Dispense di I.U.M. Modellazione Geometrica

© Leila De Floriani, Paola Magillo

Cap.2: Guscio convesso e algoritmi

Rif.: cap. 3 libro di Preparata e Shamos.

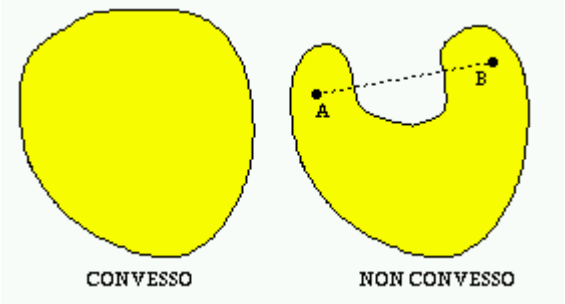
GUSCIO CONVESSO

Def. Guscio convesso (CH = Convex Hull)

Il **guscio convesso** di un insieme di `k` punti `P1,P2...P(k)` nel piano e' il piu' piccolo insieme convesso che contiene `P1,P2...P(k)`.

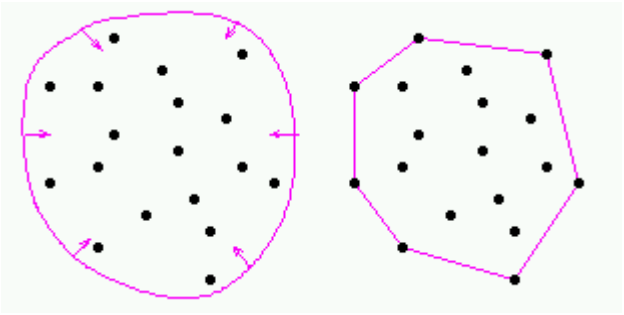
Def. Insieme convesso

Un insieme `I` e' **convesso** sse, per ogni coppia di punti `A, B` in `I`, il segmento di retta `AB` e' completamente interno a `I`.



Risultati sul guscio convesso

IDEA: avvolgere i punti `P1,P2...P(k)` in un involucro elastico e poi lasciare che si restringa.



Si ferma quando l'involucro e' teso attorno ai punti dati. Quello e' il contorno del guscio convesso.

- Il guscio convesso e' un insieme poligonale convesso
- Ha come vertici un sottoinsieme dei punti dati

Ora traduciamo in termini piu' FORMALI.

Def. Insieme poligonale convesso

Un insieme poligonale convesso e' l'intersezione di un insieme finito di semipiani chiusi (dove un semipiano e' la porzione di piano che giace da un lato di una retta).

Tale insieme e' **convesso** poiche' un semipiano e' convesso, e l'intersezione di insiemi convessi e' convessa.

Se e' limitato, un insieme poligonale e' una regione piana (poligonale convessa).

Teorema (Mc Mullan, Shephard, 1971)

Il guscio convesso di un insieme di punti e' un insieme poligonale convesso limitato. Viceversa, un insieme poligonale convesso limitato e' il guscio convesso di un insieme finito di punti.

Gli algoritmi determinano il **contorno** del guscio convesso. Chiameremo, per semplicita', **guscio convesso anche il contorno.**

Sia `L` e' un insieme finito di punti. Notazione:

- **`CH(L)` = il poligono di contorno del guscio convesso**
- **`Conv(L)` = la regione piana corrispondente**

Formulazione del problema

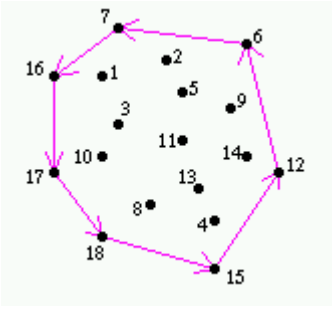
Esistono due versioni fondamentali del problema del guscio convesso.

Problema CH1 (Convex Hull)

Dato un insieme `S` di `N` punti nel piano, costruire il guscio convesso di `S` (vale a dire, una descrizione completa del poligono `CH(S)`).

Problema CH2 (Extreme Points)

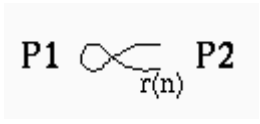
Dato un insieme `S` di `N` punti nel piano, identificare quei punti che sono i vertici di `Conv(S)`.



