intorno" per una curva chiusa; notiamo come la frontiera dell'intersezione di due poligoni convessi è una curva chiusa. Il lemma che segue mostra in che modo sia possibile indurre un intorno per $(\overline{x}_i, \overline{x}_j)$ partendo dalla frontiera di $Pol_i(\overline{x}_i) \cap Pol_j(\overline{x}_j)$.

Lemma 3.1 *Siano* $\overline{x}_i, \overline{x}_j \in \mathbf{R^3}$ *tali che:*

$$Pol_i(\overline{x}_i) \cap Pol_j(\overline{x}_j) \neq \emptyset;$$

sia $\overline{\mathcal{F}}$ *una banda di ampiezza non nulla avente come supporto*

$$\mathcal{F} = Frontiera(Pol_i(\overline{x}_i) \cap Pol_j(\overline{x}_j)).$$

Allora l'insieme $U \subset \mathbf{R^6}$ *definito come:*

$$U = \{(\overline{y}_i, \overline{y}_j) | Frontiera(Pol_i(\overline{y}_i) \cap Pol_j(\overline{y}_j)) \subset \overline{\mathcal{F}}\}$$

è un intorno del punto $(\overline{x}_i, \overline{x}_j)$.

Dimostrazione Dobbiamo far vedere che esiste un $\delta > 0$, per il quale

$$\{(\overline{y}_i, \overline{y}_j) |\ ||(\overline{x}_i, \overline{x}_j) - (\overline{y}_i, \overline{y}_j)|| < \delta\} \subset U.$$

Se $(\overline{x}_i, \overline{x}_j)$ è interno ad U il lemma è dimostrato.

Supponiamo per assurdo che $(\overline{x}_i, \overline{x}_j)$ appartenga alla frontiera di U, allora esiste una direzione u tale che:

$$\forall\ \delta > 0\ (\overline{x}_i, \overline{x}_j) + \delta u \notin U.$$

Questo implica l'esistenza di un punto di $\mathcal{F}$ nel quale la banda $\overline{\mathcal{F}}$ ha ampiezza nulla, che è assurdo per ipotesi.

□

Mostriamo ora un lemma con il quale dimostriamo la continuità delle Ψ_{ij}:

Lemma 3.2 *Le funzioni*

$$\Psi_{ij},\ i = 1, \ldots, NumPezzi - 1,\ j = i + 1, \ldots, NumPezzi$$

sono continue.

Dimostrazione Per definizione di continuità dobbiamo dimostrare che

$$\forall (\overline{x}_i, \overline{x}_j) \in \mathbf{R^3} \times \mathbf{R^3}$$

e per ogni intorno V di $y = \Psi_{ij}(\overline{x}_i, \overline{x}_j)$, esiste un intorno $U = (U_1 \times U_2)$ di $(\overline{x}_i, \overline{x}_j)$ tale che $\Psi_{ij}(U_1, U_2) \subset V$.

Se $\overline{x}_i$ e $\overline{x}_j$ sono tali che:

$$Pol_i(\overline{x}_i) \cap Pol_j(\overline{x}_j) = \emptyset$$

allora la Ψ_{ij} è nulla in tutto un intorno di $(\overline{x}_i, \overline{x}_j)$ e quindi contiunua nello steso punto.

Supponiamo, allora, di essere nel caso

$$Pol_i(\overline{x}_i) \cap Pol_j(\overline{x}_j) \neq \emptyset;$$

siano $V' \subset V$ una palla di raggio $\epsilon > 0$ e di centro y, $\mathcal{F}$ la frontiera di $Pol_i(\overline{x}_i) \cap Pol_j(\overline{x}_j)$ e $\overline{\mathcal{F}}$ una sua banda di ampiezza non nulla tale che

$$\int \overline{\mathcal{F}} \ \leq\ \epsilon.$$

L'intorno $(U_1 \times U_2)$ può essere definito, in modo corretto per il lemma precedente, nel seguente modo:

$$(U_1, U_2) = \{(\overline{y}_i, \overline{y}_j) \in \mathbf{R^3} \times \mathbf{R^3} |\ Frontiera(Pol_i(\overline{y}_i) \cap Pol_j(\overline{y}_j)) \subset \overline{\mathcal{F}}\}.$$

Verifichiamo che $\Psi_{ij}(U_1, U_2) \subset V$. Presi $\overline{y}_i \in U_1$ e $\overline{y}_j \in U_2$, consideriamo

$$P' = Pol_i(\overline{y}_i) \cap Pol_j(\overline{y}_j)$$

e

$$P = Pol_i(\overline{x}_i) \cap Pol_j(\overline{x}_j).$$

P' differisce da P a meno di $\overline{\mathcal{F}}$:

$$((P \cup P') \setminus (P \cap P')) \subset \mathcal{F}.$$

Questo implica che

$$|\Psi_{ij}(\overline{x}_i, \overline{x}_j) - \Psi_{ij}(\overline{y}_i, \overline{y}_j)| = \int ((P \cup P') \setminus (P \cap P')) < \epsilon$$

e quindi:

$$\Psi_{ij}(\overline{y}_i, \overline{y}_j) \in V' \subset V.$$

□

Con il lemma appena esposto è facile dimostrare la continuità di Ψ:

Teorema 3.1 *La funzione Ψ è continua.*

Dimostrazione Nel paragrafo 2.3 abbiamo visto che la funzione Ψ può essere scomposta come:

$$\Psi(\overline{x}_1, \ldots, \overline{x}_{NumPezzi}) = \Psi_0 + 2 \sum_{i=0}^{NumPezzi-1} \sum_{j=i+1}^{NumPezzi} \Psi_{ij}(\overline{x}_i, \overline{x}_j).$$

Quindi condizione necessaria e sufficiente per la continuità di Ψ è che siano continue le Ψ_{ij}: dal lemma precedente segue direttamente la tesi. □

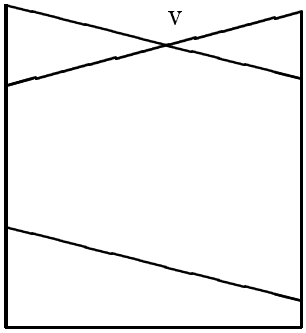


Figura 3.2: Istanza di riferimento

3.2 Differenziabilità

Cerchiamo di capire qualcosa riguardo il differenziale di Ψ, cominciando col mostrare un esempio che evidenzia la sua non continuità. Consideriamo l'istanza di PLMT mostrata in fig.3.2: sia f_1 l'indicatrice di P_1 e f_2 l'indicatrice di P_2.

Vediamo quanto valgono $\frac{d}{dx_{11}}\Psi(\overline{x}_1, \overline{x}_2)$ e $\frac{d}{d(-x_{11})}\Psi(\overline{x}_1, \overline{x}_2)$. Dalla definizione di derivata direzionale abbiamo:

$$\begin{aligned}
\frac{d}{dx_{11}}\Psi(\overline{x}_1, \overline{x}_2) &= \lim_{\epsilon \to 0^+} \frac{\Psi(\overline{x}_1 + (\epsilon, 0, 0), \overline{x}_2) - \Psi(\overline{x}_1, \overline{x}_2)}{\epsilon} = \lim_{\epsilon \to 0^+} \frac{\epsilon a}{\epsilon} = a \\
\frac{d}{d(-x_{11})}\Psi(\overline{x}_1, \overline{x}_2) &= \lim_{\epsilon \to 0^-} \frac{\Psi(\overline{x}_1 + (\epsilon, 0, 0), \overline{x}_2) - \Psi(\overline{x}_1, \overline{x}_2)}{\epsilon} = 0
\end{aligned}$$

e quindi il differenziale di Ψ non è continuo. Indichiamo, abusando in notazione, con $x_{ih}, i = 1, \ldots, NumPezzi, h = 1, 2, 3$ i versori unitari di $\mathbf{R}^{\mathbf{3NumPezzi}}$; supponiamo poi di avere un algoritmo che calcoli

$$\frac{d}{dx_{ih}}\Psi(\overline{x}_1, \ldots, \overline{x}_{NumPezzi}), \forall i = 1, \ldots, NumPezzi, \ \forall\ h = 1, 2, 3 \qquad (3.1)$$

e

$$\frac{d}{d(-x_{ih})}\Psi(\overline{x}_1, \ldots, \overline{x}_{NumPezzi}), \forall i = 1, \ldots, NumPezzi, \ \forall\ h = 1, 2, 3. \qquad (3.2)$$

Se in $(\overline{x}_1, \ldots, \overline{x}_{NumPezzi})$ Ψ è differenziabile con continuità allora per ogni $i = 1, \ldots, NumPezzi$ e per ogni $h = 1, 2, 3$ vale la seguente identità:

$$\frac{d}{dx_{ih}}\Psi(\overline{x}_1, \ldots, \overline{x}_{NumPezzi}) = \frac{d}{d(-x_{ih})}\Psi(\overline{x}_1, \ldots, \overline{x}_{NumPezzi});$$

il viceversa non è sempre vero. È vero però che se, per qualche $i = 1, \ldots, NumPezzi$, $h = 1, 2, 3$, è:

$$\frac{d}{d(\pm x_{ih})}\Psi(\overline{x}_1, \ldots, \overline{x}_{NumPezzi}) < 0$$

allora il versore $\pm x_{ih}$ è una direzione di decrescita.

Concentriamo ora la nostra attenzione sul calcolo delle (3.1) e (3.2); in particolare faremo vedere come è possibile calcolare

$$\frac{d}{dx_{k1}}\Psi(\overline{x}_1, \ldots, \overline{x}_{NumPezzi})$$

Usando la (2.1) e il teorema di linearità delle derivate abbiamo

$$\frac{d}{dx_{k1}}\Psi(\overline{x}_1, \ldots, \overline{x}_{NumPezzi}) = 2 \sum_{i=0}^{NumPezzi-1} \sum_{j=i+1}^{NumPezzi} \frac{d}{dx_{k1}}\Psi_{ij}(\overline{x}_i, \overline{x}_j);$$

essendo alcuni termini costanti rispetto a x_k, possiamo ancora scrivere:

$$\begin{aligned}
\frac{d}{dx_{k1}}\Psi(\overline{x}_1, \ldots, \overline{x}_{NumPezzi}) &= 2\sum_{i=0}^{k-1} \frac{d}{dx_{k1}}\Psi_{ik}(\overline{x}_i, \overline{x}_k) + \\
&+ 2 \sum_{j=k+1}^{NumPezzi} \frac{d}{dx_{k1}}\Psi_{kj}(\overline{x}_k, \overline{x}_j) = \\
2\sum_{i=0}^{k-1} \frac{d}{dx_{k1}}\Psi_{ik}(\overline{x}_i, \overline{x}_k) &+ 2 \sum_{i=k+1}^{NumPezzi} \frac{d}{dx_{k1}}\Psi_{ik}(\overline{x}_i, \overline{x}_k)
\end{aligned}$$

A questo punto vediamo come si calcola

$$\frac{d}{dx_{i1}}\Psi_{ij}(\overline{x}_i, \overline{x}_j) \tag{3.3}$$

distinguendo i casi $j = 0$ e $j = 1, \ldots, NumPezzi$: noi partiremo da quest'ultimo.

Ricordiamo che:

$$\Psi_{ij}(\overline{x}_i, \overline{x}_j) = \int_S f_i(\overline{x}_i, \overline{y}) f_j(\overline{x}_j, \overline{y}) d\overline{y}$$

dove $f_i(\overline{x}_i, \overline{y})$ e $f_j(\overline{x}_j, \overline{y})$ sono le funzioni indicatrici generalizzate dei poligoni convessi $Pol_i(\overline{x}_i)$ e $Pol_j(\overline{x}_j)$ rispettivamente.

Se i due poligoni hanno intersezione vuota la funzione

$$\epsilon \mapsto \int_S f_i(\overline{x}_i + (\epsilon, 0, 0), \overline{y}) f_j(\overline{x}_j, \overline{y}) d\overline{y}$$

è costante e nulla in un intorno destro di $\epsilon = 0$, quindi la derivata direzionale (3.3) risulta nulla.

Occupiamoci ora del caso più interessante: $Pol_i(\overline{x}_i) \cap Pol_j(\overline{x}_j) \neq \emptyset$; indichiamo con

$$\begin{aligned} V_i(\overline{x}_i) &= \{(x,y) \mid (x,y) \text{ è un vertice di } Pol_i(\overline{x}_i)\} \\ V_j(\overline{x}_j) &= \{(x,y) \mid (x,y) \text{ è un vertice di } Pol_j(\overline{x}_j)\} \\ V_{ij}(\overline{x}_i, \overline{x}_j) &= \{(x,y) \mid (x,y) \text{ è un vertice di } Pol_i(\overline{x}_i) \cap Pol_j(\overline{x}_j)\}. \end{aligned}$$

Se consideriamo il fascio di rette verticali che passano per i vertici di

$$V_i(\overline{x}_i) \cup V_j(\overline{x}_j) \cup V_{ij}(\overline{x}_i, \overline{x}_j),$$

queste inducono una partizione, in trapezi disgiunti, per $Pol_i(\overline{x}_i)$ e $Pol_j(\overline{x}_j)$ di $n_i(\overline{x}_i)$ e $n_j(\overline{x}_j)$ elementi rispettivamente:

$$\begin{aligned} Pol_i(\overline{x}_i) &= \bigcup_{h=1}^{n_i(\overline{x}_i)} P_{ih}(\overline{x}_i) \\ Pol_j(\overline{x}_j) &= \bigcup_{h=1}^{n_j(\overline{x}_j)} P_{jh}(\overline{x}_j) \end{aligned}$$

Indicando con f_{ih} e f_{jh} le funzioni indicatrici di P_{ih} e P_{jh} rispettivamente, possiamo scrivere:

$$
\begin{aligned}
f_i(\overline{x}_i,\overline{y}) &= \sum_{h=1}^{n_i(\overline{x}_i)} f_{ih}(\overline{x}_i,\overline{y}) \\
f_j(\overline{x}_j,\overline{y}) &= \sum_{h=1}^{n_j(\overline{x}_j)} f_{jh}(\overline{x}_j,\overline{y})
\end{aligned}
$$

da cui:

$$
\begin{aligned}
&\int_S f_i(\overline{x}_i,\overline{y}) f_j(\overline{x}_j,\overline{y}) d\overline{y} = \\
&= \int_S \sum_{h=1}^{n_i(\overline{x}_i)} f_{ih}(\overline{x}_i,\overline{y}) \sum_{h=1}^{n_j(\overline{x}_j)} f_{jh}(\overline{x}_j,\overline{y}) d\overline{y} = \\
&= \int_S \sum_{h=1}^{n_i(\overline{x}_i)} \sum_{k=1}^{n_j(\overline{x}_j)} f_{ih}(\overline{x}_i,\overline{y}) f_{jk}(\overline{x}_j,\overline{y}) d\overline{y} = \\
&= \sum_{h=1}^{n_i(\overline{x}_i)} \sum_{k=1}^{n_j(\overline{x}_j)} \left(\int_S f_{ih}(\overline{x}_i,\overline{y}) f_{jk}(\overline{x}_j,\overline{y}) d\overline{y} \right)
\end{aligned}
$$

e quindi:

$$
\begin{aligned}
&\frac{d}{dx_{i1}} \int_S f_i(\overline{x}_i,\overline{y}) f_j(\overline{x}_j,\overline{y}) d\overline{y} = \\
&= \sum_{h=1}^{n_i(\overline{x}_i)} \sum_{k=1}^{n_j(\overline{x}_j)} \frac{d}{dx_{i1}} \int_S f_{ih}(\overline{x}_i,\overline{y}) f_{jk}(\overline{x}_j,\overline{y}) d\overline{y};
\end{aligned}
$$

vediamo, allora, come può essere calcolata la seguente derivata direzionale:

$$
\frac{d}{dx_{i1}} \int_S f_{ih}(\overline{x}_i,\overline{y}) f_{jk}(\overline{x}_j,\overline{y}) d\overline{y}. \tag{3.4}
$$

Tenendo presente che, per costruzione, i trapezi delle partizioni hanno le basi parallele, studiamo $P_{ih}(\overline{x}_i) \cap P_{jk}(\overline{x}_j)$. I casi che si possono presentare sono tre:

Caso 1: $P_{ih}(\overline{x}_i) \cap P_{jk}(\overline{x}_j) = \emptyset$ è il caso banale;

Caso 2: $P_{ih}(\overline{x}_i) \cap P_{jk}(\overline{x}_j) =$"segmento verticale" è il caso in cui i trapezi $P_{ih}(\overline{x}_i)$ e $P_{jk}(\overline{x}_j)$ sono adiacenti;

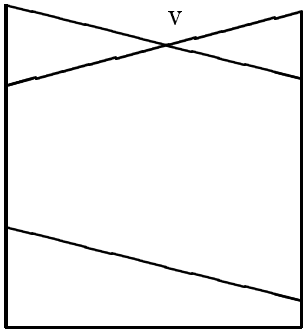


Figura 3.3: Due trapezi sovrapposto la cui intersezione non è un trapezio

Caso 3: $P_{ih}(\overline{x}_i) \cap P_{jk}(\overline{x}_j)$ ="trapezio" è il caso in cui i trapezi $P_{ih}(\overline{x}_i)$ e $P_{jk}(\overline{x}_j)$ sono sovrapposti con le basi che giacciono su due rette parallele.

