Algoritmy používané ve výpočetní geometrii

Hrubá síla. Inkrementální metoda. Zametací přímka. Heuristiky. Rozděl a panuj.

Tomáš Bayer | bayertom@natur.cuni.cz

Katedra aplikované geoinformatiky a kartografie. Přírodovědecká fakulta UK.

Obsah přednášky

1. Úvodní informace o předmětu

2. Algoritmy ve výpočetní geometrii
   - Incremental Insertion
   - Incremental Construction
   - Sweep Line
   - Divide and Conquer
   - Heuristic Algorithms
   - Greedy Algorithm
   - Randomizované algoritmy

1. Plán přednášek

Tematické celky:

(1) Algoritmy ve výpočetní geometrii.

(2) Geometrické vyhledávání.

(3,4) Konvexní obálky.

(5,6,7) 2D triangulace, DMT.

(8) Voroného diagramy.

(9) Topologická kostra.

(10, 11) Kartografická generalizace.

(12) Operace s uzavřenými oblastmi v GIS.

Cvičení:

Implementace algoritmů v jazyce C++/Java.

Literatura:

[1] Rourke O. J.: Computational Geometry in C, 2005, Cambridge University Press.

[2] de Berg, van Kreveld, Overmars M., Schwarzkopf O.: Computational Geometry, 2000, Springer.

[3] Bayer T.: Algoritmy v digitální kartografii, 2008, UK v Praze.

2. Algoritmy ve výpočetní geometrii

Kartografie: počítačová grafika doplněná pravidly (geometrická, atributová, sémantická).

Sféry zájmu: výpočetní geometrie, počítačová grafika, informatika, matematika.

Neexistuje univerzální technika vedoucí k nalezení řešení.

Řešení neumíme verifikovat: velký problém kartografických úloh.

Někdy požadováno přesné řešení, jindy pouze přibližné (NP problémy).

V závislosti na typu problému/řešení/vstupní množiny nutné zvolit vhodnou strategii.

Přehled technik návrhu algoritmů:

- Metoda hrubé síly (Brute Force Algorithm).
- Inkrementální algoritmy (Incremental Algorithms).
- Zametací přímka (Sweep Line).
- Rozděl a panuj (Divide and Conquer).
- Heuristické algoritmy (Heuristic Algorithm).
- Aproximační algoritmy.
- Randomizované algoritmy.

3. Metoda hrubé síly

Vyzkoušení všech variant řešení problému, vybráno nejlepší z nich.

Počet operací nutných k nalezení řešení roste

$$C_k(n) = \left(\begin{array}{c} n \\ k \end{array} \right), \qquad k = n \Rightarrow C = n!.$$

Výhodou je jednoduchost implementace.

Lze aplikovat pouze na malé datové soubory, zpravidla $n <= 10$.

Pro rozsáhlé soubory nelze vyzkoušet všechny kombinace.

Postupy založené na hrubé síle bývají nazývány *naivními algoritmy*,

Neobsahují žádnou hlubší myšlenku.

Využití např. u NP problémů (pro malá n).

Lepších výsledků pro tyto problémy dosahují *heuristické algoritmy.*

4. Inkrementální algoritmy

Postupné přidávání/výběr jednoho prvku ze vstupní množiny/existujícího řešení.

Po každém kroku je řešení aktualizováno.

Druhy inkrementálních algoritmů:

- Inkrementální vkládání.
- Inkrementální konstrukce.
- Inkrementální výběr.

Vlastnosti:

- Algoritmy implementačně poměrně jednoduché, za předpokladu, že přidání / odebrání jednoho prvku příliš neovlivní výsledné řešení.
- Vhodné i pro značně složité problémy.
- Snadno lze analyzovat změny řešení v závislosti na vstupní množině dat.
- Lze použít jako on-line algoritmy, po přidání vstupu vidíme výsledné řešení nad podmnožinou zpracovaného data setu.

Jedny z nejčastěji používaných algoritmů ve výpočetní geometrii:

konvexní obálky, Delauany triangulace, Voronoi diagram, skeletony...

5. Inkrementální vkládání (Incremental Insertion)

Postupná konstrukce řešení S nad množinou $P = \{p\}_{i=1}^{n}$.

Průběžné řešení pro k prvků z řešení pro $k-1$ prvků + update Δ přidáním prvku p.

$$S(P(k)) = S(P(k-1)) + \Delta S(p), \quad 1 \leq k \leq n.$$

Po dobu běhu algoritmu nemusí být k dispozici celá vstupní množina dat.

Data mohou být získávána průběžně (tj. i v době běhu algoritmu), on-line metoda.

Data vybírána ze vstupní množiny buď randomizovaně nebo dle určitých pravidel.

Jeden z nejčastěji používaných postupů v CG.

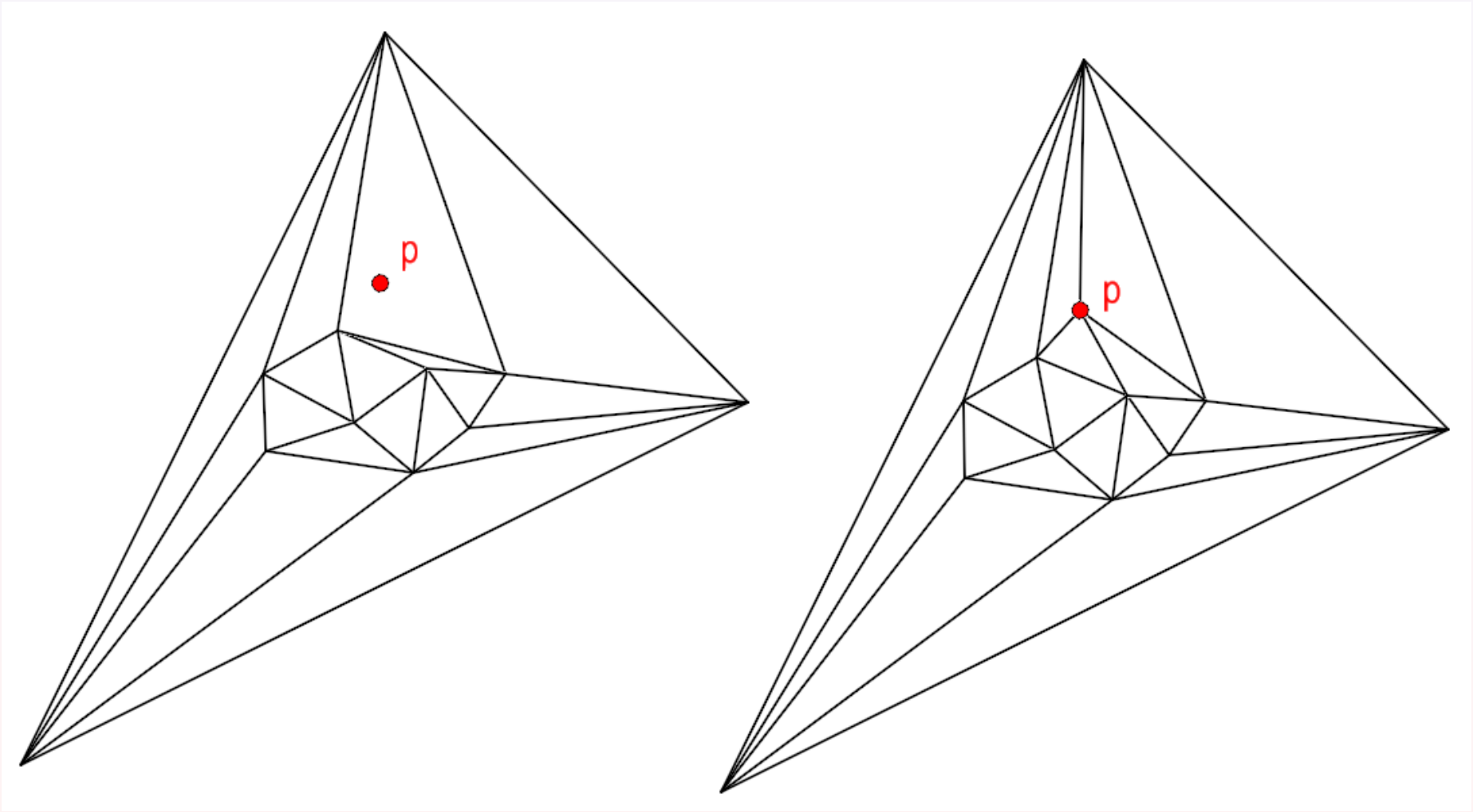
V paměti udržováno většinou pouze výsledné řešení, pro některé situace i průběžná řešení

