Konvexe Hülle

Definition konvexe Menge:
Für je zwei beliebige Punkte, die zur Menge gehören, liegt auch stets deren Verbindungsstrecke ganz in der Menge.

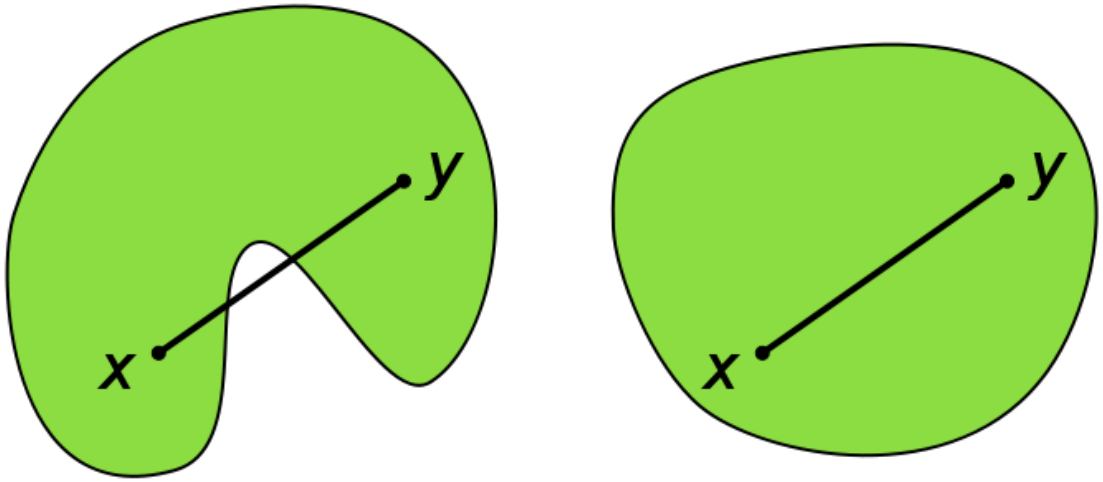


Abbildung: [Wikipedia]: Nicht-konvexe Menge (links), konvexe Menge (rechts)

Konvexe Hülle

Definition konvexe Menge:
Für je zwei beliebige Punkte, die zur Menge gehören, liegt auch stets deren Verbindungsstrecke ganz in der Menge.

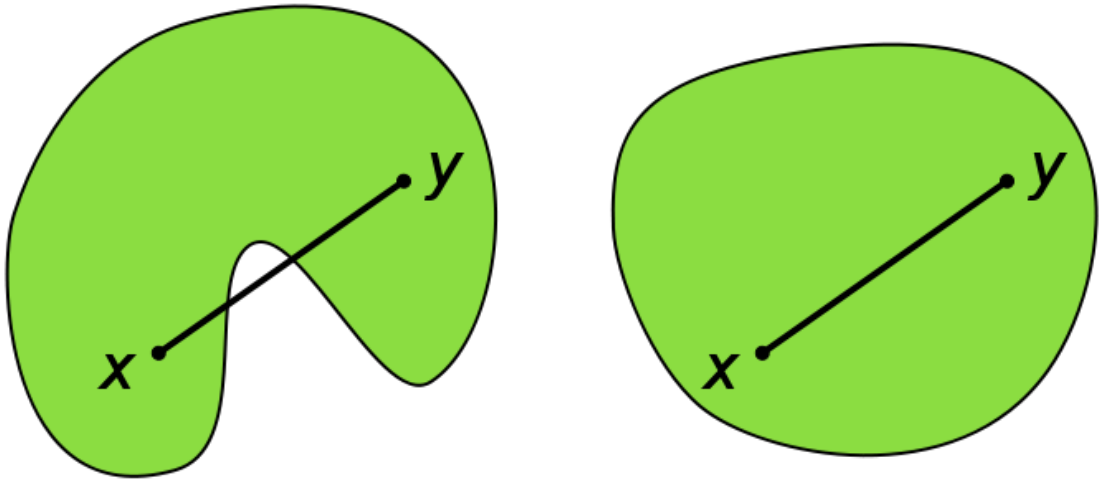


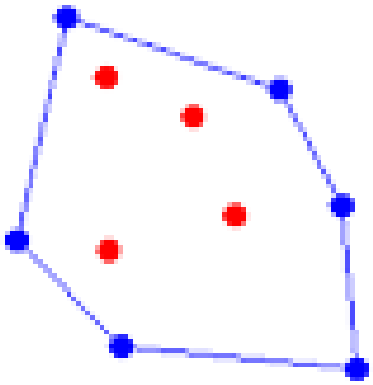
Abbildung: [Wikipedia]: Nicht-konvexe Menge (links), konvexe Menge (rechts)

Definition konvexe Hülle CH(S):
Kleinste konvexe Menge, die S enthält.

