Sull'algoritmo DIRECT e le sue evoluzioni

Gian Maria Negri Porzio

Luglio 2016

Indice

1 Introduzione

In molte applicazioni è necessario calcolare il minimo globale di una funzione a valori reali. Spesso non abbiamo a disposizioni informazioni algebriche, ma solamente i valori puntuali: in letteratura si parla di *black-box functions*. È dunque chiara l'importanza di sviluppare metodi che non utilizzino informazioni sulle derivate, conosciuti in letteratura come *derivative-free methods* [7]. In questa relazione, che nasce come supporto per un seminario, introdurremo l'algoritmo DIRECT. Abbiamo deciso di seguire un percorso storico: descriveremo inizialmente l'algoritmo di Shubert, di cui DIRECT si può considerare diretto discendente [8]; in seguito analizzeremo la versione originale di DIRECT seguendo le orme dell'articolo di Jones, Perttunen e Stuckman [6]; infine parleremo di un'importante modifica dei primi anni Duemila [3]: Finkel e Kelley avevano scoperto che in alcune situazioni la velocità

di convergenza di DIRECT diminuiva bruscamente: proposero dunque una nuova scalatura del metodo per ovviare a questo problema. Daremo infine una breve lista dei programmi che implementano DIRECT (o sue semplici modificazioni) nei più importanti linguaggi di programmazione di analisi numerica, come MATLAB, C++.

La principale forza di DIRECT risiede nella debolezza delle ipotesi di regolarità che chiediamo per la funzione f che vogliamo minimizzare: mentre per numerosi metodi necessitiamo che f sia derivabile con continuità, nel nostro caso sarà sufficiente che sia una funzione Lipschitziana.

2 Il problema

Iniziamo descrivendo il contesto del nostro problema. Benché alcune delle seguenti definizioni possano essere espresse nella generalità degli spazi metrici, noi ci limiteremo al caso di $\mathbb{R}^n$, che è quello che ci servirà nella pratica.

Definizione 2.1. Un sottoinsieme $R \subset \mathbb{R}^n$ si dice *iperrettangolo (chiuso)* se può essere scritto nella forma

$$R = \bigotimes_{i=1}^{n} [a_i, b_i],$$

con $a_i, b_i \in \mathbb{R}$.

Definizione 2.2. Sia $f : R \to \mathbb{R}$ una funzione definita su un iperrettangolo $R \subset \mathbb{R}^n$ a valori reali. Dato $K \in \mathbb{R}^+$ diremo che f è *K-lipschitziana* se vale

$$\big|f(x) - f(y)\big| \leq K\|x - y\| \qquad \forall x, y \in R.$$

Se non siamo interessati alla costante, diremo semplicemente che f è lipschitziana.

Possiamo ora formalizzare il nostro problema nella versione più generale.

Problema. Data $f : R \to \mathbb{R}$ funzione lipschitziana da un iperrettangolo $R \subset \mathbb{R}^n$ a $\mathbb{R}$, vogliamo $\bar{x}$ tale che

$$f(\bar{x}) \leq f(y) \qquad \forall y \in R,$$

ovvero x è un minimo globale di f.

Osservazione 2.3. Il problema è ben posto, ovvero il minimo globale di f esiste sempre. Infatti la lipschitzianità implica la continuità e un iperrettangolo chiuso è un insieme compatto, dunque f ammette minimo e massimo per il teorema di Heine-Cantor.

3 Uno sguardo all'algoritmo di Schubert

Nell'articolo originale Shubert è interessato a calcolare il massimo locale di una funzione [8]. Per coerenza con le parti successive di questa relazione considereremo il calcolo del minimo. Restringeremo le nostre valutazioni al caso monodimensionale. Sarà infatti poi evidente il motivo per cui, a livello computazionale, l'algoritmo di Schubert non sia utilizzabile in più dimensioni.

Sia f una funzione K-lipschitziana da un intervallo $[a, b]$ a $\mathbb{R}$. L'idea dietro questo metodo è molto intuitiva e non troppo sorprendente. Prima di poterne parlare, abbiamo bisogno di alcune premesse. Dalla lipschitzianità seguono le seguenti disuguaglianze:

$$f(x) \geq f(a) - K(x - a), \tag{1}$$
$$f(x) \geq f(b) + K(x - b). \tag{2}$$

Definiamo ora:

$$X(a, b, f, K) = \frac{a + b}{2} + \frac{f(a) - f(b)}{2K}, \tag{3}$$
$$B(a, b, f, K) = \frac{f(a) + f(b)}{2} - K(b - a). \tag{4}$$