Notiamo che non è mai possibile che l'intersezione di due trapezi della partizione non sia un trapezio; supponiamo per assurdo che ciò non sia vero (vedi fig.3.3), questo vuol dire che due lati si intersecano per formare un vertice v. Ma ciò è assurdo perché v dovrebbe appartenere a $V_{ij}(\overline{x}_i, \overline{x}_j)$ e quindi i due trapezi P_{ih} e P_{jk} neanche dovrebbero esistere.

Calcoliamo la derivata direzionale (3.4) nei tre casi appena mostrati.

Caso 1: per esempio i trapezi potrebbero essere disposti come in fig. 3.4. Il limite del rapporto incrementale che definisce la derivata (3.4) è:

$$\lim_{\epsilon \to 0^+} \frac{\int_S f_{ih}(\overline{x}_i + (\epsilon, 0, 0), \overline{y}) f_{jk}(\overline{x}_j, \overline{y}) d\overline{y} - \int_S f_{ih}(\overline{x}_i, \overline{y}) f_{jk}(\overline{x}_j, \overline{y}) d\overline{y}}{\epsilon} =$$

$$= \lim_{\epsilon \to 0^+} \frac{\int_S f_{ih}(\overline{x}_i + (\epsilon, 0, 0), \overline{y}) f_{jk}(\overline{x}_j, \overline{y}) d\overline{y}}{\epsilon} = 0$$

perché in un intorno di $\epsilon = 0$ l'integrale del numeratore è costante e nullo.

Caso 2: per semplicità calcoliamo la derivata (3.4) per la disposizione di fig. 3.5; per le altre disposizioni si procede in modo analogo.

È ovvio che:

$$\int_S f_{ih}(\overline{x}_i, \overline{y}) f_{jk}(\overline{x}_j, \overline{y}) d\overline{y} = 0;$$

inoltre, con semplici ragionamenti di tipo geometrico, è facile verificare che per

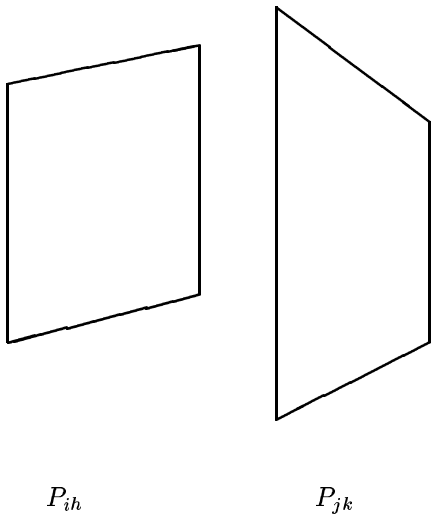


Figura 3.4: Due trapezi della partizione che non si intersecano

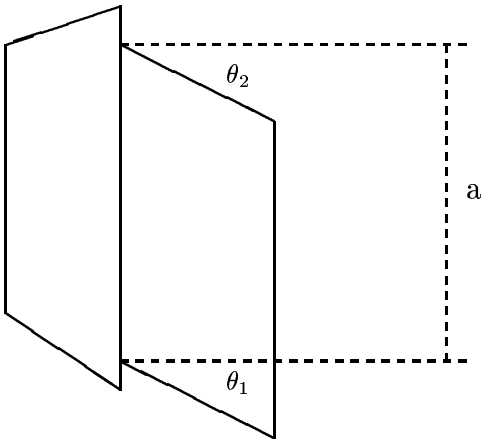


Figura 3.5: Due trapezi della partizione che si affacciano

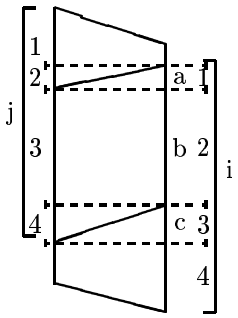


Figura 3.6: Due trapezi della partizione che si sovrappongono con una loro partizione

$\epsilon \geq 0$ è:

$$\int_S f_{ih}(\overline{x}_i + (\epsilon, 0, 0), \overline{y}) f_{jk}(\overline{x}_j, \overline{y}) d\overline{y} = \epsilon a + \frac{\epsilon^2 \tan\theta_1}{2} - \frac{\epsilon^2 \tan\theta_2}{2}.$$

Quindi per il rapporto incrementale che definisce la (3.4) si ottiene:

$$\lim_{\epsilon \to 0^+} \frac{\int_S f_{ih}(\overline{x}_i + (\epsilon, 0, 0), \overline{y}) f_{jk}(\overline{x}_j, \overline{y}) d\overline{y} - \int_S f_{ih}(\overline{x}_i, \overline{y}) f_{jk}(\overline{x}_j, \overline{y}) d\overline{y}}{\epsilon} =$$

$$= \lim_{\epsilon \to 0^+} \frac{2\epsilon a + \epsilon^2 \tan\theta_1 - \epsilon^2 \tan\theta_2}{2\epsilon} = a$$

Generalizzando possiamo dire che nel caso 2, la derivata (3.4) è pari alla lunghezza del segmento $P_{ih}(\overline{x}_i) \cap P_{jk}(\overline{x}_j)$.

Caso 3: come per il caso 2, calcoliamo la derivata (3.4) per la disposizione dei trapezi mostrata in fig.3.6; le altre disposizioni si trattano in modo analogo.

Per facilitare i conti partizioniamo, com'è nostra abitudine fare, i trapezi $P_{ih}(\overline{x}_i)$ e $P_{jk}(\overline{x}_j)$ così come mostrato in fig.3.6, in modo da poter scrivere:

$$\begin{aligned} f_{ih}(\overline{x}_i, \overline{y}) &= f^1_{ih}(\overline{x}_i, \overline{y}) + \ldots + f^4_{ih}(\overline{x}_i, \overline{y}) \\ f_{jk}(\overline{x}_j, \overline{y}) &= f^1_{jk}(\overline{x}_j, \overline{y}) + \ldots + f^4_{jk}(\overline{x}_j, \overline{y}). \end{aligned}$$

Considerando che non tutti gli elementi della partizione si intersecano, possiamo scrivere:

$$\begin{aligned} f_{ih}(\overline{x}_i,\overline{y})f_{jk}(\overline{x}_j,\overline{y}) &= f^1_{ih}(\overline{x}_i,\overline{y})f^2_{jk}(\overline{x}_j,\overline{y}) + \\ &+ f^2_{ih}(\overline{x}_i,\overline{y})f^3_{jk}(\overline{x}_j,\overline{y}) + f^3_{ih}(\overline{x}_i,\overline{y})f^4_{jk}(\overline{x}_j,\overline{y}); \end{aligned}$$

e passando prima sotto il segno d'integrale e poi sotto quello di derivata direzionale:

$$\begin{aligned} \frac{d}{dx_{i1}}\int_S f_{ih}(\overline{x}_i,\overline{y})f_{jk}(\overline{x}_j,\overline{y})d\overline{y} &= \frac{d}{dx_{i1}}\int_S f^1_{ih}(\overline{x}_i,\overline{y})f^2_{jk}(\overline{x}_j,\overline{y})d\overline{y} + \\ +\frac{d}{dx_{i1}}\int_S f^2_{ih}(\overline{x}_i,\overline{y})f^3_{jk}(\overline{x}_j,\overline{y})d\overline{y} &+ \frac{d}{dx_{i1}}\int_S f^3_{ih}(\overline{x}_i,\overline{y})f^4_{jk}(\overline{x}_j,\overline{y})d\overline{y}. \end{aligned}$$

Calcoliamo le derivate termine per termine:

primo termine

$$\begin{aligned} &\frac{d}{dx_{i1}} \int_S f^1_{ih}(\overline{x}_i,\overline{y})f^2_{jk}(\overline{x}_j,\overline{y})d\overline{y} = \\ &\lim_{\epsilon\to 0^+} \frac{\int_S f^1_{ih}(\overline{x}_i+(\epsilon,0,0),\overline{y})f^2_{jk}(\overline{x}_j,\overline{y})d\overline{y} - \int_S f^1_{ih}(\overline{x}_i,\overline{y})f^2_{jk}(\overline{x}_j,\overline{y})d\overline{y}}{\epsilon} = \\ &\lim_{\epsilon\to 0^+} \frac{(l-\epsilon)(a-\epsilon\frac{a}{l})\frac{1}{2} - \frac{la}{2}}{\epsilon} = \lim_{\epsilon\to 0^+} \frac{(l-\epsilon)(a-\epsilon\frac{a}{l}) - la}{2\epsilon} = \\ &\lim_{\epsilon\to 0^+} \frac{la - \epsilon a - \epsilon a + \epsilon^2\frac{a}{l} - la}{2\epsilon} = -a \end{aligned}$$

secondo termine

$$\begin{aligned} &\frac{d}{dx_{i1}} \int_S f^2_{ih}(\overline{x}_i,\overline{y})f^3_{jk}(\overline{x}_j,\overline{y})d\overline{y} = \\ &\lim_{\epsilon\to 0^+} \frac{\int_S f^2_{ih}(\overline{x}_i+(\epsilon,0,0),\overline{y})f^3_{jk}(\overline{x}_j,\overline{y})d\overline{y} - \int_S f^2_{ih}(\overline{x}_i,\overline{y})f^3_{jk}(\overline{x}_j,\overline{y})d\overline{y}}{\epsilon} = \\ &\lim_{\epsilon\to 0^+} \frac{b(l-\epsilon) - bl}{\epsilon} = -b \end{aligned}$$

terzo termine

$$\begin{aligned} &\frac{d}{dx_{i1}} \int_S f^3_{ih}(\overline{x}_i,\overline{y})f^4_{jk}(\overline{x}_j,\overline{y})d\overline{y} = \\ &\lim_{\epsilon\to 0^+} \frac{\int_S f^3_{ih}(\overline{x}_i+(\epsilon,0,0),\overline{y})f^4_{jk}(\overline{x}_j,\overline{y})d\overline{y} - \int_S f^3_{ih}(\overline{x}_i,\overline{y})f^4_{jk}(\overline{x}_j,\overline{y})d\overline{y}}{\epsilon} = \end{aligned}$$

$$\lim_{\epsilon \to 0^+} \quad \frac{\frac{lc}{2} - \frac{\epsilon^2 \tan\theta}{2} - \frac{lc}{2}}{\epsilon} = 0$$

Possiamo quindi scrivere:

$$\frac{d}{dx_{i1}} \int_S f_{ih}(\overline{x}_i, \overline{y}) f_{jk}(\overline{x}_j, \overline{y}) = -a - b.$$

Con quest'ultima formula dovremmo aver convinto il lettore che è sempre possibile calcolare:

$$\frac{d}{d(\pm x_{ih})} \int_S f_i(\overline{x}_i, \overline{y}) f_j(\overline{x}_j, \overline{y}) d\overline{y}$$

per ogni $i = 1, \ldots, NumPezzi$ e per ogni $h = 1, 2, 3$. Da questo risultato si ricava immediatamente che è sempre possibile calcolare:

$$\frac{d}{d(\pm x_{ih})} \Psi_{ij}(\overline{x}_i, \overline{x}_j)$$

per ogni $i = 1, \ldots, NumPezzi$ e per ogni $h = 1, 2, 3$.

Per concludere il capitolo resta da far vedere che è definito:

$$\frac{d}{d(\pm x_{ih})} \Psi_{i0}(\overline{x}_i, \overline{x}_0) \tag{3.5}$$

per ogni $i = 1, \ldots, NumPezzi$ e per ogni $h = 1, 2, 3$; questo risultato si ottiene facilmente partizionando $S \setminus S'$ come in fig. 3.7, dalla quale deduciamo che possiamo scrivere:

$$f_0(\overline{x}, \overline{y}) = f_0^1(\overline{x}, \overline{y}) + \ldots + f_0^4(\overline{x}, \overline{y})$$

dove le $f_0^1(\overline{x}, \overline{y}), \ldots, f_0^4(\overline{x}, \overline{y})$ sono le indicatrici di rettangoli e quindi di poligoni convessi.

Da quest'ultima equazione ricaviamo:

$$\begin{aligned} \frac{d}{d(\pm x_{ih})} \Psi_{i0}(\overline{x}_i, \overline{x}_0) &= \frac{d}{d(\pm x_{ih})} \int_S f_i(\overline{x}_i, \overline{y}) f_0^1(\overline{x}, \overline{y}) d\overline{y} + \ldots \\ &+ \frac{d}{d(\pm x_{ih})} \int_S f_i(\overline{x}_i, \overline{y}) f_0^4(\overline{x}, \overline{y}) d\overline{y}; \end{aligned}$$

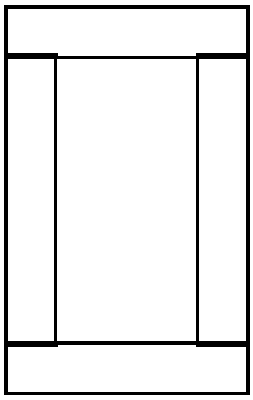


Figura 3.7: Una partizione di $S \setminus S'$.

notiamo come i termini a destra dell'uguale coinvolgono solo pezzi convessi: da questa affermazione segue che la (3.5) è sempre definita per ogni $i = 1, \ldots, NumPezzi$ e per ogni $h = 1, 2, 3$.