Lze analyzovat vliv dat na řešení, případně provést Undo.

Pro první kroky (“malá” n) je nutno vstupní množinu doplnit vhodně volenými daty (simplex).

Simplex po zpracování celé množiny n prvků z řešení dodatečněs odstranit.

6. Inkrementální vkládání: konstrukce DT



7. Inkrementální konstrukce

= Incremental Construction.

Generuje řešení nad podmnožinou $Q \subset P$ průběžně.

Přidání dalšího vstupu zpravidla *neovlivní* dílčí řešení (neopravuje).

Data zpracovávána podle požadavků algoritmu, nemohou být přidávána na vstup randomizovaně.

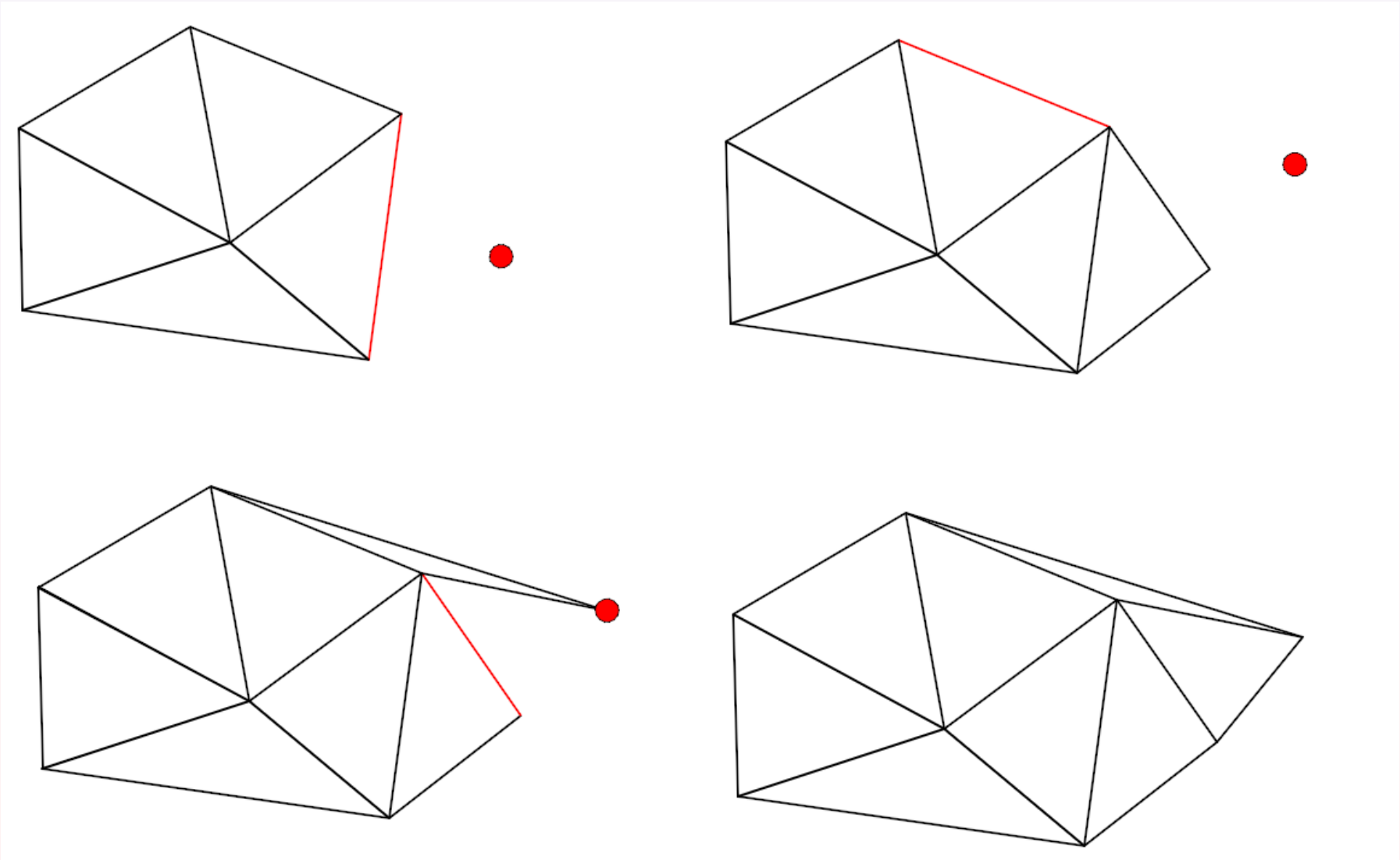
V době běhu algoritmu musí být k dispozici celá vstupní množina dat.

Pro urychlení běhu algoritmu je vhodné data *předzpracovat*, např. vhodně setřídit.

Inkrementální konstrukce často používán ve výpočetní geometrii (Delaunay triangulace).

Většinou pomalejší než inkrementální vkládání.

8. Inkrementální konstrukce: DT



9. Zametací přímka (Sweep-line)

Používána pro řešení 2D/3D problémů.

Zobecněná technika inkrementální konstrukce.

Nad vstupní množinou se pohybuje nadrovina (zpravidla L->P).

2D: přímka (častější), 3D (rovina).

Rozdělení na LHP (již nalezené řešení) a RHP (hledá se řešení).

2 Strategie - řešení v LHP se:

a] již neopravuje (Bentley&Ottman) ,

b] aktualizuje s přihlédnutím k RHP $(\mathcal{H}, \mathcal{V})$.

Vyžaduje *předzpracování* $O(N \lg N)$, objekty setříděny dle souřadnice x.

Posun “diskrétní”, probíhá po prvcích vstupní množiny.

V každém roku přidávám k řešení 1 prvek.

Alternativně prioritní fronta.

10. Sweep Line

Složitost $O(N \lg N)$-$O(N^2)$.