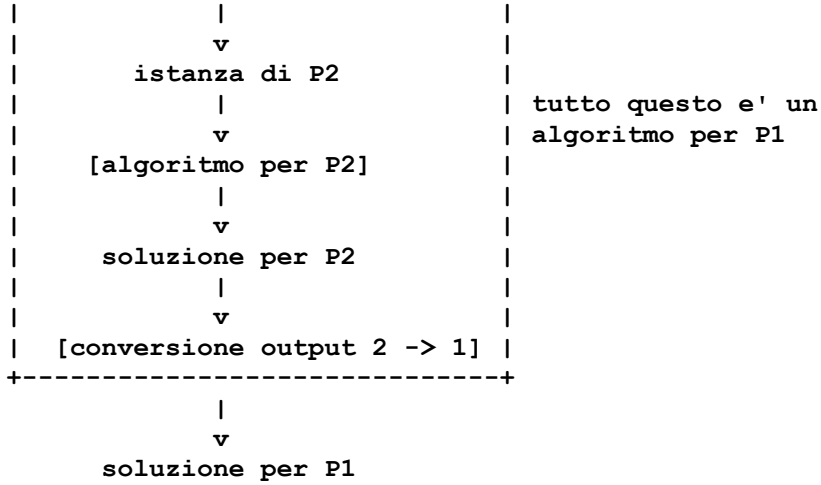
- **soluzione al problema CH1: lista `[15,12,6,7,16,17,18] = CH(S)`**
- **soluzione al problema CH2: insieme `{12,7,6,16,15,18,17}`**

Riduzione di problemi

Un problema P1 si riduce in tempo `r(n)` ad un problema P2 se e' possibile risolvere P1 utilizzando a scatola chiusa un algoritmo che risolve P2 + un costo di conversione dell'input e dell'output in O(`r(n)`).
Dove `n` e' la dimensione dell'input per P1.

Notazione: 


```
       istanza di P1
            |
            v
+-----------------------------+
|  [conversione input 1 -> 2]  |
```



Il costo delle conversioni e' in O($r(n)$).

Trasferimento di upper/lower bound

Se so che P1 si riduce in tempo $r(n)$ a P2 allora so che:

1. **P1 non e' piu' difficile di P2 + conversione. Se il costo di conversione non supera il costo di P2, allora P1 non e' piu' difficile di P2. In questo caso un upper bound alla complessita' di P2 e' anche un upper bound alla complessita' di P1.**

   Trasferimento di upper bound da P2 a P1: P2 = O($f(n)$) implica P2 = O($f(n)$), se $f(n)>=r(n)$.

2. **P2 + conversione non e' piu' facile di P1. Se il costo di conversione non supera il costo di P1, allora P2 non e' piu' facile di P1. In questo caso un lower bound alla complessita' di P1 e' anche un lower bound alla complessita' di P2.**

   Trasferimento di lower bound da P1 a P2: P1 = Omega($g(n)$) implica P2 = Omega($(g(n)$), se g(n)>=r(n).

In genere considereremo casi in cui il costo di conversione e' costante o lineare.

Relazione fra i due problemi di guscio convesso

Il problema CH2 (extreme points) e' riducibile al problema CH1 (convex hull) in tempo lineare.
Infatti una soluzione valida per CH2 e' ottenuta prendendo l'output di CH1 (lista ordinata) e trasformandolo in una lista non ordinata (un insieme).

Teorema (vale solo in 2D)

Il problema dell'ordinamento di `N` numeri reali (sorting) e' riducibile al problema CH1 (convex hull) in tempo lineare.

Vale a dire che posso ordinare `N` numeri calcolando il guscio convesso di `N` punti opportunamente scelti.

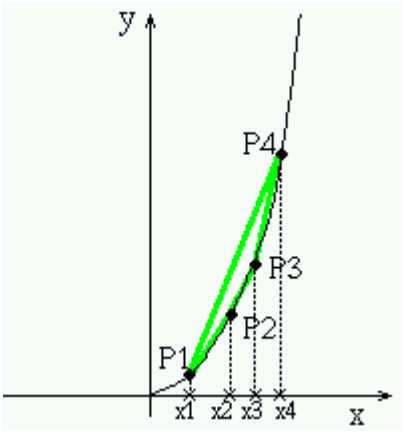
Quindi, trovare il guscio convesso di `N` punti richiede un tempo in Omega(`N log N`).

Dimostrazione

Dimostriamo la riducibilita'. Il secondo risultato segue dalla proposizione sul trasferimento del lower bound (Rif. cap.1 Preparata e Shamos).

Dati `N` numeri reali positivi `x1,x2....x(N)` dobbiamo dimostrare che un algoritmo per il calcolo del guscio convesso puo' essere usato per ordinarli con un "overhead" lineare.

CONVERSIONE DELL'INPUT: In corrispondenza di ogni numero `x(i)` costruiamo un punto nel piano `P(i) = (x(i), x(i)^2).`



Tali punti giacciono sulla parabola `y = x^2`. Il guscio convesso di questo insieme, se letto a partire dal punto di ascissa minima, consiste in una lista di punti ordinati per ascissa crescente.

CONVERSIONE DELL'OUTPUT: Un passo attraverso tale lista ci permette di leggere le ascisse `x(i)` in ordine.

Osservazione

Il risultato espresso dal teorema vale in qualsiasi dimensione maggiore di 1.

In dimensione 1, il guscio convesso (= il piu' piccolo intervallo che contiene `N` punti dati sulla retta) puo' essere calcolato in tempo lineare: basta trovare il min e il max dell'insieme.

