Gymnázium Evolution Jižní Město

Eric Dusart

Polytop maximální dimenze a minimálního obvodu s vrcholy v dané množině bodů

Ročníková práce

Školitel práce:

Adam Klepáč

Školní rok: 2023/2024

Poděkování

Děkuji bohovi Adamovi. KLEPY = BŮH!!!

Prohlášení

V Praze dne 23. dubna 2024 Eric Dusart

Abstrakt

Klíčová slova: graf, polytop, algoritmus

Abstract

Keywords: graph, polytope, algorithm

Obsah

Použitá notace

Symbol	Význam
$\mathbb{R}^+$	$\{x \in \mathbb{R} \mid x \geq 0\}$.
$\binom{n}{k}$	Kombinační číslo: $\frac{n!}{k!(n-k)!}$.
$p(a,b,c)$	Obvod trojúhelníku a, b, c.
$\Delta(a,b,c)$	Trojúhelník a, b, c.
$d(a,b)$	Vzdálenost bodu a od b.
$w(a,\ldots,n)$	Váha cesty z a do n.
min, max	Funkce minimum a maximum.
$\#V$	Velikost množiny V.
${}_n x_k$	n-tá souřadnice k-tého bodu

Základní definice

Definice 1 (Polytop). *Polytop* dimenze $n \in \mathbb{N}$ je uzavřená podmnožina $P \subseteq \mathbb{R}^n$ definovaná induktivně:

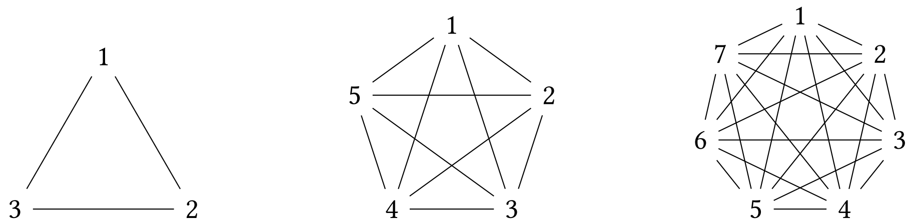
- Polytop dimenze 1 je úsečka.
- Polytop dimenze n je slepením polytopů dimenze $n-1$, jež spolu mohou sdílet stěny libovolné dimenze, kde *stěnou* polytopu rozumíme jeho libovolnou podmnožinu jsoucí rovněž polytopem. Zároveň neexistuje nadrovina (podprostor dimenze $n-1$), která by obsahovala všechny jeho vrcholy. [Ada24]

Definice 2 (Bod). Bod je uspořádaná n-tice $({}_1x, {}_2x, \ldots, {}_nx) \in \mathbb{R}^n$. Ve $2D$ budu používat značení $a = (a_x, a_y)$.

Definice 3 (Vzdálenost). Zobrazení $d : \mathbb{R}^n \times \mathbb{R}^n \longrightarrow \mathbb{R}^+$ nám určí vzdálenost dvou bodů $u, v \in \mathbb{R}^n$ podle předpisu $d(u, v) \coloneqq \sqrt{({}_1v - {}_1u)^2 + ({}_2v - {}_2u)^2 + \ldots + ({}_nv - {}_nu)^2}$.

Definice 4 (Ohodnocený graf). $G = (V, E, w)$ je ohodnocený graf, kde V je množina vrcholů, E je množina dvouprvkových podmnožin $E \subseteq \binom{V}{2}$ a w je libovolné zobrazení $E \longrightarrow \mathbb{R}^+$, které hranám přiřazuje jejich váhu.

Definice 5 (Úplný ohodnocený graf). Úplný ohodnocený graf $G = (V, E, w)$ má každé dva vrcholy spojeny hranou, neboli $E = \binom{V}{2}$. Takový graf můžeme také zapsat jako $K_n \coloneqq (V, \binom{V}{2}, w)$.



Obrázek 1: Úplné grafy K_3, K_5 a K_7

Definice 6 (Podgraf). Graf $H = (V', E')$ je podgraf grafu $G = (V, E)$, pokud $V' \subseteq V$ a $E' \subseteq E \cap \binom{V'}{2}$.

Definice 7 (Cesta). Cestou v grafu nazveme posloupnost **různých** vrcholů $v_1, \ldots, v_n$, pokud $\forall i \in \{1, \ldots, n-1\}$ platí $\{v_i, v_{i+1}\} \in E$.

Definice 8 (Váha cesty). Pokud cestu tvoří posloupnost vrcholů $v_1, \ldots, v_n$, tak váha cesty je rovna

$$w(v_1, \ldots, v_n) = \sum_{i=1}^{n-1} w(v_i, v_{i+1}).$$

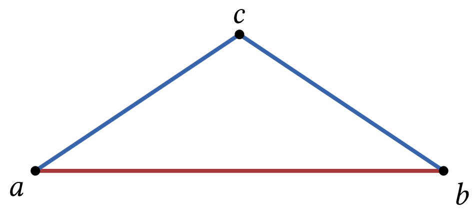
Definice 9 (Cyklus). Cyklus je posloupnost vrcholů $v_1, \ldots, v_n, v_1$, kde $v_1, \ldots, v_n$ je cesta a $\{v_1, v_n\}$ je hrana v množině hran E.