Různé implementace: předtřídění, prioritní fronta.

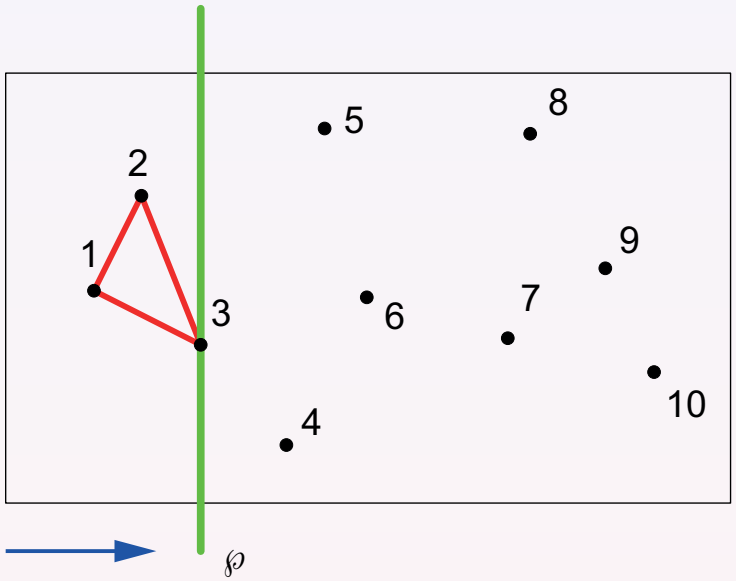
Algoritmus 1: Sweep Line (P, S)

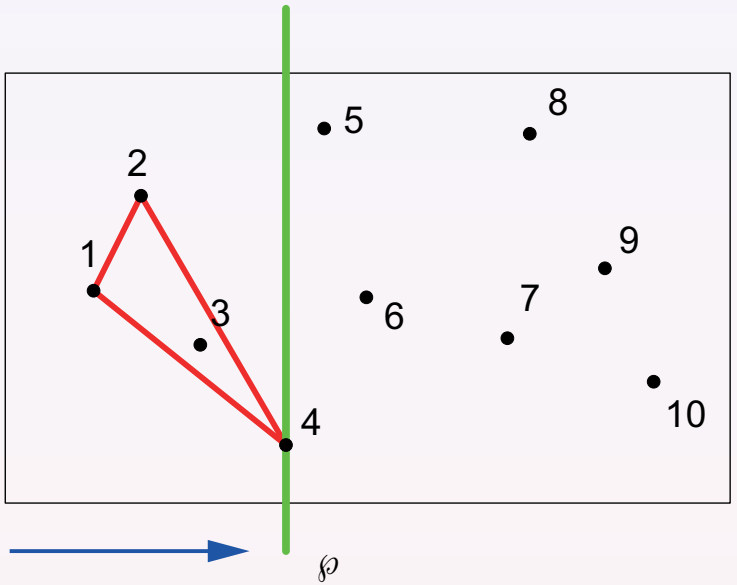
1: Sort $P_s = sort(P)$ by x
2: $S \leftarrow S(3)$. //Iniciální řešení pro $i \leq 3$
3: $i \leftarrow 4$.
4: Opakuj pro $\forall p_i \in P_s$
5: Posun $\wp(p_i)$ //Posun na aktuální prvek množiny
6: $S(P_s(i)) = S(P_s(i-1)) + \Delta S(p_i)$ //Aktualizace řešení
6: $i \leftarrow i + 1$ //Inkrementace

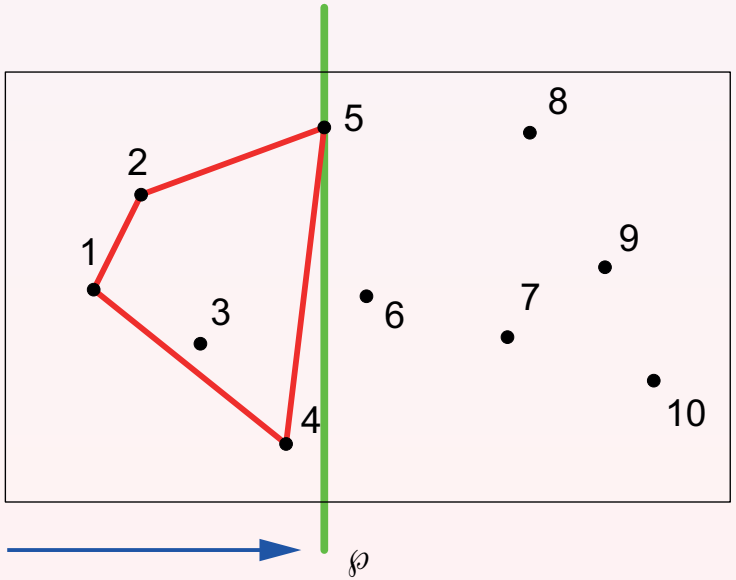
Iterační část zpravidla pro $i \geq 3$.

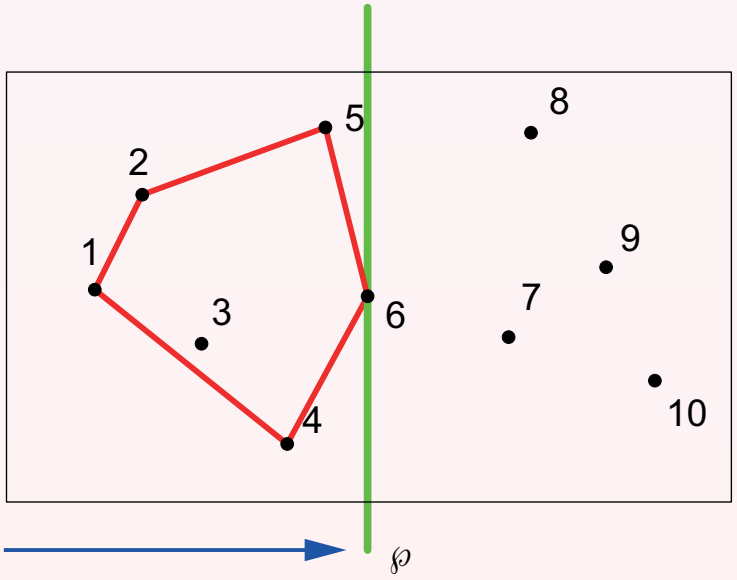
Iniciální řešení snadno dokážeme nalézt, např. trojúhelník u $\mathcal{H}$.

11. Sweep Line, konvexní obálka









12. Divide & Conquer

Opakovaném rozdělování problému na menší a jednodušší podproblémy.

Jejich řešení podobné původnímu problému.

Důsledek návrhu algoritmu technikou “Shora dolů”.

Hodí se pro **dekomponovatelné úlohy**.

Úlohy rozdělitelné na podúlohy stejného nebo podobného typu.

Dělení prováděno dokud není řešení podproblému triviální (tj. většinou přímé).

Takové řešení umíme zpravidla nalézt bez složitých výpočtů.

Podproblémy řešeny nezávisle na sobě.

Poté jejich řešení *spojena* v celek, čímž získáme řešení původního problému.

Odhad složitosti: $\Theta(n \log n)$, ale $O(N^2)$.

