UNIUNEA EUROPEANĂ

GUVERNUL ROMÂNIEI

Instrumente Structurale
2007-2013

Platformă de e-learning și curriculă e-content pentru învățământul superior tehnic

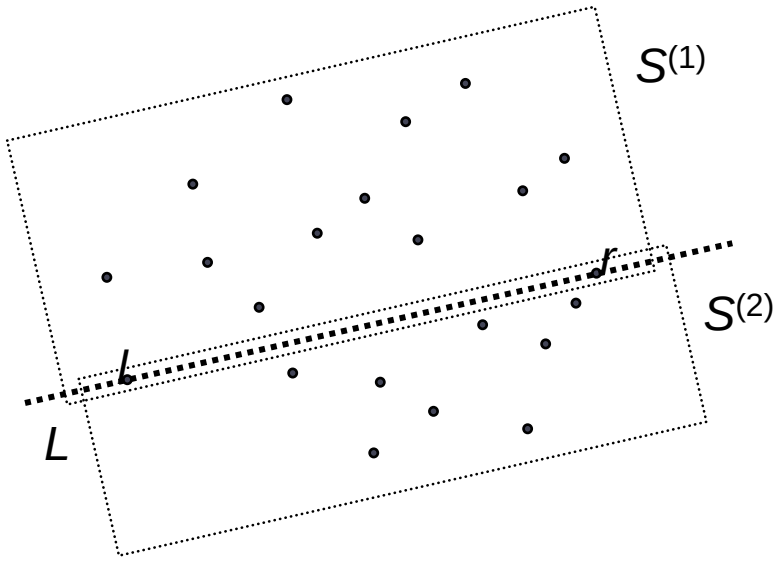
Geometrie computationala

4. Acoperiri convexe in plan: Algoritmul Quick Hull. Divide et Impera. Infasuratori convexe in 2D si 3D

Algoritmul Quick Hull (1)

Partitia initiala

- S este partitionata de o dreapta L determinata de punctele $l, r \in S$ cu cea mai mica si cea mai mare abscisa (garantat distincte)
- $S^{(1)} \subseteq S$ este submultimea lui S deasupra L.
- $S^{(2)} \subseteq S$ este submultimea lui S sub L.
- $\{S^{(1)}, S^{(2)}\}$ nu este o partitie stricta a lui S, $S^{(1)} \cap S^{(2)} \supseteq \{l,r\}$.
- Urmeaza sa se construiasca $CH(S^{(1)})$ si $CH(S^{(2)})$, si apoi concatenate in $CH(S)$.
- Procesul este identic pentru $S^{(1)}$ si $S^{(2)}$, vom detalia numai $S^{(1)}$.



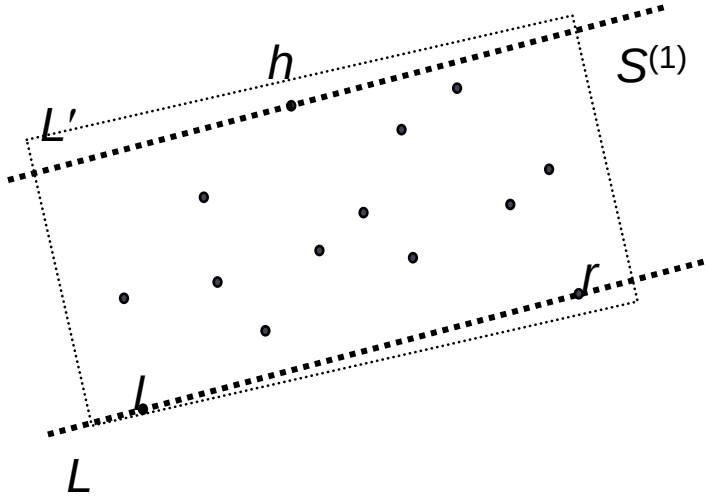
Algoritmul Quick Hull (2)

Alegerea extremului

- Se cauta punctul $h \in S^{(1)}$ astfel incat
  1. triunghiul *hlr* are cea mai mare arie posibila
  2. daca *h* nu este unic determinat, se alege cel pentru care unghiul *hlr* este maxim.
- Aceste conditii implica $h \in H(S)$. De ce?
- Se construieste prin *h* linia L' paralela cu L
- Conditia (1) impune ca nu vor exista puncte din $S^{(1)}$ (sau S) deasupra L',
- Pot fi mai multe puncte pe L', dar *h* va fi cel mai din stanga, conform (2)

$\Rightarrow h \in H(S)$.

