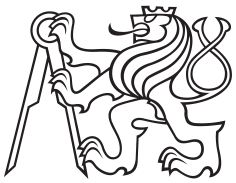


České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačů

Optimalizace na nekonvexní obálce

Tomáš Hořovský

Školitel: Ing. Jan Kubr, Ph.D.
Květen 2021

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Hořovský** Jméno: **Tomáš** Osobní číslo: **456997**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Specializace: **Kybernetická bezpečnost**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Optimalizace na nekonvexní obálce

Název diplomové práce anglicky:

Non Convex Hull Optimalization

Pokyny pro vypracování:

Prostudujte problematiku směrování v senzorových bezdrátových sítích s využitím distribuovaného tvarování vyzařovacího výkonu.
Navrhněte metody výběru prvků, které se budou účastnit komunikace s využitím distribuovaného tvarování vyzařovacího výkonu. Zaměřte se na výběr prvků na nekonvexní obálce grafu.
Efektivitu navržených metod experimentálně ověřte pomocí simulace. Výsledky simulace zhodnoťte.

Seznam doporučené literatury:

Kubr. Směrování v bezdrátových Ad-Hoc a senzorových sítích. Praha,
2017. Dizertační práce. České vysoké učení technické v Praze, Fakulta
elektrotechnická, Katedra telekomunikační techniky.
C.A. Balanis. Antenna Theory: Analysis and Design. Wiley, 2015.
Vaclav Skala, Michal Smolik, and Zuzana Majdisova. Reducing the Number of
Points on the Convex Hull Calculation Using the Polar Space Subdivision in E2.
In Graphics, Patterns and Images (SIBGRAPI), 2016 29th SIBGRAPI Conference
on, pages 40–47. IEEE, 2016.

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Jan Kubr, Ph.D., katedra počítačové grafiky a interakce FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **12.08.2021** Termín odevzdání diplomové práce: **04.01.2022**

Platnost zadání diplomové práce: **19.02.2023**

Ing. Jan Kubr, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

. | Datum převzetí zadání | Podpis studenta

Poděkování

Děkuji vedoucímu práce, Ing. Janu Kubrovi, Ph.D. za podporu a jeho připomínky při tvorbě této práce. Také děkuji svojí rodině za neutichající podporu v strastiplných časech.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípdavě vysokoškolských závěrečných prací.

V Praze dne 4.1.2022

.....
Tomáš Hořovský

Abstrakt

Cílem tohoto projektu je navrhnout a vytvořit platformu pro implementaci a testování algoritmů pro tvorbu obálek anténních polí v 2D prostoru. Platforma by měla umožňovat generování náhodných testovacích případů, vyhodnocení síly signálu mezi anténními poli a zobrazení a ohodnocení vypočítaných obálek. Dalším cílem je navrhnout takový algoritmus a otestovat jeho vlivu na výběr prvků pro beamforming pomocí vyvinuté platformy.

Klíčová slova: C++, anténní řady, sensor, konvexní obálka, konkávní obálka, grafika

Školitel: Ing. Jan Kubr, Ph.D. Praha, Resslova 307/9 (vstup z Karlova náměstí 13), místnost: E-414

Abstract

The goal of this project was to design and develop a platform for implementation and testing of algorithms for construction of hulls of antenna arrays in 2D space as well as testing the influence of hulls on the selection of elements used for beamforming. The platform allows for generating random test cases, evaluation of signal strength between antenna arrays and visualisation and evaluation of constructed hulls. Another goal was to design, implement and test such algorithm using the before mentioned platform.

Keywords: C++, antenna arrays, sensor, convex hull, concave hull, graphics

Title translation: Optimisation on non-convex hull

Obsah

Kapitola 1
Úvod a definice

Senzorové sítě mají využití v mnoha různých oblastech od sledování životního prostředí přes internet věcí a vesmírné technologie až po vojenské účely. Jsou složené z mnoha jednoduchých autonomních modulů. Kromě požadavků daných konkrétní aplikací senzorové sítě musí často tyto senzory umět komunikovat mezi sebou, případně s jinými senzorovými sítěmi k čemuž používají různých druhů antén.

Při spojení mezi různými senzorovými sítěmi může nastat situace kdy jsou od sebe senzorové sítě vzdáleny více než je dosah jednotlivých antén. Tímto problémem se zabývá ve své dizertační práci [11] vedoucí této práce, Ing. Jan Kubr, Ph.D. Právě na jeho dizertační práci navazuji touto prací.

Dílčím problémem navázání spojení mezi senzorovými sítěmi je nalezení dvojice antén, které budou dohromady mít dostatečný dosah pro navázání spojení. V této práci se zabývám tvorbou nástroje pro vývoj a testování algoritmů, které tento úkol ulehčí. Kromě toho se také zabývám návrhem, implementací a otestováním jednoho takového algoritmu. Po dohodě s vedoucím práce jsem se s ohledem na své zaměření převážně zabýval vývojem platformy umožňující vývoj takových algoritmů.

Problematika anténních polí a senzorových sítí pouze definuje úlohu, avšak její hloubkový rozbor není předmětem této práce a proto zde bude popsána jen povrchně.

1.1 Senzorová síť

Bezdrátová senzorová síť je množina modulů, kde každý je vybavený anténou a jednoduchou ovládací jednotkou. Senzorová síť může mít centrální prvek, který shromažďuje a zpracovává data z modulů a celou senzorovou síť řídí. Množinu modulů v senzorové síti si můžeme představit jako neorientovaný graf, kde uzly jsou jednotlivé moduly a hrany mezi uzly zaznamenávají schopnost dvou antén navázat přímé spojení. Zpravidla pro každé dva uzly v senzorové síti musí existovat cesta přímých spojení. Tato vlastnost se dá v grafu jednoduše popsat pomocí konceptu komponenty souvislosti.

Jak bylo řečeno dříve, senzorové sítě mají mnohá využití v praxi. Jednou z nejznámnějších aplikací senzorových sítí je například projekt pro sledování mikroklimatu v hnízdech buřňáčků malých [20]. V tomto případě bylo v roce

2002 nainstalováno 14 senzorů (později rozšířeno až na 100 senzorů), které snímaly data o místním mikroklimatu. Data byla v reálném čase odesílána do základové stanice k vyhodnocení.

Dalším projektem, kde se senzorové sítě využívají je CodeBlue [13]. V tomto projektu se vybavovaly oběti nehod senzory, které monitorovaly různé životní funkce. Lékaři v nemocnicích mohli sledovat zdravotní stav pacientů a věděli včas o krizových situacích. Tyto sítě obsahovaly nejen senzory pacientů, ale i osobní počítače a komunikátory zdravotníků. V tomto případě senzorové sítě obsluhovaly velké oblasti s malou hustotou modulů.

1.2 Anténní pole a beamforming

Senzorové sítě nemusí být vždy souvislé. Může nastat situace, kdy jsou vzdálenosti mezi prvky uvnitř senzorové sítě větší, než je vysílací dosah jednoho uzlu. Také může nastat situace, kdy se senzorová síť snaží kontaktovat jinout senzorovou síť, která je mimo běžný dosah. Právě tímto problémem se zabývá tato práce.

Existuje více způsobů překlenutí větších vzdáleností mezi vysílači a přijímači. Jedním z nich je použití směrových antén. Tyto antény mohou v některém směru vyzařovat energii s větším efektivním dosahem než všesměrové antény. Tato práce se směrovými anténami nezabývá. Předpokládám, že jsou senzorové sítě vybaveny pouze všesměrovými anténami.

Všesměrové antény mohou také simulovat směrovou anténu pomocí interference signálu z několika všesměrových antén. Takovéto kombinace antén nazýváme Anténními poli.

Anténní pole je množina antén, ze kterých jsou současně vysílané signály kombinovány do jednoho paprsku. Anténní pole obsahuje dvě a více antén, které mohou mít různé vlastnosti. Využívají se pro zesílení signálu v určitém směru, určení směru příchozích signálů nebo také pro filtraci rušení. Anténní pole jako celek mohou mít vyšší dosah, než mají jednotlivé antény, ze kterých se skládá. K tomu lze využít techniky zvané **beamforming**.

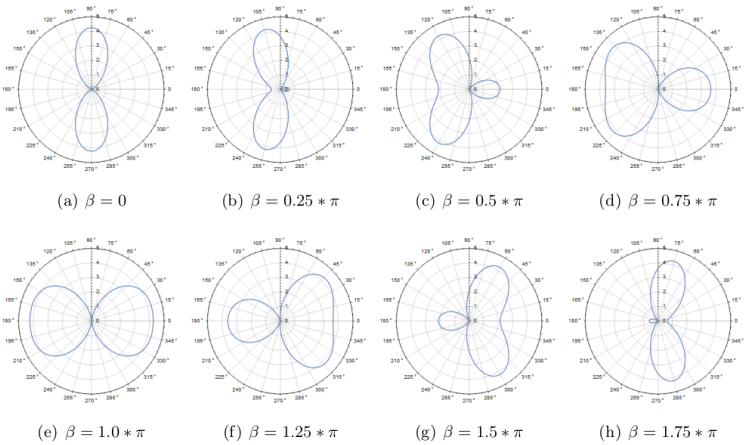
Technika beamformingu dokáže zkombinovat několik současně vysílaných všesměrových signálů do jednoho směrového signálu, který může mít vyšší dosah, než měly původní signály. Dříve využívaný analogový beamforming nebyl příliš ekonomický, jelikož vyžadoval drahé komponenty a byl nestabilní. S nástupem digitální techniky přišla i technika digitálního beamformingu, který se dá lépe ovládat a je ekonomicky dostupnější.

Obou typů beamformingu lze dosáhnout dvěma různými postupy.

- Technika časového zpoždění, která formuje signál vysíláním stejného signálu s velmi přesným časovým odstupem. Tato technika je nezávislá na frekvenci a šířce pásma, ale výhody jsou vyváženy nutností provádět vysoce přesnou (a tedy nákladnou) synchronizaci.

- Technika fázového posunu mezi vysílanými signály dosahuje stejného efektu, ovšem nevyžaduje takovou úroveň přesnosti při synchronizaci.

Podobně jako u směrových antén lze i pomocí beamformingu tvarovat vyzařovaný signál do jednoho či více směrových laloků. Efekt této techniky pro různé hodnoty fázového posunu β jsou zřejmé z obrázku 1.1.