Techniky řešení: Rekurze, iterace.

Příklady: třídící algoritmy (Merge Sort, Quick Sort), významné pro úlohy výpočetní geometrie (Convex Hull, Voronoi Diagram, Delaunay Triangulace...), kartografická generalizace (Douglas Peucker Algorithm)

13. Divide and Conquer

Zpravidla obtížná implementace, paralelizace

Interval indexů $< a, b >$ vstupní podmnožiny P.

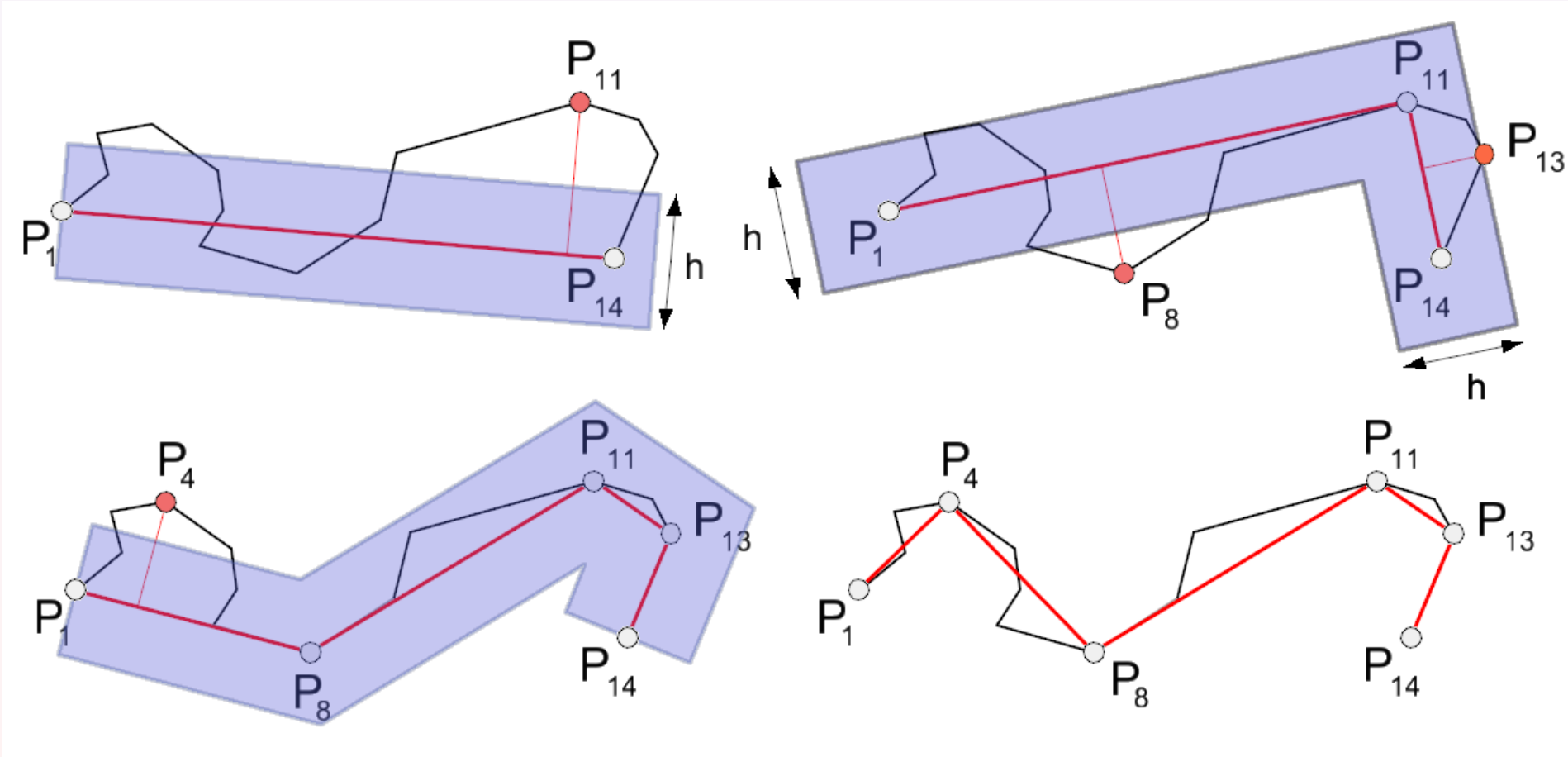
Dílčí řešení $S(P(a, b))$ nad intervalem prvků $< p_a, p_b >$.

Algoritmus 1: function DAC(P, S, a, b)

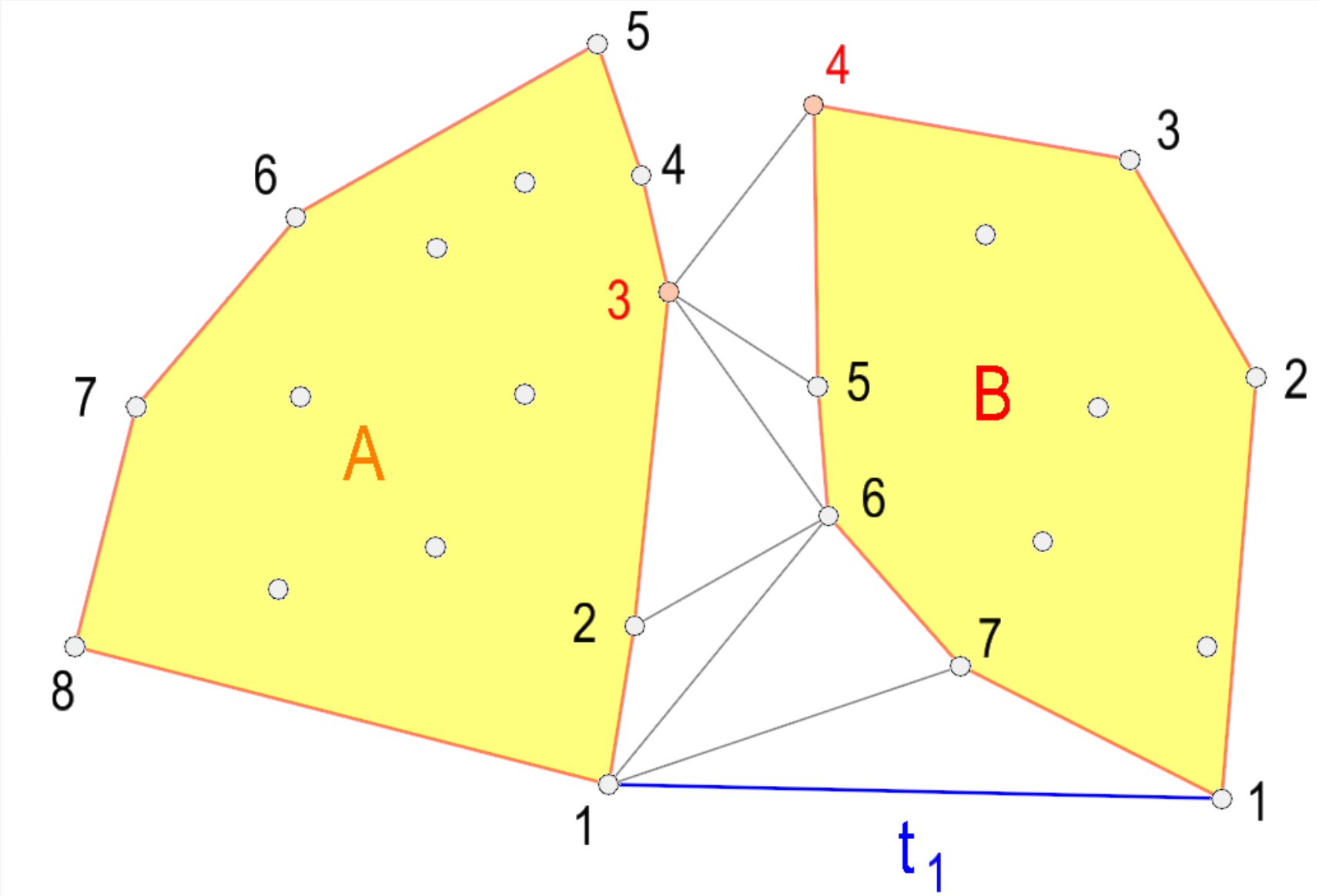
1: if (a, b) nedělitelné nebo $S(P(a, b))$ jednoduché
2: $S \leftarrow S(P(a, b))$ //Jednoduché řešení
3: else //Složité řešení, dělení intervalu
4: $i = E(a, b)$ //Střední hodnota
5: $S_1 \leftarrow$DAC(P, S, a, i) //Rekurze nad 1. intervalem
6: $S_2 \leftarrow$DAC(P, S, i, b) //Rekurze nad 2. intervalem
7: $S \leftarrow \mathrm{merge}(S_1, S_2)$ //Spojení obou řešení