Definice 10 (Váha cyklu). Pokud cyklus tvoří posloupnost vrcholů $v_1, \ldots, v_n$, tak váha cyklu je rovna

$$w(v_1, \ldots, v_n, v_1) = d(v_1, v_n) + \sum_{i=1}^{n-1} w(v_i, v_{i+1}).$$

Definice 11 (Soused). V grafu $G = (V, E, w)$ je vrchol u soused vrcholu v, pokud hrana $\{u, v\} \in E$.

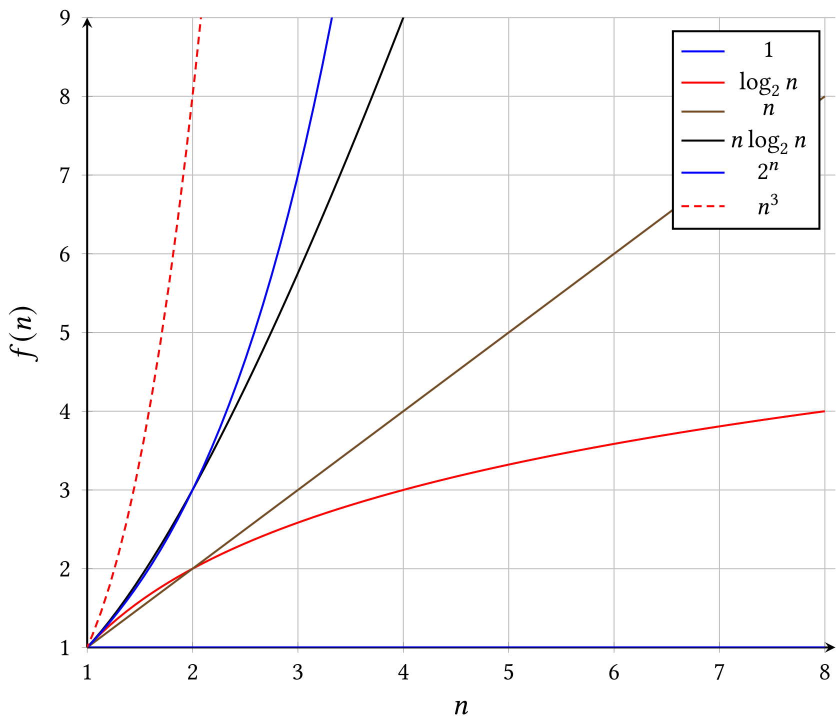
Definice 12 (Trojúhelníková nerovnost). Trojúhelníková nerovnost říká, že pro každé tři různé body a, b, c platí $d(a, b) + d(c, b) \geq d(a, b)$, neboli vzdálenost mezi dvěma body je vždy menší nebo rovna součtu vzdáleností mezi těmito body a třetím bodem.



Obrázek 2: Trojúhelníková nerovnost

Definice 13 (Obvod trojúhelníku). Definujeme zobrazení $p : \mathbb{R} \to \mathbb{R}^+$ podle předpisu: $p(a, b, c) := d(a, b) + d(b, c) + d(a, c)$, které nám určuje obvod $\Delta(a, b, c)$.

Definice 14 (Big O notation). Vizte například [Bae19]. Krátce řečeno: notace $O(\ldots)$ představuje počet kroků, kolik algoritmus musí udělat v nejhorším případě. O algoritmu můžeme říct, že má časovou složitost například $O(n^2)$.



Obrázek 3: Grafy různých funkcí

Úvod

Tohle tady nebude, to nějak změním...

Najít polytop maximální dimenze a minimálního obvodu v dané množině bodů je složité, hlavně při vyšším počtu bodů. Zdá se, že k vyřešení problému je třeba vyzkoušet všechny možnosti, ale tomu se budeme vyhýbat.

Začněme analýzou tohoto problému. Cílem je najít polytop maximální dimenze a minimálního obvodu v dané množině bodů. Ve 2D budeme problém řešit trochu jinak, než ve vyšších dimenzích.

Část I

Teoretická část

Kapitola 1

Problém v 1D

Problém v 1D je velice jednoduchý. Vstupem bude množina čísel $V \subset \mathbb{R}$ a výstupem bude nejkratší úsečka $\{a, b\}$, kde $a, b \in V$. Nejprve množinu čísel seřadíme pomocí algoritmu Quicksort (vizte [Hoa62]) a poté zkotrolujeme vzdálenost každých dvou po sobě jdoucích čísel a zapamatujeme si tu minimální, která je řešením tohoto problému.