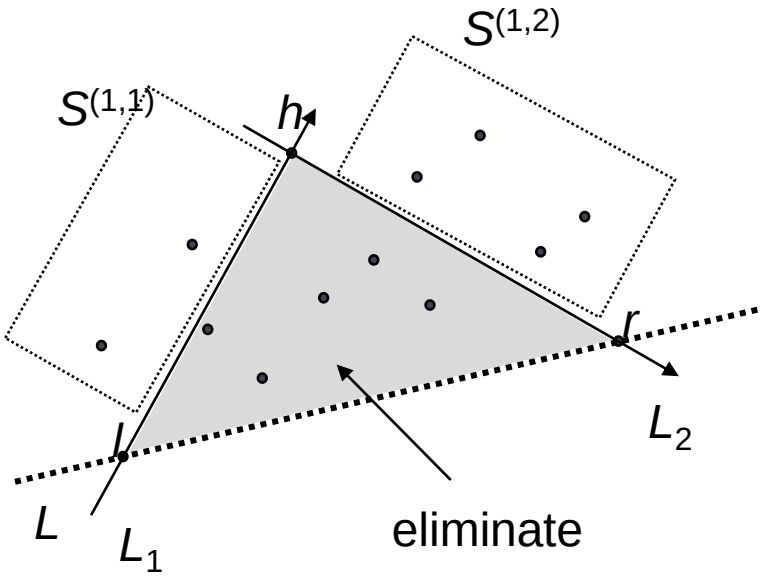
- *h* poate fi gasit in $O(N)$ verificand fiecare punct din $S^{(1)}$.



Algoritmul Quick Hull (3)

Partitionarea

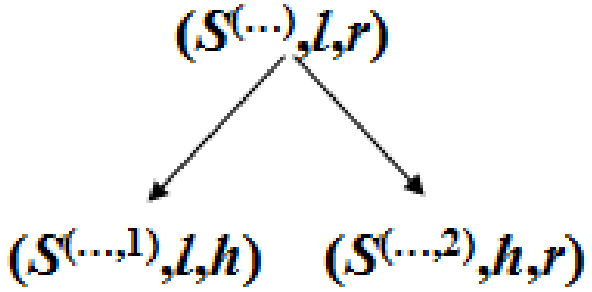
- Se construiesc doua linii *orientate*, L_1 dinspre l catre h, si L_2 dinspre h catre r.
- Fiecare punct din $S^{(1)}$ se poate clasifica relativ la L_1 si L_2
- Nici un punct din $S^{(1)}$ nu poate fi situat in acelasi timp la stanga L_1 si L_2.
- Punctele la dreapta L_1 si L_2 nu apartin *CH(S)* fiind in interiorul triunghiului *hlr* si sunt eliminate
- Punctele la stanga L_1 formeaza $S^{(1,1)}$.
- Punctele la stanga L_2 formeaza $S^{(1,2)}$.