3.3 Determinazione di una direzione di decrescita e suo significato geometrico

Nel paragrafo precedente abbiamo visto come la funzione Ψ non è differenziabile con continuità, anche se ammette in ogni punto in cui è definita le derivate direzionali lungo i versori. Questo genera non pochi problemi, infatti supponiamo di essere in un punto $(\overline{x}_1, \ldots, \overline{x}_{NumPezzi})$ per il quale valgono le seguenti relazioni:

$$\frac{d}{d(\pm x_{kh})}\Psi(\overline{x}_1, \ldots, \overline{x}_{NumPezzi}) > 0,\ k = 1, \ldots, NumPezzi,\ h = 1, 2, 3;$$

allora le direzioni rappresentate dai versori $\pm x_{hk}$ non sono di decrescita per il punto $(\overline{x}_1, \ldots, \overline{x}_{NumPezzi})$; però, in linea di principio, non è detto che non ne esista una.

Se riusciamo a dimostrare che Ψ non è differenziabile con continuità in un insieme avente misura nulla (*differenziabile quasi ovunque*) possiamo ai fini pratici, considerare Ψ differenziabile con continuità in tutto il suo dominio. Infatti sia $\{x_i\}_{i=1,\ldots,n}$ l'insieme dei punti di $\mathbf{R^{3NumPezzi}}$ generati da un algoritmo (d'ottimizzazione non convessa); allora la probabilità che un x_i sia

di non differenziabilità per Ψ è nulla. In altre parole possiamo supporre che un'implementazione di un algoritmo d'ottimizzazione genererà un punto di non differenziabilità per Ψ con una probabilità tale da poter essere considerata nulla.

Per dimostrare che Ψ è differenziabile quasi ovunque basta dimostrare, per il teorema di Rademacher, la lipschitzianità delle Ψ_{ij}:

Lemma 3.3 *La funzione Ψ_{ij} è lipschitziana.*

Dimostrazione Dobbiamo verificare che esiste una costante $M > 0$ tale che:

$$\forall \overline{x}_1, \overline{x}_2 \in \mathbf{R^3} \ \|\Psi_{ij}(0,0,0,\overline{x}_1) - \Psi_{ij}(0,0,0,\overline{x}_2)\| \leq M\|\overline{x}_1 - \overline{x}_2\|.$$

Poniamo:

$$A = |\Psi_{ij}(0,0,0,\overline{x}_1) - \Psi_{ij}(0,0,0,\overline{x}_2)| :$$

è facile rendersi conto che A si maggiora con l'area della parte di piano toccata dal poligono j-esimo quando viene spostato dal punto $\overline{x}_1$ al punto $\overline{x}_2$. Chiamiamo A' il maggiorante così costruito.

Indicando con S l'area del poligono j-esimo si ha:

$$\begin{aligned} A' &\leq S|x_{11} - x_{21}| + S|x_{12} - x_{22}| + S|x_{13} - x_{23}| \leq \\ &\leq S|\overline{x}_1 - \overline{x}_2| \end{aligned}$$

□

Teorema 3.2 *La funzione Ψ è differenziabile quasi ovunque.*

Dimostrazione Per il teorema di Rademacher abbiamo che l'insieme in cui Ψ_{ij} non è differenziabile ha misura nulla; poiché l'unione di una famiglia finita di insiemi a misura nulla ha misura nulla, anche la Ψ è differenziabile quasi ovunque. □

L'idea che sta dietro il teorema 3.2 è che i punti $(\overline{x}_1, \ldots, \overline{x}_{NumPezzi})$ in cui Ψ non è differenziabile con continuità, sono tutti quelli che per qualche i e qualche j:

$$Pol_i(\overline{x}_i) \cap Pol_j(\overline{x}_j) \neq \emptyset$$
$$\Psi_{ij}(\overline{x}_i, \overline{x}_j) = 0.$$

L'euristica che proponiamo per la determinazione di una direzione di decrescita d, che si basa ovviamente sul teorema 3.2, è la seguente:

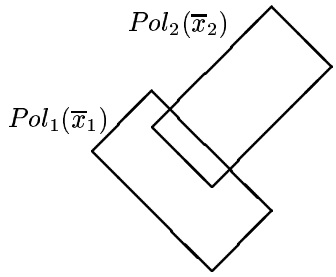


Figura 3.8: Un'istanza di PLMT.

$$d_{ih} = -\frac{d}{d\ x_{ih}}\Psi(\overline{x}_1, \ldots, \overline{x}_{NumPezzi})\ \forall i = 1, \ldots, NumPezzi,\ \forall j = 1, 2, 3.$$

Facciamo notare fin da adesso che useremo quest'euristica all'interno dell'algoritmo di minimizzazione che proporremo più avanti.

Vediamo con un esempio il significato geometrico di direzione di decrescita per la funzione Ψ. Consideriamo l'istanza di PLMT mostrata in figura 3.8; la disposizione dei pezzi corrisponderà ad una particolare coppia $(\overline{x}_1, \overline{x}_2)$.

Si può facilmente verificare che

$$\begin{array}{llll}
\dfrac{d}{dx_{11}} & \Psi(\overline{x}_1, \overline{x}_2) = -a & \dfrac{d}{d(-x_{21})}\Psi(\overline{x}_1, \overline{x}_2) = -b \\
\dfrac{d}{dx_{12}} & \Psi(\overline{x}_1, \overline{x}_2) = -a & \dfrac{d}{d(-x_{22})}\Psi(\overline{x}_1, \overline{x}_2) = -b \\
\dfrac{d}{dx_{13}} & \Psi(\overline{x}_1, \overline{x}_2) = 0 & \dfrac{d}{d(-x_{23})}\Psi(\overline{x}_1, \overline{x}_2) = 0
\end{array}$$

per opportuni a e b positivi.

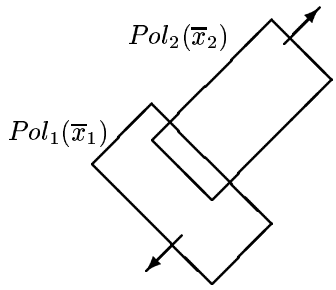


Figura 3.9: Come si spostano i pezzi.

Se consideriamo come direzione di decrescita:

$$d = (-a, -a, 0, b, b, 0),$$

i pezzi dell'esempio vengono spostati come indicato in figura 3.9.

Consideriamo ora il seguente problema di ricerca lineare:

$$\begin{array}{ll} min & \Psi((\overline{x}_1, \overline{x}_2) + \lambda d) \\ t.c. & \lambda \in \mathbf{R}^+. \end{array}$$

La sua risoluzione determina un valore di ottimo $\overline{\lambda}$ che induce un nuovo punto:

$$(\overline{y}_1, \overline{y}_2) = (\overline{x}_1, \overline{x}_2) + \overline{\lambda} d$$

che corrisponde alla configurazione dei pezzi di figura 3.10.

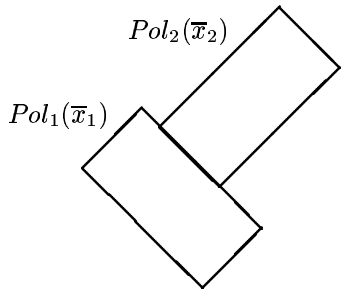


Figura 3.10: Nuova disposizione dei pezzi.

Capitolo 4

Calcolo di Ψ e delle derivate direzionali lungo i versori.

Come è facile immaginare, l'algoritmo di minimizzazione che useremo per risolvere FDB usa pesantemente due operazioni:

- Il calcolo del valore di Ψ in un punto;
- Il calcolo del differenziale di Ψ in un punto.

Per questa ragione porremo particolare attenzione agli algoritmi che effettuano queste operazioni, cercando di tenere la complessità più bassa possibile.

Nel primo e nel terzo paragrafo mostreremo come possono essere calcolati $\Psi(x)$ e $\frac{d}{d\ x_{kh}}\Psi(x)$, rispettivamente. Va notato come i metodi usati non sono *iterativi* ma *diretti*; questo implica bassa complessità e precisionedelle soluzioni trovate.

Nel secondo paragrafo studieremo un metodo "veloce" per calcolare le derivate direzionali lungo i versori, che si basa su un particolare modo di scandire in senso antiorario i vertici dei poligoni:

$$Pol_i(\overline{x}_i) \cap Pol_j(\overline{x}_j),\ i = 1, \ldots, NumPezzi - 1,\ j = i + 1, \ldots, NumPezzi.$$

4.1 Calcolo di Ψ

Abbiamo già fatto vedere nel paragrafo 2.2 che vale la seguente identità:

$$\Psi(\overline{x}_1, \ldots, \overline{x}_{NumPezzi}) = \Psi_0 + 2 \sum_{i=0}^{NumPezzi-1} \sum_{j=i+1}^{NumPezzi} \Psi_{ij}(\overline{x}_i, \overline{x}_j)$$

che può essere riscritta come:

$$\begin{aligned} \Psi(\overline{x}_1, \ldots, \overline{x}_{NumPezzi}) = \Psi_0 \quad &+ \quad 2 \sum_{j=1}^{NumPezzi} \Psi_{0j}(\overline{x}_j) \\ &+ \quad 2 \sum_{i=1}^{NumPezzi-1} \sum_{j=i+1}^{NumPezzi} \Psi_{ij}(\overline{x}_i, \overline{x}_j) \end{aligned}$$

dove

$$\begin{aligned} \Psi_0 &= \sum_{i=0}^{NumPezzi} \int_S f_i(\overline{x}_i, \overline{y})\ d\overline{y} \\ \Psi_{0j}(\overline{x}_j) &= \int_S f_0(0,0,0,\overline{y}) f_j(\overline{x}_j, \overline{y})\ d\overline{y} \\ \Psi_{ij}(\overline{x}_i, \overline{x}_j) &= \int_S f_i(\overline{x}_i, \overline{y}) f_j(\overline{x}_j, \overline{y})\ d\overline{y} \end{aligned}$$

In quello stesso paragrafo abbiamo fatto notare anche che Ψ_0 è una quantità costante che misura l'area di tutti i pezzi e che $\Psi_{ij}(\overline{x}_i, \overline{x}_j)$ misura l'area dell'intersezione dei pezzi i-esimo e j-esimo quando sono disposti come indicato in $\overline{x}_i$ e $\overline{x}_j$ rispettivamente. Nel caso particolare in cui $i = 0$, Ψ_{0j} misura l'area dell'intersezione del pezzo j con $S \setminus S'$.

Da queste considerazioni scaturisce spontaneamente il seguente algoritmo per il calcolo di Ψ:

Algoritmo 1 (PSI)

```
function PSI(x:array[1..3*NumPezzi] of real):real;

begin
  for i:=1 to NumPezzi do
    begin
      /*Ruota il pezzo i-esimo di un angolo x[i*3] */
```

```
      Ruota(i,x[i*3]);

      /*Trasla il pezzo i-esimo di un vettore
      (x[i*3-2],x[i*3-1]) */
      Trasla(i,x[i*3-2],x[i*3-1])
    end;

  s:=0;
  for j:=1 to NumPezzi do
    s:=s+PSI_0j(j);

  for i:=1 to NumPezzi-1 do
    for j:=i+1 to NumPezzi do
      s:=s+PSI_ij(i,j);

  return (s)
end
```

dove

- PSI_{0j}(j) calcola $\Psi_{0j}(\overline{x}_j)$;
- PSI_{ij}(i,j) calcola $\Psi_{ij}(\overline{x}_i, \overline{x}_j)$;
- il vettore x, da $3NumPezzi$ elementi, specifica la posizione e l' orientamento dei pezzi. Conveniamo che $\overline{x}_i$ = (x[i*3-2],x[i*3-1],x[i*3]);
- Ruota(i,α) ruota il pezzo i-esimo di un angolo α con complessità lineare rispetto al numero di vertici del poligoni i-esimo;
- Trasla(i,x,y) trasla il pezzo i-esimo di un vettore (x,y) con complessità lineare rispetto al numero di vertici del poligoni i-esimo.

Descriviamo la funzione PSI_{ij}(i,j). Osserviamo che se il più piccolo rettangolo che contiene il pezzo $Pol_i(\overline{x}_i)$ non si interseca con il più piccolo rettangolo che contiene il pezzo $Pol_j(\overline{x}_j)$ allora il pezzo i-esimo ed il pezzo j-esimo hanno intersezione vuota; questa condizione viene verificata dalla funzione VerificaIntersezione(i,j), con complessità lineare nel numero di vertici dei pezzi $Pol_i(\overline{x}_i)$ e $Pol_j(\overline{x}_j)$. La funzione EnumeraVertici(i,j) enumera i vertici dell'intersezione dei poligoni $Pol_i(\overline{x}_i)$ e $Pol_j(\overline{x}_j)$. La funzione Area($\{(x_1, y_1), \ldots, (x_n, y_n)\}$) calcola l'area del poligono supposto convesso di vertici $(x_1, y_1), \ldots, (x_n, y_n)$; l'algoritmo usato è quello presentato in appendice di complessità $O(n \log n)$. Per maggiori dettagli sulle procedure usate dall'algoritmo 2, si rimanda all'appendice.

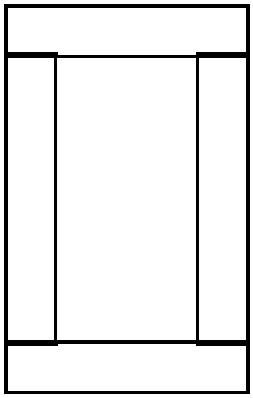


Figura 4.1: Scomposizione del bordo in poligoni convessi.

Algoritmo 2 (PSI_{ij})
function PSI_{ij}*(i,j):real;*

```
begin
  if VerificaIntersezione(i,j)
    then
      return (0)
    else
      begin
        int:=EnumeraVertici(i,j);
        return (Area(int))
      end
end
```

Esaminiamo ora la PSI_{0j}: questa deve calcolare l'area dell'intersezione di $Pol_j(\overline{x}_j)$ con $S \setminus S'$. Per fare ciò ci conviene scomporre $S \setminus S'$ in 4 poligoni convessi (dei rettangoli, vedi fig.4.1) di indici

$$NumPezzi + 1, \ldots, NumPezzi + 4$$

e poi procedere come descritto nel seguente:

Algoritmo 3 (PSI_{0j})
function PSI_{0j}*(j:integer):real;*

```
begin
  s:=0;
  for k:=1 to 4 do
    s:=s+PSI_ij(NumPezzi+k,j);
  return (s)
end
```

Studiamo ora la complessità nel caso pessimo delle funzioni proposte. Denotiamo con v_i, $i = 1, \ldots, NumPezzi$, il numero dei vertici del poligono i-esimo e con v il più grande dei v_i.

Verifichiamo facilmente che la funzione PSI_ij(i,j) ha una complessità nel caso pessimo pari ad $O((v_i + v_j)\log(v_i + v_j))$, infatti

- la VerificaIntersezione() ha complessità $O(v_i + v_j)$;
- la EnumeraVertici() ha complessità $\theta(v_i + v_j)$ (vedi appendice);
- l'esecuzione della chiamata "Area(int)" ha una complessità

$$O((v_i + v_j)\log(v_i + v_j)).$$

Verifichiamo altrettanto facilmente che la funzione PSI_0j ha una complessità nel caso pessimo pari ad $O(v_j \log(v_j))$, infatti:

- la "PSI_ij" viene chiamata esattamente 4 volte;
- l'esecuzione di una chiamata di "PSI_ij" ha complessità

$$O((4 + v_j)\log(4 + v_j)).$$

Per concludere, la complessità nel caso pessimo di PSI si valuta facilmente in $O(v\log(v)NumPezzi^2)$, infatti:

- Il suo primo ciclo costa $O(vNumPezzi)$;
- il secondo ciclo costa $O(v\log(v)NumPezzi)$;
- l'ultimo ciclo costa $O(v\log(v)NumPezzi^2)$.