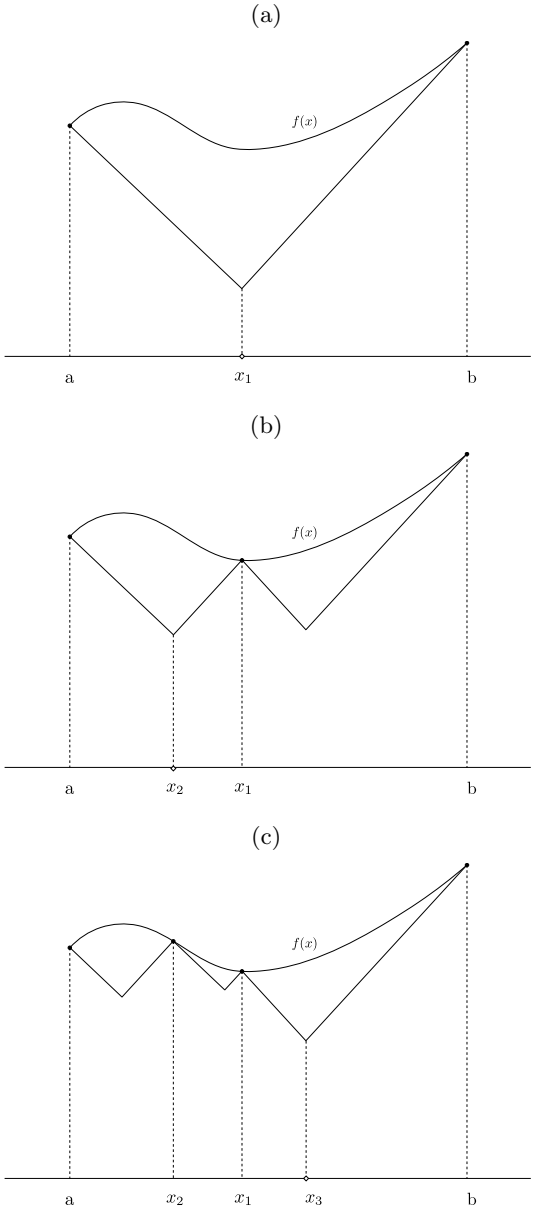
Queste due quantità corrispondono alla ascissa e all'ordinata del minimo valore che può assumere la nostra funzione nell'intervallo $[a, b]$. Seguendo la figura 1, possiamo vedere i primi passi dell'algoritmo in maniera informale. Iniziamo valutando f nei due estremi a e b, memorizziamo il valore minimo $f(a)$ e determiniamo $x_1 = X(a, b, f, K)$ (figura 1a). In questo modo otteniamo i due intervalli $[a, x_1]$ e $[x_1, b]$. Calcoliamo $f(x_1)$, che è il nostro nuovo minimo, valutiamo $X(\cdot, \cdot, f, K)$, scegliendo l'intervallo che minimizza $B(\cdot, \cdot, f, K)$ (in 1b è $[a, x_1]$). Calcoliamo $f(x_2)$ e manteniamo $f(x_1)$ come minimo di f. Nella figura 1c vediamo la successiva iterazione applicata agli intervalli $[a, x_2]$, $[x_2, x_1]$ e $[x_1, b]$, che ci restituisce il minimo di $X(\cdot, \cdot, f, K)$ in x_3. Terminiamo infine queste iterazioni quando la differenza tra due successivi aggiornamenti del minimo è minore di una predeterminata soglia ε. Nel listing 1 troviamo lo pseudocodice relativo.

La dimostrazione di convergenza dell'algoritmo di Shubert si può trovare sull'articolo originale, ma l'idea intuitiva è abbastanza chiara già dopo questi primi passaggi [8]. Stiamo infatti approssimando dal basso f con una funzione lineare a tratti il cui modulo della derivata è dato da K.

L'algoritmo di Shubert sembra fare tutto ciò che aveva promesso: trova il minimo globale di f senza usare informazioni algebriche. Tuttavia due carenze importanti sono ancora presenti:

- La conoscenza della costante di Lipschitz.
- L'inefficienza per problemi multidimensionali.

Figura 1: Primi tre passi dell'algoritmo di Shubert.



Listing 1: L'algoritmo di Shubert.

```
function Fmin = shubert(f,a,b,K)
  Fmin = min{f(a), f(b)};
  Inter = {[a,b]};
  while termine
      [l,u] = min_B(f,K){Inter};
      x = X(l,u,f,K);
      update(Fmin); Inter.del([l,u]);
      Inter.add([l,x]); Inter.add([x,u]);
      check(termine);
  end
  return Fmin
end
```

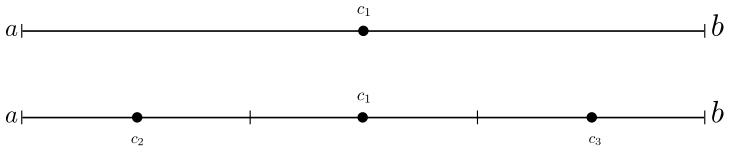
Abbiamo infatti utilizzato K per trovare i punti x_i. È sì vero che quest'ultima non rientra nell'insieme delle informazioni sulla differenziabilità di f, tuttavia calcolarla attraverso i valori puntuali della funzione non è una strada percorribile. Si è quindi obbligati ad utilizzare stime dall'alto, spesso ricavabili solo dal contesto del problema specifico; queste per loro natura non possono essere molto precise, dunque la convergenza al minimo locale non sarà sufficientemente veloce. Il secondo punto sancisce definitivamente l'inapplicabilità del metodo nei problemi reali. L'algoritmo richiede infatti la valutazione della funzione in esame agli estremi dell'intervallo, che in più dimensioni diventano iperrettangoli chiusi: i vertici sono 2^n, perciò il costo computazionale aumenta esponenzialmente.

4 L'algoritmo DIRECT

Nel 1993 Jones, Perttunen e Stuckman presentarono un algoritmo che risolveva i principali difetti del lavoro di Shubert [6]. Con un carino gioco di parole, lo chiamarono DIRECT, in quanto si basa sul suddividere lo spazio di partenza in sotto-rettangoli (*DIviding RECTangles*) ed è un metodo diretto e deterministico, ovvero valuta la funzione obiettivo in punti generati dalle iterazioni precedenti dell'algoritmo.

L'intuizione per abbassare il costo computazionale dell'algoritmo è di valutare la funzione nel centro dell'iperrettangolo, invece che nei suoi vertici. Per semplicità ne esporremo prima l'idea in una dimensione, per poi generalizzare al caso multidimensionale.

Figura 2: Suddivisione dell'intervallo $[a, b]$.



4.1 DIRECT in una dimensione

Sia f una funzione K-lipschitziana da un intervallo $[a, b]$ a $\mathbb{R}$ e sia $c = \frac{a+b}{2}$ il suo punto medio. Valgono le disuguaglianze:

$$f(x) \geq f(c) + K(x - c) \qquad \forall x \leq c, \tag{5}$$
$$f(x) \geq f(c) - K(x - c) \qquad \forall x \geq c. \tag{6}$$

Vogliamo ora utilizzare (5) e (6) per suddividere l'intervallo in sotto-intervalli più piccoli. Vogliamo anche conservare la proprietà che $c = c_1$ e i punti successivi siano al centro dei corrispettivi intervalli. È dunque naturale una suddivisione ternaria $[a, \frac{b+a}{3}]$, $[\frac{b+a}{3}, \frac{2(b+a)}{3}]$ e $[\frac{2(b+)a}{3}, b]$, come vediamo nella figura 2.