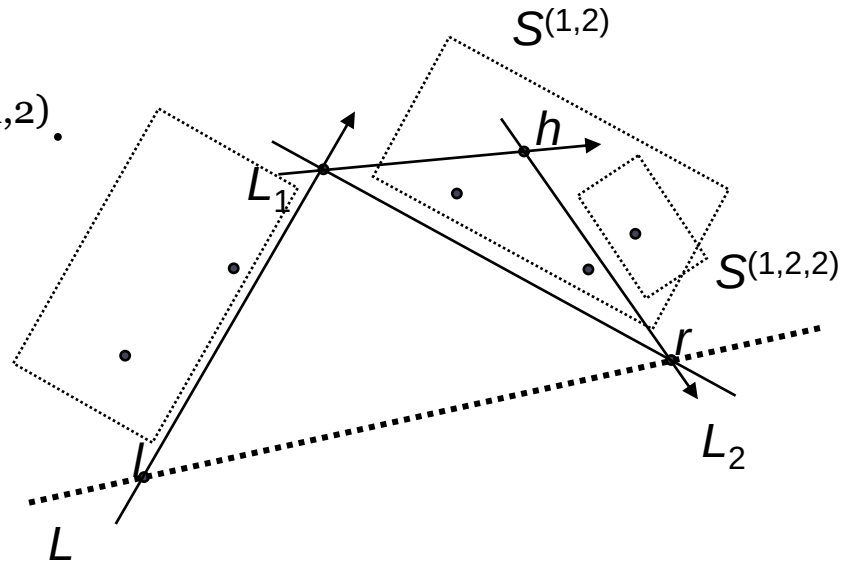


Algoritmul Quick Hull (4)

Pasul recursiv

- Procesul se reia pentru $S^{(1,1)}$ si $S^{(1,2)}$.





- Recurenta continua pana cand $S^{(\dots)}$ are 0 puncte (toate punctele interne au fost eliminate), adica lr este o muchie din $CH(S)$.

Complexitate

- Gasirea extremelor initiale se face in timp O(n).
- La fiecare impartire, este nevoie de maxim *n* pasi pentru a determina *h*, dar timpul total al apelului recursiv depinde de marimile multimilor $S^{(1)}$ si $S^{(2)}$.

Cazul favorabil Partitia este balansata

- $T(n) = 2T(n/2) + O(n)$ → $T(n)=O(n \log n)$.
- $O(n \log n)$ se intampla pentru puncte distribuite aleator.

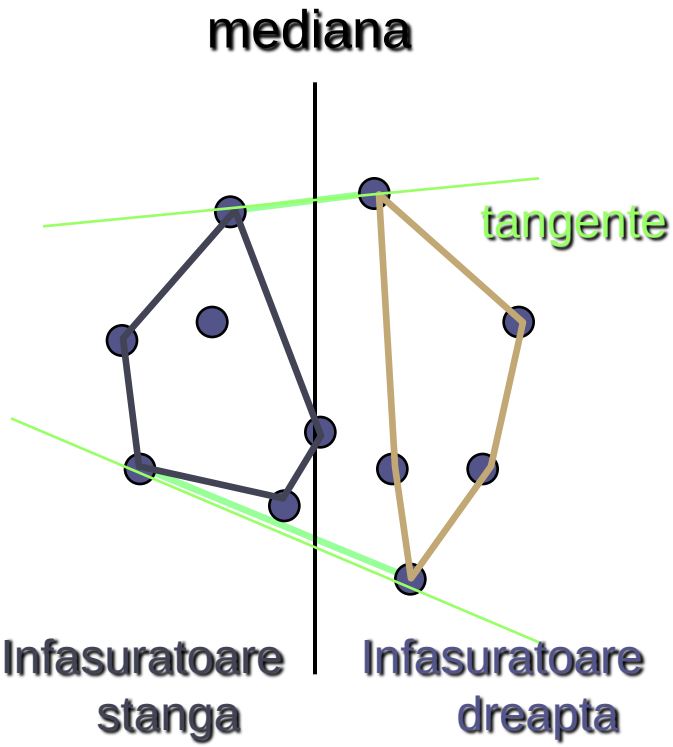
Cazul defavorabil Partitie disproportionata

- $T(n) = T(n - 1) + O(n) = T(n - 1) + cn$ → $T(n) = O(n^2)$.