4.2 Derivazione veloce

Quando nel secondo capitolo abbiamo fatto vedere che Ψ ammette in ogni punto le derivate direzionali

$$\frac{d}{d(\pm x_{kh})}\Psi(\overline{x}_1, \ldots, \overline{x}_{NumPezzi}),\ k = 1, \ldots, NumPezzi,\ h = 1, 2, 3. \qquad (4.1)$$

abbiamo, in effetti, mostrato un procedimento per calcolarle. Da questo procedimento potrebbe scaturire, in modo ovvio, un algoritmo per il calcolo delle (4.1), che però risulterebbe poco efficiente a causa dell'alto numero di partizioni da costruire. Vediamo un altro metodo per calcolare le (4.1): in realtà mostreremo, equivalentemente, com'è possibile calcolare

$$\frac{d}{d\ x_{ki}}\Psi_{ki}(\overline{x}_k, \overline{x}_i), \tag{4.2}$$

per ogni $k = 1, \ldots, NumPezzi - 1$, $i = k + 1, \ldots, NumPezzi$.

Indichiamo con:

$$\begin{aligned} V &= \{vertici(Pol_k(\overline{x}_k) \cap Pol_i(\overline{x}_i))\} \\ P &= Pol_k(\overline{x}_k) \cap Pol_i(\overline{x}_i); \end{aligned}$$

partendo dai punti di V e dalla frontiera di P possimo costruire tre particolari partizioni per Pol_k e Pol_i:

1. quella indotta dalle rette orizzontali che passano per i punti di V e dalla frontiera di P;
2. quella indotta dalle rette verticali che passano per i punti di V e dalla frontiera di P;
3. quella indotta dalle circonferenze di centro (x_{k1}, x_{k2}) che passano per i punti di V e dalla frontiera di P.

Considerando la partizione 1, possiamo supporre che esistano due famiglie di funzioni indicatrici generalizzate $\mathcal{F}_k^1$ e $\mathcal{F}_i^1$ tali che:

$$\begin{aligned} f_k(\overline{x}_k, \overline{y}) &= \sum_{\mathbf{f}_k \in \mathcal{F}_k^1} \mathbf{f}_k(\overline{x}_k, \overline{y}) \\ f_i(\overline{x}_i, \overline{y}) &= \sum_{\mathbf{f}_i \in \mathcal{F}_i^1} \mathbf{f}_i(\overline{x}_i, \overline{y}). \end{aligned}$$

Se consideriamo:

$$\mathcal{F}^1 = \left\{ (\mathbf{f}_k, \mathbf{f}_i) \in \mathcal{F}_k^1 \times \mathcal{F}_i^1 | \, \{\overline{y} \mid \mathbf{f}_k(\overline{x}_k, \overline{y}) = 1\} \cap \{\overline{y} \mid \mathbf{f}_i(\overline{x}_i, \overline{y}) = 1\} \neq \emptyset \right\},$$

possiamo scrivere:

$$\Psi_{ki}(\overline{x}_k, \overline{x}_i) = \int_S \sum_{(\mathbf{f}_k, \mathbf{f}_i) \in \mathcal{F}^1} \mathbf{f}_k(\overline{x}_k, \overline{y}) \mathbf{f}_i(\overline{x}_i, \overline{y}) d\overline{y}.$$

In un modo del tutto analogo a quello appena mostrato, possiamo costruire due famiglie di coppie di funzioni $\mathcal{F}^2$ e $\mathcal{F}^3$, associate rispettivamente alle partizioni 2 e 3, tali che:

$$\Psi_{ki}(\overline{x}_k, \overline{x}_i) = \int_S \sum_{(\mathbf{f}_k, \mathbf{f}_i) \in \mathcal{F}^2} \mathbf{f}_k(\overline{x}_k, \overline{y}) \mathbf{f}_i(\overline{x}_i, \overline{y}) d\overline{y},$$
$$\Psi_{ki}(\overline{x}_k, \overline{x}_i) = \int_S \sum_{(\mathbf{f}_k, \mathbf{f}_i) \in \mathcal{F}^3} \mathbf{f}_k(\overline{x}_k, \overline{y}) \mathbf{f}_i(\overline{x}_i, \overline{y}) d\overline{y}.$$

Forse si è già capito che le partizioni 1,2 e 3 servono per calcolare le (4.2) nei casi h=1, h=2 e h=3, rispettivamente.

Proprietà 4.1 *Sia* $V = \{v_1, \ldots, v_n, \ldots, v_m\}$ *ordinato in modo che:*

- v_1 *è il vertice ad ordinata minima più a destra;*
- v_n *è il vertice ad ordinata massima più a sinistra;*
- v_j *segue* v_{j-1} *in senso antiorario,* $j = 2, \ldots, m$*;*
- v_1 *segue* v_m *in senso antiorario.*

Allora

$$\frac{d}{d\ x_{k1}} \Psi_{ki}(\overline{x}_k, \overline{x}_i) = -|\pi_y(v_1 - v_n)| + \sum_{j=1}^{m} s_j \tag{4.3}$$

dove

$$s_j = \begin{cases} 0 & \text{se } \overline{v_j\ v_{j+1}} \text{ appartiene alla frontiera di } Pol_i, \\ & j = 1, \ldots, n; \\ |\pi_y(v_j - v_{j+1})| & \text{se } \overline{v_j\ v_{j+1}} \text{ appartiene alla frontiera di } Pol_k, \\ & j = 1, \ldots, n; \\ |\pi_y(v_j - v_{j+1})| & \text{se } \overline{v_j\ v_{j+1}} \text{ appartiene alla frontiera di } Pol_i, \\ & j = n+1, \ldots, m-1; \\ 0 & \text{se } \overline{v_j\ v_{j+1}} \text{ appartiene alla frontiera di } Pol_k, \\ & j = n+1, \ldots, m-1; \\ |\pi_y(v_m - v_1)| & \text{se } \overline{v_m\ v_1} \text{ appartiene alla frontiera di } Pol_i, \\ 0 & \text{se } \overline{v_m\ v_1} \text{ appartiene alla frontiera di } Pol_k; \end{cases}$$

π_y è la proiezione canonica lungo l'asse delle ordinate.

Dimostrazione All'inizio del paragrafo abbiamo visto com'è possibile scrivere

$$\Psi_{ki}(\overline{x}_k, \overline{x}_i) = \int_S \sum_{(\mathbf{f}_k, \mathbf{f}_i) \in \mathcal{F}^1} \mathbf{f}_k(\overline{x}_k, \overline{y}) \mathbf{f}_i(\overline{x}_i, \overline{y}) d\overline{y};$$

derivando rispetto alla direzione x_{ki} e applicando il teorema di linearità delle derivate abbiamo:

$$\frac{d}{d\ x_{ki}} \Psi_{ki}(\overline{x}_k, \overline{x}_i) = \sum_{(\mathbf{f}_k, \mathbf{f}_i) \in \mathcal{F}^1} \frac{d}{d\ x_{ki}} \int_S \mathbf{f}_k(\overline{x}_k, \overline{y}) \mathbf{f}_i(\overline{x}_i, \overline{y}) d\overline{y}.$$

Calcoliamo quindi:

$$\frac{d}{d\ x_{ki}} \int_S \mathbf{f}_k(\overline{x}_k, \overline{y}) \mathbf{f}_i(\overline{x}_i, \overline{y}) d\overline{y} \qquad \forall (\mathbf{f}_k, \mathbf{f}_i) \in \mathcal{F}^1, \tag{4.4}$$

enumerando tutti i possibili casi che si possono presentare quando prendiamo una coppia $(\mathbf{f}_k, \mathbf{f}_i)$ in $\mathcal{F}^1$; ricordiamo che per definizione di $\mathcal{F}^1$ abbiamo sempre che gli insiemi:

$$\{\overline{y} \mid \mathbf{f}_k(\overline{x}_k, \overline{y}) = 1\} \cap \{\overline{y} \mid \mathbf{f}_i(\overline{x}_i, \overline{y}) = 1\} \tag{4.5}$$

sono dei trapezi non nulli.

caso 1 : $\mathbf{f}_k \subset P,\ \mathbf{f}_i \subset P$

Sia h l'altezza del trapezio (4.5), allora con semplici ragionamenti di tipo geometrico è facile dimostrare che la (4.4) relativa al caso in questione è pari a $-h$. Questo spiega la presenza del primo termine nella (4.3).

caso 2 : $\mathbf{f}_k \subset P,\ \mathbf{f}_i \not\subset P$

In questo caso l'insieme (4.5) fa parte della frontiera di P, diciamo che è il segmento $\overline{(v_j v_{j+1})}$. Se $j \in \{1, \ldots, n\}$ allora la relativa (4.4) è pari a $|\pi_y(v_j - v_{j+1})|$; se, invece, $j \in \{n+1, \ldots, m\}$ allora la relativa (4.4) è nulla.

caso 3 : $\mathbf{f}_k \not\subset P,\ \mathbf{f}_i \subset P$

Anche in questo caso l'insieme (4.5) fa parte della frontiera di P, diciamo ancora che è il segmento $\overline{(v_j v_{j+1})}$. Se $j \in \{1, \ldots, n\}$ allora la relativa (4.4) è nulla; se, invece, $j \in \{n+1, \ldots, m\}$ allora la relativa (4.4) è pari a $|\pi_y(v_j - v_{j+1})|$.

□

Proprietà 4.2 *Sia* $V = \{v_1, \ldots, v_n, \ldots, v_m\}$ *ordinato in modo che:*

- v_1 *è il vertice ad ascissa massima più in basso;*
- v_n *è il vertice ad ascissa minima più in alto;*
- v_j *segue* v_{j-1} *in senso antiorario,* $j = 2, \ldots, m$*;*
- v_1 *segue* v_m *in senso antiorario.*

Allora

$$\frac{d}{d\, x_{k2}} \Psi_{ki}(\overline{x}_k, \overline{x}_i) = -|\pi_y(v_1 - v_n)| + \sum_{j=1}^{m} s_j$$

dove

$$s_j = \begin{cases} 0 & \text{se } \overline{v_j\ v_{j+1}} \text{ appartiene alla frontiera di } Pol_i, \\ & j = 1, \ldots, n; \\ |\pi_y(v_j - v_{j+1})| & \text{se } \overline{v_j\ v_{j+1}} \text{ appartiene alla frontiera di } Pol_k, \\ & j = 1, \ldots, n; \\ |\pi_y(v_j - v_{j+1})| & \text{se } \overline{v_j\ v_{j+1}} \text{ appartiene alla frontiera di } Pol_i, \\ & j = n+1, \ldots, m-1; \\ 0 & \text{se } \overline{v_j\ v_{j+1}} \text{ appartiene alla frontiera di } Pol_k, \\ & j = n+1, \ldots, m-1; \\ |\pi_y(v_m - v_1)| & \text{se } \overline{v_m\ v_1} \text{ appartiene alla frontiera di } Pol_i, \\ 0 & \text{se } \overline{v_m\ v_1} \text{ appartiene alla frontiera di } Pol_k; \end{cases}$$

π_y è la proiezione canonica lungo l'asse delle ascisse.

Dimostrazione Simile a quella per la proprietà 4.1.

□

Proprietà 4.3 *Sia $c = (x_{k1}, x_{k2})$ il centro del poligono $Pol_k(\overline{x}_k)$ e sia $V = \{v_1, \ldots, v_n, \ldots, v_m\}$ ordinato in modo che:*

- *v_1 è uno dei vertici più vicini a c;*
- *v_n è uno dei vertici più lontani da c;*
- *gli elementi di V sono ordinati in senso antiorario.*

Allora

$$\frac{d}{d\ x_{k3}} \Psi_{ki}(\overline{x}_k, \overline{x}_i) = -\pi_z(c - v_1, c - v_n) + \sum_{j=1}^{m} s_j$$

dove

$$s_j = \begin{cases} 0 & \text{se } \overline{v_j\ v_{j+1}} \text{ appartiene alla frontiera di } Pol_i, \\ & j = 1, \ldots, n; \\ \pi_y(c - v_j, c - v_{j+1}) & \text{se } \overline{v_j\ v_{j+1}} \text{ appartiene alla frontiera di } Pol_k, \\ & j = 1, \ldots, n; \\ \pi_y(c - v_j, c - v_{j+1}) & \text{se } \overline{v_j\ v_{j+1}} \text{ appartiene alla frontiera di } Pol_i, \\ & j = n+1, \ldots, m-1; \\ 0 & \text{se } \overline{v_j\ v_{j+1}} \text{ appartiene alla frontiera di } Pol_k, \\ & j = n+1, \ldots, m-1; \\ \pi_y(c - v_m, c - v_1) & \text{se } \overline{v_m\ v_1} \text{ appartiene alla frontiera di } Pol_i, \\ 0 & \text{se } \overline{v_m\ v_1} \text{ appartiene alla frontiera di } Pol_k; \end{cases}$$

$\pi_z : \mathbf{R^2} \times \mathbf{R^2} \to \mathbf{R}^+$ *è definita come* $\pi_z(v_1, v_2) = \left| \| v_1 \|^2 - \| v_2 \|^2 \right|$.

Dimostrazione Simile a quella della proprietà 4.1.

□

4.3 Calcolo di $\frac{d}{d(\pm x_{kh})}\Psi(x_1, \ldots, x_{NumPezzi})$

In questo paragrafo mostreremo un algoritmo per calcolare la derivata direzionale

$$\frac{d}{d(\pm x_{kh})}\Psi(x_1, \ldots, x_{NumPezzi}) \tag{4.6}$$

con $k = 1, \ldots, NumPezzi$ e $h = 1, 2, 3$, usando il teorema 3.2 e i risultati mostrati nel paragrafo precedente. Abbiamo già visto che vale la seguente identità:

$$\begin{aligned} \frac{d}{d\ x_{kh}}\Psi(\overline{x}_1, \ldots, \overline{x}_{NumPezzi}) &= 2\frac{d}{d\ x_{kh}}\Psi_{0k}((0,0,0), \overline{x}_k) + \\ 2\sum_{i=1}^{k-1}\frac{d}{d\ x_{kh}}\Psi_{ik}(\overline{x}_i, \overline{x}_k) &+ 2\sum_{i=k+1}^{NumPezzi}\frac{d}{d\ x_{kh}}\Psi_{ki}(\overline{x}_k, \overline{x}_i), \end{aligned}$$

che può essere riscritta come:

$$2\frac{d}{d\ x_{kh}}\Psi_{0k}((0,0,0),\overline{x}_k) \quad + \quad 2\sum_{i=1}^{k-1}\frac{d}{d\ x_{kh}}\Psi_{ik}(\overline{x}_i,\overline{x}_k) + \\ + \quad 2\sum_{i=k+1}^{NumPezzi}\frac{d}{dx_{kh}}\Psi_{ik}(\overline{x}_i,\overline{x}_k)$$

perché è vera l'uguaglianza:

$$\frac{d}{d\ x_{kh}}\Psi_{ki}(\overline{x}_k,\overline{x}_i) = \frac{d}{d\ x_{kh}}\Psi_{ik}(\overline{x}_i,\overline{x}_k).$$

Da questa considerazione ricaviamo il seguente algoritmo:

Algoritmo 4 (dPSI)
function dPSI(k,h):real;

begin
  s:=$dPSI_0$*(k,h);*

  for i:=1 to k-1 do
    s:=s+$dPSI_{ij}$*(k,i,h);*

  for i=k+1 to NumPezzi do
    s:=s+$dPSI_{ij}$*(k,i,h);*
  return (s)
end

dove

- $dPSI_0$(k,h) calcola

$$\frac{d}{d\ x_{kh}}\Psi_{0k}((0,0,0),\overline{x}_k); \tag{4.7}$$

- $dPSI_{ij}$(k,i,h) calcola

$$\frac{d}{d\ x_{kh}}\Psi_{ik}(\overline{x}_i,\overline{x}_k). \tag{4.8}$$

Mostriamo ora la funzione $dPSI_{ij}$(k,i,h). Come abbiamo già fatto notare per l'algoritmo 2, se il più piccolo rettangolo che contiene $Pol_i(\overline{x}_i)$ non si interseca con il più piccolo rettangolo che contiene $Pol_j(\overline{x}_j)$, allora il pezzo i-esimo ha intersezione vuota con il pezzo j-esimo: in questo caso possiamo subito dire che la derivata (4.8) è nulla.