(a) $\beta = 0$ (b) $\beta = 0.25 * \pi$ (c) $\beta = 0.5 * \pi$ (d) $\beta = 0.75 * \pi$

(e) $\beta = 1.0 * \pi$ (f) $\beta = 1.25 * \pi$ (g) $\beta = 1.5 * \pi$ (h) $\beta = 1.75 * \pi$

Obrázek 1.1: Příklad vyzařovací charakteristiky při použití beamformingu pro různé hodnoty fázového posunu β
Zdroj: Obrázek převzat z [11]

V dizertační práci [11], ze které jsem čerpal je dopodrobna popsán efekt fázového posunu a změny tvaru a dosahu vyzařovaného signálu pro anténní pole. Ze závěrů práce je zřejmé, že pro spojování dvou senzorových sítí je dostatečné zvolit do anténního pole dvě antény. Při použití většího množství antén není přidaná hodnota dostatečná k ospravedlnění výrazně vyšší složitosti hledání správné kombinace.

1.2.1 Úloha

Mějme dvě senzorové sítě, které nazveme **vysílací** a **přijímací**. Pro jednoduchost jsou moduly v těchto sítích, vybaveny stejnými všesměrovými anténami a jednotlivé antény jsou rozloženy v dvojrozměrné ploše. Dalším významným zjednodušením je zanedbání efektu odrazu signálu od zemského povrchu, který může velmi ovlivnit sílu signálu.

Definice 1. *S je množina antén vysílací senzorové sítě.*

Definice 2. *R je množina antén přijímací senzorové sítě.*

Vysílací senzorová síť se aktivně pokouší navázat spojení s přijímací senzorovou sítí.

Definice 3. *P_t je výkon, který jde do vysílače a P_r je očekávaný výkon na přijímači*

Definice 4. *$P_r^{\min}$ je minimální výkon na přijímači, který je potřebný pro úspěšný přenos.*

Pokud $\exists a \in S, \exists c \in R$ takové, že při vysílání signálu z a je $P_r(c) \geq P_r^{\min}$, pak je úloha navázání spojení triviální. Úloha je také triviální, pokud vysílací

senzorová síť zná polohu antén přijímací senzorové sítě. V takovém případě může senzorová síť rychlým výpočtem získat optimální dvojici vysílacích antén, ze kterých vytvoří anténní pole. V opačném případě je úloha nalezení takové dvojice antén netriviální a právě tímto problémem se zabývá tato práce.

Definice 5. *Sílu signálu na přijímači $c \in R$ při vysílání anténním polem z vysílačů $a \in S, b \in S$ označíme jako: $K(a, b, c, \beta)$, kde $0 \leq \beta < 2\pi$ je fázový posun mezi vysílačí a, b.*

Definice 6. *Maximální možnou sílu signálu na přijímači c, při vysílání anténním polem skládajícím se z vysílaču a, b označíme jako: $E(a, b, c) = \max_{0 \leq \beta < 2\pi} K(a, b, c, \beta)$.*

Definice 7. *Množinu všech řešení úlohy U definujeme jako:*

$$U(S, R) = \left\{(a, b, c) \mid a, b \in S, c \in R, E(a, b, c) \geq P_r^{\min}\right\} \tag{1.1}$$

Definice 8. *Optimální řešení u^* definujeme jako:*

$$u^* = \operatorname*{argmax}_{u \in U} E(u_a, u_b, u_c) \tag{1.2}$$

1.2.2 Výběr antén

Výběr optimální dvojice antén má v obecném případě asymptotickou složitost $O(n^2)$. V praxi je ovšem nezbytné vzít v úvahu i velikost konstant, které asymptotický výpočet opomíjí. V tomto případě je nutné počítat s vysokou časovou náročností hodnotícího kritéria pro dané anténní pole. Pozice prvků R není senzorové síti předem S známa a to značně komplikuje navázání spojení.

Vybraná dvojice antén a, b musí provést správnou modulaci signálu pomocí fázového posunu. Jedná se tedy o optimalizační úlohu o jedné proměnné. Kritérium této optimalizační úlohy je síla signálu na přijímači $E(a, b, c)$. Zpětnou vazbu o hodnotě kritéria dostanou přijímače až ve chvíli, kdy je hodnota kritéria vyšší než $P_r^{\min}$.

Z těchto faktů vyplývá, že nalezení optimálního fázového posunu je netriviální úloha, jejíž řešení je potenciálně časově náročné. Výběr vhodných kandidátů na dvojici vysílačů a a b je tedy kritický.

1.2.3 Cíl projektu

Analytické vyhodnocení kvality různých přístupů k výběru dvojice vysílačů a, b není jednoduché. Proto jsem se rozhodl pro hodnocení pomocí simulace. Jelikož jsem nenašel žádný existující systém, který by byl vhodný pro tyto účely, rozhodl jsem se vytvořit vlastní simulátor.

Vytvořený simulátor umožňuje uživateli zadat či vygenerovat množiny S, R. Nadále systém poskytuje uživateli rozhraní ve kterém může naprogramovat vlastní algoritmus pro tvorbu obálky S. Systém poté spočítá množinu řešení úlohy, zobrazí optimální řešení (řešení s nejyšší silou signálu na přijímacím uzlu) a spočítá různé statistické údaje, které se dají použít pro hodnocení

zvoleného algoritmu pro výpočet obálky. Systém by měl také uživateli umožnit využívat výpočetní knihovnu ve vlastních programech, což může být užitečné například pro zadávání dávkových výpočtů.

V rámci této práce jsem tento systém naimplementoval a nazval ho **Antenna Array Calculator**, zkráceně **AAC**.

Kapitola 2
Rozbor problematiky

V této kapitole se budu zabývat podrobnějším popisem nejdůležitějších řešených problémů, jako je výpočet síly signálu, výběr optimálního fázového posunu nebo rozbor výběru obálky a mnou navrhovaného řešení.

2.1 Výpočet síly signálu

Mějme dvě vysílací antény $a, b \in S$ a přijímací anténu $c \in R$. Potom pro fázový posun β mezi vysílanými signály můžeme spočítat sílu signálu $\vec{E_t}$ na anténě c.

$$K(a, b, c, \beta) = \left\| \vec{E_t}(c) \right\|$$

V dizertační práci [11] je vzorec 3.40, kterým lze $E_t(x)$ vypočítat:

$$\vec{E_t} = \sum_{i=1}^{N} \hat{a_\theta} \cdot j \cdot \eta \cdot \frac{k \cdot I_0 \cdot l_i}{4\pi r_i} \cdot e^{-j(kr_i + \beta_i)} \cdot \sin\theta \tag{2.1}$$

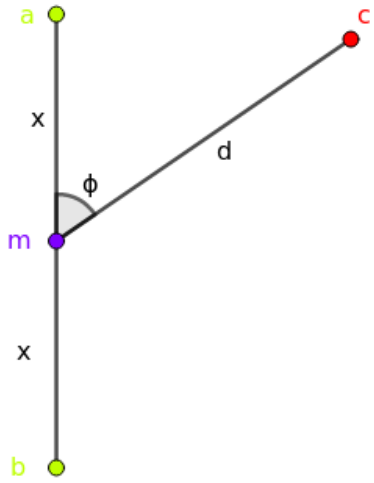
Tento vzorec je poněkud komplexní a proto jsem se rozhodl rozložit výpočet na několik dílčích kroků.

2.1.1 Rozklad

Mějme dvě vysílací antény $a, b \in S$, jejichž pozici určují vektory $\vec{a}, \vec{b}$ a přijímací anténu $c \in R$, s pozicí $\vec{c}$. Nejprve si připravíme nezbytné proměnné:

$$\vec{m} = \frac{\vec{a} + \vec{b}}{2} \qquad x = \|\vec{a} - \vec{m}\|$$
$$\vec{r_a} = \vec{c} - \vec{m} \qquad d = \|\vec{r_a}\|$$
$$\vec{b_a} = \vec{b} - \vec{m} \qquad \cos\phi = \frac{\vec{b_a} \cdot \vec{r_a}}{\left\|\vec{b_a}\right\| \cdot d}$$

Zde $\vec{m}$ je střed anténního pole, x je vzdálenost vysílačů od středu anténního pole, d je vzdálenost přijímače od středu anténního pole a ϕ je úhel ve kterém leží přijímač vůči spojnici vysílačů.



Obrázek 2.1: Nákres situace při výpočtu

Výrazným zjednodušením je předpoklad, že obě antény v anténním poli mají stejné parametry. Efekt těchto parametrů lze shrnout konstantou ψ:

$$\psi = i \cdot k \cdot I_0 \cdot l \cdot \sin\theta$$

kde $k = \frac{2\pi}{\lambda}$ je betafaktor, $l = \frac{\lambda}{2}$ je velikost dipólu, I_0 je proud na anténách a $i = \sqrt{-1}$.

Vzdálenosti obou antén od přijímače lze spočítat pomocí polárních souřadnic:

$$r_0 = \sqrt{(r\sin\theta\cos\phi + x)^2 + (r\sin\theta\sin\phi)^2 + (r\cos\theta)^2}$$
$$r_1 = \sqrt{(r\sin\theta\cos\phi - x)^2 + (r\sin\theta\sin\phi)^2 + (r\cos\theta)^2}$$

kde $\theta = \frac{\pi}{2}$ určuje, že se výpočet provádí v ploše.

Poté můžeme spočítat intenzitu elektrického pole na přijímači:

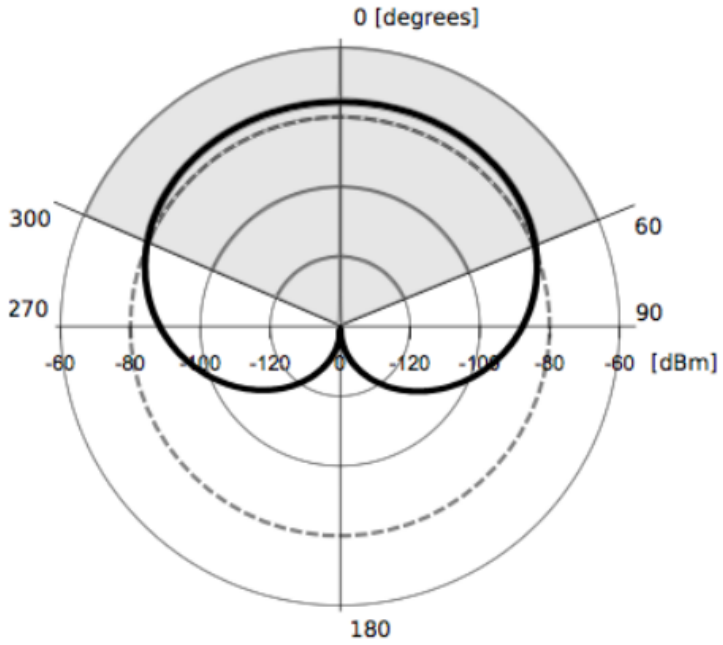
$$h_0 = \frac{\psi \cdot e^{-i\cdot(k\cdot r_0+\beta)}}{4\pi \cdot r_0}$$
$$h_1 = \frac{\psi \cdot e^{-i\cdot(k\cdot r_1)}}{4\pi \cdot r_1}$$
$$\vec{E_t} = \frac{1}{2} \cdot \eta \cdot |h_0 + h_1|^2$$

Zde $\eta = 120\pi$ je impedance volného prostoru a $\lambda = 0.125m$ je vlnová délka. Hodnoty konstant jsou převzaty z experimentů v dizertační práci [11].