1.1 Algoritmus

1. Nejprve body seřadíme pomocí algoritmu quicksort.
2. Projdeme všechny body $x_1 \dots x_n \in V$ a spočítáme vzdálenost všech po sobě jdoucích bodů: $d(x_i, x_{i+1})$, kde $i \in \{1, ..., \#V - 1\}$. Tuto vzdálenost si uložíme, a pokud je menší, než ta doposud uložená, změníme ji.
3. Po tom, co projdeme všechny body, je uložená minimální vzdálenost řešením problému.

Tvrzení 1. *Algoritmus na hledání nejkratší úsečky je korektní.*

Důkaz. Algoritmus skončí, protože prochází konečnou množinu bodů a je korektní, protože spočítá všechny vzdálenosti po sobě jdoucích bodů a vybere tu minimální. □

Poznámka 1. Pseudokód je popis jednotlivých kroků v algoritmu s použitím základní logiky programovacích jazyků. Následuje náš algoritmus napsaný v pseudokódu.

Algoritmus 1: Algoritmus na hledání úsečky s minimální délkou.

input : množina čísel $V \subset \mathbb{R}$.
output: $a, b \in V, d(a, b)$

$points \leftarrow quicksort(V)$;
$shortest\_distance \leftarrow \infty$;
$closest\_points \leftarrow \emptyset$;
for $i \in \{1, \dots, \#V - 1\}$ **do**
  if $d(x_i, x_{i+1}) < shortest\_distance$ **then**
    $shortest\_distance \leftarrow d(x_i, x_{i+1})$;
    $closest\_points \leftarrow (x_i, x_{i+1})$;
return $closest\_points, shortest\_distance$

Kapitola 2

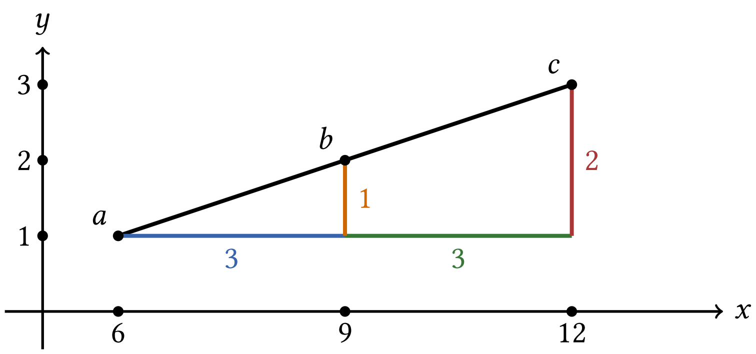
Problém ve 2D

Ve 2D budu hledat trojúhelník s minimálním obvodem. Prvním možným přístupem je algoritmus, který vyzkouší všechny trojúhelníky a vybere ten s minimálním obvodem. Ten ale vůbec není časově efektivní. Pokusím se najít, naprogramovat a dokázat správnost nějakého, který časově efektivní je.

2.1 Podobnost trojúhelníků

V sekci 2.3 Vám představím algoritmus na hledání trojúhelníku s minimálním obvodem. Velmi důležitou částí tohoto algoritmu je kontrolování, jestli body tvoří trojúhelník, to znamená, jestli body neleží na přímce. K tomuto máme následující tvrzení.

Tvrzení 2 (Podobnost trojúhelníků a body na jedné přímce). *Jsou dány 3 body ležící v $\mathbb{R}^2$: $a = (a_x, a_y)$, $b = (b_x, b_y)$, $c = (c_x, c_y)$. Tyto body jsou na přímce, pokud platí, že trojúhelníky $\Delta((a_x, a_y), (b_x, a_y), (b_x, b_y))$ a $\Delta((a_x, a_y), (c_x, a_y), (c_x, c_y))$ mají stejný poměr stran, proto platí rovnice: $(c_y - a_y)(b_x - a_x) = (b_y - a_y)(c_x - a_x)$.*



Obrázek 2.1: Příklad podobnosti trojúhelníků

Důkaz. Předpokládejme, že existuje lineární funkce $\mathfrak{f} : y = mx + k$, na které leží všechny tři body. Po dosazení bodů do rovnice nám vznikne soustava rovnic o třech neznámých.

$$a_y = ma_x + k$$
$$b_y = mb_x + k$$
$$c_y = mc_x + k$$

Poté odečteme třetí rovnici od první a první od druhé. Tím dostaneme:

$$\begin{aligned} a_y - c_y &= ma_x + k - mc_x - k \\ b_y - a_y &= mb_x + k - ma_x - k \end{aligned}$$

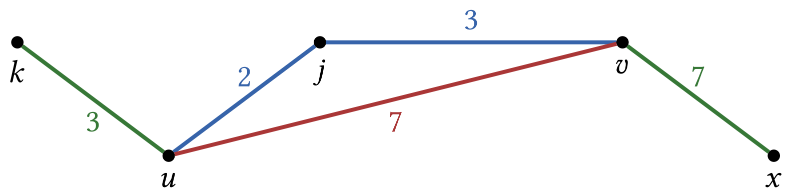
Vytknutím m dostaneme:

$$\begin{aligned} a_y - c_y &= m(a_x - c_x) \\ b_y - a_y &= m(b_x - a_x) \end{aligned}$$