In caso contrario calcoliamo l'insieme V dato dall'insieme dei vertici di $Pol_i(\overline{x}_i) \cap Pol_k(\overline{x}_k)$, per poi passarlo alla funzione dPARTIZIONE() per l'uso che vedremo più avanti. L'insieme V viene costruito usando la funzione EnumeraVertici() (vedi appendice).

Algoritmo 5 ($dPSI_{ij}$)
function $dPSI_{ij}$(k,i,h):real;

```
begin
  if VerificaIntersezione(i,k) then
    return (0);

  V:=EnumeraVertici(i,k);
  return (dPARTIZIONE(V,k,i,h))
end
```

Per spiegare il funzionamento della funzione dPARTIZIONE() è necessario capire perché viene costruito V. Nel paragrafo precedente abbiamo fatto vedere che per calcolare la (4.8) dobbiamo costruire tre partizioni diverse per $Pol_k(\overline{x}_k)$ e $Pol_i(\overline{x}_i)$ in funzione del valore di h:

- Se $h = 1$ la partizione da considerare è quella indotta dalle rette orizzontali che passano per i punti di V e dalla frontiera di $Pol_k(\overline{x}_k) \cap Pol_i(\overline{x}_i)$;
- se $h = 2$ la partizione da considerare è quella indotta dalle rette verticali che passano per i punti di V e dalla frontiera di $Pol_k(\overline{x}_k) \cap Pol_i(\overline{x}_i)$;
- se $h = 3$ la partizione da considerare è quella indotta dalle circonferenze aventi il centro in (x_{k1}, x_{k2}) e passanti per i punti di V e dalla frontiera di $Pol_k(\overline{x}_k) \cap Pol_i(\overline{x}_i)$.

Per quanto appena detto dovrebbe risultare ovvia la seguente definizione di dPARTIZIONE:

Algoritmo 6 (dPARTIZIONE)
function dPARTIZIONE(V,k,i,h):real;

```
begin
  case h do
    1 : return(dPARTIZIONE1(V,k,i));
    2 : return(dPARTIZIONE2(V,k,i));
    3 : return(dPARTIZIONE3(V,k,i))
  end
end
```

dove le funzioni dPARTIZIONE1(), dPARTIZIONE2() e dPARTIZIONE3(), che trattano rispettivamente i casi $h = 1, h = 2$ e $h = 3$, sono definite come segue:

Algoritmo 7 (dPARTIZIONE1)

```
function dPARTIZIONE1(V,k,i):real;

begin
  "Ordina i vertici di V in senso antiorario in modo tale
  che V[1] sia il vertice ad ascissa maggiore tra quelli
  ad ordinata minima;

  "Sia n l'indice del vertice in V ad ascissa minore tra
  quelli ad ordinata massima";

  m:=cardinalità(V);

  s:=-norma2(proiezione_x(V[n]-V[1]));

  for j:=1 to n do
    if "il segmento (V[j],V[j+1]) appartiene alla frontiera di Pol[k]"
      then s:=s+norma2(proiezione_x(V[j]-V[j+1]));

  for j:=n+1 to m-1 do
    if "il segmento (V[j],V[j+1]) appartiene alla frontiera di Pol[i]"
      then s:=s+norma2(proiezione_x(V[j]-V[j+1]));

  if "il segmento (V[m],V[1]) appartiene alla frontiera di Pol[i]"
    then s:=s+norma2(proiezione_x(V[m]-V[1]));

  return(s)
end
```

dove

- V rappresenta un insieme di vertici (elementi di $\mathbf{R^2}$);

- cardinalità() ritorna il numero di elementi dell'insieme V in $O(1)$;
- norma2() calcola la norma 2 di un vettore di $\mathbf{R^2}$ in $O(1)$;
- proiezione$_x$() proietta un vettore di $\mathbf{R^2}$ sull'asse delle ascisse in $O(1)$.

Algoritmo 8 (dPARTIZIONE2)
function dPARTIZIONE2(V,k,i):real;

begin

"Ordina i vertici di V in senso antiorario in modo tale
che V[1] sia il vertice ad ordinata minore tra quelli
ad ascissa massima";

"Sia n l'indice del vertice in V ad ordinata maggiore
tra quelli ad ascissa minima";

m:=cardinalità(V);

s:=-norma2(proiezione$_y$(V[n]-V[1]));

for j:=1 to n do
if "il segmento (V[j],V[j+1]) appartiene alla frontiera di Pol[k]"
then s:=s+norma2(proiezione$_y$(V[j]-V[j+1]));

for j:=n+1 to m-1 do
if "il segmento (V[j],V[j+1]) appartiene alla frontiera di Pol[i]"
then s:=s+norma2(proiezione$_y$(V[j]-V[j+1]));

if "il segmento (V[m],V[1]) appartiene alla frontiera di Pol[i]"
then s:=s+norma2(proiezione$_y$(V[m]-V[1]));

return(s)

end

dove

- proiezione$_y$() proietta un vettore di $\mathbf{R^2}$ sull'asse delle ordinate in $O(1)$.

Algoritmo 9 (dPARTIZIONE3)
function dPARTIZIONE3(V,k,i):real;

begin
*c:=(x[k*3-2],x[k*3-1]); /* E' il centro del poligono k-esimo. */*

"Ordina i vertici di V in senso antiorario in modo tale
che V[1] e' uno dei vertici piu' vicini a c";

"Sia n l'indice di uno dei vertice di V piu' lontani da c";

m:=cardinalità(V);

s:=-proiezione$_z$(V[n],V[1]);

for j:=1 to n do
if "il segmento (V[j],V[j+1]) appartiene alla frontiera di Pol[k]"
then s:=s+proiezione$_z$(c-V[j],c-V[j+1]);

for j:=n+1 to m-1 do
if "il segmento (V[j],V[j+1]) appartiene alla frontiera di Pol[i]"
then s:=s+proiezione$_z$(c-V[j],c-V[j+1]);

if "il segmento (V[m],V[1]) appartiene alla frontiera di Pol[i]"
then s:=s+proiezione$_z$(c-V[m],c-V[1]);

return(s)

end

dove

- proiezione$_z$() prende due vettori v_1 e v_2 e restituisce $\left| \| v_1 \|^2 - \| v_2 \|^2 \right|$ in $O(1)$.

Descriviamo ora la funzione dPSI_0(): come già fatto per la funzione PSI_0j() consideriamo il bordo $S \backslash S'$ scomposto in poligoni convessi di indici $NumPezzi+1, \ldots, NumPezzi+4$.

Algoritmo 10 (dPSI$_0$)
function dPSI$_0$(k,h):real;

begin

```
  s:=0;

  for i:=NumPezzi+1 to NumPezzi+4
    s:=s+dPSI_ij(k,i,h);

  return(s)
end;
```

Studiamo ora la complessità nel caso pessimo delle funzioni proposte; denotiamo, come già fatto nel paragrafo precedente, con v_i il numero di vertici del poligono i-esimo e con v il più grande dei v_i.

Cominciamo con lo studio delle dPARTIZIONE1(), dPARTIZIONE2() e dPARTIZIONE3(), che hanno tutte complessità $O(m\log(m))$: infatti

- l'ordinamento dei vertici di V costa, in tutti i tre casi, $O(m\log(m))$;
- la determinazione dell'indice n costa $O(m)$;
- il calcolo della cardinalità di V costa $O(1)$;
- il calcolo delle varie proiezioni costa $O(1)$;
- il corpo dei due cicli for può essere fatto in $O(1)$ se facciamo la seguente considerazione. L'insieme V contiene i vertici dell'intersezione $Pol_i(\overline{x}_i) \cap Pol_j(\overline{x}_j)$; in più possiamo supporre

  che l'insieme V contiene anche l'indice del poligono d'appartenenza dei suoi vertici. Notiamo che questa ipotesi non è restrittiva perché possiamo supporre che la EnumeraVertici() associ ai vertici che restituisce, l'indice del poligono d'apparteneza quando è possibile;
- l'esecuzione dei due for ha, quindi, complessità $O(m)$.

Passiamo ora alla funzione dPSI$_{ij}$(); constatiamo facilmente che essa ha complessità $O((v_i + v_k)\log(v_i + v_k))$, infatti:

- VerificaIntersezione() ha, come abbiamo fatto già notare nel paragrafo precedente, complessità $O(v_i + v_k)$;
- EnumeraVertici() ha complessità $O(v_i + v_k)$, vedi appendice;
- dPARTIZIONE() ha complessità $O((v_i + v_k)\log(v_i + v_k))$.

Capitolo 5

Un algoritmo di minimizzazione

Lo scopo di questo capitolo è quello di mostrare una procedura per la risoluzione di FDB. A tale fine mostreremo nel primo paragrafo un algoritmo di risoluzione per PLMT; notiamo come quest'algoritmo debba minimizzare una funzione non convessa, cosa che non è tra le più facili.

Nel secondo paragrafo mostreremo l'algoritmo di risoluzione per FDB, che usa come procedura l'algoritmo di risoluzione per PLMT.

Nel terzo paragrafo mostreremo una serie di test effettuata sull'algoritmo per misurarne la bontà.

5.1 Un algoritmo per PLMT

L'algoritmo di risoluzione per PLMT è un particolare algoritmo di minimizzazione non convessa per funzioni reali definite su un ipercubo D di $\mathbf{R^n}$, ovvero risolve problemi del tipo:

Definizione 5.1 (NCP - NonConvexProblem)

$$\begin{array}{rl} min & f(x) \\ t.c. & x \in D. \end{array}$$

Nel nostro caso si ha che:

- la funzione da minimizzare è:

$$\int_S \left(\sum_{i=0}^{NumPezzi} f_i(\overline{x}_i, \overline{y}) \right)^2 d\overline{y};$$

- il dominio sul quale va ricercato il minimo è:

$$\begin{array}{rcl} x_{i1} & \in & [0, S'.base] \\ x_{i2} & \in & [0, S'.altezza] \\ x_{i3} & \in & [0, 2\pi] \\ \forall & & i = 1, \ldots, NumPezzi. \end{array}$$

Mostriamo lo schema generale di un algoritmo per la risoluzione di NCP ampiamente trattato in letteratura (vedi per esempio [6]); partendo da questo e apportando le opportune particolarizzazioni, dedurremo un algoritmo per PLMT.

Definizione 5.2 (Regione d'attrazione) *Sia x un minimo locale per NCP, allora la* regione d'attrazione attr(x) *per x è il più grande insieme contenuto in D tale che per ogni y $\in$ attr(x), l'algoritmo "infinitely small step steepest decent" applicato al punto y converge verso x.*

Questa definizione ha un'iterpretazione molto pittoresca: supponiamo di avere il plastico del fondale di un bacino fluviale e una piccola sfera di vetro. Se appoggiamo la sfera in un punto y del plastico questa, essendo immersa nel campo magnetico terrestre, inizia a muoversi per spostarsi in un punto x a potenziale locale minimo: x è un punto d'attrazione per y.

Un possibile schema di risoluzione per NCP consta di due fasi:

1. Ricerca di un punto d'attrazione del minimo globale;
2. Ricerca dell'ottimo globale partendo dal punto trovato nella fase 1.

Generalmente la fase 1 consiste in un campionamento del dominio D; le tecniche disponibili sono varie e generalmente la scelta della migliore deve essere guidata dalla struttura del problema da risolvere. La fase 2 consiste generalmente nell'applicazione di un algoritmo di ricerca locale; anche in questo caso le tecniche utilizzabili sono diverse e solo la struttura del problema da risolvere può fornire un'indicazione su quale usare.

Mostriamo come abbiamo particolarizzato lo schema di risoluzione per NCP al caso PLMT, cioè mostriamo come abbiamo implementato le due fasi sopra esposte.

fase 1

Per questa fase abbiamo deciso d'usare un algoritmo tipo "simulated annealing". La scelta è stata motivata dalle seguenti considerazioni:

- mancano informazioni specifiche sulla funzione da minimizzare. Questo implica che non abbiamo sufficienti elementi per costruire un algoritmo ad hoc; inoltre i metodi probabilistici sono quelli che generalmente si applicano senza troppe ipotesi;
- notoriamente il simulated annealing si applica per la minimizzazione di funzioni non convesse, determina soluzioni di alta qualità ed è *robusto*, nel senso che la scelta della soluzione iniziale "non influenza" la soluzione determinata.

Il simulated annealing che abbiamo implementato è leggermente diverso da quello "classico" riportato in appendice: la modifica sostanziale consiste nel cambiamento della funzione d'accettazione. Questa modifica è stata suggerita dalla seguente considerazione: supponiamo che in una generica iterazione del metodo l'ottimo corrente x vale v. Quando viene generata una nuova soluzione y riusciamo a stabilire "velocemente" se essa vale meno di v?

Consideriamo la nostra funzione da minimizzare; essa ha una particolare struttura che può essere schematizzata come segue:

$$f(x) = \sum_{i=0}^{n} f_i(x).$$

Estendiamo la $f(x)$ nel modo seguente:

$$f(x,v) = \sum_{i=0}^{\overline{n}} f_i(x),$$

dove

$$\overline{n} = max \left\{ n \mid \sum_{i=0}^{n} f_i(x) \leq v \right\}.$$

È banale verificare che se $f(y,v) < v$ allora y è una soluzione migliore di x e che $f(y) = f(y,v)$; se invece $f(y,v) \geq v$ allora y è una soluzione peggiore o al

più equivalente a x. Quello che va notato è che il passaggio da una soluzione x ad una peggiore y non comporta il calcolo di tutte le $f_i(y)$.

Se supponiamo di voler usare questo sistema per determinare quando una soluzione è migliore o peggiore di un'altra, dobbiamo accettare il fatto che in caso di peggioramento non sappiamo di quanto peggioriamo: questo implica il cambiamento della funzione d'accettazione.