Dobbiamo ora capire quali sono gli intervalli migliori dove continuare la nostra ricerca del minimo globale. Facciamo prima un passo intermedio. Supponiamo che $[a, b]$ sia stato suddiviso in sotto-intervalli disgiunti $[a_i, b_i]$ di centro c_i. Rappresentiamo ognuno di questi intervalli $[a_i, b_i]$ come un punto di $\mathbb{R}^2$, la cui ascissa è c_i e l'ordinata è $f(c_i)$. Questa rappresentazione grafica, che vediamo in figura 3, ha il pregio di sottolineare la bontà di ciascun sotto-intervallo in funzione del rapporto tra ricerca globale e ricerca locale: quanto è maggiore l'ascissa, tanta più percentuale dell'intervallo originario stiamo considerando; viceversa, quando consideriamo l'ordinata, ci stiamo concentrando sui valori della funzione nella sua "località".

Supponiamo per ora di conoscere la costante di Lipschitz K. La famiglia delle rette $y = K(x - c_i) + f(c_i)$ evidenzia in $x = 0$ il valore minimo che f potrebbe assumere nell'intervallo $[a_i, b_i]$. Sarebbe dunque logico continuare la ricerca nell'intervallo $[a_j, b_j]$ che minimizza questo valore. Avevamo però iniziato questa sezione dicendo di voler fare a meno dell'informazione sulla costante di Lipschitz. Ci chiediamo dunque quali siano gli intervalli che minimizzano $f(c_i) - \tilde{K}c_i$ al variare di $\tilde{K} \in \mathbb{R}^+$. È facile vedere che la risposta è data dall'inviluppo convesso dei punti che rappresentano gli intervalli, come vediamo in figura 4. Più formalmente, nel caso generale:

Definizione 4.1 (Iperrettangoli potenzialmente ottimi)**.** Sia R un iperrettangolo in $\mathbb{R}^n$ partizionato in R_i per $i = 1, \ldots, m$ e siano c_i i loro centri. Sia f una funzione lipschitziana definita su R a valori reali e sia $f_{\min}$ il valore minimo finora computato. Diciamo che R_j è un *iperrettangolo potenzialmente*

Figura 3: Selezione del sotto-intervallo ottimale, K nota.

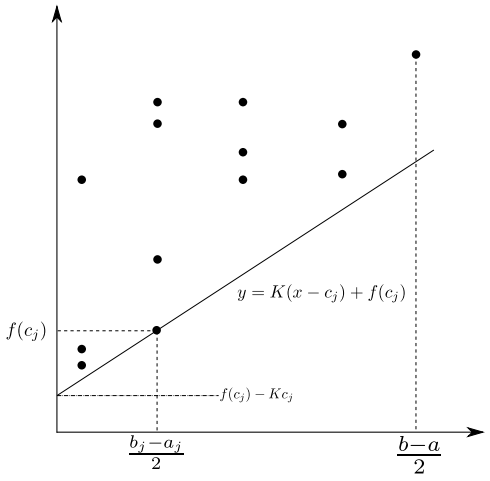
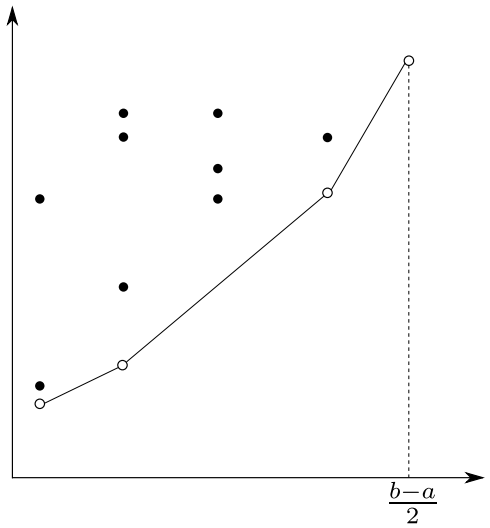


Figura 4: Selezione dei sotto-intervalli ottimali, K ignota.



Listing 2: DIRECT in $\mathbb{R}$.

```
function Fmin = direct(f,a,b, maxit)
 Int = {[a,b]} # insieme degli intervalli
 Fmin = f( (a+b)/2 )
 for it =1:maxit
   S = optInt(Int)
   for j in S
     divide(j)
     update(Fmin) # nuovo valore minimo di f
     update(Int) # aggiungiamo i nuovi intervalli
   endfor
 endfor
 return Fmin
end
```

ottimo se esiste $\tilde{K}$ tale che:

$$f(c_j) - \tilde{K}c_j \leq f(c_i) - \tilde{K}c_i \qquad \forall i = 1, \ldots, m, \tag{7}$$

$$f(c_j) - \tilde{K}c_j \leq f_{\min} - \varepsilon|f_{\min}|\,, \tag{8}$$

dove $\varepsilon > 0$ è una costante definita a priori.

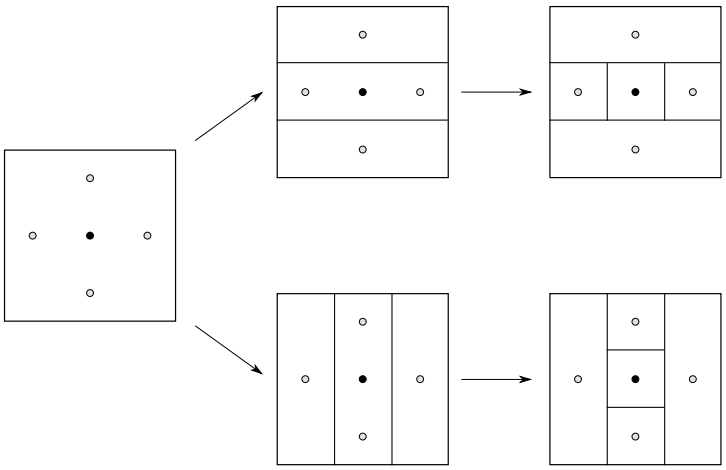
Come si può intuire, la costante ε serve solo da un punto di vista computazionale: non vogliamo aggiungere iperrettangoli se questi non migliorano di un'adeguata percentuale il minimo attuale. Vedremo però che il ruolo di ε è più importante di quanto sembri [3]. Possiamo trovare lo pseudo codice in stile MATLAB di DIRECT in una dimensione nel listing 2.