A tím pádem $m_1 = (a_y - c_y)/(a_x - c_x)$ a $m_2 = (b_y - a_y)/(b_x - a_x)$. My ale víme, že m_1 a m_2 jsou stejná čísla, protože funkce $\mathfrak{f}$ protíná všechny tři body. Proto můžeme sestavit rovnici $(a_y - c_y)/(a_x - c_x) = (b_y - a_y)/(b_x - a_x)$. Nakonec můžeme rovnici upravit do tvaru $(a_y - c_y)(b_x - a_x) = (b_y - a_y)(a_x - c_x)$. Tímto je tvrzení 2 dokázáno. $\square$

2.2 Adaptace Dijkstrova algoritmu

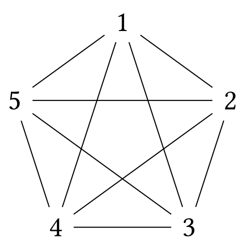
Velmi důležitou části algoritmu je Dijkstrův Algoritmus, který v ohodnoceném grafu dokáže najít cestu s minimální váhou mezi dvěma body v čase $O(\#V^2)$. [BS99] Obecně taková cesta může mít několik vrcholů, ale protože náš graf je převzatý z roviny, povede právě přes jeden vrchol a bude se skládat ze dvou hran. To vyplývá z trojúhelníkové nerovnosti. Kdyby náš graf nebyl převzatý z roviny, mohla nastat situace na obrázku 2.2. Dijkstrův algoritmus ale lehce upravíme, aby byl rychlejší. Jelikož Dijkstrův algoritmus neví, že náš graf je převzatý z roviny (to znamená, že v grafu platí trojúhelníková nerovnost), tak ho ukončíme dřív, aby nehledal cestu s minimální váhou přes více vrcholů. Tímto časovou náročnost zredukujeme na $O(\#V)$.



Obrázek 2.2: V obecném grafu nemusí platit trojúhelníková nerovnost.

2.2.1 Popis adaptace Dijkstrova algoritmu

Vstupem je bod a a cílem je bod b. Cestu s minimální váhou mezi těmito body můžeme najít všechny cesty, které vedou z a do b a vybrat tu nejkratší. V algoritmu 2 je adaptace Dijkstrova algoritmu funkce $path$.



Obrázek 2.3: Graf K_5 na kterém si můžete představit jak by tento algoritmus fungoval.

2.2.2 Popis Dijkstrova algoritmu

Dijkstrův algoritmus počítá také nejkratší vzdálenosti z bodu a do všech ostatních. Tyto vzdálenosti by teoreticky šly využít na zrychlení celého algoritmu, ale je otázkou co by bylo rychlejší: nechat doběhnout celý Dijkstrův algoritmus a pak z něho brát informace nebo využít adaptaci Dijkstrova algoritmu, která je rychlejší. Následuje popis Dijkstrova algoritmu.

1. Vytvoříme množinu všech nenavštívených bodů a vybereme startovní a cílový bod. Označíme všechny body nenavštívenými.

2. Každému bodu přiřadíme vzdálenost od počátečního bodu; prozatím na ∞. Vzdálenost počátečního bodu od sebe samého nastavíme na 0.

3. Přesuneme na nenavštívený bod s minimální vzdáleností od počátečního bodu (poprvé to bude počáteční bod). Tento bod označíme jako aktuální a začneme kontrolovat jeho sousedy. Je-li součet vzdálenosti od počátečního bodu do aktuálního s váhou hrany vedoucí k sousedu menší než vzdálenost, kterou má u sebe soused uloženou, změníme ji. Je třeba myslet na to, že když přepíšeme vzdálenost souseda od startovního bodu, souseda neoznačujeme za navštíveného. Počáteční bod odebereme z množiny nenavštívených bodů, až zkontrolujeme všechny jeho sousedy.

4. Čtvrtý bod opakujeme, dokud nevybereme za aktuální bod ten cílový. V tomto okamžiku jsme našli nejkratší cestu. Pokud bychom chtěli získat nejkratší vzdálenosti z počátečního bodu do všech ostatních, čtvrtý bod budeme opakovat dokud nebude množina nenavštívených bodů prázdná.

2.3 Algoritmus

Nechť V je množina bodů v rovině a pro každé dva body $u, v \in V$ označme $d(u, v)$ jejich vzájemnou vzdálenost. Množinu hran označíme $E = \binom{V}{2}$ a váhu, neboli ohodnocení, nám určuje zobrazení w dané předpisem $w(\{u, v\}) := d(u, v)$ pro $\forall(u, v) \in E$. Nyní můžeme definovat graf $G = (V, E, w)$. Tímto je příprava hotova. Následuje hledání cyklu délky tři s minimální váhou.