Obiettivo

Risolvere il problema CH1 (calcolo del guscio convesso).

Alcuni risultati (non troppo soddisfacenti)

La definizione di guscio convesso non e' costruttiva (non ci da direttamente un algoritmo). Proviamo a delineare un primo algoritmo semplice (e inefficiente!).

Prima idea di un algoritmo per il guscio convesso

1. **risolvo il problema CH2 (identifico i vertici del guscio convesso)**
2. **li ordino per formare il guscio convesso**

Passo 1 (soluzione di CH2)

Un punto `P` di `S` non e' vertice di `Conv(S)` se appartiene a qualche triangolo i cui vertici sono punti di `S` distinti da `P` (non lo dimostriamo).

Algoritmo che ne deriva:

- **Per ogni punto `P` di `S`, esamino i triangoli formati da terne di punti di `S - {P}`.**
  Se `P` e' interno, o appartiene ad un lato, di uno di questi triangoli, allora `P` non e' vertice di `Conv(S)`.
- **I punti che restano sono i vertici di `Conv(S)`.**

`N` punti determinano $\binom{N}{3}$ triangoli. Quindi sono necessarie O(N^3) operazioni per stabilire se `P` e' un vertice di `Conv(S)`.
La complessita' totale del passo 1 e' O(N^4).

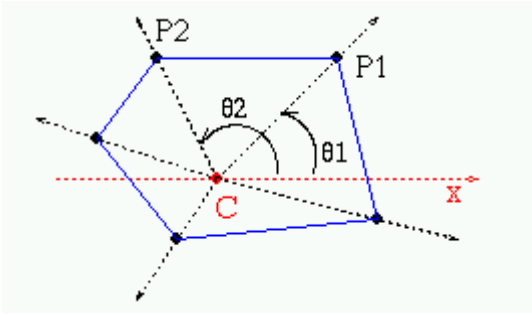
Passo 2 (ordinamento dei vertici)

L'algoritmo e' basato sui due risultati seguenti:

1. **Un raggio uscente da un punto interno di un insieme convesso `F` chiuso**

interseca il contorno di `F` in esattamente un punto (segue dal teorema di Jordan).

2. **Due vertici consecutivi di un poligono convesso sono consecutivi in ordine angolare rispetto ad un qualunque punto interno.**



P(i) = (x(i),y(i)) diventa P(i) = (r(i),theta(i)) in coordinate angolari

Algoritmo:

- **Scelgo un punto `C` interno al guscio convesso. E' sufficiente considerare tre vertici qualsiasi (in quanto vertici del guscio convesso, non sono allineati) e calcolare il baricentro del triangolo da essi formato. Tale punto e' sicuramente interno al guscio convesso. Costo: O(1).**
- **Calcolo ordinamento angolare rispetto a `C`.**

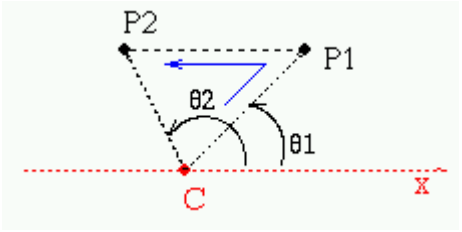
Calcolo dell'ordinamento radiale

Una possibilita:

- **Calcolo le coordinate polari `(r, theta)` di ciascun punto.**
- **Ordino i punti in base a `theta`. Il costo del passo di ordinamento e' O(`K log K`) dove `K` e' il numero di punti vertici del guscio convesso `Conv(S)`. Nel caso pessimo `K = N` e quindi il costo e' O(`N log N`).**

  NOTA: Il test chiave per effettuare l'ordinamento e' confrontare, fra due punti, quale dei due ha coordinata `theta` maggiore.
  Per fare questo confronto, NON e' necessario aver calcolato la coordinata `theta`.

  Siano `P1, P2` due punti e `theta1, theta2` le loro coordinate `theta`.
  `theta1` e' minore di `theta2` sse `[C,P1,P2]` e' una sequenza antioraria.



Equivalente a dire che la terna `C,P1,P2` definisce una svolta a sinistra, o

che l'area con segno del triangolo `C,P1,P2` e' maggiore di zero.

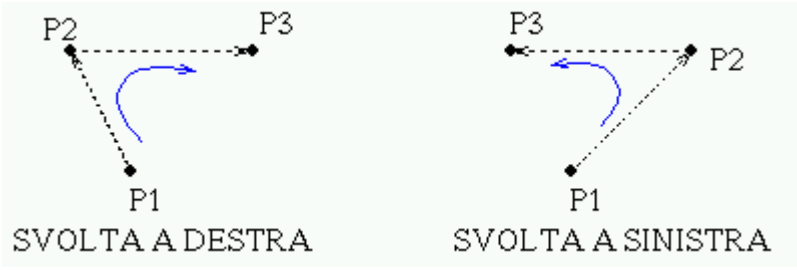
Questo test si riduce al calcolo di un determinante 3x3, quindi solo operazioni aritmetiche. Invece il calcolo `theta` richiederebbe l'uso di funzioni trigonometriche, che sono piu' costose.