Osserviamo che l'algoritmo termina dopo un numero prestabilito di iterazioni. Si potrebbe utilizzare un altro meccanismo di terminazione se si conoscesse la costante ottima di Lipschitz, in modo da poter stimare già a priori il valore minimo su ogni sotto-intervallo e terminare l'algoritmo raggiunta una certa soglia di precisione.

È interessante notare che Jones, Perttunen e Stuckman ritenevano che la scelta di ε non influenzasse il numero di passi compiuti dall'algoritmo [6]. Testarono diversi valori in $[10^{-7}, 10^{-3}]$ e consigliarono di fissare $\varepsilon = 10^{-4}$. Tuttavia Finkel e Kelley mostrarono come in alcune situazioni patologiche questa scelta rallenti notevolmente la convergenza. Ciò sarà oggetto della sezione 5.

Osservazione 4.2. Quando avevamo parlato dell'algoritmo di Shubert, avevamo scritto che un alto valore della costante di Lipschitz K causava un rallentamento della convergenza al minimo globale della funzione f. Ora possiamo vedere questo rallentamento come un'eccessiva enfasi della

Figura 5: Le due possibili divisioni del quadrato in $\mathbb{R}^2$.



ricerca globale sulla ricerca locale. L'obiettivo di DIRECT è trovare il giusto compromesso tra le due.

Osservazione 4.3. Un possibile algoritmo per trovare l'inviluppo convesso di un insieme finito di punti in R^2 è stato proposto da Graham in [5]. Lo vedremo velocemente nell'appendice.

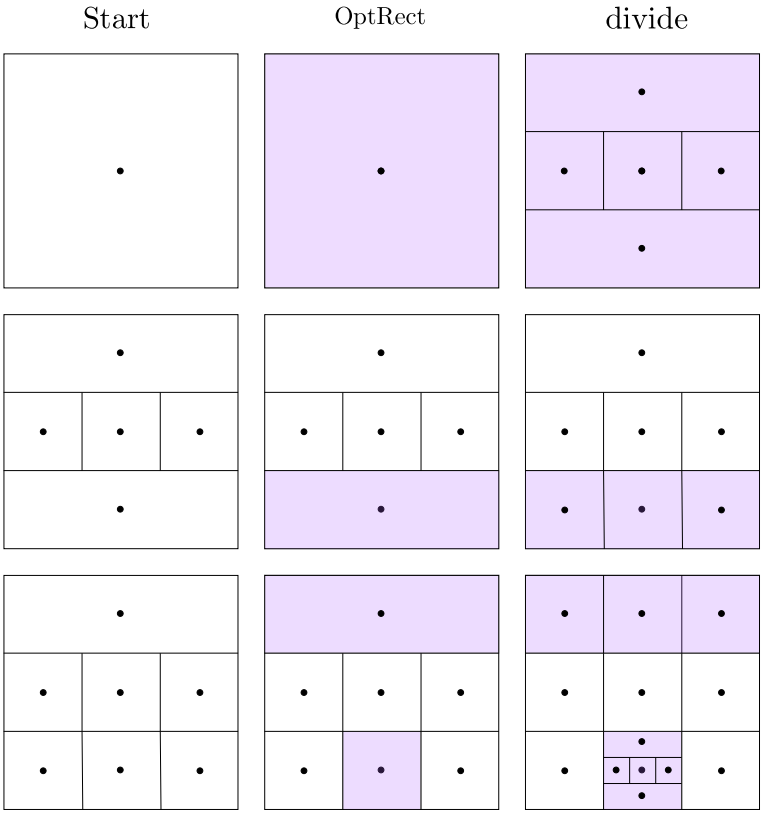
4.2 DIRECT in più dimensioni e dimostrazione della convergenza

Sia $f : R \to \mathbb{R}$ una funzione Lipschitziana su un iperrettangolo $R \subset \mathbb{R}^n$ a valori reali. A meno di riparametrizzazione possiamo supporre che R sia l'ipercubo unitario $[0,1]^n$, c il suo baricentro e $\delta = \frac{1}{3}$. La prima differenza del caso multidimensionale è scegliere come dividere R. Per fare questo consideriamo i $2n$ punti $d_i = c \pm \delta e_i$, dove e_i rappresenta l'i-esimo vettore della base canonica. In generale sono possibili $n!$ divisioni, che rappresentiamo in $\mathbb{R}^2$ nella figura 5. A livello sperimentale si osservò che i risultati migliori si ottengono quando la funzione ha valori minori sui rettangoli più grandi [6]. Euristicamente questo non ci deve stupire: stiamo "migliorando" il rapporto tra ricerca globale e locale. Più formalmente, definiamo

$$w_i = \min\{f(c_i - \delta e_i), f(c_i + \delta e_i)\}$$

e suddividiamo R secondo l'ordine crescente dei w_i. Dopo la prima divisione si saranno formati dei rettangoli. Negli step successivi sarà necessario considerare solo le dimensioni con lunghezza maggiore: in questo modo evitiamo che si restringano eccessivamente lungo un'unica direzione . Tale condizione è necessaria per la convergenza, come vedremo nel teorema 4.4.

Figura 6: I primi tre passi di DIRECT per la funzione di Branin.