Náhodně vybereme jednu hranu $\{u, v\} \in E$ a odebereme ji z množiny hran E. V grafu bez hrany $\{u, v\}$ potřebujeme najít cestu s minimální váhou mezi body u a v. K tomuto použijeme adaptaci Dijkstrova algoritmu, která nám vrátí cestu délky dva. Cesta povede právě přes jeden vrchol, protože z trojúhelníkové nerovnosti je jasné, že pokud by cesta vedla přes více vrcholů, byla by delší. Bude skládat ze dvou hran: $\{u, j\}$ a $\{j, v\}$.

Teď musíme zkontrolovat, zda tato cesta s hranou $\{u, v\}$ opravdu tvoří trojúhelník, protože se může stát, že body u, j, v leží na jedné přímce. K tomu využijeme podobnost trojúhelníků, z které vyplývá, že body se nacházejí na přímce právě tehdy, pokud rovnice $(v_y - u_y)(j_x - u_x) = (j_y - u_y)(v_x - u_x)$ je pravdivá.

Poznámka 2 (jiný způsob). Předchozí způsob funguje perfektně pro počítače, ale pro člověka je v některých případech zbytečně komplikovaný. Tyto případy nastávají, když se body nacházejí na horizontálních nebo vertikálních přímkách. To jde poznat tak, že $u_y = j_y = v_y$ nebo $u_x = j_x = v_x$. V případě, že se body nacházejí na horizontálních přímce, musíme odebrat delší hranu. To uděláme tak, že si body uspořádáme (a přejmenujeme) tak, že $a_x < b_x < c_x$, nebo a_y, b_y, c_y (podle toho, jestli jsou na horizontální, nebo vertikální přímce), kde $\{a, b, c\} = \{u, j, v\}$. Z grafu pak musíme odebrat hranu $\{a, c\}$.

Pokud tyto body tvoří trojúhelník, pak tvoří cyklus délky tři. Pokud tento cyklus bude mít váhu menší než ten, který jsme doposud našli, uložíme jej a vrátíme hranu $\{u, v\}$ do množiny hran E. Tento postup opakujeme dokud nevyzkoušíme všechny hrany. Výsledkem bude trojúhelník s minimálním obvodem.

2.3.1 Algoritmus v pseudokódu

Algoritmus 2: Algoritmus na hledání cyklu délky tři.

input : množina bodů V v rovině, kde každý bod je reprezentován jako dvojice souřadnic (v_x, v_y).

output: cyklus délky tři a jeho váha.

```
Function path(start, end):
    min_path ← ∅, ∅;
    for x ∈ V do
        path ← w(start, x, end);
        if min_path[0] > path then
            min_path ← path, (start, x, end);
    return min_path;

for u ∈ V do
    for v ∈ V do
        w({u, v}) ← d(u, v);
E ← (V choose 2);
G ← (V, E, w);
min_Δ ← ∅;
for {u, v} ∈ E do
    E ← E \ {u, v};
    {u, j, v} ← path(V, u, v);
    if (v_y − u_y)(j_x − u_x) = (j_y − u_y)(v_x − u_x) then
        if d(u, j) > d(u, v) then
            E ← E \ {u, j};
        else if d(j, v) > d(u, v) then
            E ← E \ {v, j};
        else
            continue;
    else if p(u, j, v) < p(min_Δ) then
        min_Δ ← p(u, j, v);
    E ← E ∪ {u, v};
return min_Δ, p(min_Δ);
```

2.4 Důkaz algoritmu na hledání cyklu délky tři

Abychom mohli dokázat korektnost algoritmu, musíme dokázat, že algoritmus skončí a že je správný, to znamená, že dělá přesně co chceme. Dijkstrův algoritmus, který je součástí našeho algoritmu, dokazovat nebudu, protože důkaz je příliš dlouhý a je dostupný v literatuře.

Tvrzení 3. *Dijkstrův algoritmus je korektní.*

Důkaz. Vizte [BS99, s. 113]. □

Tvrzení 4. *Adaptace Dijkstrova algoritmu je korektní.*

Důkaz. Algoritmus je konečný, protože prochází konečnou množinu vrcholů a je správný, protože ze všech cest z a do b vybere tu s minimální váhou. □

Tvrzení 5. *Algoritmus na hledání cyklu délky tři je korektní.*

Důkaz. Konečnost algoritmu je zřejmá; jediným cyklem v algoritmu je procházení všech stran. Jelikož je množina hran konečná a víme, že Dijkstrův algoritmus je korektní, můžeme říci, že náš algoritmus skončí.

Správnost algoritmu dokážeme sporem. Budeme předpokládat, že existuje trojúhelník x, y, z, který má kratší obvod, než trojúhelník a, b, c, který našel algoritmus, neboli:

$$\exists \{x, y, z\} \subseteq V : p(x, y, z) < p(a, b, c) \mid a, b, c \leftarrow algoritmus.$$

Znamená to, že náš algoritmus vybral jednu stranu trojúhelníku s minimálním obvodem špatně, a tím vznikl trojúhelník s delším obvodem. Jelikož náš algoritmus prochází všechny hrany, tak tam nemohl vybrat špatnou hranu. Tím pádem špatnou hranu musel vybrat když vybíral zbylé dvě hrany. Ty ale vybral Dijsktrův algoritmus, který je korektní. Tím vznikl spor a korektnost algoritmu je dokázána. □

Kapitola 3

Zobecnění na n dimenzí

3.1 Pomocné definice

Tyto definice nám umožní vyřešit problém zobecněný na n dimenzí.