Obere konvexe Hülle

Entwurfsprinzip: Teile und herrsche (divide and conquer)

- **Gegeben:** Punktmenge $S = \{p_1, \ldots, p_n\} \subset \mathbb{R}^2$
- **Gesucht:** Konvexe Hülle von S



$CH(S)$ = Bereich,
der von blauen Kanten
umschlossen ist

Obere konvexe Hülle

Annahmen

Annahmen oBdA:

- Sortieren geht mit EREW-PRAM in $T(n) = O(\log n)$ und $W(n) = O(n \log n)$
- Vereinfachung: $n = 2^k$ für $k \in \mathbb{N}$, Koordinaten sind eindeutig

Obere konvexe Hülle

Algorithmische Idee

- Punkte p und q mit minimaler/maximaler x-Koordinate sind Teil der CH
- p und q partitionieren CH in UCH und LCH (lower convex hull)
- **OBdA:** Betrachten Upper Convex Hull (UCH)

Obere konvexe Hülle

Algorithmische Idee

- Punkte p und q mit minimaler/maximaler x-Koordinate sind Teil der CH
- p und q partitionieren CH in UCH und LCH (lower convex hull)
- **OBdA:** Betrachten Upper Convex Hull (UCH)

Vorgehen: Teile und herrsche (divide and conquer)

- Sortiere Punkte aufsteigend nach x-Koordinate
- Seien $S_1 = \{p_1, \ldots, p_{n/2}\}$ und $S_2 = \{p_{n/2+1}, \ldots, p_n\}$

Obere konvexe Hülle

Algorithmische Idee

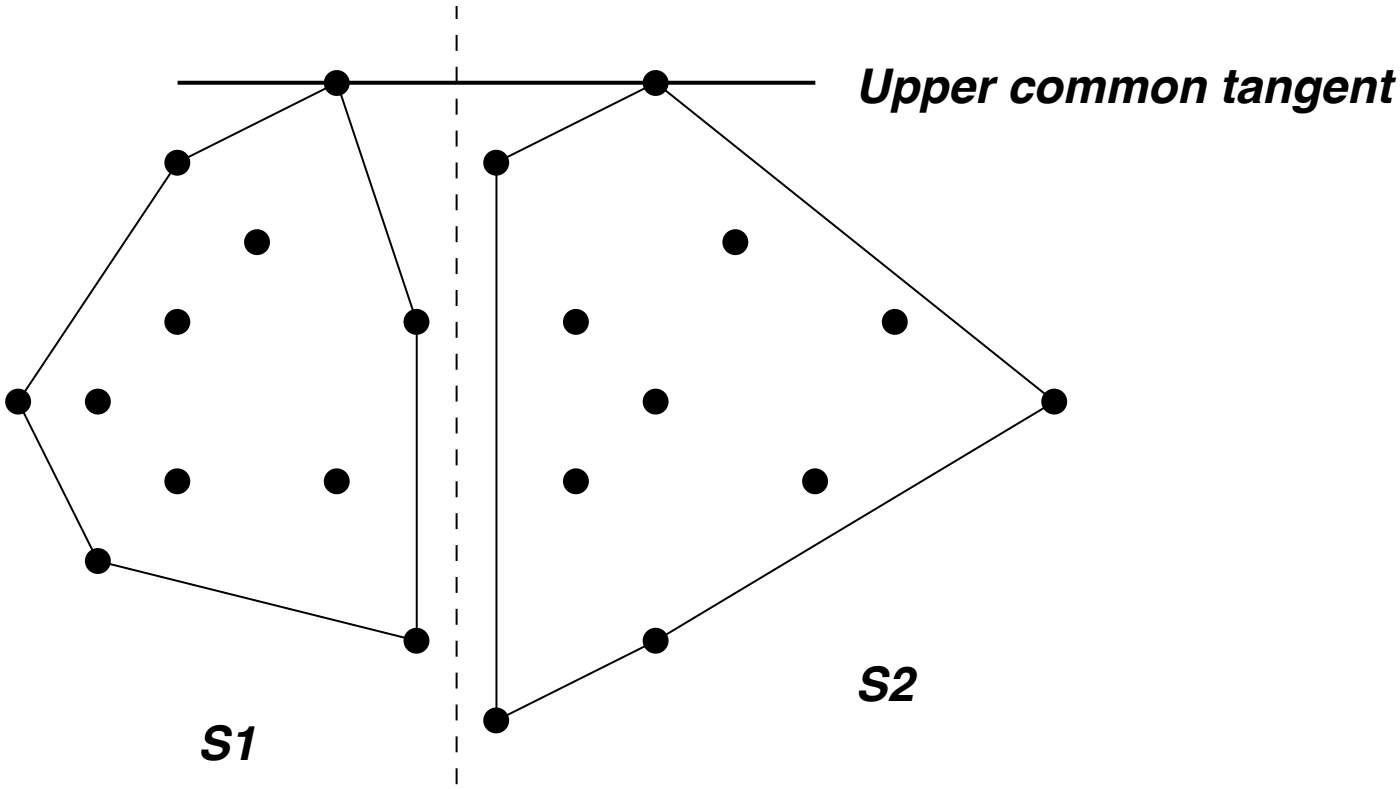
- Punkte p und q mit minimaler/maximaler x-Koordinate sind Teil der CH
- p und q partitionieren CH in UCH und LCH (lower convex hull)
- **OBdA:** Betrachten Upper Convex Hull (UCH)