Possiamo ora riportare lo pseudo codice per DIRECT in più dimensioni, come vediamo nel listing 3. Nella figura 6 rappresentiamo invece i primi 3 passi per la funzione di Branin:

$$(y - \frac{5.1}{4\pi^2}x^2 + \frac{5}{\pi}x - 6)^2 + 10(1 - \frac{1}{8\pi})\cos x + 10 \tag{9}$$

con $(x, y) \in [-5, 10] \times [0, 15]$.

Teorema 4.4 (Convergenza dell'algoritmo DIRECT)**.** *Sia $f : R \to \mathbb{R}$ una funzione Lipschitziana su un iperrettangolo $R \subset \mathbb{R}^n$ a valori reali. Sia (x_i) la successione dei punti dell'algoritmo DIRECT. Allora (x_i) ha limite $\bar{x}$ e*

$$f(\bar{x}) = \min_{x \in R} f(x)$$

Dimostrazione. L'idea della dimostrazione consiste nel provare che l'insieme dei punti campionati è denso in R, ovvero per ogni x in R e per ogni $\delta > o$ esiste un x_i tale che $\|x - x_i\| < \delta$. La successione converge infine al minimo globale perché l'algoritmo estrae ogni volta il valore minimo computato fino a quel momento. Per semplicità di notazione, supponiamo che R sia

Listing 3: DIRECT in $\mathbb{R}^n$.

```
function Fmin = direct(f, R, maxit)
  Fmin = f(c)
  Rect = {R} # insieme degli iperrettangoli
  for it =1:maxit
    S = optRect(Rett) #i potenzialmente ottimi
    for j in S
      divide(j)
      update(Fmin) # nuovo valore minimo di f
      update(Rect) # aggiungiamo i nuovi rettangoli
    endfor
  endfor
  return Fmin
end
```

l'ipercubo unitario. In questo modo le lunghezze dei lati dei rettangoli in cui DIRECT suddivide R saranno 3^{-k} per qualche $k \in \mathbb{N}$. Consideriamo ora un rettangolo A tra quelli presenti all'r-esima iterazione. Poiché abbiamo imposto che le suddivisioni si effettuino solo sui lati più lunghi, nessun lato di lunghezza $3^{-(k+1)}$ sarà considerato dall'algoritmo prima che abbia toccato tutti quelli di lunghezza 3^{-k}. Questo implica che, chiamati $j = r \mod n$ e $k = (r-j)/n$, A avrà j lati di lunghezza $3^{-(k+1)}$ e $n-j$ lati di lunghezza 3^{-k}. Di conseguenza la distanza del centro dai vertici sarà data da

$$d = \frac{\sqrt{j3^{-2(k+1)} + (n-j)3^{-2k}}}{2}. \tag{10}$$

Sia ora r_t il minimo numero di divisioni di un rettangolo all'iterazione t. Affermo che $\lim_{t\to\infty} r_t = \infty$. Se così non fosse, esisterebbe un'iterazione t' in cui viene raggiunto il limite $r_{t'}$. Sia ora N il numero di rettangoli divisi $r_{t'}$ volte alla fine dell'iterazione t'. Tutti questi avranno la stessa distanza dai vertici (il valore delle ascisse nel grafico 4), ma possono differenziarsi per il valore della funzione nel loro centro. Sia ora j il rettangolo con il minimo valore $f(c_j)$. Questo sarà potenzialmente ottimo: le equazioni (7) e (8) sono soddisfatte con $\tilde{K} > \max\{K_1, K_2\}$, ove

$$K_1 = \frac{f(c_j) - f_{\min} + \varepsilon|f_{\min}|}{d_j}$$
$$, K_2 = \max_{1\le i\le m} \frac{f(c_j) - f(c_i)}{d_j - d_i}.$$

Dunque j sarà suddiviso e all'iterazione $t'' = t' + N$ tutti gli originali N rettangoli saranno stati suddivisi a loro volta e $r_{t''} > r_{t'}$, assurdo. Perciò

$\lim_{t\to\infty} r_t = \infty$ e, conseguentemente, la distanza d (10) andrà a zero. Quindi i punti campionati sono densi in R. □

Corollario 4.5. *Data f come nel teorema 4.4 e $\varepsilon > 0$, l'algoritmo* DIRECT *restituisce un punto c tale che*

$$\left| f(c) - \min_{x\in R} f \right| < \varepsilon.$$

per un numero di iterazioni sufficientemente elevato.

Dimostrazione. Dal teorema 4.4 segue che i centri dei rettangoli sono densi in R. Dato $\bar{x}$ in cui f assume il minimo globale, esiste c con $\|\bar{x} - c\| < \delta$, con δ fissato a posteriori dipendente dal numero di iterazioni. La tesi segue quindi dalla continuità di f e dall'arbitrarietà di δ. □

Nella prossima sezione vedremo un'importante modifica a DIRECT. Altri miglioramenti erano stati proposti prima di quell'articolo. Ad esempio Gablosky suggerì di dividere al più un solo iperrettangolo potenzialmente ottimo per ciascuna taglia, anche nel caso ne fossero stati presenti più d'uno (in questa situazione se ne sceglieva uno qualsiasi)[4]. Lo stesso Jones successivamente consigliò di modificare la suddivisione dei rettangoli: invece che tagliare lungo tutte le dimensioni maggiori, si sarebbe dovuto dividere lungo una sola; questa viene scelta ogni volta in modo che non si tagli troppo spesso lungo una sola direzione. In questo modo, indipendente dalla dimensione, vengono aggiunti solo due punti alla volta. Maggiori dettagli e altre modifiche minori si possono trovare in [4, Capitolo 3].