Divide et Impera

Algoritm:

- Se gaseste un punct ce are coordonata x mediana (in timpul $O(n)$)
- Se partitioneaza multimea de puncte in 2 jumatati
- Se calculeaza infasuratoarea convexa a fiecarei jumatati (executie recursiva)
- Se combina cele doua infasuratori convexe gasind tangentele lor superioare si inferioare in $O(n)$

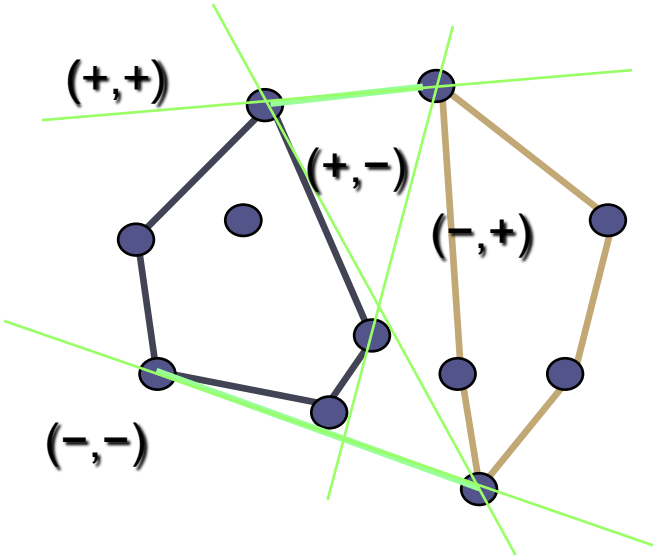


Complexitate de timp: $O(n \log n)$

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

Tangente (1)

- Oricare doua poligoane convexe disjuncte au 4 tangente care le impart in doua categorii: in intregime la stanga (+) si in intregime la dreapta (-), raportat la dreapta tangentei.

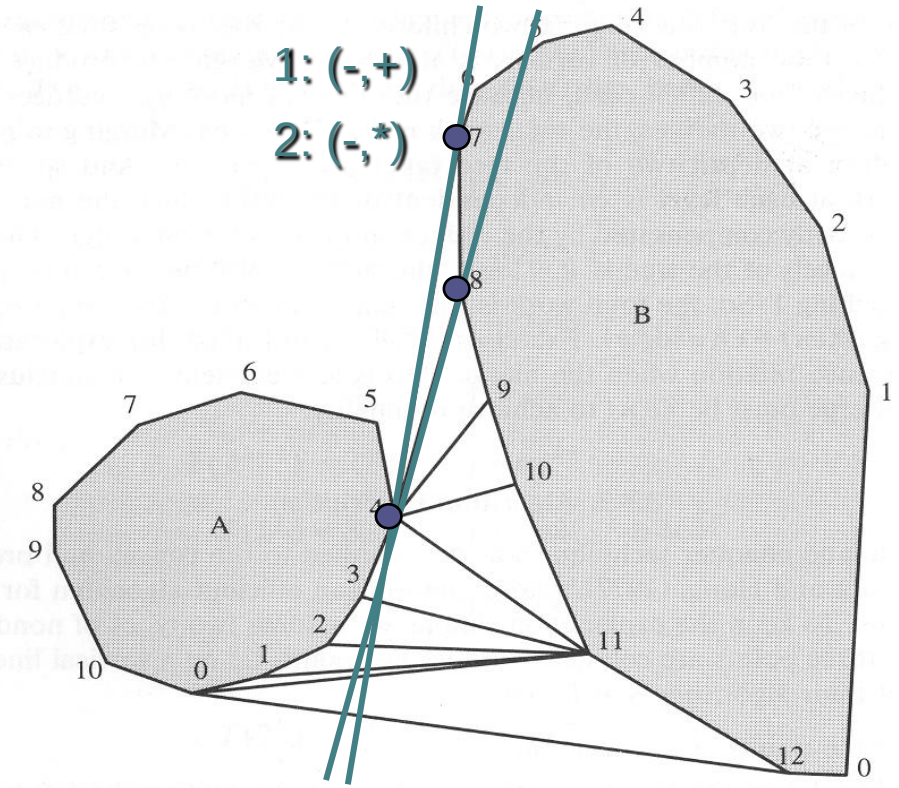


Tangente (2)

Tangenta inferioara:

Se uneste puntul cel mai din dreapta al poligonului din stanga cu punctul cel mai din stanga al poligonului din dreapta si se parcurg muchiile pana cand se atinge tangenta inferioara

Complexitate: $O(n)$.