Klíčové parametry: efektivní dělení P (medián), rychlé skládání řešení.

14. Ukázka Douglas-Peucker algoritmu



15. Ukázka konstrukce $\mathcal{H}$



16. Ukázku výpočtu časové složitosti (2 dělení)

Očekávaná složitost (dělení v mediánu):

$$\begin{aligned} T(n) &= 2T(n/2) + n, \\ &= 2[2T(n/4) + n/2] + n = 4T(n/4) + 2n, \\ &= 4[2T(n/8) + n/4] + 2n = 8T(n/8) + 3n, \\ &= 8[2T(n/16) + n/8] + 3n = 16T(n/16) + 4n, \\ &= iT(n/i) + \log_2(i)n, \\ &= n(T(1)) + \log_2(n)n, \\ &\leq n\log(n). \end{aligned}$$

Horní odhad složitosti (dělení v 2./předposledním bodu):

$$\begin{aligned} T(n) &= T(n-1) + n, \\ &= [T(n-2) + n - 1] + n = T(n-2) + n - 1 + n, \\ &= T(n-3) + n - 2 + n - 1 + n, \\ &= T(n-i) + n - i + 1 + .. + n - 2 + n - 1 + n, \\ &= T(0) + \frac{n}{2}(n+1), \\ &\leq n^2. \end{aligned}$$

17. Heuristické algoritmy

Vygenerováno velké množství potenciálních řešení.

Na rozdíl od metody hrubé síly nikoliv všechny varianty.

Z nich hledáno nejlepší možné řešení.

Pozor: Vygenerovaná množina nemusí obsahovat nejlepší možné řešení.

Heuristické algoritmy využívají heuristiku.

Nehledají exaktní řešení problému, ale tzv. *přípustné řešení*

Řešení není nejlepší možné, ale zároveň není špatné.

Nemusí být nalezeno v rychlém čase, některé heuristiky pracují poměrně dlouho.

Špatné určování časové složitosti, někdy běží rychle, jindy ne (závislost na vstupních datech).

Zvýšením počtu opakování lze dosáhnou přesnějšího řešení.

Kvalita řešení zpravidla je zpravidla funkcí počtu iterací (odmocnina) nadměrné zvyšování iterací nemá proto smysl.

Použití heuristiky: NP úlohy, optimalizační problémy.

18. Techniky heuristiky

Základní techniky heuristiky:

- *Iterativní algoritmy:*
  Postupné hledání řešení nad zmenšující se množinou možných řešení, kvalita řešení se v každém dalším kroku zlepšuje.
- *Hladové algoritmy:*
  V každém dílčím kroku hledáno optimální řešení. Hledáním lokálního minima chceme nalézt globální minimum.
- *Genetické algoritmy:*
  Založeny na principech evoluční biologie. Do dalšího kroku přežívají pouze perspektivní řešení, která jsou dále zlepšována drobnou modifikací (křížením).

Výhody a nevýhoda heuristiky:

+ Snadná implementace.

+ Často jediné řešení problémů, u nichž neexistuje exaktní řešení (NP problémy).

- Obtížné stanovení časové složitosti, závislost na konfiguraci vstupních dat.

- Řešení nelze dokázat.

- V mezních situacích nemusí být nalezené řešení vhodné.

- Není zaručeno, že takové řešení bude vždy nalezeno.

19. Kvalita řešení

Cílem nalezení přípustného řešení.

Jak je nalezené řešení dobré? (kvalita vs. počet iterací)

Hodnocení kvality nalezeného řešení:

- *Odchylka nalezeného řešení m_A od optimálního m_{opt}*
  Hodnoceno poměrem k

$$k = \frac{m_A}{m_{opt}}.$$

  Nalezené řešení představuje k% optima (např. 90%).
  Jako optimální může být posuzováno dosud nejlepší nalezené řešení.
- *Počet kroků vedoucích k nalezení řešení*
  Snaha o co nejrychlejší konvergenci algoritmu.
  Důležitá volba podmínky ukončující iteraci.

20. Hladový algoritmus (Greedy Algorithm)

GA z množiny $P = \{p_i\}_{i=1}^{m}$ vybere podmnožinu $S = \{p_j\}_{j=1}^{n}$ $S \subset P$, $n \leq m$, s minimálním ohodnocením

$$W(S(n)) = \sum_{j=1}^{n} w(p_j), \qquad w(p_j) > 0,$$

kde $w(p_j)$ ohodnocením prvku p_j *účelovou funkcí* w.

GA ve snaze najít globální minimum nad S

$$W(S(k)) = W(S(k-1)) + w(\underline{p}) = \min,$$

hledá v k-tém kroku prvek $\underline{p} \in P$ s nejlepším ohodnocením

$$\underline{p} = \arg\min_{p_i \in P}(w(p_i)).$$

Aktualizace přidáním optimálního prvku (výchozí strategie)

$$S(k) = S(k-1) \vee \underline{p}.$$

Alternativní strategie: přípustné řešení není horší než ε

$$S(k) = \begin{cases} S(k-1) \vee \underline{p}, & \text{w}(\underline{p})<\varepsilon, \\ S(k-1), & \text{jinak.} \end{cases}$$

Zaručuje nalezení lokálního minima.

21. Hladový algoritmus

Představuje lokální optimalizaci vzhledem k w.