Conclusione

Il problema del CONVEX HULL puo' essere risolto in tempo O(`N^4`) effettuando esclusivamente operazioni aritmetiche (+,-,*,/) e confronti fra numeri reali.

Problema geometrico fondamentale

Data una sequenza `[P1,P2,P3]` di tre punti nel piano, stabilire se definisce una svolta a destra o una svolta a sinistra (o se i tre punti sono allineati).



Problema equivalente

Data una retta orientata `r` definita da una coppia di punti `P2,P3`, e un punto `P1`, stabilire se `P1` giace alla destra o alla sinistra della retta orientata `r` (o se appartiene alla retta).

ALGORITMI per la determinazione del guscio convesso

- **Algoritmo di Graham (ordinamento angolare + scansione). Complessita' Theta(`N log N`).**
- **Algoritmo di Jarvis (basato su definizione di guscio convesso come intersezione di semipiani). Complessita' O(`K^2`) dove `K` e' il numero di vertici del guscio convesso (`K = N` nel caso pessimo.**
- **Algoritmo QUICKHULL ("estensione" di QUICKSORT). Complessita' O(`N^2`).**
- **Algoritmo divide-et-impera ("estensione" di MERGESORT). Complessita' Theta(`N log N`).**

- **Algoritmi dinamici ("on-line"): non necessitano di conoscere tutti i punti all'inizio.**

Algoritmo di GRAHAM (1972)

Idea

Utilizzare un ordinamento preventivo dei punti di `S` per evitare di dover esaminare tutti gli Omega(`N^3`) triangoli ad ogni passo.

Schema dell'algoritmo

Dato un insieme `S` formato da `N` punti:

- **Determinare un punto `O` interno a `S` da scegliersi come origine del sistema di coordinate (e' una semplice traslazione per gli `N` punti di `S`).**
- **Ordinare gli `N` punti di `S` per coordinate polari crescenti intorno all'origine `O`. Se due punti hanno la stessa coordinata `theta`, vengono discriminati per la distanza rispetto all'origine (nota: il confronto non richiede il calcolo della distanza dall'origine, o del quadrato di questa, in quanto i punti sono allineati).**
  Costo: O(`N log N`).
- **I punti ordinati vengono organizzati in una lista circolare doppia.**
- **L'algoritmo effettua un'unica scansione della lista degli `N` punti ordinati, durante la quale i punti interni al guscio convesso sono eliminati.**

Determinazione del punto interno

METODO 1:
Come punto interno, prendo il baricentro dell'insieme dei vertici del guscio convesso.

Il baricentro di un insieme di `N` punti `Q1,Q2...Q(N)` e' il punto `Q = (Q1+Q2+...+Q(N)) / N`.

E' noto che il baricentro di un insieme convesso e' interno al guscio convesso dell'insieme.
Per il calcolo del baricentro sono necessarie O(N) operazioni aritmetiche.

METODO 2 (Graham, 1972):

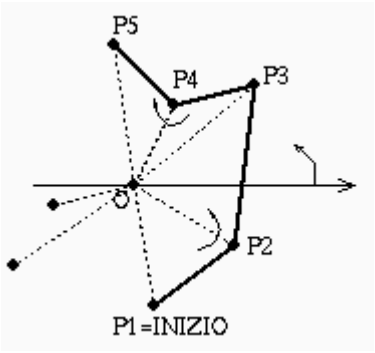
Basato sulla proprieta' che il baricentro di tre punti non allineati di un insieme convesso e' interno all'insieme.

- **Considero due punti arbitrariamente scelti ed esamino gli altri `N-2` fino a trovarne uno non allineato.**
- **A questo punto ho un triangolo, ne calcolo il baricentro.**

La complessita' e' O(`N`) nel caso peggiore, ma sempre costante in pratica.

La scansione di Graham

1. **Come punto iniziale (detto INIZIO) si prende il punto di ordinata minima situato piu' a destra. In questo modo, si parte da un punto che appartiene sicuramente al guscio convesso.**
2. **Si esaminano iterativamente terne di punti `P(i),P(i+1),P(i+2)` consecutivi in senso antiorario e si controlla l'orientamento della svolta.**
   Se la terna `P(i) P(i+1) P(i+2)` definisce una svolta a destra (angolo reflex) oppure i punti `P(i) P(i+1) P(i+2)` sono allineati, `P(i+1)` e' interno al guscio convesso in quanto e' interno al triangolo `o P(i) P(i+2).`



3. **In base all'orientamento della svolta, si decide come procedere nella scansione:**
   - **se `P(i) P(i+1) P(i+2)` definiscono una svolta a destra, o sono allineati, si elimina il vertice `P(i+1)` e si passa a controllare la terna `P(i-1),P(i),P(i+2).`**
   - **se `P(i) P(i+1) P(i+2)` definiscono svolta a sinistra, si avanza nella scansione e si passa a controllare la terna `P(i+1),P(i+2),P(i+3).`**
4. **La scansione termina quando, dopo aver esaminato i punti di `S` in ordine, si ritorna al punto INIZIO.**