2.1.2 Výběr optimálního fázového posunu

Výpočet hodnoty funkce $K(a, b, c, \beta)$ popsaný vzorcem 2.1 sice vypadá komplikovaně, ale jedná se o operaci, kterou počítač vyhodnotí asymptoticky v konstantním čase. Mnohem větší komplikací je optimalizace přes různé úhly β fázového posunu, která se provádí při výpočtu hodnoty funkce $E(a, b, c)$ (z definice 6).

V dizertační práci [11] je v kapitole 4.2.3 obrázek, který ilustruje sílu signálu na přijímači pro různé hodnoty fázového posunu β:



Obrázek 2.2: Přijímaný výkon v závislosti na fázovém posunu vysílání dvou vysílačů
Zdroj: Dizertační práce [11]

Na obrázku 2.2 je tlustou čarou znázorněna síla signálu na přijímači a citlivost přijímače je naznačena přerušovanou čarou. Z dat na tomto diagramu můžeme vyvodit několik předpokladů.

1. Kriteriální funkce pro maximalizaci je na intervalu $\beta \in (0, 2\pi)$ hladká, konkávní a má právě jedno maximum.

2. Velikost intervalu $\Gamma = (\beta_1, \beta_2)$, pro které je $K(a, b, c, \beta) \geq P_r^{\min}$, je závislá na vzdálenosti anténních polí. Na obrázku je interval $\Gamma = (-60, 60)$ stupňů.

3. V praxi stačí nalézt jakoukoliv hodnotu $\beta \in \Gamma$, ovšem **AAC** musí najít co nejlepší odhad pro optimální hodnotu β^*.

Nadále bududu předpokládat, že tyto předpoklady jsou splněné. Z těchto předpokladů plyne, že **AAC** nemůže použít algoritmus pro hledání optimálního fázového posunu z dříve zmíněné kapitoly 4.2.3. Proto jsem použil pro hledání β^* algoritmus Golden-section search který popíšu v kapitole 5 a jehož předpoklady se shodují s předpoklady z bodu 1. Tento algoritmus hledá extrém funkce na zvoleném intervalu s předem stanovenou přesností. Tuto přesnost jsem nastavil v souladu s omezeními zvoleného programovacího jazyka na $1 \cdot 10^{-6}$.

2.1.3 Dosah anténního pole

Z dizertační práce:

> "Z výše uvedených vzorců vyplývá, že v ideálním případě díky anténní řadě tvořené dvěma prvky je možné při stejném výkonu vkládaném do obou antén překonat až vzdálenost $d' = 2 \cdot d \cdot \sqrt{2}$, kde d je původní vzdálenost překlenutá původní anténou s původním výkonem. Pokud se původní výkon rozdělí mezi obě antény, čili použije se stejný výkon jako pro jednu anténu, je možné překlenout vzdálenost $d' = 2 \cdot d$." [11, Kap. 3.2, str.67]

Jinými slovy, maximální dosah anténního pole je až dvojnásobkem maximálního dosahu jeho vysílačů. Jelikož se v této práci zabývám pouze senzorovými sítěmi, které se skládají ze stejného typu antén, označím maximální vzdálenost, na kterou může jedna takováto anténa navázat spojení s jinou anténou jako $d_{\max}$.

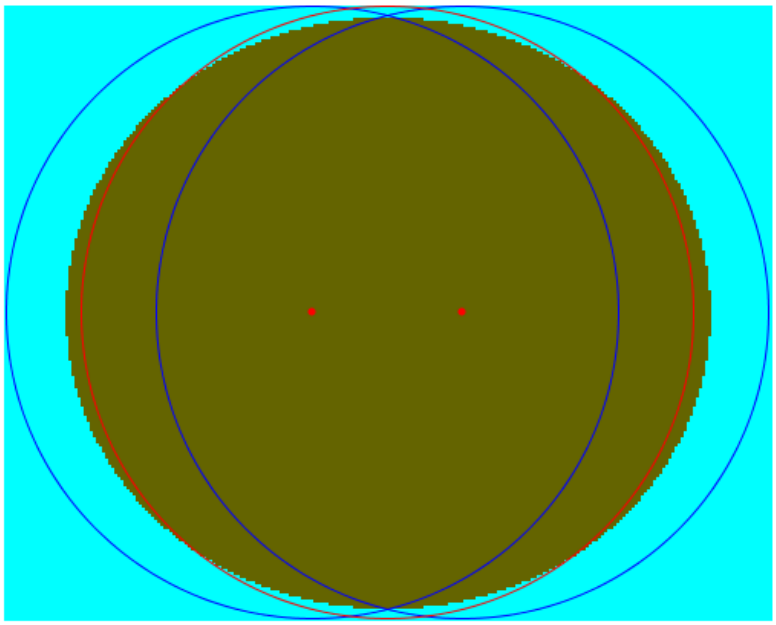
Tvrzení 2.1.3 jsem se rozhodl ověřit experimentálně a výsledek ilustroval na obrázku 2.3. Dva červené body jsou vysílače v anténním poli, plná béžová značí dosah anténního pole, červená kružnice vyznačuje vzdálenost $2d_{\max}$ od středu anténního pole a dvě modré kružnice jsou vzdálenosti $2d_{\max}$ od jednotlivých vysílačů v anténním poli.

Jak je vidět z obrázku, tvar oblasti dosažitelné anténní řadou je mírně eliptický. Tento tvar je daný vzdáleností jednotlivých vysílačů v anténním poli. Na obrázku 2.3 je $\|a - b\| \approx 0.9 \cdot d_{\max}$. Teoretické optimum o kterém se zmiňuje dizertační práce je dosažitelné pro vysílače, které jsou od sebe vzdáleny méně než jednu vlnovou délku $\lambda \ll d_{\max}$.

Přesné parametry této elipsy pravděpodobně lze spočítat, nicméně v době psaní této práce mi vzorce pro jejich výpočet nejsou známé. Elipsa však prochází průsečíky modrých kružnic a je uvnitř jejich sjednocení.

Tvrzení: Pro všechny úlohy $\forall u \in U(S, R)$ platí, že pokud $\|u_a - u_b\| \leq d_{\max}$, pak $\min\left(\|u_c - u_a\|, \|u_c - u_b\|\right) \leq 2d_{\max}$

Toto tvrzení jsem experimentálně ověřil pro $\|a - b\| \in \left\{\frac{d_{\max}}{10}, \frac{d_{\max}}{2}, \frac{9 \cdot d_{\max}}{10}\right\}$. Jelikož se v této práci nezabývám případy kdy je $\|a - b\| > d_{\max}$, budu pokládat tvrzení 2.1.3 za pravdivé.



Obrázek 2.3: Ilustrace maximální vzdálenosti na kterou naváže anténní pole spojení

2.2 Volba dvojice vysílacích antén

Jednou z hypotéz na kterých je dizertační práce [11] postavena je i tato:

> "**Pracovní hypotéza 4:** Z pohledu energetické výhodnosti je vhodné volit komunikaci na nejmenší možnou vzdálenost. Pokud existují dvě oblasti pokryté senzory, s velkou pravděpodobností se dvojice nejbližších senzorů zvolených z obou oblastí bude nacházet na okraji těchto oblastí." [11, Kap. 1.3, str.8]

Z experimentů provedených v rámci dizertační práce [11] vyplývá, že tato hypotéza platí pro většinu testovaných případů. Podmnožina, ze které bude senzorová síť vybírat dvojice uzlů, by tedy měla obsahovat spíše okraje oblasti. Z toho vyplývá, že pro volbu této množiny by mohly být vhodné algoritmy pro výpočet obálek množin bodů. Autor pro tento účel zvolil konvexní obálku.

2.3 Konvexní obálka

Definice 9. *Množina S je **konvexní** právě tehdy když:*

$$\{(1-\alpha)p + \alpha q, \forall p, q \in S, \forall \alpha \in (0,1)\} \subseteq S \tag{2.2}$$

Definice 10. ***Konvexní obálka** množiny bodů M je nejmenší konvexní množina S taková, že $M \subseteq S$. Ve 2D se dá konvexní obálka definovat také*

jako nejmenší konvexní polygon, který obsahuje všechny body z množiny M.

2.3.1 Porovnání výsledků s dizertační prací

Jelikož jsem měl k dispozici i předpočítané výsledky z dizertační práce, mohl jsem ověřit rozdíly ve vypočtených hodnotách. **AAC** nacházelo více možných řešení, než bylo v předpočítaných datech a proto některé statistiky vycházely jinak. Například podíl případů, kdy existuje řešení takové, že oba uzly a, b leží na konvexní obálce vyšel z výpočtů **AAC** na 70.35%, zatímco v předpočítaných výsledcích to bylo jen 67%.

Rozdílné výsledky jsou pravděpodobně způsobeny optimalizačním algoritmem, který autor využívá ve svých skriptech ve Wolfram Mathematice. Jednoduchým experimentem jsem ověřil, že se výsledek spočítaný Wolfram Mathematicou liší v závislosti na počáteční hodnotě argumentu optimalizace. Optimalizátor se pro některé počáteční hodnoty β zastaví v lokálním maximu. Jednou z možných příčin můžou být i hodnoty kriteriální funkce, které jsou velmi malé (řádově se optimální hodnoty pohybují okolo $1 \cdot 10^{-8}$).