Finding the lower tangent: from (4, 7) to (0, 12).

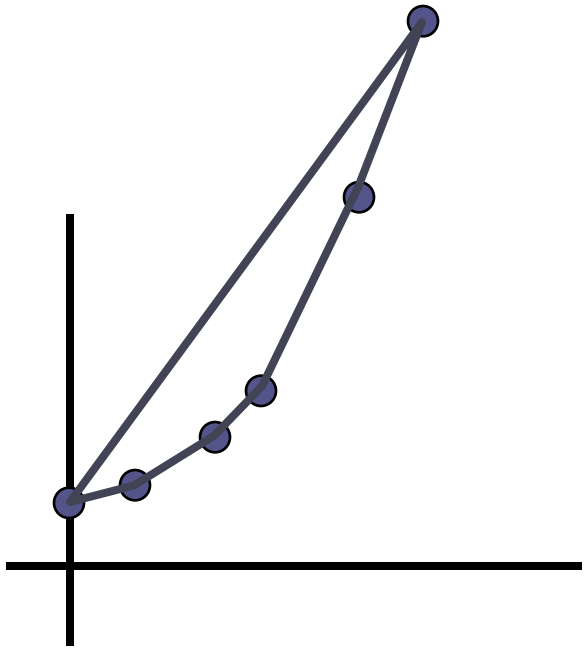
Algorithm: LOWER TANGENT
$a \leftarrow$ rightmost point of A.
$b \leftarrow$ leftmost point of B.
while $T = ab$ not lower tangent to both A and B do
    while T not lower tangent to A do
        $a \leftarrow a - 1$
    while T not lower tangent to B do
        $b \leftarrow b + 1$

Limita inferioara pentru infasuratori convexe in 2D

Presupunere: Calcularea infasuratorii convexe dureaza $\Theta(n \log n)$

Demonstratie: reducere de la Sortare la Infasuratoare Convexa:

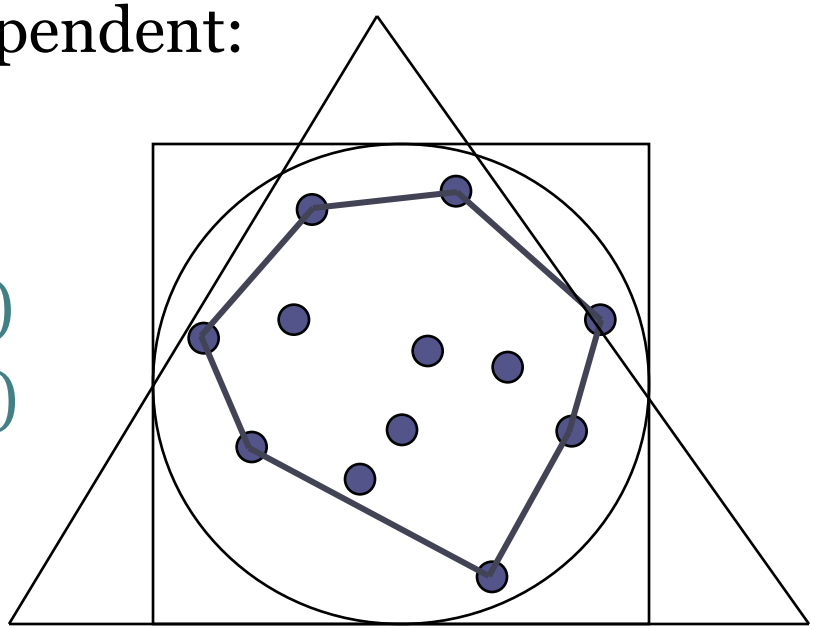
- Fiind date n valori reale x_i, se genereaza n puncte pe graficul unei functii convexe, ex. (x_i, x_i^2).
- Se calculeaza infasuratoarea convexa (ordonata) a punctelor.
- Ordinea punctelor infasuratorii convexe corespunde ordinii x_i.