Nell'esempio le terne esaminate sono:

- **`P1,P2,P3`: si avanza**
- **`P2,P3,P4`: si avanza**
- **`P3,P4,P5`: si elimina `P4`**
- **`P2,P3,P5`: si avanza**
- **ecc.**

La scansione (detta scansione di Graham) ha complessita' O(N) poiche':

- **Il test sull'angolo (passo 2) e' effettuato in tempo costante (calcolo di un determinante 3x3).**
- **Ad ogni test o si avanza o si elimina un punto. Poiche' S ha N punti, la scansione non puo' avanzare piu' di N volte, ed inoltre non e' possibile cancellare piu' di N punti.**

Complessita' dell'algoritmo di Graham

L'algoritmo di Graham calcola il guscio convesso in tempo O(N log N) (dovuto al passo di ordinamento) utilizzando O(N) posizioni di memoria. Poiche' il lower bound per il problema del guscio convesso e' Omega(N log N), l'algoritmo di Graham e' asintoticamente ottimale nel caso pessimo.

Osservazione

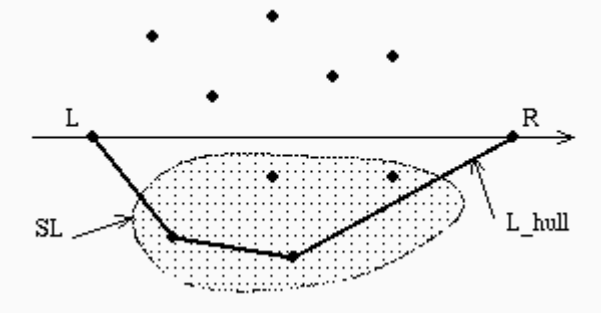
La scansione di Graham puo' essere applicata a punti ordinati per coordinata x crescente (invece che per coordinata theta crescente attorno ad un'origine o).

Dati N punti nel piano, basta determinare l'estremo sinistro L e l'estremo destro S, e costruire la retta r passante per L ed S ed orientata da L a S.

Questa retta partiziona l'insieme S dei punti in due sottoinsiemi SL ed SU:

- **SL e' l'insieme dei punti di S che stanno al di sotto di r (L=lower);**
- **SU e' l'insieme dei punti di S che stanno al di sopra di r (U=upper).**

A partire da SL costruiro' L_hull (lower hull, catena monotona rispetto all'asse x).
A partire da SU costruiro' U_hull (upper hull, catena monotona rispetto all'asse x).
L'unione di L_hull e U_hull mi dara' il guscio convesso di S.



La scansione di Graham applicata ad SL unito {L,R} restituisce il lower hull, ed applicata ad SU unito {L,R} restiruisce l'upper hull.

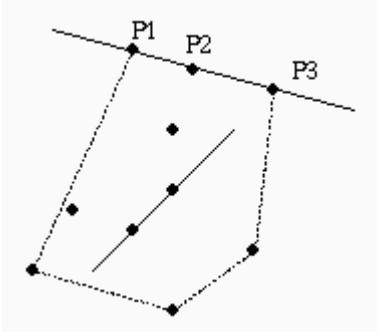
Vantaggi e svantaggi dell'algoritmo di Graham

- **Vantaggi:**
  - **semplice**
  - **ottimale (asintoticamente nel caso pessimo)**
- **Svantaggi:**
  - **non generalizzabile a 3D**
  - **non parallelizzabile (in modo naturale)**
  - **non on-line (richiede che tutti i punti siano disponibili prima dell'elaborazione)**

Algoritmo di Jarvis (cenno)

Basato sulla determinazione degli spigoli del guscio convesso `CH(S)` e sul seguente risultato:

Un segmento `l` definito da due punti di `S` e' un lato del guscio convesso sse tutti gli altri punti di `S` appartengono a `l` o giacciono dalla stessa parte della retta passante per `l`.



Algoritmo piu' semplice

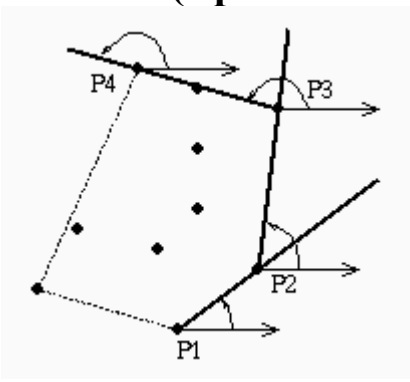
Esistono $\binom{N}{2}$ **= O(`N^2`) rette determinate dagli `N` punti di `S`. Per ogni retta si esaminano gli `N-2` punti di `S` che non definiscono la retta in questione. Complessita': O(`N^3`).**

Algoritmo di Jarvis

1. **Si parte dal punto `P1` di ordinata minima che sta piu' a destra (come per il metodo di Graham).**
2. **Come vertice consecutivo `P2` si prende il punto di `S` che ha la coordinata `theta` minima in un sistema di coordinate polari centrato in `P1` (l'angolo `theta` e' misurato rispetto all'asse `x` positivo); se due o**

piu' punti hanno la stessa coordinata `theta`, si prende il punto fra questi piu' distante da `P1`.