Jako příklad uvedu výslednou sílu signálu pro množiny:

$$S = \{(36.3328, 1.01682), (71.1329, 81.5245)\}, R = \{(44.5769, 216.247)\}$$

Pro počáteční nastavení $\beta = 0$ vyjde Mathematice síla signálu: $8.17124 \cdot 10^{-9} < P_r^{\min}$. Avšak pro počáteční nastavení $\beta = 1$ vyjde Mathematice síla signálu: $7.07861 \cdot 10^{-8} > P_r^{\min}$.

Dalším zjevným důvodem rozdílů v některých metrikách je duplicita řešení ve výstupních datech dizertační práce. Tento rozdíl vyplývá z odlišné interpretace úlohy. **AAC** využívá pro danou dvojici vysílačů a, b nanejvýš jedno řešení s optimální silou signálu pro výpočet metrik obálek. V předpočítaných datech se ovšem objevují všechny trojice (a, b, c), pro které platí $E(a, b, c) \geq P_r^{\min}$.

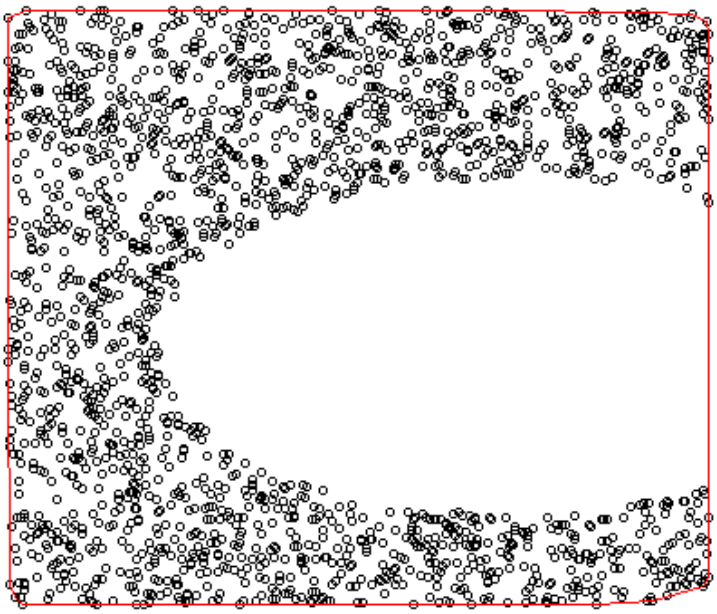
2.3.2 Nedostatky konvexní obálky

Použití konvexní obálky pro výběr uzlů má signifikantní nedostatky. V dizertační práci byla data generována pouze do obdélníkových oblastí, ovšem v praxi se s takovými tvary setkáme jen zřídka. Pro nekonvexní oblasti se budou výsledky konvexní obálky pravděpodobně výrazně lišit. Tuto hypotézu ověřím v kapitole 3, nicméně v následujících sekcích ji budu pokládat za pravdivou.

Kromě toho, že pro nekonvexní tvary senzorových sítí je výběr pomocí konvexní obálky nedostačující, tak navíc neposkytuje příliš dobré návrhy pro výběr druhého uzlu. Z těchto nedostatků vyplývá prostor pro zlepšení algoritmu. Nekonvexní oblasti by měly mít těsnější obálku, která by lépe reprezentovala tvar vygenerovaných dat.

2.4 Nekonvexní obálka

Hlavním nedostatkem konvexní obálky je její chování pro nekonvexní množiny.



Obrázek 2.4: Chování konvexní obálky pro nekovnexní množiny

V praxi se ukazuje, že takovéto případy jsou poměrně běžné a proto je nutné vylepšit odhad zahrnutím uzlů v konkávních křivkách.

Pro správnou modulaci signálu je třeba provést synchronizaci, která se obtížně provádí přes prostředníky. Proto jsem po dohodě s vedoucím práce vynechal z výpočtu pro beamforming dvojice antén, jejichž vzdálenost je větší než $d_{\max}$ (tedy se nedokážou spojit přímo). Tedy uzly na okrajích hran delších než $d_{\max}$ na obálce nemohou být dobrými kandidáty pro beamforming.

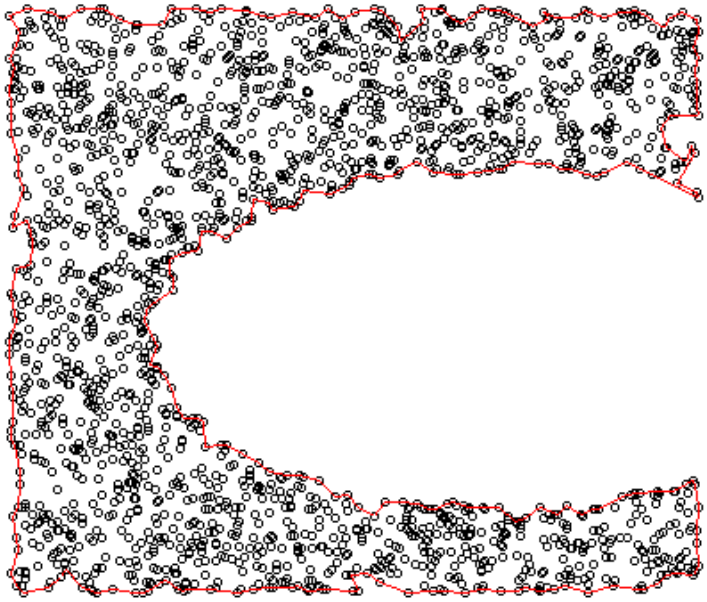
Prvotní nápad vycházel z algoritmu Jarvis wrapping pro sestavení konvexní obálky, který je popsán v kapitole 5.3.1. Po doběhnutí tohoto algoritmu by se hrany obálky, které jsou delší než $d_{\max}$ odstranily a nahradily konkávními segmenty. Tyto konkávní segmenty by byly vytvářeny pomocí pozměněného algoritmu pro konvexní obálku, kde by se vzaly v úvahu pouze hrany kratší než $d_{\max}$. Nastavení konstanty $d_{\max}$ by výrazně změnilo výslednou obálku. Pro výběr hodnoty se nabízí maximální dosah jediné antény z anténního pole.

Pokud by vzdálenost dvou shluků uzlů v rámci jednoho anténního pole byla vyšší než $d_{\max}$, mohlo by se stát, že některé body nebudou do konkávní obálky zahrnuty. Tento případ lze jednoduše detekovat pomocí algoritmu Ray casting popsaného v kapitole 5.3.2, pro zjištění zda je bod uvnitř polygonu. Poté by se na takto zjištěnou komponentu souvislosti spustil algoritmus znovu.

Tento algoritmus poskytne lepší odhady pro volbu prvního uzlu než algoritmus konvexní obálky, jelikož konvexní obálka je podmnožinou takto vzniklé konkávní obálky. Nicméně ani tato obálka nevystihuje příliš dobře tvar oblasti

s vysílacími uzly a proto jsem se pokusil tento algoritmus vylepšit.

Po několika týdnech experimentů a konzultací jsem nalezl již existující algoritmus pro tvorbu konkávní obálky, který pracuje na velmi podobném principu. Z toho důvodu jsem zanechal vývoje svého vlastního algoritmu a raději upravil existující algoritmus tak, aby vyhovoval úloze. Princip a funkci tohoto algoritmu popíšu v kapitole 5.3.7. Tento algoritmus jsem naimplementoval a otestoval na stejných datech, která jsou na obrázku 2.4:



Obrázek 2.5:

Na této obálce leží mnohem více uzlů než na konvexní obálce, což může být problematické pro její použití. Jak bylo uvedeno dříve, anténní pole se pokusí navázat spojení právě z uzlů, které leží na obálce. Proto je jedním z kritérií kvality obálky i počet uzlů na obálce, který bychom se měli snažit minimalizovat.

2.4.1 Výběr parametrů obálky

Jednoznačným nedostatkem konkávní obálky je vysoký podíl uzlů na obálce. Tento faktor ovšem můžeme ovlivnit vhodnou úpravou parametrů obálky.

Algoritmus pro tvorbu konkávní obálky přijímá jako vstupní parametr celé číslo k. Úpravami hodnoty tohoto parametru lze ovlivnit rozlišení obálky a tím redukovat počet uzlů na obálce. Číslo k ovlivňuje minimální počet sousedů, kteří jsou součástí rozhodnutí o každém uzlu, který leží na obálce. Jelikož se jedná o minimum počtu sousedů, mohou různá nastavení k mít stejný výstup. Nastavení tohoto parametru ovlivňuje nejen tvar obálky, ale i čas jejího sestavení.

Kromě možnosti specifikovat k poskytuje algoritmus možnost specifikovat i číslo r, které určuje poloměr kružnice ve které se budou nejbližší sousedé

hledat. Pro některá nastavení parametrů k, r obálku nelze sestavit, nicméně jejich použití může výrazně snížit poměr uzlů na obálce a tím zvýšit přesnost tohoto algoritmu.

Pro účely hodnocení kvality obálek zavedu několik termínů:

- Řešení je dvojice uzlů $a, b \in S$, pro které $\exists c \in R$ takové, že $(a, b, c) \in U(S, R)$
- M_α je množina řešení, které mají alespoň jeden uzel na obálce
- M_β je množina řešení, které mají oba uzly na obálce

Změny parametru k

Definice 11. *Mějme oblast M, , pak označíme hustotu oblasti M jako M_H.*

$$M_H = \frac{|M| \cdot \pi \cdot (P_r^{\min})^2}{V(M)}$$

Zde $P_r^{\min}$ je maximální dosah jednoho uzlu a $V(M)$ je obsah polygonu definujícího oblast M.

Intuitivně M_H vyjadřuje průměrný teoretický počet uzlů $a \in M$ se kterými se může každý uzel $b \in M$ přímo spojit.

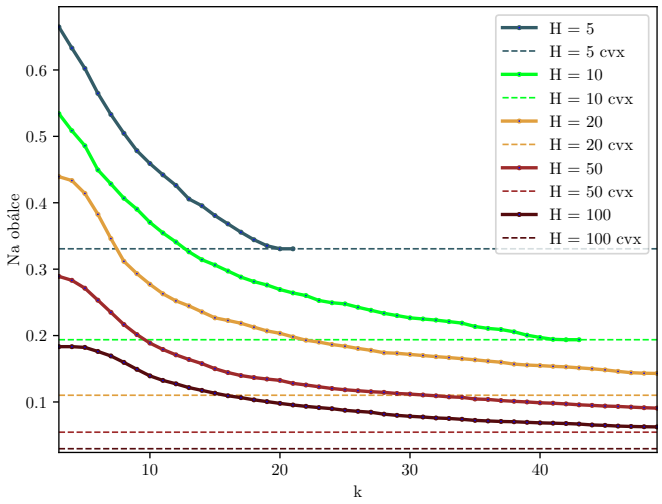
Vliv parametru k na detail obálky je významný. Zvýšením tohoto parametru snížíme detail obálky. Pro $k \geq |M|$ bude výsledná obálka běžnou konvexní obálkou.