5 Una modifica a DIRECT: il problema dell'additive scaling

Finkel e Kelley si accorsero in un articolo del 2006 che l'algoritmo DIRECT com'era stato proposto fino a quel momento era sensibile all'*additive scaling*, ovvero la convergenza di funzioni del tipo

$$f(x) + M$$

con $M \gg 1$ peggiorava notevolmente [3]. Notarono che tutto questo era dovuto al valore della costante ε in (8). Formalmente lo possiamo vedere nei due teoremi 5.1 e 5.2. Il primo ci mostra che l'iperrettangolo A contenente il valore ottimo non sarà mai campionato fintanto che i suoi vicini non avranno la stessa dimensione; il secondo riporta invece il caso peggiore in assoluto: l'iperrettangolo A non verrà preso in considerazione fintanto che tutti quanti gli altri iperrettangoli avranno la sua dimensione. Si nota che entrambe le condizioni (11) e (15) vengono verificate quando la funzione soffre appunto di "bad additive scaling". Riporteremo solo la dimostrazione di 5.1, mentre la seconda può essere trovata in [3].

Teorema 5.1. *Sia $f : R \to \mathbb{R}$ una funzione K-lipschitziana definita su un iperrettangolo $R \subset \mathbb{R}^n$. Sia S l'insieme degli iperrettangoli creati da* DIRECT*, sia A un ipercubo con centro c e lunghezza dei lati 3^{-l} e d_T la distanza del centro dai vertici nell'iperrettangolo T . Supponiamo che*

1. $d_A \leq d_T \quad \forall T \in S$.
2. $f(c) = f_{min} \neq 0$.

Se inoltre

$$d_A < \frac{\varepsilon|f(c)|}{2K}\left(\sqrt{n+8} - \sqrt{n}\right) \tag{11}$$

allora A non sarà un iperrettangolo potenzialmente ottimo fintanto che tutti gli iperrettangoli di centro $c+3^{-l}e_i$ $\forall i = 1, \ldots, n$ avranno la stessa dimensione di A.

Dimostrazione. Affinché A sia potenzialmente ottimo, devono valere le condizioni (7) e (8); la prima implica che deve esistere $\tilde{K}$ tale che

$$\tilde{K} \leq \frac{f(c_T) - f(c)}{d_T - d_A}$$

per ogni $T \in S$ tale che $d_T > d_A$. Prendiamo il miglior $\tilde{K}$ possibile, ovvero

$$\tilde{K} = \min_{T \in S} \frac{f(c_T) - f(c)}{d_T - d_A} = \frac{f(c_{\tilde{T}}) - f(c)}{d_{\tilde{T}} - d_A} \tag{12}$$

e mostriamo che, se esiste un iperrettangolo adiacente più grande di A, allora (8) non viene verificata. Sappiamo che $d_A = 3^{-l}\frac{\sqrt{n}}{2}$ e dunque l'ipotesi diventa

$$\frac{2K}{\sqrt{n+8} - \sqrt{n}} < \frac{\varepsilon|f(c)|}{\frac{3^{-l}}{2}\sqrt{n}} \tag{13}$$

Il più piccolo $T \in S$ di dimensione maggiore di A avrà $n-1$ lati di lunghezza 3^{-l} e uno di lunghezza $3^{-(l-1)}$, da cui

$$d_T = \frac{1}{2}\sqrt{(n-1)3^{-2l} + 3^{-2(l-1)}} = \frac{3^{-l}}{2}\sqrt{n+8} \quad .$$

Dalla definizione di lipschitzianità segue

$$f(c_{\tilde{T}}) - f(c) = f(c \pm 3^{-l}e_i) - f(c) \leq K3^{-l} \quad .$$

Dunque

$$\tilde{K} = \frac{f(c_{\tilde{T}}) - f(c)}{d_{\tilde{T}} - d_A} \leq \frac{2K}{\sqrt{n+8} - \sqrt{n}}, \tag{14}$$

e unendo (13) e (14) otteniamo

$$\tilde{K} < \frac{\varepsilon\left|f(c)\right|}{\frac{3^{-l}}{2}\sqrt{n}} \quad .$$

Dal fatto che $f(c) = f_{\min}$ otteniamo

$$f(c) - \tilde{K}d_A > f_{\min} - \varepsilon|f_{\min}|$$

in aperto contrasto con (8). □

Teorema 5.2. *Siano f, c, A, R, S come sopra e sia*

$$f^* = \min_{x \in R} f(x).$$

Se

$$f^* > \frac{K\sqrt{n}}{\varepsilon(\sqrt{1+\frac{8}{n}}-1)} \tag{15}$$

allora A non è potenzialmente ottimo se esiste $A' \in S$ più grande di A.

Dimostrazione. La dimostrazione si trova in [3]. □

Come abbiamo visto, la dimostrazione si regge sul ruolo della costante ε e sull'equazione (8) nella definizioni di rettangoli potenzialmente ottimi. L'idea di Finkel e Kelley è modificare questa condizione e fare in modo che non subisca gli effetti negativi dell'additive scaling. Definiscono perciò f^t_{median} come la media aritmetica dei valori di f calcolati fino all'iterazione t e gli iperrettangoli potenzialmente ottimi diventano:

Definizione 5.3 (Iperrettangoli potenzialmente ottimi secondo Finkel e Kelley)**.** Sia R un iperrettangolo e sia f una funzione lipschitziana ivi definita a valori reali. Supponiamo di essere all'iterazione t e chiamiamo $f_{\min}$, f^t_{median} il minimo e la media aritmetica dei valori calcolati fino a quel momento. Diciamo che A è un *iperrettangolo potenzialmente ottimo* secondo Finkel e Kelley se esiste $\tilde{K}$ tale che

$$f(c_A) - \tilde{K}\frac{d_A}{2} \leq f(c_T) - \tilde{K}\frac{d_T}{2} \qquad \forall T \text{ iperrettangolo} \tag{16}$$

$$f(c_A) - \tilde{K}d_A \leq f_{\min} - \varepsilon\left|f_{\min} - f^t_{\text{median}}\right| \tag{17}$$

dove $\varepsilon > 0$ è una costante definita a priori.