3. **Il passo 2 e' ripetuto finche' non si arriva al punto di ordinata massima (e piu' a sinistra). Nell'esempio il punto `P4`.**



4. **La catena di spigoli sul guscio convesso dal punto di ordinata massima al punto iniziale si costruisce agendo in modo simmetrico rispetto al passo 2.**

Complessita' temporale

Determinazione del punto successivo (passo 2): O(`N`) operazioni (solo operazioni aritmetiche e confronti).

Se `K` e' il numero di vertici del guscio convesso allora si hanno O(`NK`) operazioni, poiche' vengono effettuate O(`N`) operazioni per ogni punto del guscio convesso.

La complessita' dell'intero algoritmo O(`N^2`) nel caso pessimo (`H` = O(`N`)).

Osservazione

Il metodo e' generalizzabile a 3 o piu' dimensioni (metodo "gift-wrapping").

Algoritmo QUICKHULL

Tecnica simile al QUICKSORT.

Dato un insieme `S` di `N` punti:

Passo iniziale

Si considerano i punti di `S` di ascissa minima e massima: `L, R`.

L'insieme `S` e' "suddiviso" in due sottoinsiemi `S1` ed `S2` dalla retta orientata `r` da `L` a `R`:

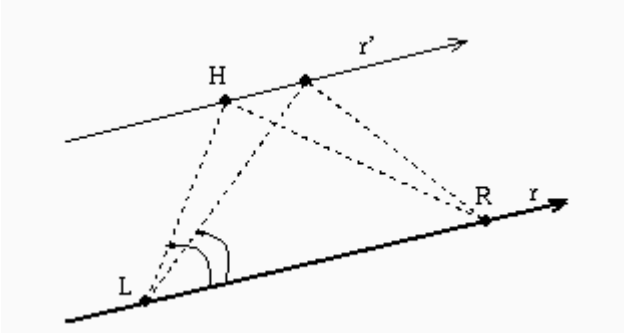
- **`S1` = punti di `S` che si trovano al di sopra della retta orientata `r`**
- **`S2` = punti di `S` che che si trovano al di sotto della retta orientata `r`**

Nota: i punti appartenenti ad `r` (e quindi situati fra `L` ed `R`) non possono appartenere al guscio convesso e quindi vengono scartati.

Passi successivi

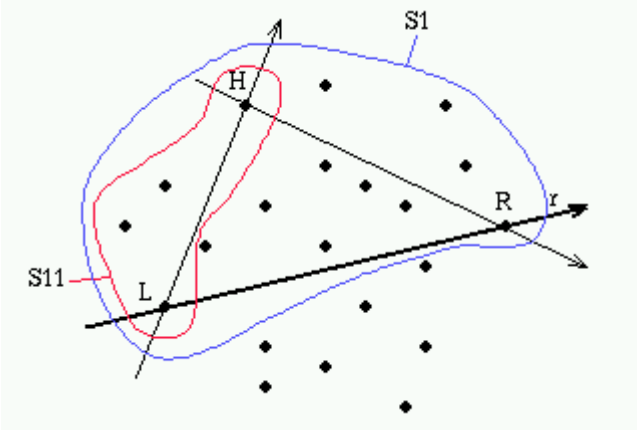
Consideriamo `S1` (o, equivalentemente, `S2`).

- **Consideriamo il punto `H` di `S1` tale che il triangolo `LHR` ha l'area massima fra tutti i triangoli `LKR` con base `LR` e vertice opposto `K` in `S1`. Basta prendere `H` come il punto di massima distanza da `LR`.**
  Se il triangolo di area massima non e' unico, prendiamo quello che massimizza l'angolo `HLR` (vale a dire il punto `H` piu' a sinistra fra i potenziali candidati).



Il punto `H` appartiene al guscio convesso.
Dimostrazione: Se tracciamo la retta `r'` parallela ad `LR` per `H`, non ci sono punti di `S` al di sopra di `r'`. Ci possono essere altri punti di `S` su `r'` oltre ad `H`, ma abbiamo scelto il punto piu' a sinistra. Percio', `H` non puo' essere espresso come combinazione convessa di altri due punti di `S`.