Complexitate(CH)=$\Omega(n \log n)$
Dar exista un algoritm in timpul $O(n \log n)$, rezulta
Complexitate(CH)= $\Theta(n \log n)$

Infasuratori convexe in 2D: complexitate prezisa

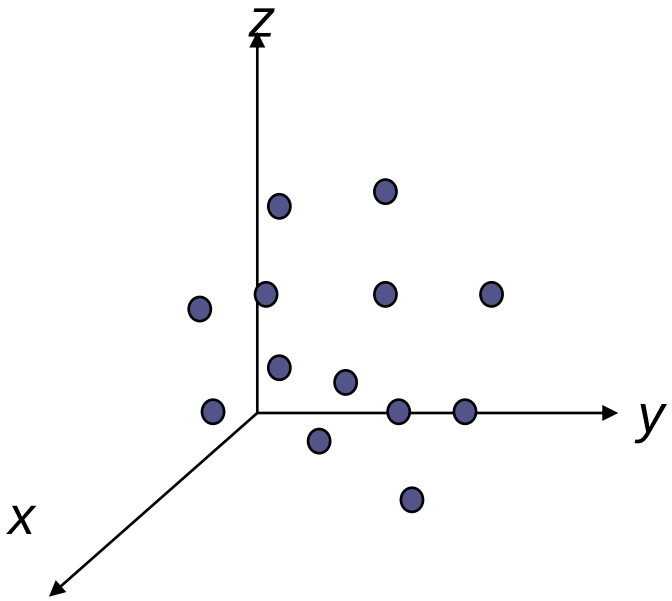
- Numarul prezis de varfuri ale infasuratorii convexe a n puncte alese uniform si independent:
  - pe un disc: $O(n^{1/3})$
  - pe un patrat: $O(\log n)$
  - pe un triunghi: $O(\log n)$



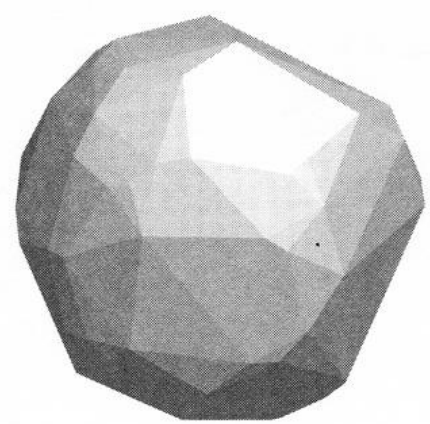
Infasuratoarea convexa a unui poligon poate fi calculata in O(n).

Infasuratori convexe in 3D

- Idee: se generalizeaza procedurile din 2D



Puncte in 3D



Infasuratoare convexa

Infasuratori convexe in dimensiuni superioare

Problema: fiind date n puncte in $\boldsymbol{R}^d$, sa se gaseasca infasuratoarea lor convexa (numita si *politop* convex).

- Fetele devin hiperfete de dimensiunea 2,3,…,d–1.
- Hiperfetele formeaza o structura de graf unde adiacentele intre diferite entitati de dimensiunea i si i-1 sunt stocate.
- Cativa din algoritmii prezentati mai devreme sunt aplicabili in dimensiuni superioare (in principiu), cu anumite extensii.

Teorema 1: Infasuratoarea convexa de n puncte in spatiul d-dimensional are cel mult $\Omega(n^{\lfloor d/2 \rfloor})$ hiperfete.

Teorema 2: infasuratoarea convexa poate fi calculata folosind algorimul "Gift Wrapping" in $\Omega(n^{\lfloor d/2 \rfloor+1}) + \Omega(n^{\lfloor d/2 \rfloor} \log n)$