Ricordiamo brevemente che, nel simulated annealing, la funzione d' accettazione rappresenta la probabilità con la quale viene accettata una soluzione peggiore di quella corrente; quella standard è del tipo:

$$\exp\left(\frac{-\Delta f}{T}\right)$$

dove

- $-\Delta f$ rappresenta il valore del peggioramento della funzione obiettivo;
- T rappresenta la temperatura corrente del metodo.

La funzione d'accettazione che proponiamo non può ovviamente dipendere da Δf; inoltre deve fare in modo che per alti valori di T vengano accettate quasi tutte le soluzioni proposte, al decrescere di T la probabilità d'accettazione deve decrescere fino a raggiungere un valore di soglia sotto il quale è quasi nulla. Questa funzione è:

$$\frac{1}{1+\exp(-T+k)}$$

con k opportuna costante.

Una descrizione del simulated annealing può essere trovata in appendice, qui riportiamo solo la scelta di alcuni parametri che ne influenzano l'esecuzione:

- la temperatura iniziale viene posta a

  $$\Psi((0,0,0),\ldots,(0,0,0))$$

  che è uno dei valori più alti che può ritornare la Ψ;
- ad ogni iterazione la nuova temperatura viene determinata moltiplicando la temperatura del passo precedente per una costante $c \in [0.8, 1[$;
- l'intorno utilizzato è quello euclideo;
- la temperatura viene diminuita dopo $O(NumPezzi)$ iterazioni, questo per permettere una visita più o meno completa dell'intorno.

fase 2

Supponiamo d'eseguire la fase 1 e di determinare, di conseguenza, un punto $x \in D$; per ipotesi, supponiamo anche che x appartenga alla regione d'attrazione di un ottimo globale x^*. Lo scopo della seconda fase consiste proprio nell'applicazione di un algoritmo di ricerca locale che partendo da x cerchi di determinare x^*.

L'algoritmo che abbiamo usato per la fase 2 è l'algoritmo dei gradienti coniugati (d'ora in poi AGC); questa scelta è stata motivata dall'alta velocità di convergenza di questo metodo.

Le ipotesi d'applicazione e lo schema generale dell'AGC si trova in appendice; qui diamo solo il seguente commento. L'AGC si applica solo a funzioni differenziabili con continuità, quindi, a rigor di logica, non potrebbe essere applicato per minimizzare Ψ. Però per il toerema 3.2, possiamo supporre di non generare mai un punto di non differenziabilità.

Concludiamo il paragrafo mostrando l'algoritmo di risoluzione per PLMT:

Algoritmo 11 (MinPLMT)

```
function MinPLMT(var x;S'):real;

begin
  /* Fase 1 */
  SimulatedAnnealing(x,S');

  /* Fase 2 */
  return(GradientiConiugati(x,S'))
end;
```

dove

- la procedura SimulatedAnnealing() accetta in input un punto x, passato per riferimento, e una striscia S': x è il punto dal quale parte con la ricerca, S' è la striscia sulla quale va costruita l'istanza di PLMT da risolvere. Una volta determinato il minimo, questo viene reso disponibile all'esterno attraverso la x;
- la funzione GradientiConiugati() accetta in input un punto x, passato per riferimento, e una striscia S', così come la procedura SimulatedAnnealing(). Una volta determinato il minimo, questo viene reso disponibile all'esterno attraverso la variabile x e ne viene ritornato anche il valore.

5.2 Un algoritmo per FDB

Nel capitolo 1 abbiamo introdotto un modello per FDB, che riportiamo per comodità:

$$\begin{array}{ll} \min & S'.altezza \\ t.c. & \int_S \left(\sum_{i=0}^{NumPezzi} f_i(\overline{x}_i, \overline{y}) \right)^2 d\overline{y} = A \\ & x_{i1} \in [0, S'.base] \\ & x_{i2} \in [0, S'.altezza] \\ & x_{i3} \in [0, 2\pi] \\ & \forall i = 1, \ldots, NumPezzi \\ & S'.altezza > 0 \\ & S = [-d, S'.base + d] \times [-d, S'.altezza + d]. \end{array}$$

Osserviamo come i vincoli presenti in FDB sono equivalenti al seguente predicato in x:

$$MinPLMT(x, S') = A;$$

dove $S' = [0, S'.base] \times [0, S'.altezza]$. Posto:

$$\phi(S'.altezza) = \begin{cases} 1 & \text{se } MinPLMT(x, S') > A \\ 0 & \text{se } MinPLMT(x, S') = A \end{cases}$$

possiamo riscrivere, in modo equivalente, il modello per FDB nel seguente modo:

Definizione 5.3 (FDB)

$$\begin{array}{ll} \min & S'.altezza \\ t.c. & \phi(S'.altezza) = 0. \end{array}$$

La seguente proposizione ci permetterà di risolvere FDB usando una particolarità della ϕ:

Proposizione 5.1 *Fissata una particolare istanza di FDB, la funzione ϕ è un gradino del tipo:*

$$\phi(h) = \begin{cases} 1 & \text{se } h < h_{min} \\ 0 & \text{se } h \geq h_{min} \end{cases}$$

dove h_{min} è l'altezza minima ammissibile per l'istanza in esame di FDB.

Dimostrazione La tesi discende direttamente dalle seguenti considerazioni:

1. se $h > 0$ è un valore per il quale $\phi(h) = 0$, allora per ogni $h' > h$ è $\phi(h') = 0$: se una striscia alta h è ammissibile allora la striscia lunga h' è ancora ammissibile;
2. se $h > 0$ è un valore per il quale $\phi(h) = 1$, allora per ogni $h' \in (0, h)$ è $\phi(h') = 1$: se una striscia h è inammissibile allora la striscia lunga h' è ancora inammissibile.

□

Questa proposizione suggerisce un modo abbastanza ovvio di risolvere FDB: utilizzare un algoritmo tipo bisezione per la ricerca del punto di commutazione di ϕ; anche se questo modo di procedere risulta corretto da un punto di vista teorico, risulta inutilizzabile da un punto di vista pratico. Infatti non è rara la situazione in cui MinPLMT(x, S') ritorna un valore più grande di A quando l'altezza di S' è di poco più grande rispetto a quella minima.

Per ovviare a questo inconveniente proponiamo un'altra metodologia che si basa sulla seguente considerazione. Sia S' una striscia sufficientemente alta per la quale abbiamo che MinPLMT(x_1, S')=A: il vettore x_1 restituito dalla MinPLMT() corrisponde ad una disposizione ammissibile dei pezzi in S'. Se supponiamo di abbassare leggermente la striscia S' in modo tale che ammetta ancora una disposizione ammissibile, è lecito aspettarsi che il vettore x_2 calcolato con:

1. x_2:=x_1;
2. MinPLMT(x_2, S');

sia "simile" ad x_1, nel senso che la disposizione dei pezzi corrispondente ad x_1 è simile a quella corrispondente ad x_2.

L'algoritmo che quindi proponiamo è:

Algoritmo 12 (MinFDB)

```
function MinFDB(var soluzione:array[1..NumPezzi*3] of real):real;

begin
  x:=InitSoluzione;
  S':=InitStriscia;
  delta:=InitDelta;
  h:=S'.altezza;

  while delta ≥ delta_min do
    begin
      intersezione:=0;
      while intersezione=0 do
        begin
          S'.altezza:=h;
          intersezione:=MinPLMT(x,S');
          if intersezione=0 then
            begin
              soluzione:=x;
              h_min:=h
            end;
          h:=h-delta
        end;
      h:=h_min;
      x:=soluzione;
      RiduciDelta(delta);
      h:=h-delta
    end;
  return (h_min)
end
```

dove :

- InitSoluzione() è una funzione che assegna in modo casuale dei valori agli elementi di x; questo è come dire che inizialmente i pezzi vengono disposti sulla striscia in modo casuale;
- InitStriscia() è una funzione che ritorna una striscia che ammette una disposizione ammissibile. Sia $S'.base$ la lunghezza della base di S' ed A la somma delle aree di tutti i pezzi: sicuramente

$$h_{min} = \frac{A}{S'.base}$$

è il più piccolo valore che è possibile assegnare ad $S'.altezza$. Per determinare in un modo molto semplice un valore iniziale per $S'.altezza$, e quindi un valore iniziale per

$$S' = [0, S'.base] \times [0, S'.altezza],$$

poniamo:

$$S'.altezza = k \times h_{min},$$

con k costante opportunamente grande;

- InitDelta() è una funzione che ritorna un valore iniziale per *delta*; nella nostra implementazione *delta* viene inizializzata ad un quinto dell'altezza iniziale di S';
- $delta_{min}$ è il più piccolo valore per *delta*.

5.3 Prove d'esecuzione

Le soluzioni per FDB sono rappresentate da un vettore $\overline{x}$ di $R^{3NumPezzi}$ e da una striscia S', vedi capitolo 2. Un modo per stimare la "bontà" di una soluzione $(\overline{x}, S')$ è il seguente. Indichiamo con A la somma delle aree di tutti i pezzi: è ovvio che nessuna soluzione per FDB avrà un'altezza $S'.altezza$ minore di $A/S'.base$. In particolare questo vale anche per la soluzione ottima $(\overline{x}_{opt}, S'_{opt})$:

$$S'_{opt}.altezza \geq \frac{A}{S'.base}.$$

Data quindi una soluzione $(\overline{x}, S')$ per FDB, una possibile stima per la sua bontà è data dal seguente indice:

$$\epsilon_h = \frac{S'.altezza - \frac{A}{S'.base}}{\frac{A}{S'.base}} \geq \frac{S'.altezza - S'_{opt}.altezza}{S'_{opt}.altezza}.$$

Un'altra misura della bontà delle soluzioni $(\overline{x}, S')$ è la percentuale di superfice non utilizzata:

$$wa = 100 \cdot \frac{S'.altezza \cdot S'.base - A}{S'.altezza \cdot S'.base}.$$

Per cultimo, ma non in ordine di importanza, consideriamo anche il tempo d'esecuzione necessario per il calcolo della soluzione.

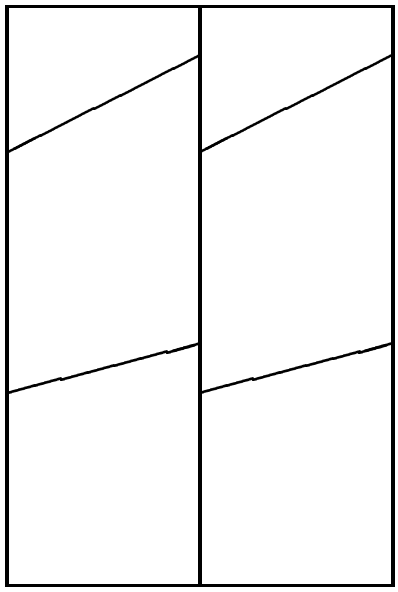


Figura 5.1: Una soluzione ottima per FDB.

L'istanza di FDB che abbiamo tentato di risolvere con *MinFDB* è quella che si ottiene dalla partizione indicata in figura 5.1.

È ovvio che per questa particolare istanza, la soluzione ottima deve avere:

$$\begin{aligned} \epsilon_h &= 0; \\ wa &= 0\%. \end{aligned}$$

I risultati ottenuti dall'esecuzione su di una Sun di una batteria di prove sono riportati nella tabella seguente:

N. prova	ϵ_h	$wa(\%)$
1	0.77	43.56
2	0.49	33
3	0.72	41.93
4	0.60	37.61
5	0.60	37.62
6	0.53	34.57

Il tempo d'esecuzione per ogni prova si aggira intorno ai 40–50 minuti. La prima cosa che viene da commentare è che il tempo d'esecuzione è troppo alto

per solo sei pezzi, inoltre i due indici sono troppo lontani dallo zero. In definitiva possiamo dire che la versione testata di *MinFDB* è inutilizzabile. Questo fatto ci spinge a modificare MinFDB aggiungendo una forte componente euristica.

Capitolo 6

Un altro algoritmo

Nel capitolo precedente abbiamo presentato un algoritmo probabilistico che cercava di risolvere **FDB** e che si è rilevato improponibile a causa dell'alto tempo di calcolo. Presentiamo in questo capitolo un altro algoritmo (deterministico) per la risoluzione di **FDB** e che ha il seguente schema:

1. Determinazione di una soluzione iniziale usando un'approssimazione dei poligoni;
2. Compattare la soluzione così determinata.

Nel paragrafo 6.1 mostriamo come si può determinare una soluzione iniziale; nel paragrafo 6.2 mostriamo la tecnica da noi proposta per il compattamento di poligoni. Per finire, nel paragrafo 6.3 illustreremo i risultati che si ottengono procedendo in quest'altra maniera.

6.1 Determinazione di una soluzione iniziale

Un metodo per determinare una soluzione ammissibile per **FDB** è il seguente: prima si approssimano i Pol_i con i più piccoli rettangoli R_i tali che $Pol_i \subseteq R_i$, $i = 1, \ldots, NumPezzi$. Successivamente si dispongono i rettangoli R_i sulla striscia con un qualunque algoritmo greedy. Quanto detto può essere formalizzato con il seguente:

Algoritmo 13 (Init)
procedure Init(var x:array[1..NumPezzi]) of real;

```
var
  R:array[1..NumPezzi] of box;

begin
  S'.altezza:=0;
  for i:=1 to NumPezzi do
    InRettangolo(Pol_i,R_i);
  for i:=1 to NumPezzi do
    DisponiRettangolo(R_i);
  InPoligoni(x,R)
end
```

dove:

- *box* è definito come

  ```
  type box =
    record
      base,altezza:real;
      centrox,centroy:real;
      orientamento:real
    end;
  ```

  Il campo *orientamento* serve per passare dai rettangoli ai poligono, come si vedrà più avanti.

- la procedura *InRettangolo*(Pol_i,R_i) determina il più piccolo rettangolo R_i che inscrive il poligono Pol_i risolvendo il seguente problema:

$$\begin{array}{ll} min & bh \\ t.c. & (vx_{ij}, vy_{ij})R(\theta) \in \left[-\frac{b}{2}, \frac{b}{2}\right] \times \left[-\frac{h}{2}, \frac{h}{2}\right] \\ & j = 1, \ldots, NumVertici_i \\ & \theta \in \left[0, \frac{\pi}{2}\right] \\ & b, h > 0 \end{array}$$

  dove:

  - $R(\cdot)$ è la matrice di rotazione;
  - $NumVertici_i$ è il numero di vertici dell'i-esimo poligono;
  - (vx_{ij}, vy_{ij}) sono le coordinate del vertice j dell'i-esimo poligono.

Indicando con $\overline{b}, \overline{h}, \overline{\theta}$ i valori delle variabili del problema precedente corrispondenti all'ottimo, la procedura *InRettangolo(Pol_i,R_i)* effettua i seguenti assegnamenti:

$$\begin{aligned} &ret.base := \overline{b}; \\ &ret.altezza := \overline{h}; \\ &ret.orientamento := \overline{\theta}. \end{aligned}$$

- La procedura *DisponiRettangolo(R_i)* dispone il rettangolo i- esimo nella striscia S', eventualmente aumentando il valore di $S'.altezza$, usando un qualche criterio. La nostra implementazione prevede che i rettangoli siano disposti partendo dal vertice inferiore sinistro di S' andando da sinistra verso destra, in caso di overflow la disposizione si sposta di una riga verso l'alto. Questa procedura modifica i campi *centrox, centroy* degli R_i in base alla allocazione dei rettangoli.