- **Si considerano le due rette orientate `r1` ed `r2`: `r1` = retta orientata da `L` ad `H`, `r2` = retta orientata da `H` ad `R`.**
  Non esistono punti di `S1` che giacciono simultaneamente alla sinistra di `r1` ed `r2`.
  I punti di `S1` che giacciono alla destra di entrambe sono interni al triangolo `LRH` e quindi possono essere eliminati dalla considerazione.
  I punti alla sinistra di `r1` (o appartenenti ad `r1`) e alla destra di `r2` costituiscono l'insieme `S11`
  I punti alla sinistra di `r2` (o appartenenti ad `r21`) e alla destra di `r1` costituiscono l'insieme `S12`



- **Il processo e' attivato ricorsivamente su `S11` e su `S12` separatamente.**

Pseudocodice del passo generico

```
Algorithm QUICKHULL (S,L,R)
begin
  if (S={L,R}) then return ([L,R]);
  /* condizione di terminazione del processo ricorsivo */
  else
  /* S contiene piu' di due punti */
  begin
    H := FURTHEST(S,L,R);
    /* H e' il punto piu' distante dalla retta LR
       secondo quanto spiegato prima */
    S1 := punti di S giacenti a sinistra della retta LH;
    S2 := punti di S giacenti a sinistra della retta HR;
    return ( CONCATENATE (QUICKHULL(S1,L,H), QUICKHULL(S2,H,R) );
    /* CONCATENATE concatena le due liste eliminando il punto H
       dalla seconda lista */
   else /* else */
end /* QUICKHULL */
```

Complessita'

- **Passo iniziale: la determinazione di `S1` ed `S2` richiede O(`N`) operazioni.**
- **Passi successivi: la determinazione del punto `H`, dei punti interni al triangolo (punti da scartare) e dei due sottoinsiemi `Si1` ed `Si2` costa O(`Ni`), dove `Ni` e' la cardinalita' dell'insieme corrente `Si`.**

Se ad ogni attivazione ricorsiva gli insiemi `S1` ed `S2` hanno circa la stessa cardinalita', allora la complessita' e' O(`N log N`).
Nel caso peggiore la complessita' e' O(`N^2`), come nel caso di QUICKSORT.

Vantaggi e svantaggi dell'algoritmo QUICKHULL

- **Vantaggio: facilmente parallelizzabile**
- **Svantaggio: Difficolta' nel controllo delle dimensione dei due sottoproblemi in cui e' suddiviso. Puo' portare a complessita'**

quadratica.

Algoritmi di tipo divide-et-impera

Obiettivo:
Sviluppare un algoritmo con complessita O(`N log N`) e facilmente parallelizzabile (sommare i vantaggi di Graham e QUICKHULL).
A questo scopo usiamo una tecnica di divide-et-impera.

Idea

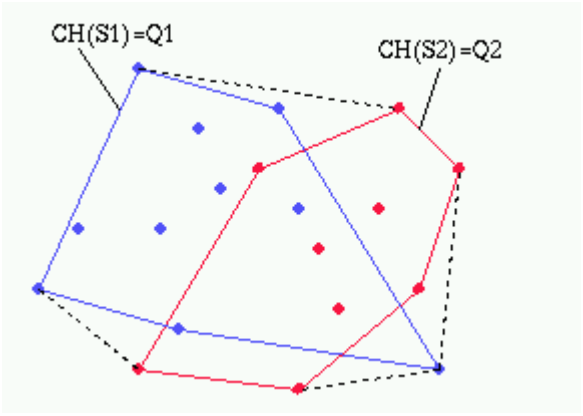
- **Si partiziona l'insieme `S` in due sottoinsiemi `S1` ed `S2`, ciascuno contenente la meta' dei punti.**
- **Si calcolano, separatamente e ricorsivamente, `CH(S1)` e `CH(S2)`.**
- **Si costruisce `CH(S) = CH(S1 unito S2)` da `CH(S1)` e `CH(S2)`.**

  `CH(S1 unito S2) = CH ( CH(S1) unito CH(S2) )`

`CH(S1)` e `CH(S2)` sono poligoni convessi. Percio' il problema chiave e' il calcolo del guscio convesso dell'unione di due poligoni convessi.

Guscio convesso dell'unione di due poligoni convessi

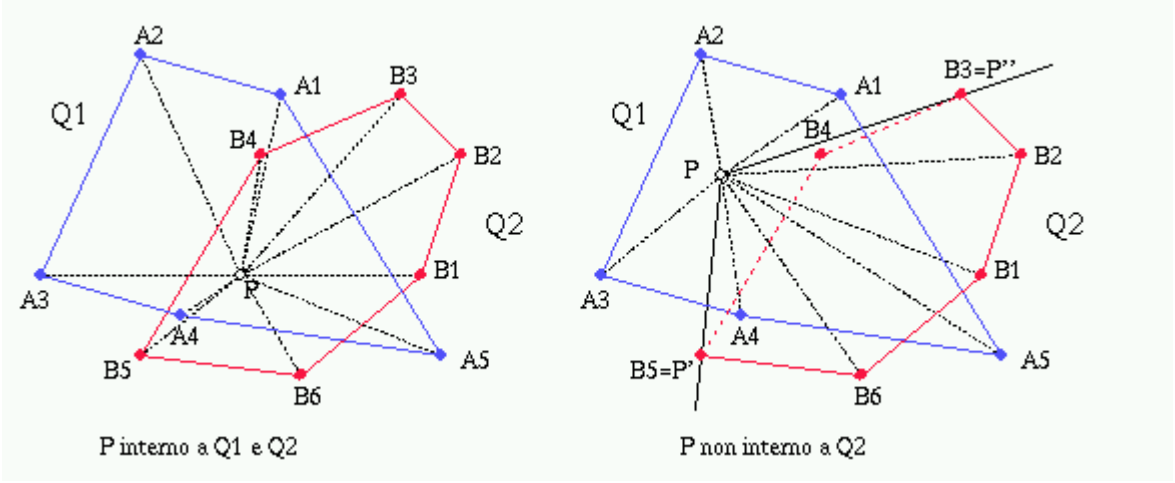
Dati due poligoni convessi `Q1` e `Q2`, determinare il guscio convesso dell'unione di `Q1` e `Q2`.