Vorgehen: Teile und herrsche (divide and conquer)

- Sortiere Punkte aufsteigend nach x-Koordinate
- Seien $S_1 = \{p_1, \ldots, p_{n/2}\}$ und $S_2 = \{p_{n/2+1}, \ldots, p_n\}$
- Angenommen, $UCH(S_1)$ und $UCH(S_2)$ sind bekannt
- **Dann:** Brauchen obere gemeinsame Tangente (Stützgerade) von $UCH(S_1)$ und $UCH(S_2)$

Obere konvexe Hülle

Veranschaulichung der oberen gemeinsamen Tangente



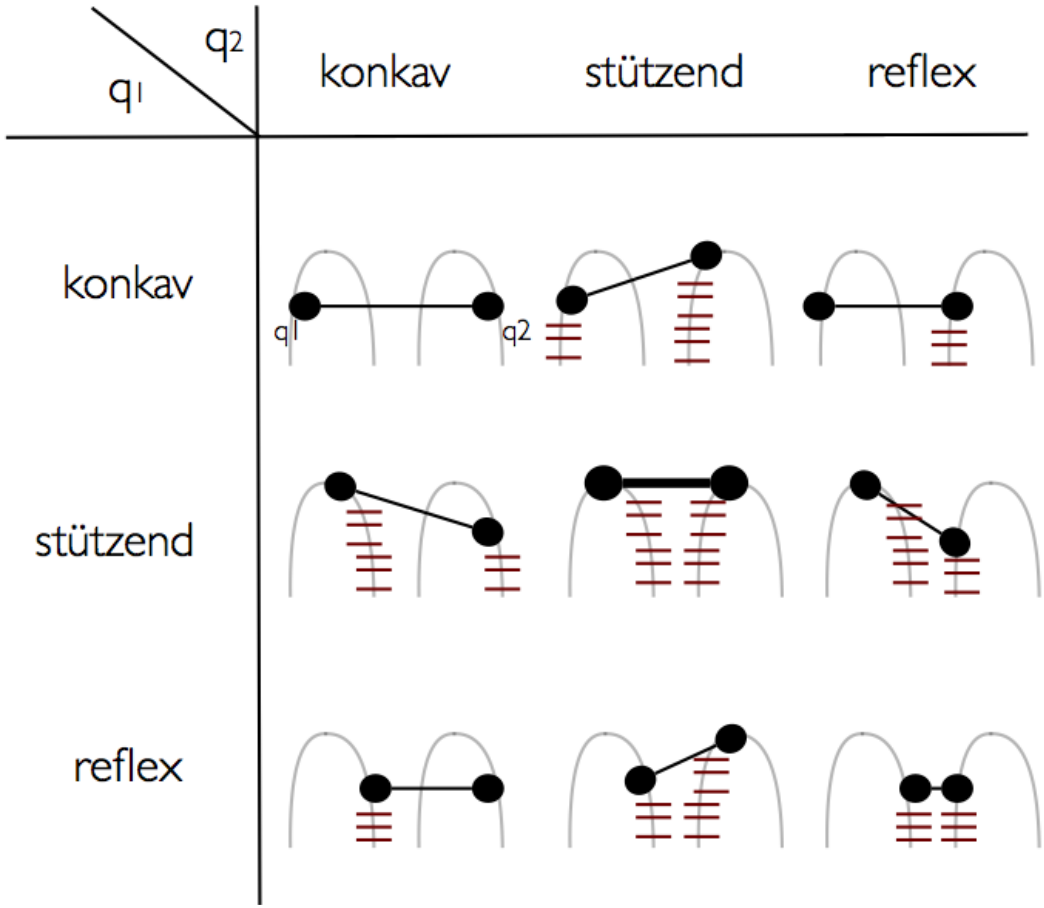
http://www.cs.yale.edu/homes/arvind/cs424/readings/pram.pdf, 06.07.2015