Vliv změn tohoto parametru jsem zkoumal experimentálně a proto jsem vyhodnotil data pro všechny testovací případy pro $3 \leq k < 50$. Následující tabulky ukazují výsledky v závislosti na změnách parametru k a hustoty M_H na testovacím případu řeky (ten je popsán v kapitole 3.5):

Hustota	k = 3	k = 6	k = 10	k = 15	k = 20	k = 30
5	0.665	0.565	0.459	0.381	0.331	-
10	0.534	0.449	0.370	0.306	0.269	0.227
20	0.439	0.383	0.277	0.227	0.203	0.172
50	0.289	0.253	0.189	0.150	0.133	0.112
100	0.183	0.176	0.139	0.113	0.098	0.078

Tabulka 2.1: Závislost podílu uzlů na obálce na k a hustotě

Z tabulky 2.1 jasně vyplývá, že vhodným nastavením k lze výrazně snížit podíl uzlů na obálce. Tato závislost je také zobrazena na grafu 2.6.



Obrázek 2.6: Graf podílu uzlů na obálce v závislosti na k pro testovací případ řeka

Toto chování není nijak překvapivé, jelikož se zvyšováním k počet uzlů na obálce konverguje k počtu uzlů na konvexní obálce. Rychlost této konvergence se však pro různé hodnoty hustoty liší.

Způsob, jakým se tyto úpravy dotknou přesnosti obálky ilustruje tabulka 2.2.

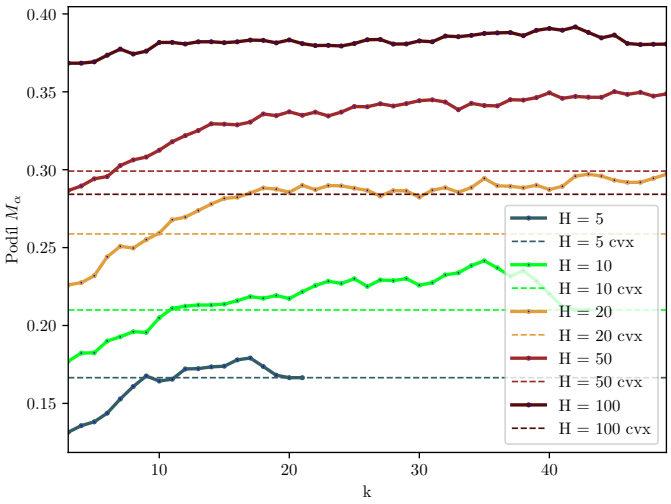
Hustota	k = 3	k = 6	k = 10	k = 15	k = 20	k = 30
5	0.132	0.144	0.164	0.174	0.167	-
10	0.177	0.190	0.205	0.214	0.217	0.226
20	0.226	0.244	0.259	0.282	0.286	0.282
50	0.287	0.296	0.313	0.329	0.337	0.344
100	0.368	0.373	0.382	0.382	0.383	0.383

Tabulka 2.2: Podíl M_α vůči počtu uzlů na obálce pro různé hodnoty parametru k

Tyto výsledky ukazují, že v případě řeky se zvětšováním parametru k snižuje počet uzlů na obálce. Proto přesnost odhadu se zvyšováním parametru k mírně roste. Závislosti jsou ještě lépe vidět na grafu 2.7.

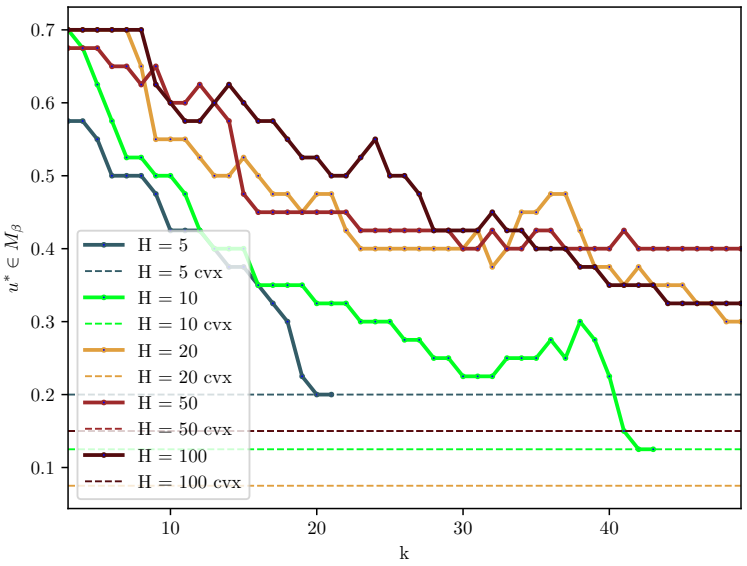
Grafy pro nižší hustoty ukazují podobné chování. Pro nízká nastavení k je přesnost obálky dokonce nižší než u konvexní obálky. S růstem hodnoty k přesnost obálky stoupá až do maxima a poté prudce klesá zpět k výsledkům konvexní obálky. Hodnoty k pro které dosahuje zobrazená metrika maxima se u hustot 5 a 10 blíží celkovému počtu uzlů v senzorové síti.

Grafy vyšších hustot mají jiné chování, ovšem vzhledem k vyššímu počtu uzlů je zobrazený interval pro kompletní analýzu nedostatečný. Vliv k na podíl M_α pro sítě s vysokými hustotami by mohlo být předmětem dalšího výzkumu.



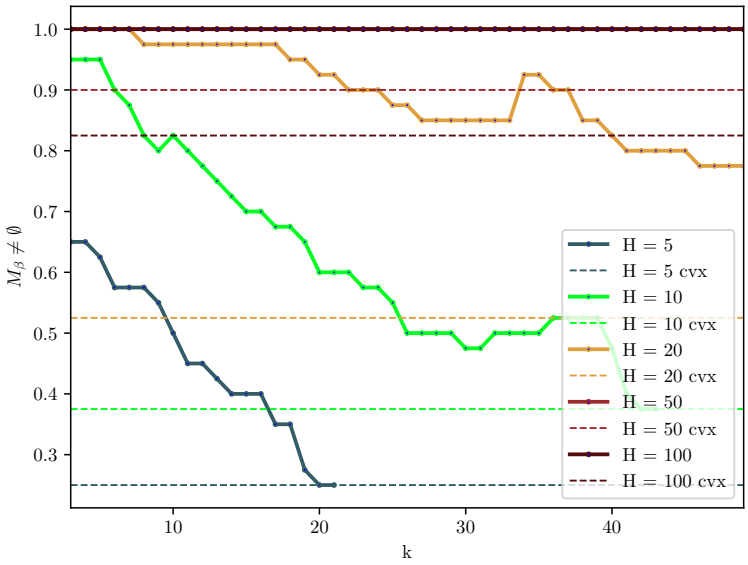
Obrázek 2.7: Graf podílu M_α v závislosti na k pro testovací případ řeka

Graf 2.8 zobrazuje vliv k na existenci optimálního řešení na obálce. Tato metrika klesá se zvyšováním parametru k. Zajímavé je chování pro hustoty $10, 20$, které mají pro hodnoty $k \approx 37$ lokální maximum.



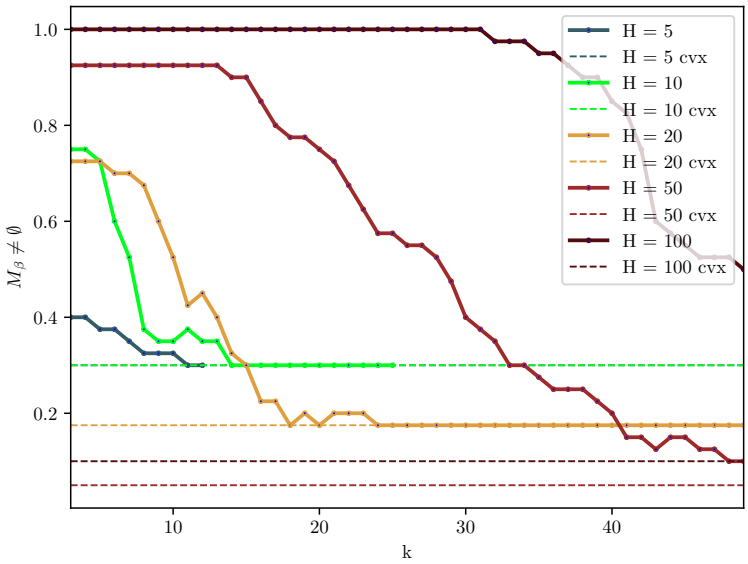
Obrázek 2.8: Graf podílu případů, kdy $u^* \in M_\beta$ v závislosti na k pro testovací případ řeka

Podobné lokální maximum je i na grafu 2.9, který zobrazuje pravděpodobnost existence jakéhokoliv řešení s oběma uzly na obálce.



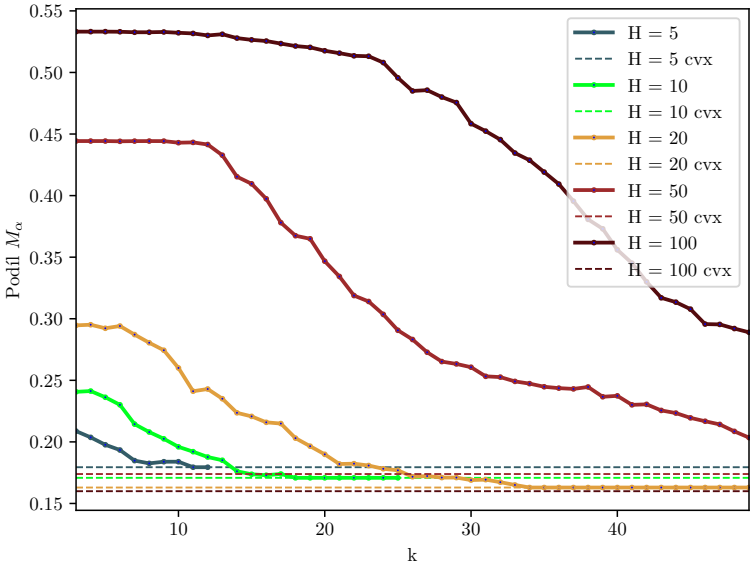
Obrázek 2.9: Graf podílu případů, kdy $M_\beta \neq \emptyset$ v závislosti na k pro testovací případ řeka

Uvedu ještě výsledky testovacího případu Vnitřní elipsa (popsaného v kapitole 3.3). Vývoj počtu uzlů na obálce je stejný jako u testovacího případu řeky. Na grafu 2.10 je zobrazen vliv hodnot k na existenci řešení s oběma uzly na obálce.