Algoritmo "unione di poligoni convessi" (Shamos, 1978)

1. **Si determina un punto `P` interno a `Q1` (baricentro di tre vertici di `Q1`)**
2. **Si stabilisce se `P` e' interno a `Q2` oppure no (algoritmo "point-in-polygon")**
3. **Se `P` e' interno a `Q2`, allora i vertici di `Q1` e `Q2` possono essere ordinati attorno a `P` e le due liste risultanti possono essere immerse, ottenendo una lista ordinata `Q` dei vertici di `Q1` e `Q2`.**

4. **Se `P` non e' interno a `Q2`, allora il poligono `Q2` giace in un settore con vertice `P` i cui raggi infiniti passano per due vertici di `P'` e `P''` di `Q2`. I due vertici `P'` e `P''` suddividono `Q2` in due catene monotone rispetto alla coordinata polare `theta`.**
   La catena "piu' vicina" a `P` puo' essere eliminata in quanto non appartiene al guscio convesso.
   La catena "esterna" risulta ordinata angolarmente rispetto a `P` ed immersa con la lista ordinata dei vertici di `Q1`. Ottengo cosi' una lista ordinata `Q` dei vertici di `Q1` piu' i vertici della catena esterna di `Q2`.
5. **Si applica la scansione di Graham alla lista dei vertici `Q` ottenuta ai passi 3 o 4. Questo restituisce il guscio convesso di `Q1` unito `Q2`.**



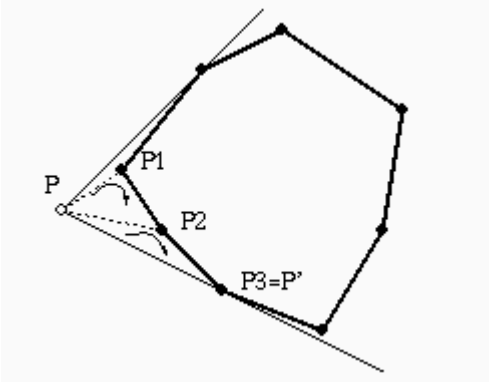

```
Q1 = [A1,A2,...,A5]
Q2 = [B1,B2,...,B6]
```

Risultato `= [B1,B2,B3,A1,B4,A2,A3,A4,B5,B6,A5]`

Determinazione dei due punti di tangenza

Calcolo del punto di tangenza inferiore `P'`:

- **Si Prende `P1` = il punto piu' vicino a `P`.**
- **A partire da `P1` si scorre il contorno di `Q2` in senso antiorario fintanto che la terna `[P,P(i),P(i+1)]` svolta a destra.**
- **Il punto centrale della prima terna che svolta a sinistra e' `P'`.**



Il punto di tangenza superiore `P''` si determina a partire da `P1` operando in maniera simmetrica.

Complessita' temporale

Supponiamo che $Q1$ abbia m vertici e che $Q2$ abbia n vertici.

- **Passo 1 (determinazione del punto interno P): costa O(1).**
- **Passo 2 (test su P e poligono $Q2$): $O(n)$ (= complessita' dell'algoritmo che determina dove si trova un punto rispetto ad un poligono convesso).**
- **Passo 3 (se P interno a $Q2$): $O(n+m)$ (per l'immersione delle due liste ordinate $Q1$ e $Q2$).**
- **Passo 4 (se P esterno a $Q2$): $O(n+m)$ (la determinazione dei due punti P' e P'' costa $O(n)$, l'immersione delle due liste ordinate costa $O(n+m)$).**
- **Passo 5 (scansione di Graham): $O(n+m)$.**

Complessita' dell'algoritmo divide-et-impera

La complessita' totale dell passo di fusione nell'algoritmo divide-et-impera per il calcolo del guscio convesso e' $O(n+m)$.

Sia $T(N)$ il tempo richiesto per determinare il guscio convesso di un insieme S di N punti.

$$T(N) < 2T(N/2) + O(N)$$

Ne segue che $T(N) = O(N \log N)$.