- La procedura *InPoligono(x,R)* effettua i seguenti assegnamenti per ogni $i = 1, \ldots, NumPezzi$:

$$\begin{aligned} x[i*3-2] &:= ret[i].centrox; \\ x[i*3-1] &:= ret[i].centroy; \\ x[i*3] &:= ret[i].orientamento. \end{aligned}$$

6.2 Algoritmo di compattamento.

Mostriamo ora un nostro algoritmo per il compattamento di poligoni convessi. Si tratta di un'euristica che si basa sul fatto che quando iscriviamo un poligono convesso con un rettangolo, otteniamo una approssimazione che induce uno spreco che è stimabile intorno al 25%. In altre parole, possiamo dire che nella soluzione determinata dall'algoritmo 13, i poligoni sono circondati da molta area libera.

L'algoritmo di compattamento può essere suddiviso in due fasi:

1. **Precompattamento**: i pezzi vengono spostati verso la parte centrale della striscia facendoli muovere solo lungo la direzione y;
2. **Compattamento fine**: viene usata una particolare versione dei *gradienti coniugati* per ottenere un compattamento più fine.

Esaminiamo queste due fasi con maggior dettaglio.

Nella fase di precompattamento, la striscia S' viene suddivisa esattamente in due con un taglio parallelo all'asse delle x: indichiamo con S'_1 la parte superiore della suddivisione e con S'_2 quella inferiore. I pezzi che risultano avere il loro baricentro in S'_1 vengono etichettati con **giù**, tutti gli altri con **su**.

A questo punto, senza effettuare né rotazioni né traslazioni lungo l'asse x, i pezzi vengono compattati il più possibile spostando quelli etichettati **giù** verso il basso e quelli etichettati **su** verso l'alto.

La seconda fase è stata denominata *compattamento fine* perché cerca di ottenere un impaccamento migliore; illustriamo l'idea di base considerando un pezzo Pol_i marcato **giù**. È naturale pensare di spostare Pol_i verso il basso consentendo, questa volta, anche rotazioni e traslazioni lungo l'asse delle x.

Per effettuare questo spostamento consideriamo il seguente insieme:

$$I = \{(x,y) \quad | \quad y = oy + k\delta_y, k = 1, \ldots, K_y \wedge \\ x = ox + k\delta_x, k = -K_x, \ldots, -1, 0, 1, \ldots, K_x\}$$

dove:

- (ox, oy) è il baricentro di Pol_i;
- δ_x, δ_y sono due numeri reali che regolano la *densità* di I;
- K_x e K_y sono due numeri naturali che regolano la cardinalità di I.

Partendo dai punti ad ascissa minima di I, posizioniamo Pol_i facendo in modo che il suo baricentro coincida con uno dei punti di I: nel caso in cui Pol_i dovesse intersecarsi con qualche pezzo o uscire fuori dalla striscia useremo, per poche ($O(1)$) iterazioni, il metodo dei gradienti coniugati per minimizzare la funzione:

$$\sum_{j=0}^{i-1} \Psi_{ij} + \sum_{j=i+1}^{NumPezzi} \Psi_{ij}.$$

Questo stratagemma permette una ricerca alquanto accurata di un nuovo punto dove piazzare Pol_i e che possibilmente aumenti il grado di compattezza dei pezzi. Ovviamente quanto esposto non ha nessun fondamento teorico e rappresenta, quindi, solo un'euristica.

In modo del tutto analogo spostiamo i pezzi etichettati **su** verso l'alto. Indichiamo con *fase discendente* l'azione dello spostare tutti i pezzi etichettati **giù** e con *fase ascendente* l'azione dello spostare tutti i pezzi etichettati **su**. L'algoritmo di compattamento che deriva da quanto detto finora è il seguente:

Algoritmo 14 (FineComp)
procedure FineComp(var x);

```
begin
  Precompattamento(x);
  finito:=false;
  while not finito do
    begin
      finito:=FaseAscendente(x);
      if not finito then
        begin
          sol:=x;
          finito:=FaseDiscendente(x);
          if not finito then
            sol:=x
        end
    end;
  x:=sol
end
```

dove:

- la procedura *Precompattamento()* implementa la fase di precompattamento sopra illustrata;
- la funzione *FaseAscendente()* implementa la fase ascendente sopra illustrata, essa ritorna *true* sse non si riescono a spostare più di $\frac{NumPezzi}{6}$ pezzi. Idem per la funzione *FaseDiscendente()*

6.3 Prove d'esecuzione

Consideriamo ancora l'istanza di FDB costruita nel paragrafo 5.3 per usarla come pietra di paragone tra i due algoritmi. Siccome il metodo che abbiamo appena presentato è deterministico, non ha senso effettuare una batteria di prove, ma basta farne una sola.

Alla fine della fase di precompattamento, la disposizione dei pezzi provoca uno spreco di striscia pari ad $wa = 41.418\%$. Dopo la fase di compattamento fine, la situazione migliora portando lo spreco ad $wa = 37.714\%$; il parametro ϵ_h vale 0.608 ed il tempo di calcolo è di 19 minuti su un computer basato su un 80286.

Notiamo subito che il secondo algoritmo ha trovato una soluzione equivalente a quelle determinate dal primo algoritmo in un tempo che è nettamente minore.

Abbiamo provato a disporre manualmente i pezzi per determinare la soluzione iniziale evitando l'utilizzo del greedy, ottenendo uno spreco pari a $wa = 17.427\%$. L'algoritmo, dopo il compattamento fine, propone una soluzione con $wa = 4.101\%$, $\epsilon_h = 0.043$ in 38 minuti. Questo esperimento mette in luce il ruolo essenziale del calcolo della disposizione iniziale dei pezzi.

Mostriamo ora l'andamento dell'algoritmo su un'istanza formata da 12 pezzi di varie forme (6 rettangoli, 2 triangoli, 3 quadrilateri, 1 pentagono irregolare e 1 esagono irregolare):

N. prova	$wa(\%)$	tempo(min)
1	35.129	4
2	38.108	25
3	45.065	28

Questi risultati non sono ancora confortanti essendo lo spreco della soluzione finale troppo elevato. Per concludere possiamo dire che determinare un buon algoritmo di risoluzione per FDB non è una cosa così semplice, a riprova della difficoltà intrinseca del problema.

Capitolo 7

Conclusioni

Mostriamo alcuni possibili sviluppi del presente lavoro che ci sembrano particolarmente interessanti e promettenti.

Nel primo paragrafo vederemo come è possibile estendere il modello FDB al caso in cui i pezzi da piazzare sono dei poligoni non convessi. Nel secondo paragrafo vedremo come è possibile esprimere, con poca fatica, particolari tipi di vincoli sul posizionamento finale dei pezzi. Nel terzo ed ultimo paragrafo, illustreremo alcune idee che potrebbero migliorare le prestazioni dell'algoritmo.

7.1 Estensioni al caso non convesso

Sia $\{Pol_i\}_{i=1,\ldots,NumPezzi}$ una famiglia di poligoni non convessi tale che:

$$(0,0) \in Pol_i, \forall i = 1, \ldots, NumPezzi$$

e che (vedi anche fig.7.1):

- $Pol_i = \bigcup_{j=1}^{n_i} Pol_{ij}$;
- $\forall\ j_1, j_2 \in \{1, \ldots, n_i\},\ j_1 \neq j_2,\ \int Pol_{ij_1} \cap Pol_{ij_2} = 0$;
- Pol_{ij} è un poligono convesso, $\forall j \in \{1, \ldots, n_i\}$.

Se indichiamo con f_{ij} la funzione indicatrice generalizzata di Pol_{ij} con $i = 1, \ldots, NumPezzi$ e $j = 1, \ldots, n_i$, allora possiamo scrivere:

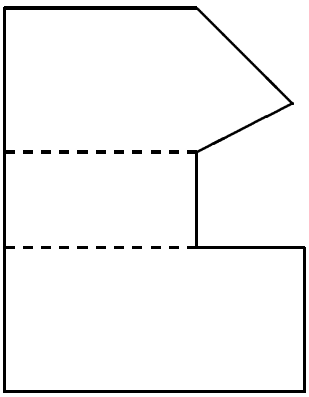


Figura 7.1: Scomposizione di un poligono non convesso in poligoni convessi.

$$f_i(\overline{x}_i, \overline{y}) = \sum_{j=1}^{n_i} f_{ij}(\overline{x}_i, \overline{y}).$$

Riprendiamo la definizione di Ψ:

$$\begin{aligned} \Psi &= \int_S \left(\sum_{i=0}^{NumPezzi} f_i(\overline{x}_i, \overline{y}) \right)^2 d\overline{y} = \\ &= \Psi_0 + 2 \sum_{i=0}^{NumPezzi-1} \sum_{j=i+1}^{NumPezzi} \Psi_{ij}(\overline{x}_i, \overline{x}_j) \end{aligned}$$

dove Ψ_0 rappresenta la somma delle aree dei Pol_i, mentre $\Psi_{ij}(\overline{x}_i, \overline{x}_j)$ rappresenta l'area dell'intersezione $Pol_i(\overline{x}_i) \cap Pol_j(\overline{x}_j)$.

Nel caso non convesso Ψ_0 si calcola nel modo ovvio. Per vedere com'è possibile calcolare $\Psi_{i_1 i_2}$, $i_1 = 0, \ldots, NumPezzi - 1$, $i_2 = i_1 + 1, \ldots, NumPezzi$, consideriamo la seguente identità:

$$\begin{aligned} \Psi_{i_1 i_2}(\overline{x}_{i_1}, \overline{x}_{i_2}) &= \int_S f_{i_1}(\overline{x}_{i_1}, \overline{y}) f_{i_2}(\overline{x}_{i_2}, \overline{y}) d\overline{y} = \\ &= \int_S \sum_{j_1=1}^{n_{i_1}} f_{i_1 j_1}(\overline{x}_{i_1}, \overline{y}) \sum_{j_2=1}^{n_{i_2}} f_{i_2 j_2}(\overline{x}_{i_2}, \overline{y}) d\overline{y} = \end{aligned}$$

$$= \sum_{j_1=1}^{n_{i_1}} \sum_{j_2=1}^{n_{i_2}} \int_S f_{i_1 j_1}(\overline{x}_{i_1}, \overline{y}) f_{i_2 j_2}(\overline{x}_{i_2}, \overline{y}) d\overline{y}.$$

Questa ci dice che il calcolo di $\Psi_{i_1 i_2}$ nel caso non convesso si riduce al calcolo di un'opportuna serie di $\Psi_{j_1 j_2}$ nel caso convesso. Da questo risultato scaturisce anche la continuità di Ψ nel caso non convesso.

La differenziabilità si tratta in modo analogo, consideriamo infatti la seguente uguaglianza:

$$\frac{d}{d\, x_{kh}} \Psi(\overline{x}_1, \ldots, \overline{x}_{NumPezzi}) = 2 \sum_{i_1=1}^{NumPezzi-1} \sum_{i_2=i_1+1}^{NumPezzi} \frac{d}{d\, x_{kh}} \Psi_{i_1 i_2}(\overline{x}_{i_1}, \overline{x}_{i_2})$$

con questa abbiamo ridotto, al solito, la differenziabilità di Ψ alla differenziabilità di un opportuno insieme di funzioni $\Psi_{i_1 i_2}$. Queste ultime si riducono al caso convesso con la seguente formula:

$$\frac{d}{d\, x_{kh}} \Psi_{i_1 i_2}(\overline{x}_{i_1}, \overline{x}_{i_2}) = \sum_{j_1=1}^{n_{i_1}} \sum_{j_2=1}^{n_{i_2}} \frac{d}{d\, x_{kh}} \int_S f_{i_1 j_1}(\overline{x}_{i_1}, \overline{y}) f_{i_2 j_2}(\overline{x}_{i2}, \overline{y}) d\overline{y}.$$

Facciamo notare che questi risultati sono stati già usati implicitamente per modellare $S \setminus S'$ che, addirittura, non è un poligono.

7.2 Alcuni vincoli

Nell'industria tessile la stoffa può essere tagliata solo in particolare versi, per esempio: *il taglio del davanti di una camicia a quadri deve rispettare la geometria dei quadri stessi.*

Se indichiamo con I l'insieme degli indici dei pezzi che devono rispettare particolari orientamenti, allora il vincolo appena esposto può essere formalizzato come segue:

$$x_{i3} \in \{\theta_1, \theta_2, \ldots\},\ i \in I$$

con $\theta_1, \theta_2, \ldots$ opportuni.

Un altro vincolo presente in sartoria ed esprimibile facilmente è il seguente: *il taschino di una camicia a righe deve avere le sue righe che coincidono con*

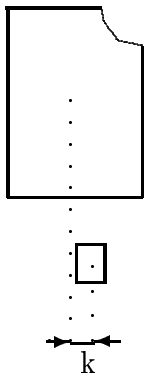


Figura 7.2: Davanti e taschino.

le righe del davanti sul quale viene cucito. Vediamo come si formalizza tale vincolo.

Siano Pol_1 il davanti a righe e Pol_2 il taschino da cucire, supponiamo anche che le righe percorrano la stoffa in senso verticale. Sotto queste ipotesi i vincoli da aggiungere ad FDB sono i seguenti:

$$\begin{aligned} x_{13} &\in \{0, \pi\} \\ x_{23} &= x_{13} \\ x_{21} &= x_{11} + k \end{aligned}$$

dove k è un opportuna costante che serve per stabilire dove andrà posizionato il taschino, vedi fig.7.2.

Per concludere analizziamo il seguente vincolo: *la distanza minima tra due pezzi deve essere maggiore di una costante d.* Questo si esprime sostituendo ai Pol_i dei Pol'_i ottenuti dai primi per un opportuno ingrandimento. Risolto FDB per i Pol'_i questi vengono risostituiti con i Pol_i.

7.3 Implementazioni migliori

7.3.1 Enumerazione parziale

Il primo è più importante miglioramento che si potrebbe apportare all' implementazione è la sostituzione del simulated annealing con un altro algoritmo di

"ricerca globale" deterministico. Una possibile idea è la seguente: consideriamo fisso l'orientamento di tutti i pezzi e discretizziamo il rettangolo. Così facendo abbiamo costruito uno spazio delle soluzioni

$$\Sigma = (\{1, \ldots, k_1\} \times \{1, \ldots, k_2\})^{NumPezzi}.$$

Definiamo una nuova Ψ definita su Σ partendo dalla Ψ del capitolo 2, ponendo:

$$\Psi(\sigma) = \Psi(\overline{x}_1, \ldots, \overline{x}_{NumPezzi}),$$

dove

- $\sigma = ((\alpha_1, \beta_1), \ldots, (\alpha_{NumPezzi}, \beta_{NumPezzi})) \in \Sigma$;
- $\overline{x}_i = (\alpha_i, \beta_i, 0)$.