Obrázek 2.10: Graf podílu případů, kdy $M_\beta \neq \emptyset$ v závislosti na k pro testovací případ Vnitřní elipsa

V tomto případě je vývoj této metriky mnohem strmější než u případu řeky. I zde se objevuje lokální maximum pro hustoty 10, 20. Nápadný je rozdíl mezi výsledky konkávní a konvexní obálky, primárně pro vysoké hustoty.



Obrázek 2.11: Graf podílu M_α a počtu uzlů na obálce pro testovací případ Vnitřní elipsa

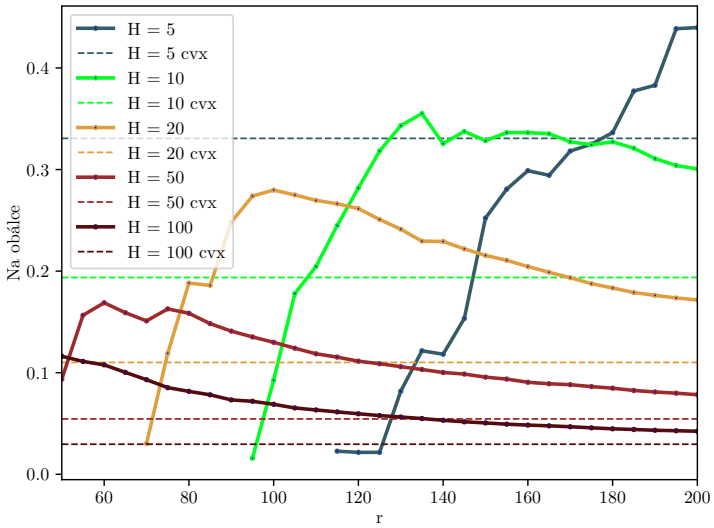
Na grafu 2.11 není stoupavá tendence, jaká se objevovala v grafu 2.7. Z porovnání těchto dvou testovacích případů je zřejmé, že optimální volba k je vysoce závislá na topologii a hustotě senzorové sítě.

Změny parametru r

Hodnota parametru r udává maximální vzdálenost na kterou se budou hledat dvojice uzlů pro obálku. Zřejmou volbou hodnoty by mohla být $r = d_{\max}$. Pro hodnocené testovací případy je $d_{\max} = 100$ a proto jsem všechny testovací případy vyhodnotil pro hodnoty $r \in \langle 50, 200 \rangle$.

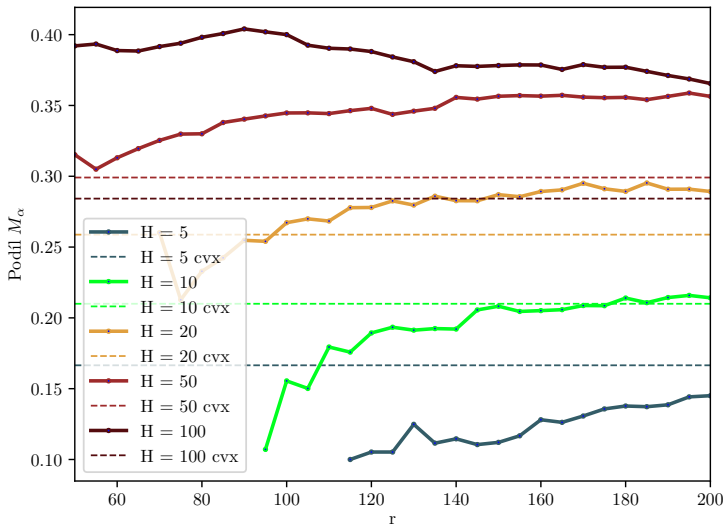
Stejně jako tomu bylo u k, i zde pro některé hodnoty parametru r nelze obálku sestavit. Pro nižší hustoty se může stát, že jsou vzdálenosti mezi uzly větší než r a proto taková souvislá obálka neexistuje.

Na grafu 2.12 je zobrazen podíl uzlů na obálce v závislosti na nastaveních r. Jak bylo zmíněno, pro nízké hustoty a nízké hodnoty r obálka nelze sestavit. Podíly uzlů na obálce jsou výrazně nižší než při volbách parametru k.



Obrázek 2.12: Graf podílu uzlů na obálce v závislosti na r pro testovací případ řeka

Vliv parametru r na podíl uzlů na obálce, které jsou součástí alespoň jednoho řešení je zanesen v grafu 2.13



Obrázek 2.13: Graf podílu M_α v závislosti na r pro testovací případ řeka

Rozbor pro různé hodnoty parametrů k, r (například v poměru k celkovému počtu uzlů sítě) by mohl být předmětem dalšího zkoumání. Algoritmus umožňuje specifikovat oba parametry zároveň, což může být výhodné pro specifické tvary a topologie sítě.

Kapitola 3

Výsledky obálek

V této sekci se budu zabývat kvalitami a porovnáním odhadů obou obálek.

Pro zhodnocení výsledků obálek jsem definoval pět testovacích případů a pro každý případ vygeneroval padesát náhodných instancí pro čtyři různá nastavení hustoty antén: $1, 3, 8, 15$. Tabulky uvádějí následující údaje pro jednotlivé hodnoty hustot:

- Podíl uzlů na obálce z celkového počtu uzlů v S
- Podíl testovacích případů ve kterých $M_\alpha \neq \emptyset$. Tedy případů, kdy existuje alespoň jeden uzel na obálce, který je součástí některého řešení.
- Podíl testovacích případů ve kterých $M_\beta \neq \emptyset$. Tedy případů, kdy existuje alespoň jedna dvojice uzlů na obálce, která je řešením úlohy.
- Podíl testovacích případů ve kterých je optimální řešení $u^* \in M_\beta$. Tedy případů, kdy jsou oba uzly optimálního řešení na obálce.
- Podíl uzlů na obálce, které jsou součástí alespoň jednoho řešení z M_α
- Podíl $|M_\beta|$ vůči počtu všech kombinací dvou uzlů na obálce $\binom{|M|}{2}$. Tato metrika vyjadřuje pravděpodobnost, že se při náhodném výběru dvojice uzlů z obálky trefíme do některého řešení úlohy.

Na ilustracích jsou vysílací senzorové sítě zobrazeny černě a přijímací senzorové sítě červeně. Pro výpočty konkávní obálky jsem použil výchozí nastavení parametrů $k = 3, r = \infty$. Konkávní obálka je pro tuto kombinaci parametrů velmi detailní a tím trpí podíly M_α, M_β. Vývoj algoritmu pro hledání optimálních hodnot k, r by mohl být předmětem dalšího výzkumu.

3.1 Dva obdélníky

Tento případ simuluje testovací případy, na kterých byl algoritmus konvexní obálky testován v dizertační práci [11].

Obrázek 3.1: Dva obdélníky

Výsledky pro konvexní obálku:

Hustota	Na obálce	$M_\alpha \neq \emptyset$	$M_\beta \neq \emptyset$	$u^* \in M_\beta$	Podíl M_α	Podíl M_β
5	0.829	1	0.825	0.800	0.370	0.084
10	0.544	1	0.775	0.750	0.258	0.039
20	0.332	1	0.700	0.600	0.211	0.023
50	0.168	1	0.575	0.350	0.159	0.012
100	0.095	0.975	0.550	0.275	0.151	0.011

Tabulka 3.1: Dva obdélníky s konvexní obálkou

A pro konkávní obálku:

Hustota	Na obálce	$M_\alpha \neq \emptyset$	$M_\beta \neq \emptyset$	$u^* \in M_\beta$	Podíl M_α	Podíl M_β
5	0.858	1	0.875	0.850	0.368	0.084
10	0.698	1	0.875	0.875	0.215	0.026
20	0.575	1	0.900	0.850	0.144	0.012
50	0.415	1	0.975	0.925	0.102	0.006
100	0.306	1	1	0.875	0.096	0.006

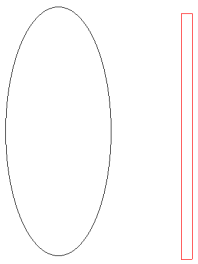
Tabulka 3.2: Dva obdélníky s konkávní obálkou

Z dat v této testovací sadě vyplývá, že konvexní obálka se skládá z mnohem menšího počtu uzlů, což je velmi výhodné z hlediska hledání řešení. Oba typy obálek poskytují dobrý odhad pro M_α, nicméně konkávní obálka má mnohem častěji oba uzly optimálního řešení na obálce.

Jelikož je na konkávní obálce výrazně vyšší množství uzlů, její přesnost je relativně nízká.

3.2 Elipsa a obdélník

V tomto případě se jedná o dvě konvexní oblasti, pouze různých tvarů. Elipsa je pro oba typy obálek tím nejméně výhodným tvarem vzhledem k podílu uzlů na obálce, ovšem tento efekt by se měl projevit jen u velmi vysokých hustot.