Definice 15 (Nadrovina). Nadrovina je podprostor $\mathbb{R}^n$ dimenze $n-1$.

Definice 16 (Lineární kombinace). Bod $a_1x_1+\cdots+a_kx_k$, kde $a_1,\dots,a_k \in \mathbb{R}$ je lineární kombinací bodů $x_1,\dots,x_k \in \mathbb{R}^n$. [Jin05, s. 67]

Definice 17 (Lineární závislost). Množina bodů $x_1,\dots,x_n \in \mathbb{R}$ se nazývá lineárně závislá, jestliže lze nějaký bod z této množiny vyjádřit jako lineární kombinaci ostatních bodů této množiny. [Jin05, s. 78]

Definice 18 (Afinní závislost). Body $x_1,\dots,x_k \in \mathbb{R}^n$ jsou afinně závislé právě tehdy, když jejich rozdíly $x_2-x_1, x_3-x_1,\dots,x_k-x_1$ jsou lineárně závislé. Geometricky to znamená, že body leží v jedné nadrovině. [Mat03, s. 4]

Definice 19 (Determinant matice). Determinant je zobrazení značené det, které posílá čtvercové matice na reálné číslo. Determinant jde interpretovat geometricky; lze chápat jako objem obecného n-rozměrného rovnoběžnostěnu. Důležitá vlastnost determinantu je, že řádky nebo sloupce matice A jsou lineárně závislé právě tehdy, když $det(A)=0$. [DJ19, s. 224; Mat03, s. 4] Pro hlubší pochopení determinantů vizte [Jin05, s. 164] nebo [DJ19, s. 187].

3.2 Řešení problému v nD

Problém v n dimenzích bude komplikovanější a časově náročnější. Z množiny bodů $V=\{x_1,\dots,x_k\} \in \mathbb{R}^n$ vytvoříme množinu všech $n+1$-prvkových podmnožin: $C=\binom{V}{n+1}$. Vybereme podmnožinu $\mathcal{P}=\{x_1,\dots,x_{n+1}\} \in C$, která tvoří polytop s minimálním obvodem. Obvod polytopu spočítáme následovně:

Hlavní otázkou je kolik hran opravdu tvoří obvod. (Například: ve 3D tělesová úhlopříčka krychle netvoří obvod.) Je jich přesně $\binom{n+1}{2}$. Toto číslo pochází ze vzorce $\binom{n+1}{m+1}$ [Cox73, s. 120], který znamená počet m-dimenzionálních podpolytopů (dále $\mathcal{P}'_m$) v n-rozměrném polytopu ($\mathcal{P}_n$) za předpokladu, že $\mathcal{P}_n$ má minimální počet vrcholů. To je $n+1$. Když víme že počet vnějších hran v polytopu je $\binom{n+1}{2}$, musíme zjistit které hrany z celkové množiny hran to jsou. Jsou to všechny, protože náš polytop má přesně $n+1$ vrcholů. Proto můžeme místo $n+1$ v $\binom{n+1}{2}$ dosadit množinu vrcholů polytopu ($\mathcal{P} = \{x_1, \ldots, x_{n+1}\}$) a tím dostaneme $\binom{\mathcal{P}}{2}$, což jsou všechny dvouprvkové podmnožiny z $\mathcal{P}$, neboli všechny hrany polytopu. Obvod tedy spočítáme následovně:

$$\rho(\mathcal{P}) = \sum_{i=1}^{n} \sum_{j=i+1}^{n+1} d(x_i, x_j).$$

Teď, když jsme vybrali polytop s minimálním obvodem, musíme ověřit, že jeho vrcholy opravdu tvoří polytop maximální dimenze (jako ve 2D, kde jsme ověřovali, jestli body neleží na přímce).

Polytop maximální dimenze se vyznačuje tím, že jeho body neleží ve stejné nadrovině. Z definice 18 víme, že pokud rozdíly bodů jsou lineárně závislé, tak body v jedné nadrovině leží. Z toho vyplývá, že chceme, aby body nebyly afinně závislé, neboli aby tvořily polytop maximální dimenze.

Poznámka 3. Značení ${}_n x_k$ znamená n-tá souřadnice k-tého bodu, například ${}_2 x_3$ je druhá souřadnice třetího bodu, tudíž ${}_n x_k - {}_n x_1$ je rozdíl n-té souřadnice bodu x_k a x_1.