Obere gemeinsame Tangente

- Bestimmung geht sequentiell in $O(\log n)$ Zeit
- **Verfahren:** Binäre Suche auf beiden Seiten, jeweils 9 Fälle testen

Obere gemeinsame Tangente

- Bestimmung geht sequentiell in $O(\log n)$ Zeit
- **Verfahren:** Binäre Suche auf beiden Seiten, jeweils 9 Fälle testen



Schraffiert:
Für binäre Suche
uninteressant

Algorithmus UCH

Pseudocode

Eingabe: Menge S von n Punkten in der Ebene (mit obigen Annahmen)
Ausgabe: Obere konvexe Hülle von S

assert $x(p_1) < x(p_2) < \cdots < x(p_n)$
if $n \leq 4$ **then**
    1) Brute Force zur Bestimmung von u, der oberen konvexen Hülle von S
else
    2a) Teile S in $S_1 = (p_1, \ldots, p_{n/2})$ und $S_2 = (p_{n/2}, \ldots, p_n)$
    2b) Parallel: $u_1 := UCH(S_1)$ und $u_2 := UCH(S_2)$
    3a) Berechne sequentiell *uct* (obere gemeinsame Tangente) von u_1 und u_2
    3b) Bestimme u, die obere konvexe Hülle von S, mit Hilfe von *uct*
return u

Algorithmus UCH

Beispiel

Siehe Animation!

Algorithmus UCH

PRAM-Analyse

Rekurrenz für $T(n)$:

1. $T(n) = O(1)$, $W(n) = O(1)$
2. $T(n) = T(n/2) + c$, $W(n) = 2W(n/2)$
3. Sequentiell $O(\log n)$ Zeit für Bestimmung der oberen Tangente, dazu $T(n) = O(1)$ und $W(n) = O(n)$ fürs Kombinieren der beiden oberen Hüllen

Algorithmus UCH

PRAM-Analyse

Rekurrenz für $T(n)$:

1. $T(n) = O(1)$, $W(n) = O(1)$
2. $T(n) = T(n/2) + c$, $W(n) = 2W(n/2)$
3. Sequentiell $O(\log n)$ Zeit für Bestimmung der oberen Tangente, dazu $T(n) = O(1)$ und $W(n) = O(n)$ fürs Kombinieren der beiden oberen Hüllen

Also (für Konstanten a, b, c):

$$\begin{aligned} T(n) &\leq T(n/2) + a\log n + c && \in O(\log^2 n) \\ W(n) &\leq 2W(n/2) + bn && \in O(n \log n) \end{aligned}$$

Algorithmus UCH

PRAM-Analyse

Rekurrenz für $T(n)$:

1. $T(n) = O(1)$, $W(n) = O(1)$
2. $T(n) = T(n/2) + c$, $W(n) = 2W(n/2)$
3. Sequentiell $O(\log n)$ Zeit für Bestimmung der oberen Tangente, dazu $T(n) = O(1)$ und $W(n) = O(n)$ fürs Kombinieren der beiden oberen Hüllen

Also (für Konstanten a, b, c):

$$\begin{aligned} T(n) &\leq T(n/2) + a\log n + c && \in O(\log^2 n) \\ W(n) &\leq 2W(n/2) + bn && \in O(n \log n) \end{aligned}$$

Achtung: $T(n)$ ist hier nicht optimal, $W(n)$ schon!