Obrázek 3.2: Elipsa a obdélník

Hustota	Na obálce	$M_\alpha \neq \emptyset$	$M_\beta \neq \emptyset$	$u^* \in M_\beta$	Podíl M_α	Podíl M_β
5	0.700	1	0.725	0.475	0.424	0.107
10	0.447	1	0.875	0.525	0.484	0.090
20	0.299	1	0.925	0.275	0.473	0.060
50	0.169	1	1	0.275	0.471	0.059
100	0.105	1	1	0.175	0.502	0.060

Tabulka 3.3: Elipsa a obdélník s konvexní obálkou

Hustota	Na obálce	$M_\alpha \neq \emptyset$	$M_\beta \neq \emptyset$	$u^* \in M_\beta$	Podíl M_α	Podíl M_β
5	0.812	1	0.875	0.650	0.436	0.110
10	0.747	1	1	0.900	0.541	0.115
20	0.599	1	1	0.825	0.530	0.084
50	0.406	1	1	0.900	0.543	0.089
100	0.301	1	1	0.900	0.552	0.084

Tabulka 3.4: Elipsa a obdélník s konkávní obálkou

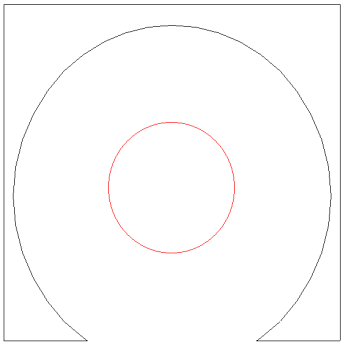
Oba typy obálek by v případech s velmi vysokými hustotami měly spočítat stejné obálky a z hlediska podílu uzlů na obálce jsou skutečně velmi blízko vzhledem k výsledkům ostatních testovacích případů. V tomto případě je podíl uzlů na obálce vysoký pro oba typy obálek.

Oba typy obálek také nalezly ve většině testovacích případů řešení s oběma uzly na obálce. Konvexní obálka má vysoký podíl M_α, avšak přesnost odhadu

je stále vyšší u konkávní obálky. Konkávní obálka navíc obsahuje oba uzly optimálního řešení v převážné většině případů

3.3 Vnitřní elipsa

Toto přednastavení simuluje typický případ konkávní oblasti.



Obrázek 3.3: Vnitřní elipsa

Hustota	Na obálce	$M_\alpha \neq \emptyset$	$M_\beta \neq \emptyset$	$u^* \in M_\beta$	Podíl M_α	Podíl M_β
5	0.600	0.775	0.300	0.050	0.179	0.012
10	0.391	0.800	0.300	0.025	0.171	0.006
20	0.240	0.825	0.175	0	0.163	0.003
50	0.112	0.925	0.050	0	0.174	0.001
100	0.067	0.900	0.100	0	0.160	0.001

Tabulka 3.5: Konvexní obálka pro vnitřní elipsu

Hustota	Na obálce	$M_\alpha \neq \emptyset$	$M_\beta \neq \emptyset$	$u^* \in M_\beta$	Podíl M_α	Podíl M_β
5	0.694	0.850	0.400	0.175	0.209	0.018
10	0.581	0.975	0.750	0.250	0.241	0.014
20	0.454	1	0.725	0.075	0.295	0.011
50	0.387	1	0.925	0.425	0.444	0.023
100	0.345	1	1	0.775	0.533	0.032

Tabulka 3.6: Konkávní obálka pro vnitřní elipsu

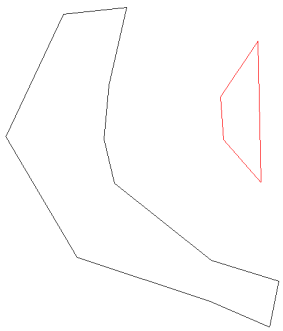
Dle očekávání zde konkávní obálka exceluje. Přestože se konkávní obálka sestává z výrazně většího počtu uzlů, poskytuje velmi dobrý odhad pro jeden i oba uzly na obálce. Obě hodnoty podílu ukazují, že konkávní obálka poskytuje výrazně přesnější odhad než konvexní obálka. Překvapivé je chování konkávní

obálky pro hustotu 20, kdy byl podíl případů s optimálním řešením na obálce výrazně nižší než pro ostatní hustoty.

Konvexní obálka pro tento testovací případ zpravidla nenachází optimální řešení, její podíl M_α se pohybuje pod 18% a $M_\beta \leq 1.2\%$. Proto je pro tento případ šance na nalezení jakéhokoliv řešení na konvexní obálce velmi nízká.

3.4 Jednoduchý polygon

V tomto testovacím případě jsou postaveny proti sobě dvě senzorové sítě, jedna ve tvaru jednoduchého polygonu a druhá ve tvaru obdélníku.



Obrázek 3.4: Jednoduchý polygon

Výsledná data:

Hustota	Na obálce	$M_\alpha \neq \emptyset$	$M_\beta \neq \emptyset$	$u^* \in M_\beta$	Podíl M_α	Podíl M_β
5	0.534	0.575	0.175	0.050	0.159	0.023
10	0.333	0.525	0.125	0.025	0.112	0.005
20	0.207	0.375	0.075	0	0.072	0.002
50	0.108	0.275	0.125	0	0.069	0.003
100	0.061	0.275	0	0	0.027	0

Tabulka 3.7: Jednoduchý polygon a konvexní obálka

Hustota	Na obálce	$M_\alpha \neq \emptyset$	$M_\beta \neq \emptyset$	$u^* \in M_\beta$	Podíl M_α	Podíl M_β
5	0.736	0.900	0.625	0.525	0.240	0.042
10	0.659	1	0.850	0.625	0.177	0.016
20	0.565	1	0.975	0.675	0.179	0.015
50	0.425	1	1	0.875	0.184	0.016
100	0.328	1	1	0.825	0.204	0.016

Tabulka 3.8: Jednoduchý polygon a konkávní obálka

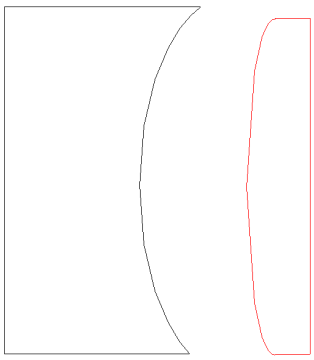
I v tomto případě se konkávní obálka skládá z velkého podílu uzlů, nicméně rozdíl v kvalitách odhadů to vyvažuje. Pro všechny měřené hodnoty hustot má konvexní obálka velmi špatné výsledky. Pro valnou většinu případů nenalezla žádná řešení a to ani s jedním uzlem na obálce. Přestože se konvexní obálka skládá z mnohonásobně menšího počtu uzlů, podíly M_α, M_β jsou nízké.

Přestože se konkávní obálka skládá z velkého podílu uzlů, podíl M_α je u ní velmi příznivý. Optimální řešení leží na konkávní obálce ve více než polovině případů pro všechny hustoty a obálka téměř vždy obsahuje alespoň jedno řešení.

I v tomto případě je konkávní obálka výrazně lepší, než ta konvexní.

3.5 Řeka

Tento testovací případ zobrazuje řeku, která odděluje dvě senzorové sítě. Tvar oblastí je podobný případu jednoduchého polygonu, avšak okraje konkávní oblasti jsou mnohem bližší k přijímací oblasti.



Obrázek 3.5: Řeka

A naměřená data:

Hustota	Na obálce	$M_\alpha \neq \emptyset$	$M_\beta \neq \emptyset$	$u^* \in M_\beta$	Podíl M_α	Podíl M_β
5	0.331	0.850	0.250	0.200	0.167	0.009
10	0.194	0.975	0.375	0.125	0.210	0.017
20	0.110	1	0.525	0.075	0.259	0.021
50	0.054	1	0.900	0.150	0.299	0.029
100	0.030	1	0.825	0.150	0.284	0.025

Tabulka 3.9: Řeka s konvexní obálkou

Hustota	Na obálce	$M_\alpha \neq \emptyset$	$M_\beta \neq \emptyset$	$u^* \in M_\beta$	Podíl M_α	Podíl M_β
5	0.665	1	0.650	0.575	0.132	0.008
10	0.534	1	0.950	0.700	0.177	0.012
20	0.439	1	1	0.700	0.226	0.013
50	0.289	1	1	0.675	0.287	0.018
100	0.183	1	1	0.700	0.368	0.028

Tabulka 3.10: Řeka s konkávní obálkou

Oba typy obálek mají na tomto testovacím případě dobré výsledky, což je překvapivé vzhledem ke konkávnímu tvaru vysílací oblasti. Podíl uzlů na konkávní obálce je oproti té konvexní vysoký. To však není nikterak překvapivé, jelikož konvexní obálka tu přeskočí celou konkávní část řeky a spojí jen dva krajní body. Nicméně i přes menší detail v konkávní oblasti okolo břehu řeky má konvexní obálka velmi dobré podíly M_α a M_β. Tyto parametry jsou i u konkávní obálky lehce vyšší, nicméně hlavním rozdílem je opět v šanci na nalezení optimálního řešení.

Výsledky konkávní obálky jsou pro hustoty $10, 20$ silně ovlivněny podílem uzlů na obálce. Výsledky by mohly být výrazně lepší pro správně zvolené hodnoty parametru k, jak je vidět z grafu 2.7.

Výsledky řeky a jednoduchého polygonu ukazují důležitost vzdálenosti okrajů konkávní oblasti od přijímací oblasti. V případě jendoduchého polygonu byly okraje, které jsou vždy na konvexní obálce, velmi vzdálené od přijímací oblasti. Výsledky konvexní obálky to také reflektovaly v hodnotách podílů M_α, M_β. V případě řeky jsou okraje konkávní sekce mnohem blíže k přijímací oblasti a výsledky konvexní obálky to jasně reflektují.

3.6 Roztržení

Tento případ se skládá ze dvou "odtržených"oblastí.

Obrázek 3.6: Roztržení

Hustota	Na obálce	$M_\alpha \neq \emptyset$	$M_\beta \neq \emptyset$	$u^* \in M_\beta$	Podíl M_α	Podíl M_β
5	0.368	0.875	0.475	0.350	0.213	0.022
10	0.217	0.925	0.500	0.025	0.261	0.018
20	0.125	1	0.800	0.075	0.325	0.030
50	0.059	1	0.925	0.150	0.363	0.041
100	0.032	1	1	0.025	0.367	0.042

Tabulka 3.11: Roztržení s konvexní obálkou

Hustota	Na obálce	$M_\alpha \neq \emptyset$	$M_\beta \neq \emptyset$	$u^* \in M_\beta$	Podíl M_α	Podíl M_β
5	0.596	0.975	0.675	0.525	0.200	0.015
10	0.512	1	0.975	0.600	0.303	0.023
20	0.392	1	1	0.500	0.358	0.025
50	0.317	1	1	0.750	0.406	0.027
100	0.244	1	1	0.775	0.447	0.031

Tabulka 3.12: Roztržení s konkávní obálkou

I zde dosahuje konvexní obálka překvapivě dobrých výsledků. Nejzajímavější anomálií je její výrazně vyšší podíl M_β, který se objevoval pro nižší hustoty i v případu řeky. Tento případ je ovšem jediný, kde má konvexní obálka výrazně lepší podíl M_β pro všechny testované hustoty.