I risultati mostrati dai due autori sulle funzioni test sono molto interessanti. Considerano, tra le altre altre, la funzione di Branin (9), le funzioni m-dimensionali di Shekel

$$-\sum_{i=1}^{m}\left(\sum_{j=1}^{4}(x_{ij} - C_{ij})^2 + B_i\right)^{-1},$$

e di Hartmann

$$\sum_{i=1}^{4} a_i \exp\left(-\sum_{j=1}^{m} A_{ij}(x_j - P_{ij})^2\right) \quad .$$

Mostrano che la loro modifica non comporta conseguenze sul numero di passi per i problemi originali: entrambe le versioni di DIRECT convergono in $\approx 10^2$ valutazioni. Tuttavia, quando considerano i problemi perturbati $f(x) + 10^5$, DIRECT originale non è in grado di trovare il minimo entro $200,000$ valutazioni, mentre la versione modificata non subisce alcun rallentamento [3]. La maggior parte di queste funzioni test è già implementata in MATLAB e in R. I valori consigliati e altre funzioni test si possono trovare in [1] oppure online in [9].

A Il software oggi disponibile

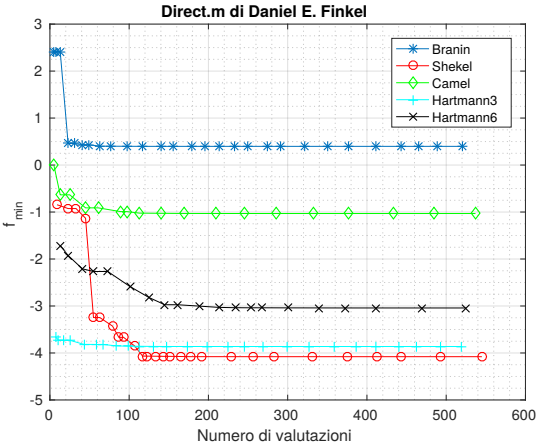
L'algoritmo DIRECT è sicuramente una delle soluzioni preferite per risolvere i problemi di minimo globale. È quindi naturale che oggi si trovino numerose implementazioni in tutti i più importanti linguaggi utilizzati in campo numerico, quali C/C++ e MATLAB. Nel 2012 Rios e Sahinidis hanno pubblicato un esaustivo resoconto in cui comparano le diverse performance su numerosi problemi test [7]. Tra i software ivi presentati spiccano senza dubbio quelli programmati dalla TOMLAB, una piattaforma che si occupa di ottimizzazione perfettamente integrata con MATLAB. Più precisamente

- GLCFAST: implementazione naturale di DIRECT.
- MULTIMIN: tenta di trovare di tutti i minimi locali con un metodo multi-start attraverso un risolutore non-lineare.
- GLCCLUSTER: utilizza GLCFAST come ricerca globale per trovare dei punti che vengono analizzati da un algoritmo di clustering per ottenere un insieme di partenza; su questo si utilizza una ricerca locale e l'output ottenuto viene migliorato nuovamente da GLCFAST.

Come il lettore può immaginare, tutto questo è software proprietario, dunque abbiamo di fronte i due consueti svantaggi. Il primo è il prezzo: per il software di base un privato deve pagare circa 1000 €, mentre in ambito accademico la cifra si ferma ai 300 €. Se tuttavia ne avessimo bisogno per un periodo molto limitato, è disponibile una prova gratuita per tre settimane. Il secondo è l'impossibilità di studiare il codice sorgente: non sappiamo, ad esempio, quale ε venga utilizzato, come vengano scelti gli iperrettangoli potenzialmente ottimali oppure che versione di DIRECT stiamo veramente utilizzando.

Per quanto riguarda alternative più open-source, la libreria C++ DAKOTA, che contiene un'implementazione di DIRECT, è distribuita attraverso la Lesser GPL, dunque scaricabile gratuitamente. Nei test non si è però comportato benissimo: lo troviamo spesso nelle ultime posizioni o nei migliori casi a metà [7]. Se invece siamo veramente interessati al codice sorgente, lo stesso Finkel aveva implementato DIRECT: possiamo trovare il sorgente nella sua pagina, la cui ultima modifica risale però al 2004 [2]. È ancora codice MATLAB, ma alcune modifiche lo possono trasformare in Octave e/o Julia. Allo stesso indirizzo è possibile trovare anche una *user guide* per utilizzarlo, tuttavia è presente un'imprecisione: i parametri opzionali vengono passati attraverso una *struct*, mentre nella guida si parla di semplici vettori. Questo può causare un po' di confusione durante la sperimentazione. Nella figura 7 possiamo vedere la velocità di convergenza di DIRECT nella versione implementata da Finkel: notiamo come in meno di 200 valutazioni si raggiunga il minimo con un'ottima precisione. Il codice delle funzioni test utilizzate con i relativi valori standard si può trovare in [9].

Figura 7: Velocità di convergenza di DIRECT.



B Lo scan di Graham per l'inviluppo convesso

Abbiamo visto che trovare gli iperrettangoli potenzialmente ottimi corrisponde a calcolare l'inviluppo convesso "inferiore" (vedasi figura 4) di un insieme finito di punti in $\mathbb{R}^2$. Uno degli algoritmi più famosi per risolvere questo problema è l'algoritmo di Graham, presentato per la prima volta in [5]. Ha un costo computazionale, nel caso generale, di $O(n \log n)$. Vediamo i passi qui di seguito e nella figura 8:

1. Troviamo il punto P_0 con ordinata minima; a parità scegliamo quello con ascissa minore. Lo facciamo in $O(n)$ operazioni