Algorithmus UCH

PRAM-Analyse

Rekurrenz für $T(n)$:

1. $T(n) = O(1)$, $W(n) = O(1)$
2. $T(n) = T(n/2) + c$, $W(n) = 2W(n/2)$
3. Sequentiell $O(\log n)$ Zeit für Bestimmung der oberen Tangente, dazu $T(n) = O(1)$ und $W(n) = O(n)$ fürs Kombinieren der beiden oberen Hüllen

Also (für Konstanten a, b, c):

$$\begin{aligned} T(n) &\leq T(n/2) + a\log n + c && \in O(\log^2 n) \\ W(n) &\leq 2W(n/2) + bn && \in O(n \log n) \end{aligned}$$

Achtung: $T(n)$ ist hier nicht optimal, $W(n)$ schon!

Brauchen für optimales $T(n)$ schnelleres Conquer!

Zusammenfassung

- Parallele Algorithmen sind nützlich
- PRAM: Abstraktes Shared-Memory-Modell
- Work-Time-Prinzip

Zusammenfassung

- Parallele Algorithmen sind nützlich
- PRAM: Abstraktes Shared-Memory-Modell
- Work-Time-Prinzip
- Parallele Entwurfskonzepte:
  - Baumparadigma
  - Teile und herrsche
  - ... ($\rightarrow$ Vorlesung Parallele Algorithmen von Prof. Sanders)

Kap. 14: Zusammenfassung

- Datenstrukturen
- Algorithmen
- Entwurfstechniken
- Analysetechniken

Zusammenfassung – Datenstrukturen

- (doppelt) verkettete Listen, unbeschränkte (zyklische) Felder, Stapel, FIFOs, deques
- (beschränktes) Hashing: verketten (universell) / lin. Suche
- sortiertes Feld
- Prioritätslisten (binärer Heap) (addressierbar)
- Implizite Repräsentation vollständiger Bäume
- Suchbäume: binär, (a, b)-Baum
- Graphen: Adjazenzfeld / Listen / Matrix
- Union-Find

Zusammenfassung – Algorithmen

- Langzahlmultiplikation
- Insertion-, Merge-, Quick-, Heap-, Bucket-, Radix-sort, Selektion
- BFS, DFS, topologisches Sortieren
- Kürzeste Wege: Dijkstra, Bellman-Ford
- MST: Jarník-Prim, Kruskal
- Parallele Summe
- Konvexe Hülle auf der PRAM

Zusammenfassung – Entwurfstechniken I

- Iteration/Induktion/Schleifen, Teile-und-Herrsche
- Schleifen- und Datenstruktur-Invarianten
- Randomisierung (universelles Hashing, Quicksort,...)
- Graphenmodelle
- Trennung Mathe $\leftrightarrow$ Funktionalität $\leftrightarrow$ Repräsentation $\leftrightarrow$ Algorithmus $\leftrightarrow$ Implementierung
- Sonderfälle vermeiden
- Zeigerdatenstrukturen
- Datenstrukturen augmentieren (z.B. Teilbaumgrößen)
- Datenstrukturen unbeschränkt machen
- Implizite Datenstrukturen (z.B. Intervallgraphen)

Zusammenfassung – Entwurfstechniken II

- Algebra
  (Karatsuba, univ. Hashfkt., Matrixmultiplikation für Graphen)
- Algorithmenschemata (z.B. DFS, lokale Suche)
- Verwendung abstrakter Problemeigenschaften
  (z.B. Schnitt/Kreis-Eigenschaft bei MST)
- Black-Box-Löser (z.B. lineare Programmierung)
- Greedy
- Dynamische Programmierung
- Systematische Suche
- Metaheuristiken (z.B. Lokale Suche)
- Parallelität

Zusammenfassung – Analysetechniken

- Summen, Rekurrenzen, Induktion, Master-Theorem, Abschätzung
- Asymptotik ($O(\cdot)$, ..., $\omega(\cdot)$), einfache Modelle
- Analyse im Mittel
- Amortisierung (z.B. unbeschränkte Felder)
- Linearität des Erwartungswertes, Indikatorzufallsvariablen
- Kombinatorik ($\approx$ Zählen): univ. Hashfunktionen, untere Sortierschranken (Informationsmenge)
- Integrale als Summenabschätzung
- Schleifen/Datenstruktur-(In)varianten (z.B. (a, b)-Baum, Union-by-rank)

Zusammenfassung – weitere Techniken

- Algorithm Engineering
- Parameter-Tuning (z.B. Basisfallgröße)
- High-Level-Pseudocode
- Dummys und Sentinels (Listen, insertion sort,...)
- Speicherverwaltung