Nel caso in cui σ ha le ultime n coppie di componenti non specificate poniamo:

$$\begin{aligned} \Psi(\sigma) &= \sum_{i=0}^{NumPezzi-n} \int_S f_i((\alpha_i, \beta_i, 0), \overline{y}) d\overline{y} + \\ &+ 2 \sum_{i=0}^{NumPezzi-1-n} \sum_{j=i+1}^{NumPezzi-n} \int_S f_i((\alpha_i, \beta_i, 0), \overline{y}) f_j((\alpha_j, \beta_j, 0), \overline{y}) d\overline{y}. \end{aligned}$$

La proprietà che segue ci permette la costruzione di un algoritmo d' enumerazione parziale per la minimizzazione di Ψ su Σ:

Proprietà 7.1 *Siano σ_1 e σ_2 due elementi di Σ con le ultime coppie di componenti non specificate:*

$$\begin{aligned} \sigma_1 &= ((\alpha_1, \beta_1), \ldots, (\alpha_i, \beta_i), (.,.), \ldots, (.,.)), \\ \sigma_2 &= ((\alpha_1, \beta_1), \ldots, (\alpha_{i+1}, \beta_{i+1}), (.,.), \ldots, (.,.)). \end{aligned}$$

Allora

$$\Psi(\sigma_1) \leq \Psi(\sigma_2).$$

Appendice A

Algoritmi usati

Questa appendice contiene alcuni risultati classici di Geometria Computazionale e Teoria dell'Ottimizzazione usati in questo lavoro. Salteremo tutte le dimostrazioni lasciando un riferimento bibliografico per maggiori chiarimenti.

A.1 Geometria computazionale

A.1.1 Area di un poligono

Consideriamo il seguente problema: *determinare l'area del poligono convesso avente come vertici i punti* $(x_1, y_1), \ldots, (x_n, y_n)$ *di* R^2. Facciamo presente che non abbiamo imposto nessun vincolo sull'ordinamento dei vertici.

L'algoritmo usato per risolvere questo problema è il seguente:

Algoritmo 15 (AreaPoligono)
function Area($\{(x_1, y_1), \ldots, (x_n, y_n)\}$*):real;*

var
  s:real;

begin
  Ordina gli (x_i, y_i) *in senso orario;*
  Sia (x_1, y_1) *il vertice più a sinistra;*
  Sia (x_m, y_m) *il vertice più a destra;*
  (x_{n+1}, y_{n+1})*:=*(x_1, y_1)*;*

```
  for i:=1 to m do
    s:=s+(y_i +y_{i+1})*(x_i -x_{i+1})/2;
  for i:=m to n+1 do
    s:=s-(y_i +y_{i+1})*(x_i -x_{i+1})/2;
  return s
end
```

esso consta di due fasi:

- ordinamento in $O(n \log n)$ dei vertici in input in modo da poterli scandire in senso orario;
- calcolo in $O(n)$ delle aree sottese dai lati del poligono.

L'algoritmo proposto ha complessità $O(n \log n)$.

A.1.2 Intersezione di due poligoni

Riportiamo in questo paragrafo alcuni risultati riguardanti il problema di trovare l'intersezione di due poligoni convessi P e Q di L e M lati rispettivamente.

Cominciamo con il seguente teorema che limita il numero di vertici di $P \cap Q$:

Teorema A.1 ([8]) *L'intersezione di P e Q è un poligono convesso avente al più $L + M$ vertici.*

Da questo teorema deduciamo che ogni algoritmo che calcola $P \cap Q$ ha una complessità $\Omega(L + M)$. Mostreremo più avanti l'esistenza di un algoritmo che enumera i vertici di $P \cap Q$ con complessità $O(L + M)$; da questi due risultati ricaviamo il seguente:

Teorema A.2 ([8]) *L'intersezione $P \cap Q$ può essere trovata con complessità $\theta(L + M)$.*

L'idea di base dell'algoritmo è quella di suddividere opportunamente il piano che contiene P e Q in L+M regioni; con questa costruzione abbiamo scomposto il problema di trovare $P \cap Q$ in $L + M$ sottoproblemi di intersezione che, come vedremo, vengono risolti in $O(1)$. La chiave di tutto il discorso sta nella scelta delle suddivisioni; seguendo [Preparata,Shamos] prendiamo quella indotta dalle rette verticale che passano per i vertici di P e Q, vedi fig.A.1.

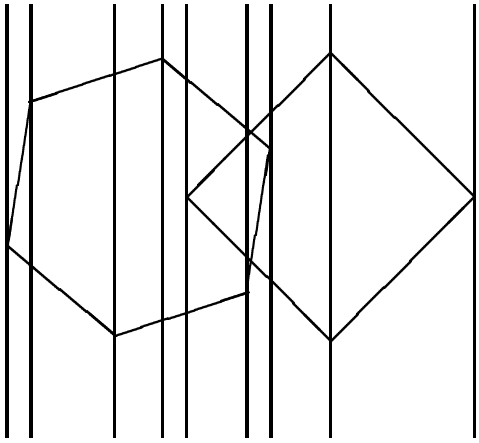


Figura A.1: Un esempio di suddivisione

È facile dimostrare che le rette prese in considerazione, inducono una partizione dei due poligoni in trapezi disgiunti e quindi, riduciamo il problema dell'intersezione dei due poligoni convessi all'intersezione di $L + M$ coppie di trapezi con le basi giacenti su due rette parallele, problema che si risolve in $O(1)$.

In base alle considerazioni appena esposte, mostriamo un algoritmo che enumera i vertici di $P \cap Q$:

Algoritmo 16 (EnumeraVertici)
function EnumeraVertici(v_1,v_2:integer):(real,real);
/ Enumera i vertici dell'intersezione dei due poligoni di indici v1 e v2.*

INPUT
v1,v2 : gli indici dei poligoni da intersecare.

OUTPUT
*Ritorna una lista contenente i vertici del poligono intersezione. */*

begin
$l_1 = \{(x_1, y_1), \ldots, (x_n, y_n)$ *:* (x_i, y_i) *è l'i-esimo vertice che incontro scorrendo in senso antiorario i vertici del poligono* v_1 *partendo dal vertice più a destra fino ad arrivare a quello più a sinistra.*$\}$

$l_2 = \{(x_1, y_1), \ldots, (x_n, y_n) : (x_i, y_i)$ *è l'i-esimo vertice che incontro*

scorrendo in senso antiorario i vertici del poligono v_1 partendo dal vertice più a sinistra fino ad arrivare a quello più a destra.}

$l_3 = \{(x_1, y_1), \ldots, (x_n, y_n) : (x_i, y_i)$ *è l'i-esimo vertice che incontro scorrendo in senso antiorario i vertici del poligono v_2 partendo dal vertice più a destra fino ad arrivare a quello più a sinistra.*}

$l_4 = \{(x_1, y_1), \ldots, (x_n, y_n) : (x_i, y_i)$ *è l'i-esimo vertice che incontro scorrendo in senso antiorario i vertici del poligono v_2 partendo dal vertice più a sinistra fino ad arrivare a quello più a destra.*}

*/*Inizializza la scansione delle liste l1,...,l4 per costruire implicitamente le suddivisione del piano.*/*

ris:={};
u:=0;
v:=min{ x_i *:* (x_i, y_i)*=top(*l_i*)};*
pop(l_i*);*

repeat
  u:=min{ x_i *:* (x_i, y_i)*=top(*l_i*)};*
  pop(l_i*);*

  */*Costruisco i due pentagoni*/*
  tr_1*:=CostruisciTrapezio(*l_1*,*l_2*,v,u);*
  tr_2*:=CostruisciTrapezio(*l_3*,*l_4*,v,u);*

  */*Interseco* pn_1 *e* pn_2 *e aggiorno la soluzione*/*
  int:=IntersecaTrapezi(tr_1*,*tr_2*);*
  ris:=Unione(ris,int);
  v:=u;

until empty(l_1*) or empty(*l_2*) or empty(*l_3*) or empty(*l_4*);*

return (ris);
end

A.2 Teoria dell'Ottimizzazione

Mostriamo in questa sezione la struttura e le principali proprietà di due noti algoritmi: il *Simulated Annealing* e il *Metodo dei Gradienti Coniugati.*

A.2.1 Simulated Annealing

Nella fisica dei materiali, l'*annealing* è conosciuto come un processo termico per ottenere solidi con uno stato a bassa energia (stato groud) mediante un *bagno caldo*. Il processo consiste dei seguenti due passi:

- aumentare la temperatura del bagno caldo finché il solido non fonde;
- far decrescere **lentamente** la temperatura del bagno caldo finché le particelle non si aggregano tra di loro per formare un solido allo stato ground.

Il processo fisico dell'annealing può essere simulato con successo dall' *algoritmo di Metropolis* che si basa su tecniche tipo "Monte Carlo" per la generazione di una successione di stati del solido: vediamo come.

Dato uno stato corrente i ad energia E_i, allora lo stato successivo j, ad energia E_j, viene generato a partire da i applicandovi un meccanismo di pertubazione che, per esempio, può corrispondere allo spostamento di qualche particella.

Se la differenza d'energia $E_j - E_i$ è minore o uguale a zero, allora lo stato j diventa il nuovo stato corrente. Se la differenza d'energia è maggiore di zero, allora lo stato j viene accettato come stato corrente con una certa probabilità data da:

$$\exp\left(\frac{E_i - E_j}{k_B T}\right)$$

dove:

- T è la temperatura del bagno caldo;
- k_B è la costante di Boltzmann.

Consideriamo la seguente:

Definizione A.1 (Istanza di problema d'ottimizzazione) *Una coppia del tipo* $(\mathcal{S}, f)$ *con*

- $\mathcal{S}$ *un insieme discreto,*
- $f : \mathcal{S} \to \mathbf{R}$ *una funzione costo,*

è un'istanza di un problema d'ottimizzazione combinatoria.

Questa definizione si presta bene alla seguente analogia tra *problema d'ottimizzazione combinatoria* e *sistema fisico con molte particelle*:

- le soluzione di un problema d'ottimizzazione combinatoria sono equivalenti agli stati di un sistema fisico;
- il costo di una soluzione è equivalente all'energia di uno stato.

Continuando con l'analogia, introduciamo anche un parametro, detto di controllo, che gioca il ruolo della temperatura nel processo dell'annealing.

Il *Simulated Annealing* può ora essere visto come un'applicazione iterata dell'algoritmo di Metropolis al decrescere del valore del parametro di controllo.

Come in un algoritmo di ricerca locale, assumiamo l'esistenza di un intorno e di un meccanismo di generazione. Consideriamo le seguenti definizioni:

Definizione A.2 (Criterio d'accettazione) *Denotiamo con $(\mathcal{S}, f)$ l'istanza di un problema d'ottimizzazione e con i e j due soluzioni con costo $f(i)$ e $f(j)$, rispettivamente. Allora, il* criterio d'accettazione *determina quando accettare j partendo da i, applicando la seguente probabilità d'accettazione:*

$$\mathbf{P}_c\{\textit{accetto } j\} = \begin{cases} 1 & \textit{se } f(i) \leq f(j) \\ \exp\left(\frac{f(i)-f(j)}{c}\right) & \textit{altrimenti} \end{cases}$$

dove $c \in \mathbf{R}^+$ denota il parametro di controllo.

Definizione A.3 (Transizione) *Una* transizione *è una azione che trasforma la soluzione corrente in un'altra. L'azione consiste dei seguenti passi:*

1. *applicazione del meccanismo di generazione;*
2. *applicazione del criterio d'accettazione.*

Sia c_k il valore del parametro di controllo e L_k il numero di transizioni generate alla k-esima iterazione dell'algoritmo di Metropolis. Allora l'algoritmo del simulated annealing può essere definito come:

Algoritmo 17 (Simulated Annealing)
procedure SIMULATED_ANNEALING;

begin
  INITIALIZE(i_{start},c_0,L_0);
  k:=0;
  i:=i_{start};
  repeat
    for l:= 1 to L_k do begin
      GENERATE(j from S_i);
      if $f(j) \leq f(i)$ then i:=j
      else
      if $\exp\left(\frac{f(i)-f(j)}{c_k}\right) >$ *random[0,1) then i:=j*
    end;
    k:=k+1;
    CALCULATE_LENGHT(L_k);
    CALCULATE_CONTROL(c_k);
  until stopcriterion
end

Tipicamente il simulated annealing si comporta nel seguente modo: inizialmente, per grandi valori di c, vengono accettate anche grandi deterioramenti; al decrescere di c solo piccoli deterioramenti vengono accettati; quando c è prossimo allo zero non vengono più accettati deterioramenti. Questa caratteristica permette all'algoritmo di scappare dai minimi locali, cosa che è nettamente in contrasto con il comportamento degli algoritmi di ricerca locale.

Notiamo che il criterio d'accettazione viene implementato mediante il confronto del valore di $\exp((f(i) - f(j))/c)$ con un numero generato casualmente e in modo uniforme sull'intervallo $[0, 1)$. Inoltre dovrebbe essere ovvio che la velocità di convergenza dell'algoritmo dipende dalla scelta dei parametri L_k e c_k, $k = 1, 2, \ldots$

A.2.2 Metodo dei Gradienti Coniugati

È un particolare algoritmo di ricerca locale per la minimizzazione di funzioni

$$f : \mathbf{R}^n \to \mathbf{R}$$

continue e differenziabili con continuità.

Lo schema dell'algoritmo è il seguente:

Algoritmo 18 (Gradienti Coniugati)
procedure GradientiConiugati(x);

begin
  k:=1;
  g_k*:=*$-f'(x)$*;*
  x_k*:=x;*
  repeat
    k:=k+1;
    x_k*:=RicercaLineare(*x_{k-1}, g_{k-1}*);*
    $g_k := -f'(x_k) + \frac{f'(x_k)f'(x_k)}{f'(x_{k-1})f'(x_{k-1})} g_{k-1}$
  until CriterioStop;
  x:=x_k
end

dove:

- *RicercaLineare*(x, g) ritorna il punto $x + \overline{\lambda}g$ con

$$\overline{\lambda} = \arg\min\{f(x + \lambda g) \mid \lambda \geq 0\};$$

- *CriterioStop* determina quando deve terminare il metodo; tipicamente questo avviene quando il miglioramento ottenuto con la nuova soluzione non è maggiore di un prefissato valore.

Il seguente teorema fornisce una misura della bontà del metodo:

Teorema A.3 *Se la funzione* $f : \mathbf{R}^n \to \mathbf{R}$ *è quadratica, strettamente convessa, l'algoritmo dei gradienti coniugati fornisce la soluzione ottima in n passi al più.*

Notiamo che se manca l'ipotesi di convessità su f, l'algoritmo determina un minimo locale.

Bibliography

[1] H. Shin, A. L. Sangiovanni-Vincentelli, C. H. Séquin. *Two-Dimensional compaction by "zone refining"*, 23rd Design Automation Conference 1986, 115-112

[2] D. F. Wong, C. L. Liu. *A new algorithm for floorplan design*, 23rd Design Automation Conference 1986, 101-107

[3] H. Spruth, G. Sigl, F. Johannes. *Parallel algorithms for slicing based final placement*, Technische Universität München 1992

[4] J.P. Uhry. *Applications of simulated annealing in operation research*, in *New methods in optimization and their industrial uses*, ISNM 87

[5] P. De Santis. *Strumenti cad per la progettazione di sistemi VLSI*, Tesi di laurea in Scienze dell'Informazione, Pisa 1989

[6] A. Törn, A. Žilinskas. *Global optimization*, Springer-Verlag 1989

[7] Aarts, Korst. *Simulated Annealing and Boltzmann Machines*, New York:Wiley 1989

[8] F. Preparata, M. Shamos. *Computational geometry: an introduction*, Springer-Verlag, 1985

[9] G. Gallo. *Dispense di TMO*, Pisa 1993