Algoritmus 1: Greedy (P, S, w, m,)

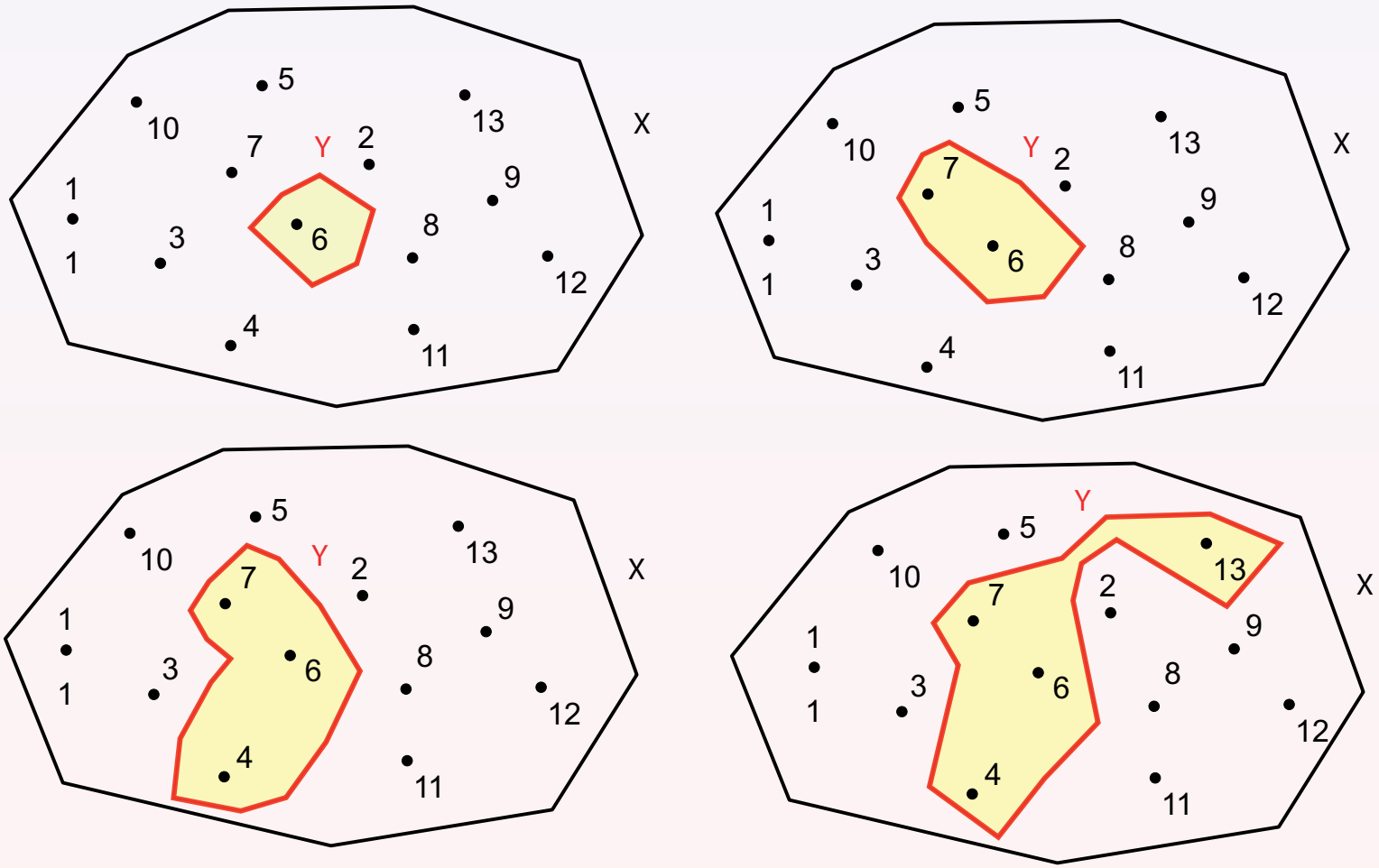
1: $S \leftarrow \emptyset$, $W = 0$
2: for $i = 1 : m$
3: $\quad \underline{p} = \arg\min_{p_i \in P}(w(p_i))$ //Nalezni optimální prvek
4: $\quad$ if $(w(\underline{p}) < \varepsilon)$ //Přípustné řešení
5: $\quad\quad W \leftarrow W + w(\underline{p})$ //Aktualizuj ohodnocení
6: $\quad\quad S \leftarrow \underline{p}$ //Přidej $\underline{p}$ do S
7: $\quad\quad P \rightarrow \underline{p}$ //Odeber $\underline{p}$ z P

Ohodnocení $w(p_i)$ můžeme předpočítat.

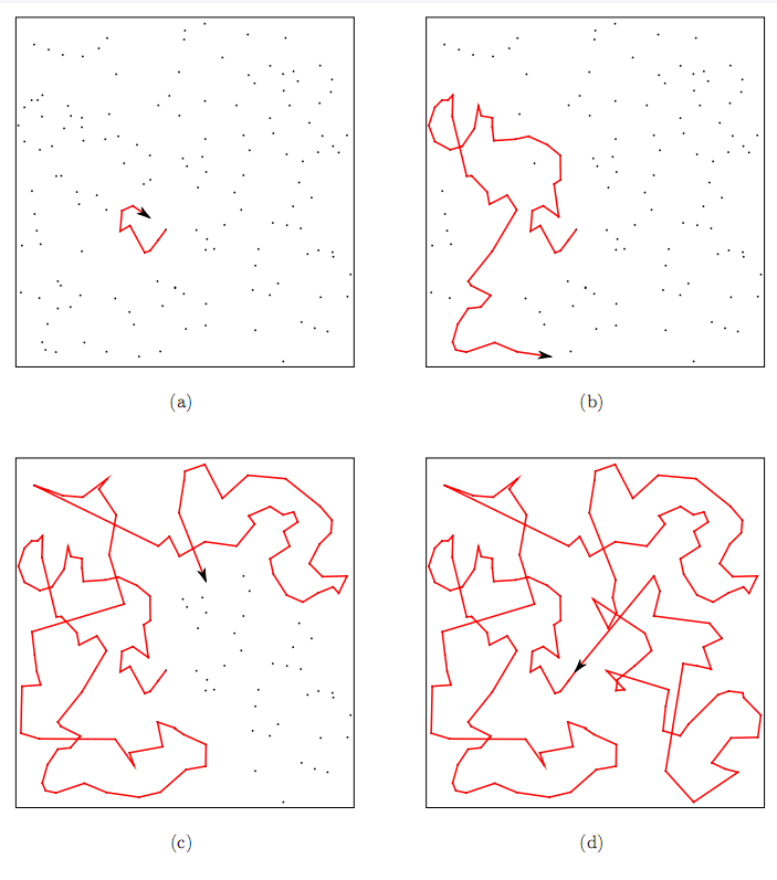
Prvky $p_i \in P$ uloženy v prioritní frontě $(w(p_i), p_i))$.

Netřeba hledat minimum v (3).