Abychom zjistili, zda jsou body afinně závislé, musíme spočítat jejich rozdíly a zjistit, jestli jsou lineárně závislé. Od bodů $x_2, \ldots, x_{n+1} \in \mathbb{R}^n$ odečteme první bod x_1. Tím nám vzniknou rozdíly $x_2 - x_1, x_3 - x_1, \ldots, x_{n+1} - x_1$, které umístíme do matice A.

$$A = \begin{pmatrix} x_2 - x_1 \\ x_3 - x_1 \\ \vdots \\ x_k - x_1 \end{pmatrix} = \begin{pmatrix} {}_1x_2 - {}_1x_1 & {}_2x_2 - {}_2x_1 & \cdots & {}_nx_2 - {}_nx_1 \\ {}_1x_3 - {}_1x_1 & {}_2x_3 - {}_2x_1 & \cdots & {}_nx_3 - {}_nx_1 \\ \vdots & \vdots & \ddots & \vdots \\ {}_1x_{n+1} - {}_1x_1 & {}_2x_{n+1} - {}_2x_1 & \cdots & {}_nx_{n+1} - {}_nx_1 \end{pmatrix}$$

Teď, když máme matici, vypočítáme $det(A)$. Podle definice 19, pokud vyjde determinant 0, znamená to, že pak rozdíly bodů jsou lineárně závislé. Z definice 18 ale víme, že pokud tyto rozdíly jsou lineárně závislé, pak body jsou afinně závislé, což znamená, že leží ve stejné nadrovině. Pokud body leží ve stejné nadrovině, musíme tento polytop odebrat z množiny všech polytopů a zopakovat celý tenhle postup pro nový polytop s minimálním obvodem.

Tímto jsme vyřešili problém v n dimenzích, kde $n \in \mathbb{N}, n > 2$.

3.3 Algoritmus v nD

V této sekci Vám představím algoritmus, který řeší problém v n dimenzích. Součástí algoritmu není algoritmus na počítání determinantu matice, protože existují různé algoritmy, a protože v sekci 4.4, kde jsem algoritmus naprogramoval, používám knihovnu NumPy, která je optimalizovaná na matematické výpočty.

Algoritmus dostane množinu bodů V. Dimenzi n algoritmus zjistí podle počtu souřadnic jednoho bodu. Komentář v algoritmu označím jako // komentář.

Algoritmus 3: Algoritmus na hledání polytopu maximální dimenze s minimálním obvodem.

input : $V = \{x_1, \dots, x_k\} \in \mathbb{R}$.
output: body, které tvoří polytop maximální dimenze s minimálním obvodem.

Function $\rho(\mathcal{P})$:
  $perimeter \leftarrow \sum_{i=1}^{n} \sum_{j=i+1}^{n+1} d(x_i, x_j)$;
  return $perimeter$;

$n \leftarrow \#x \in V$;
$S \leftarrow \binom{V}{n+1}$;
$C \leftarrow []$;
for $\mathcal{P} \in S$ **do**
  $C.append(\mathcal{P}, \rho(\mathcal{P}))$;
$C.sort()$ // Od nejmenšího po největší podle $\rho(\mathcal{P})$;
$min_{\mathcal{P}} \leftarrow \emptyset$;
while C **do**
  $\mathcal{P}, \rho(\mathcal{P}) \leftarrow C[0]$;
  $x_1, \dots x_{n+1} \leftarrow \mathcal{P}$;
  $$A \leftarrow \begin{pmatrix} {}_1x_2 - {}_1x_1 & {}_2x_2 - {}_2x_1 & \cdots & {}_nx_2 - {}_nx_1 \\ {}_1x_3 - {}_1x_1 & {}_2x_3 - {}_2x_1 & \cdots & {}_nx_3 - {}_nx_1 \\ \vdots & \vdots & \ddots & \vdots \\ {}_1x_{n+1} - {}_1x_1 & {}_2x_{n+1} - {}_2x_1 & \cdots & {}_nx_{n+1} - {}_nx_1 \end{pmatrix};$$
  if $det(A)\ ! = 0$ **then**
    $min_{\mathcal{P}} \leftarrow \mathcal{P}$;
    return $min_{\mathcal{P}}, \rho(\mathcal{P})$;
  else
    $C.remove(C[0])$;
return $\emptyset$;

Část II

Praktická část

Kapitola 4

Programování algoritmů

Moje praktická část bude programování problému, o kterým jsem psal v teoretické části. K programování použiji programovací jazyk python, ve kterém už umím programovat. Python nabízí mnoho různých knihoven, které dokážou velmi ušetřit práci. Knihovna je sbírka funkcí a metod, které jsou předem napsané a můžeme je získat jako doplněk pro snadnější vývoj softwaru. Ve své práci použiji knihovnu networkx, která je optimalizovaná pro práci s grafy. Použiji ji, protože nabízí lehké ukládání a získávání dat z grafu.