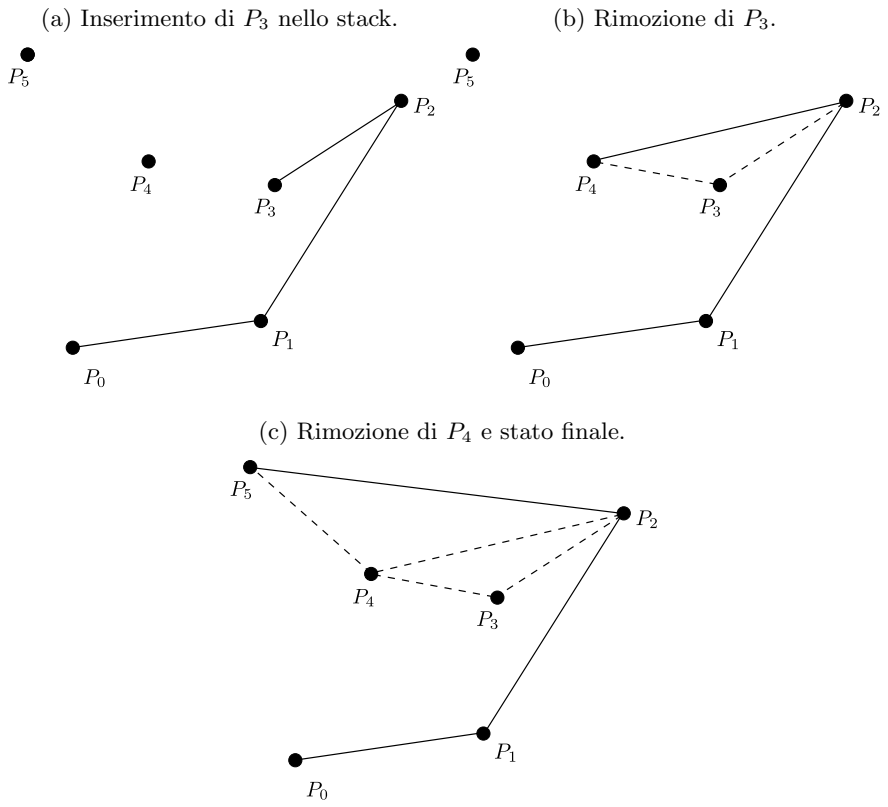
2. Ordiniamo i punti rimanenti P_i in base all'angolo formato con P_0 e l'asse delle ascisse. Si fa con $O(n \log n)$ operazioni

3. Inseriamo in uno stack vuoto S i punti P_0 e P_1. Per ogni punto non ancora analizzato P_{i+2}, consideriamo l'angolo formato da $P_{\text{end}-1}, P_{\text{end}}, P_{i+2}$, dove $P_{\text{end}-1}, P_{\text{end}}$ sono gli ultimi due elementi dello stack : se è antiorario, aggiungiamo P_{i+2} allo stack; viceversa rimuoviamo P_{end} dallo stack e ripetiamo questo passo fintanto che P_{i+2} non viene inserito nello stack (figure 8b e 8c).

4. Gli elementi rimasti nello stack formano l'inviluppo convesso dell'insieme di partenza (figura 8c).

Nel caso da noi considerato il numero di operazione scende a $O(n)$, in quanto i nostri punti sono già ordinati per ascissa, dato che erano le dimensioni degli iperrettangoli. In effetti si può ottenere di meglio: se si è attenti all'implementazione si può scendere a $O(n')$, dove n' è il numero dei punti con ascissa diversa: possiamo infatti a priori scartare i punti che hanno

Figura 8: Alcuni passi dello scan di Graham.



stessa ascissa, ma ordinata maggiore, in quanto non faranno parte del bordo "inferiore" dell'inviluppo convesso. Questo era ben visibile nella figura 4.

Riferimenti bibliografici

[1] L. C. W. Dixon e G. P. Szegő, cur. *Towards global optimisation. 2*. North-Holland Publishing Co., Amsterdam-New York, 1978, pp. vii+363. ISBN: 0-444-85171-2.

[2] D. E. Finkel. *MATLAB code for the DIRECT algorithm*. Retrieved June 26, 2016, from `http://www4.ncsu.edu/~ctk/Finkel_Direct/`.

[3] D. E. Finkel e C. T. Kelley. ≪Additive scaling and the DIRECT algorithm≫. In: *J. Global Optim.* 36.4 (2006), pp. 597–608. ISSN: 0925-5001. DOI: `10.1007/s10898-006-9029-9`. URL: `http://dx.doi.org/10.1007/s10898-006-9029-9`.

[4] Jorg Maximilian Xaver Gablonsky. *Modifications of the DIRECT algorithm*. Thesis (Ph.D.)–North Carolina State University. ProQuest LLC, Ann Arbor, MI, 2001, p. 167. ISBN: 978-0493-42568-9. URL: `http://gateway.proquest.com/openurl?url_ver=Z39.88-2004&rft_val_fmt=info:ofi/fmt:kev:mtx:dissertation&res_dat=xri:pqdiss&rft_dat=xri:pqdiss:3030042`.

[5] Ronald L. Graham. ≪An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set≫. In: *Inform. Process. Lett.* 7.1 (1972), pp. 132–133. ISSN: 0020-0190.

[6] D. R. Jones, C. D. Perttunen e B. E. Stuckman. ≪Lipschitzian optimization without the Lipschitz constant≫. In: *J. Optim. Theory Appl.* 79.1 (1993), pp. 157–181. ISSN: 0022-3239. DOI: `10.1007/BF00941892`. URL: `http://dx.doi.org/10.1007/BF00941892`.

[7] Luis Miguel Rios e Nikolaos V. Sahinidis. ≪Derivative-free optimization: a review of algorithms and comparison of software implementations≫. In: *J. Global Optim.* 56.3 (2013), pp. 1247–1293. ISSN: 0925-5001. DOI: `10.1007/s10898-012-9951-y`. URL: `http://dx.doi.org/10.1007/s10898-012-9951-y`.

[8] Bruno O. Shubert. ≪A sequential method seeking the global maximum of a function≫. In: *SIAM J. Numer. Anal.* 9 (1972), pp. 379–388. ISSN: 0036-1429.

[9] S. Surjanovic e D. Bingham. *Virtual Library of Simulation Experiments: Test Functions and Datasets*. Retrieved June 26, 2016, from `http://www.sfu.ca/~ssurjano`.