22. Princip hladového algoritmu

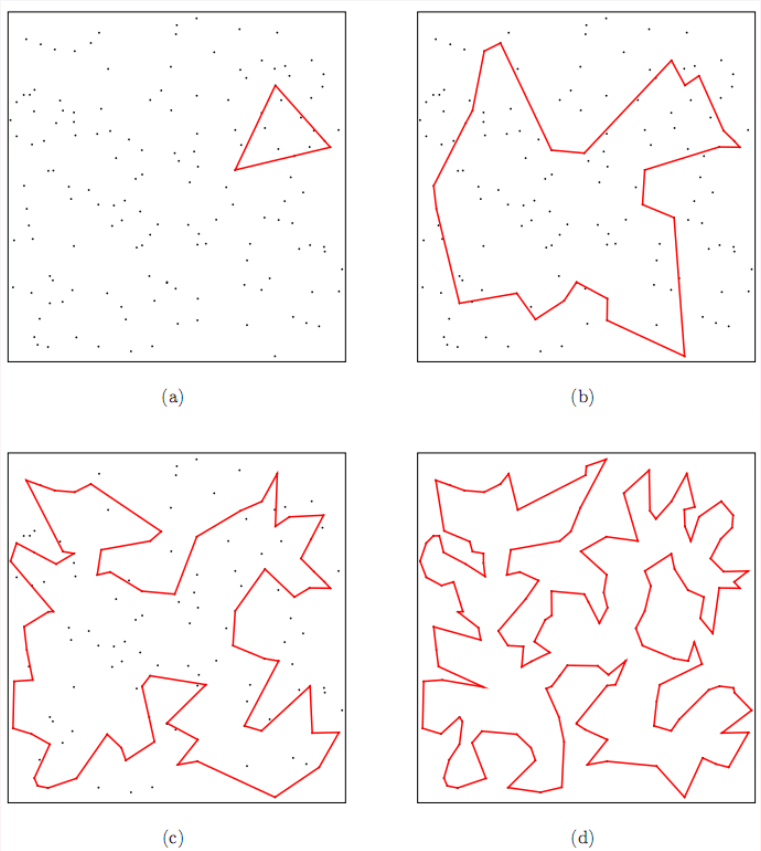


23. Heuristika: TSP, Nearest Neighbor



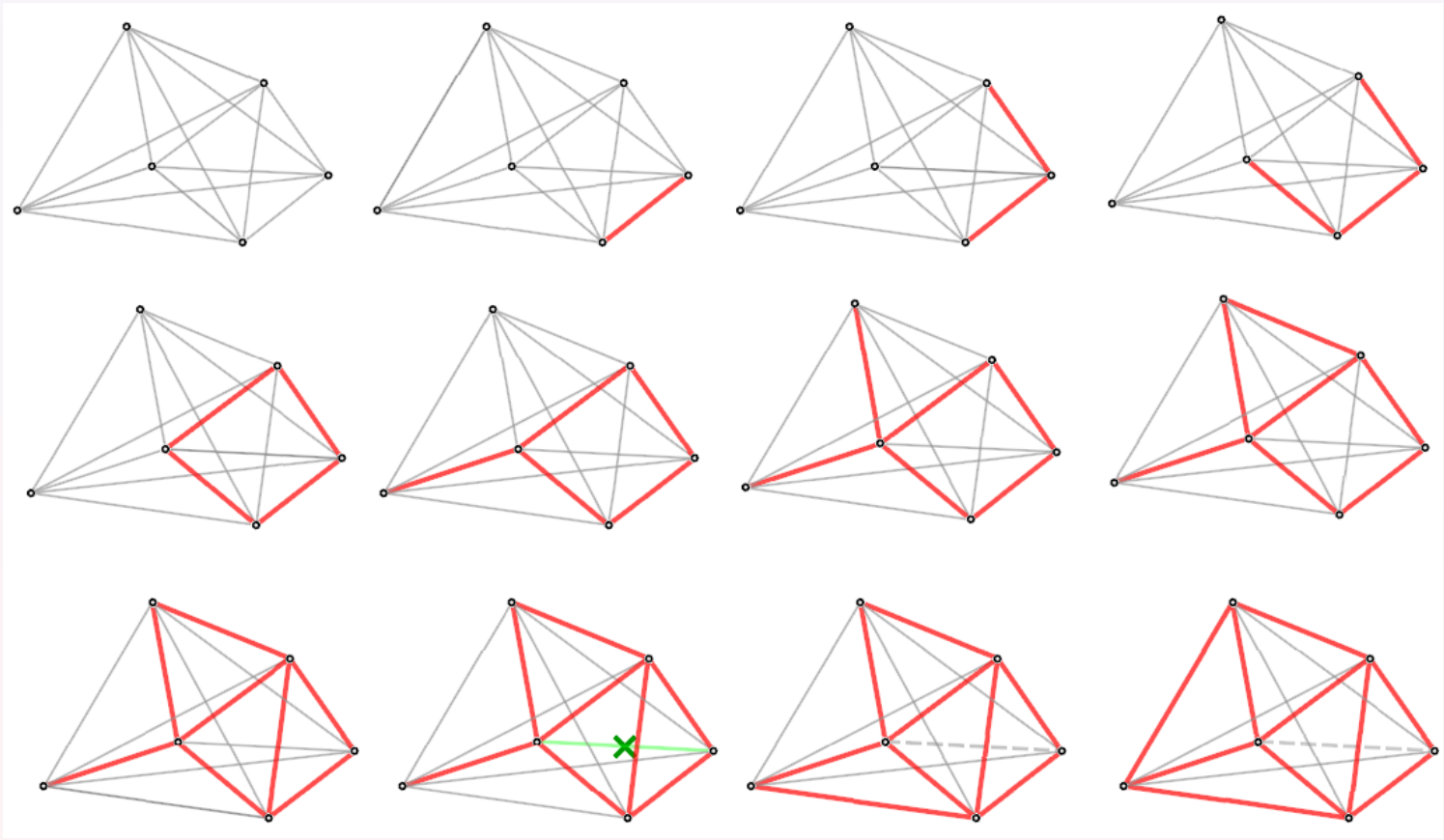
$$W(S(n)) = \sum_{j=1}^{n-1} \|p_j - p_{j-1}\|_2 = \min, p_k = \arg \min_{\forall p_i \in X} \|p_i - p_j\|_2 .$$

24. Heuristika: TSP, Best Insertion



$$W(S(n)) = \sum_{j=1}^{n-1} \|p_j - p_{j-1}\|_2 = \min, p_k = \arg \min_{\forall p_i \in X} \left(\|p_i - p_j\|_2 + \|p_i - p_{j-1}\|_2 \right).$$

25. Grafické znázornění Greedy triangulace



$$W(S(n_h)) = \sum_{j=1}^{n_h} \|h_j\|_2 = \min, h_k \equiv h_{[k]} = \arg \min_{\forall h_i \in h \setminus h_{[1]}, \ldots, \setminus h_{[k-1]}} \left(\|h_i\|_2\right),$$

$$\text{kde } h_{[1]} = \arg\min_{\forall h_i \in h} \left(\|h_i\|_2\right).$$

26. Randomizované algoritmy (1/3)

Zavádějí do procesu řešení nedeterministický prvek, tj. prvek náhodnosti.

Cílem zjednodušení implementace algoritmu při zachování Worst Case.

Máme -li štěstí (jestliže není vybrán nevhodný prvek), běží algoritmus průměrně rychleji.

Tímto postupem lze řešit úlohy, které nemusí být běžně řešitelné.