4.1 Dijkstrův algortimus

Jelikož Dijkstrův algoritmus je součástí našeho algoritmu na řešení problému ve 2D, tak ho budu muset naprogramovat. Knihovna networkx má také k dispozici Dijkstrův algoritmus, ale pro náš problém je potřeba lehce modifikovaný; musí se zastavit v ten moment, kdy se vybere cílový bod za aktuální bod.

```
def dijkstra_triangle(G:Graph, start, end):
    Q = set()
    distances = {}
    middle_point = {}
    for vertex in G:
        distances[vertex] = float('inf')
        middle_point[vertex] = None
        Q.add(vertex)

    distances[start] = 0
    while Q:
        actual = min(Q, key=lambda vertex: distances[vertex])
        if actual == end:
            return distances[end], [start, middle_point[end], end]
        Q.remove(actual)
        neighbors = Q
        if actual == start:
            neighbors.remove(end)
        for neighbor in neighbors:
            calculated_distance = distances[actual] + G.get_edge_data(
                actual, neighbor)['weight']
            if calculated_distance < distances[neighbor]:
                distances[neighbor] = calculated_distance
                middle_point[neighbor] = actual
    return distances[end], [start, middle_point[end], end]
```

4.2 Program v 1D

4.3 Program ve 2D

Ve 2D je program jednodušší, než ve více dimenzích, protože hledáme pouze 3 body, které jsou u sebe nejblíž a musíme kontrolovat pouze jestli nejsou na přímce. Ve vyšších dimenzích budeme muset kontrolovat, jestli body nejsou ve stejných rovinách nebo i nadrovinách. Tento program dostane vstup množinu bodů, kde body budou v $\mathbb{R}^2$ a výstupem budou 3 body, které tvoří trojúhelník s minimálním obvodem.

V tomto programu použiji Dijkstrův algoritmus, který jsem naprogramoval v sekci 4.1.

tohle jde udelat jeste jinak at tam neni to edges v tom algoritmu

```
import dijkstra

def find_shortest_path(G:Graph, edges):
    shortest_path = None
    smallest_triangle = float('inf')
    for edge in G.edges:
        G.remove_edge(*edge)
        path_length, path = dijkstra.dijkstra_triangle(G, *edge)
        triangle = edges[edge] + path_length
        if triangle < smallest_triangle:
            shortest_path = path
            smallest_triangle = triangle
        G.add_edge(*edge, weight=edges[edge])
    return shortest_path
```

4.4 Program v nD

Závěr

Literatura

[Hoa62] C. A. R. Hoare. „Quicksort“. In: *The Computer Journal* 5.1 (led. 1962), s. 10–16. ISSN: 0010-4620. DOI: 10.1093/comjnl/5.1.10. eprint: https://academic.oup.com/comjnl/article-pdf/5/1/10/1111445/050010.pdf. URL: https://doi.org/10.1093/comjnl/5.1.10 (cit. 09. 03. 2024).

[Cox73] H.S.M. Coxeter. *Regular Polytopes*. third. Dover Books on Advanced Mathematics. Dover Publications, 1973. ISBN: 978-0-486-61480-9. URL: https://books.google.cz/books?id=iWvXsVInpgMC (cit. 22. 04. 2024).

[BS99] Holger Benl a Helmut Schwichtenberg. „Formal Correctness Proofs of Functional Programs: Dijkstra's Algorithm, a Case Study“. In: *Computational Logic*. Ed. Ulrich Berger a Helmut Schwichtenberg. Berlin, Heidelberg: Springer, 1999, s. 113–126. ISBN: 978-3-642-58622-4. DOI: 10.1007/978-3-642-58622-4_4.

[Mat03] Jiří Matoušek. „Convexity“. In: *Introduction to Discrete Geometry*. Department of Applied Mathematics, Charles University Malostranské nám. 25, 118 00 Praha 1, Czech Republic: Springer, zář. 2003, s. 1–16. ISBN: 978-1-4613-0039-7. URL: https://kam.mff.cuni.cz/~matousek/kvg1-tb.pdf (cit. 21. 04. 2024).

[Jin05] Jindřich Bečvář. *Lineární algebra*. third. Sokolovská 83, 186 75 Praha 8: Matfyzpress, 2005. 436 s. ISBN: 80-86732-57-6. URL: https://www.karlin.mff.cuni.cz/~halas/becvar_-_linearni_algebra.pdf (cit. 20. 04. 2024).

[Bae19] Sammie Bae. „Big-O Notation“. In: *JavaScript Data Structures and Algorithms*. Berkeley, CA: Apress, 2019, s. 362. ISBN: 978-1-4842-3988-9. DOI: 10.1007/978-1-4842-3988-9_1. URL: https://doi.org/10.1007/978-1-4842-3988-9_1 (cit. 20. 04. 2024).

[DJ19] Dan Margalit a Joseph Rabinoff. *Interactive Linear Algebra*. Ve spol. s Larry Rolen. master version. Georgia Institute of Technology, 3. čvn. 2019. 455 s. URL: https://textbooks.math.gatech.edu/ila/ila.pdf (cit. 20. 04. 2024).

[Ada24] Adam Klepáč. *Definice polytopu*. 9. led. 2024.