Metody randomizace:

- *Náhodný (nedeterministický) výběr prvku ze vstupních dat*
  Zpracování dat ze vstupní množiny v náhodném pořadí, v každém okamžiku vybrán náhodný prvek.
- *Náhodné stanovení prahové veličiny*
  Výběr náhodného prvku u kterého očekáváme požadované statistické parametry (např. medián).
  Nahrazuje předzpracování ve formě statistické analýzy dat.

Výhody a nevýhody:

+ snadnost implementace,

+ vylepšení špatných vlastností algoritmů (špatný Worst Case, dobrý Average Case),

- nelze použít pro NP problémy,

- v obecných případech vede převod deterministického algoritmu na nedeterministický ke zhoršení výkonu.

27. Randomizované algoritmy (2/3)

Dělení randomizovaných algoritmů:

- *Randomizované inkrementální algoritmy:*
  Zpravidla náhodný výběr prvku ze vstupní množiny, použití např. u triangulací.
- *Randomizované Divide & Conquer algoritmy:*
  Náhodný výběr prvku majícího předpokládané statistické parametry.
  Např. QuickSort, náhodné stanovení mediánu.

Vzorkování:

Před spuštěním algoritmu obyčejně provedena statistická analýza podmnožiny dat, statistické parametry následně vztaženy na celý vstupní soubor.

Výběr vzorku důležitý, požadavky:

- reprezentativnost (vzorek reprezentuje rozložení dat v celém souboru),
- přiměřená velikost (krátká doba zpracování).

28. Randomizované algoritmy (3/3)

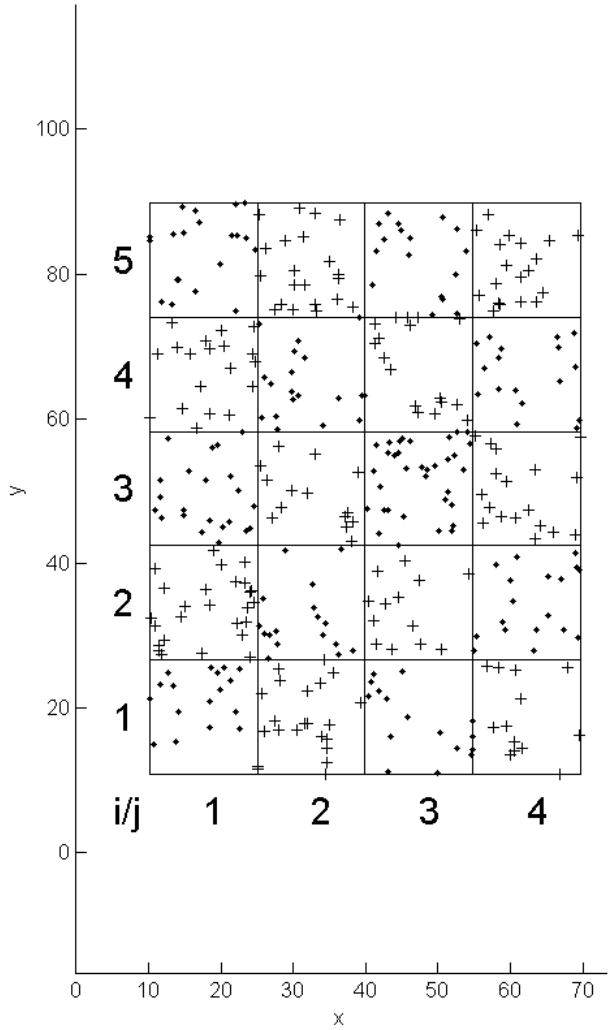
Typy vzorkování:

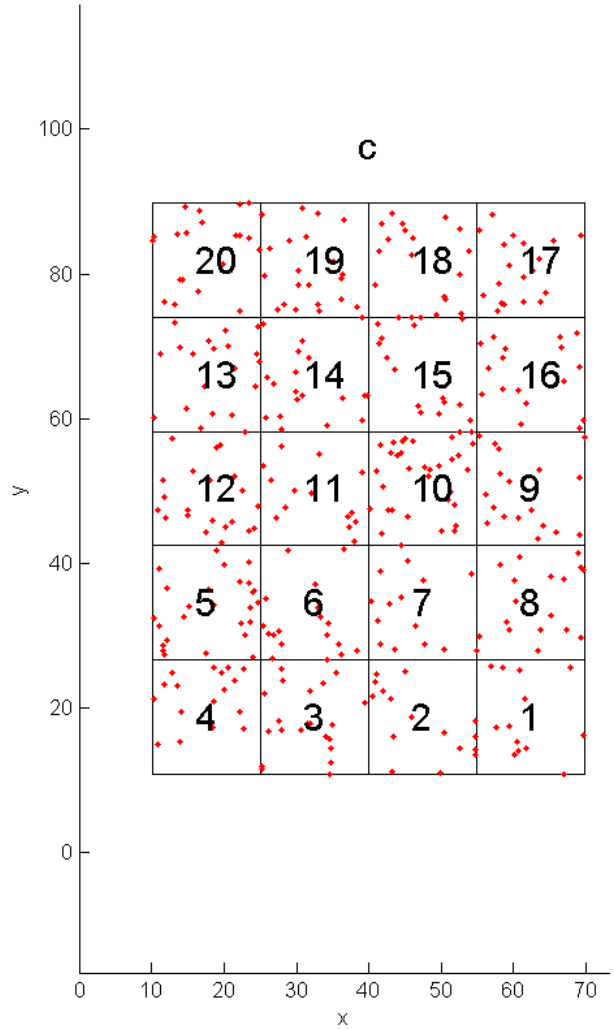
- *Jednoduché vzorkování*
  Všechny prvky vstupní množiny mají stejnou pravděpodobnost pro zařazení do vzorku.
  Položky vybrány náhodně, nemusí být rovnoměrně rozloženy.
- *Systematické vzorkování*
  Výběr každé n-té položky, první položka vybrána náhodně.
  Výhodou lepší rozložení položek v souboru.
- *Shlukové vzorkování*
  Ze vstupní množiny vybrán náhodný / předem daný počet vzorků.
  Zpracovány všechny prvky každého vybraného vzorku.

Použití:

Třídící algoritmy, konvexní obálky, triangulace, hledání průsečíků.

29. Randomizované zpracování bodů: náhodný výběr ze shluků





30. Volba algoritmu

Při volbě algoritmu nutno zohlednit:

- Jaké jsou požadavky na vstupní / výstupní data?
- Jaká je velikost vstupních / výstupních dat, tj. pro jaká n problém řeším?
- Požaduji exaktní / přibližné řešení? Jak jsem schopen ověřit kvalitu přibližného řešení?
- Je problém řešitelný pro libovolnou konfiguraci vstupních dat?
- Existují speciální případy, které musí být řešeny zvlášť nebo je lze zanedbat?
- Lze problém efektivně převést na jiný problém?
- Jak má být problém rychle vyřešen, tj. on-line, nebo mohu čekat sekundy, minuty, hodiny či dny?
- Jak dlouho má vývoj aplikace trvat, tj. kolik času a financí je do řešení problému možno investovat?
- O jaký typ problému se jedná, nepatří mezi NP problémy?
- Lze urychlit / vylepšit řešení volbou vhodných datových struktur?
- Lze urychlit / vylepšit řešení předzpracováním dat?