Konvexní obálka i zde neposkytuje příliš dobrý odhad optimálního řešení, avšak alespoň jedno řešení nachází v naprosté většině případů nezávisle na hustotě. Konkávní obálka má převahu v odhadu optimálního řešení, což je způsobeno výrazně vyšším počtem uzlů na obálce a v podílu M_α, který je i pro nízké hustoty velmi příznivý.

Kapitola 4

Uživatelský manuál

Výsledky rozebrané v kapitole 3 jsem získal pomocí simulačního programu **AAC**. Program umožňuje definovat vlastní testovací případy a vypočítat výsledky pro různé hustoty. To je velmi užitečné pro výzkumníky pracující na problematice výběru obálky. Z toho důvodu v této kapitole podrobněji popíšu jeho instalaci a způsob ovládání uživatelského rozhraní.

Program je primárně určen pro použití na operačním systému Linux. Kompilace a spuštění na jiných platformách je možná, ovšem vyžaduje změny v CMake souboru. Uvedu zde proces instalace na operačním systému Linux Ubuntu 20.04.

4.1 Instalace

Stáhněte a rozbalte zdrojový kód projektu například z repozitáře na GitLabu [7]. Pro kompilaci a sestavení projektu je potřeba několik knihoven a nástrojů. Pro jejich instalaci slouží následující sekvence příkazů:

4.1.1 Instalace knihoven

```
# Aktualizace package manageru
sudo apt update -y

# Instalace build systému a knihovny nezbytné pro funkci wxWidgets
sudo apt install build-essential libgtk-3-dev -y

# Pro sestavení projektu je také nutné mít cmake verzi vyšší než 3.17:
sudo snap install cmake --classic

# Další dependencí je knihovna CGAL
sudo add-apt-repository universe
sudo apt update -y
sudo apt install libcgal-dev -y

# Stažení a rozbalení zdrojových souborů knihovny wxWidgets.
```

```
# Verze musí být vyšší než 3.1.0
wget github.com/wxWidgets/wxWidgets/releases/download/v3.1.4/wxWidgets-3.1.4.tar
tar -xf wxWidgets-3.1.4.tar.bz2

# Tvorba složky ve které se bude kompilovat wxWidgets
mkdir wxWidgets-3.1.4/my-build
cd wxWidgets-3.1.4/my-build

# Kompilace wxWidgets
../configure --enable-stl --with-cxx=17 --disable-shared
make -j4 # Zde 4 je počet vláken, která se pro kompilaci využijí.
sudo make install
sudo ldconfig
```

Zda instalace wxWidgets proběhla úspěšně můžete ověřit z kteréhokoliv adresáře spuštěním příkazu

```
wx-config --version
# Mělo by vypsat 3.1.4
```

4.1.2 Kompilace a spuštění projektu

Pro kompilaci v adresáři s rozbaleným projektem spusťte příkazy:

```
cmake .
cmake --build . --target AAC
```

Zkompilovanou aplikaci lze spustit příkazem:

```
./AAC
```

4.2 Použití

Po spuštění aplikace se zobrazí okno s ovládacími prvky a bílým čtvercem, ve kterém je vizualizace plochy. Uživatel může přidávat vysílací antény do plochy kliknutím levého tlačítka myši a přijímací antény kliknutím pravého tlačítka myši. Vysílací antény jsou zobrazeny černou kružnicí a přijímací antény červenou kružnicí.

Uživatel může označit vysílací anténu podržením Ctrl a levým kliknutím na anténu. Pokud jsou označené dvě antény a na ploše je alespoň jeden přijímač, v ploše se vizualizuje cesta k nejbližšímu přijímači. Přitom se v pravé části okna se zobrazí detaily zobrazeného připojení a síla signálu. Zeleně označenou anténu lze z plochy smazat stiskem klávesy Delete.

V pravé části obrazovky je souhrn detailů právě označeného spojení a také seznam deseti nejlepších řešení zobrazené instance. Po kliknutí na řádek v seznamu se zvolené řešení zvýrazní ve vizualizaci a v detailech připojení.

4.2.1 Výpočet obálky

Program umí vypočítat a zobrazit obálku množiny vysílacích antén. Uživatel může zvolit typ zobrazené obálky v rozhrnovacím seznamu nad vizualizací. Uzly na vypočtené obálce lze zvýraznit odškrtnutím volby **Hull connecting lines** v menu **Display**. Na spodní liště se zobrazují metriky zvolené obálky:

- Podíl $\frac{|M_\alpha|}{|M|}$
- Počet řešení v M_β
- Pořadí nejlepšího řešení v M_β ze všech nalezených řešení
- Podíl uzlů na obálce

Po přidání či odebrání uzlů se všechna data automaticky přepočítají. Toto chování lze změnit v menu **Computation** volbou možnosti **Auto-recalculate**. Pokud je tato možnost vypnutá, data se přepočítávají pouze po stisknutí tlačítka **Recalculate**.

Program také umí zobrazit přibližný dosah jednotlivých antén zaškrtnutím příslušného tlačítka v menu **Display**.

4.2.2 Tvorba a použití oblastí

Oblasti jsou útvary v ploše na kterých se náhodně generují antény. Každá oblast generuje jeden typ antén. Uživatel může přidat čtyři typy oblastí do plochy: obdélník, elipsu, jednoduchý polygon a komplexní oblast.

Po kliknutí na příslušné tlačítko v horní části okna může uživatel kurzorem ve vizualizaci definovat oblast. Obdélník a elipsu uživatel definuje kliknutím a tažením kurzoru. Při tvorbě polygonu přidá vrchol kliknutím myši do plochy. Výsledný polygon může být konkávní, ale nesmí být degenerovaný (žádné dvě strany polygonu se nesmí protínat). Když je uživatel spokojen s tvarem tvořené oblasti, může svou volbu potvrdit stiskem klávesy **Enter**, případně operaci zrušit stiskem klávesy **Esc**.

Komplexní oblast je oblast, která vznikne postupným aplikováním množinových operací nad obdélníky, elipsami a polygony. Po kliknutí na tlačítko **Complex** uživatel zvolí tvar výchozí oblasti (obdélník, elipsa, polygon) a definuje ji v ploše. Po potvrzení tvaru výchozí oblasti enterem se zobrazí kontextová nabídka ve které lze zvolit operaci **Join** a **Intersect** a další tvar.

Po definici a potvrzení nové oblasti se zvolená množinová operace aplikuje nad dosavadní komplexní oblastí a novou oblastí. Operace **Join** připojí novou oblast k dosavadní, jinými slovy provede nad útvary matematickou operaci sjednocení. Operace **Intersect** odstraní novou oblast z dosavadní, což je ekvivalentní s matematickou operací průniku inverze nové oblasti s dosavadní. Pro dokončení definice komplexní oblasti lze v kontextové nabídce vybrat možnost **Finish** a pro zrušení **Cancel**.

Celý proces definice komplexní oblasti se dá popsat pseudokódem 4.1.

```
area = getUserDefinedArea()
running = true
repeat:
    if userSelected(CANCEL):
        return Nothing
    if userSelected(FINISH):
        return area

    newArea = getUserDefinedArea()
    if userSelected(JOIN):
        area = area.union(newArea)
    else:
        area = area.intersection(newArea)
```

Obrázek 4.1: Pseudokód popisující tvorbu komplexní oblasti

Po potvrzení se nová oblast zobrazí v seznamu v pravé části okna. Oblast lze v listu vybrat a upravovat její vlastnosti: Počet antén, které se v oblasti vygenerují a zda jsou antény přijímače či vysílače. Po kliknutí na tlačítko **Save** se vlastnosti oblasti uloží a tlačítkem **Delete** lze vybranou oblast smazat.

Instance je konrétní případ množin S, R. Pokud jsou v ploše definovány oblasti, uživatel může vygenerovat náhodnou instanci klikutím na tlačítko **Generate random areas** v menu **Areas**. Tato akce smaže předchozí instanci a vygeneruje nová náhodná data v definovaných oblastech. Pokud je v oblastech nastaveno mnoho antén, program zobrazí varování, jelikož počítání signálů pro vysoké počty antén může trvat dlouhou dobu.

4.2.3 Ukládání a nahrávání

AAC umí uložit jak vygenerovanou instanci, tak definované náhodné oblasti do souboru. Uložit nebo nahrát vygenerovanou instanci lze kliknutím na příslušné tlačítko v menu **File**. Program instanci uloží do souboru s příponou **.aai** (antenna array instance) v textovém formátu **JSON**. Data v těchto souborech lze bez problémů ručně měnit. Aplikace načte upravený **.aai** soubor, pokud jeho struktura zůstane zachována. Nahrání **.aai** souboru přepíše zobrazenou instanci daty ze souboru.

Definice náhodných oblastí lze uložit či nahrát příslušnými tlačítky v menu **Areas**. Program uloží pouze definice náhodných oblastí, neukládá vygenerovanou instanci. Oblasti se ukládají do souborů s příponou **.aam** (antenna array map), který je opět ve formátu **JSON**. Nahrání **.aam** souboru přepíše definované náhodné oblasti daty ze souboru, ale zanechá zobrazenou instanci.

4.3 Skriptovací rozhraní

Struktura programu umožňuje použít výpočetní část ve svém vlastním kódu. To může být užitečné například pro automatické vygenerování a vyhodnocení velkého množství instancí.

Knihovna se ovládá pomocí třídy `Logic::World`. Tato třída poskytuje rozhraní, které umožňuje přidávat body a oblasti do plochy, generovat instance, ukládat a nahrávat oblasti a instance a získávat výsledky výpočtů síly signálu. Knihovna používá pro reprezentaci bodů v ploše třídu `Logic::Vector2`, která implementuje většinu běžně používaných operací nad vektory.

Všechny relevantní třídy pro použití skriptovacího rozhraní lze do programu připojit pomocí hlavičkového souboru `src/console/ScriptingHeaders.h`. Vlastní skripty lze přidávat do předpřipravené složky `src/console`. Zde se také nachází příklad použití skriptovacího rozhraní v souboru `exampleScript.h`.

Targety pro kompilaci vlastních skriptů lze přidat do souboru `CMakeLists.txt` v kořenovém adresáři projektu. Na konci souboru jsou zakomentované dvě řádky, které definují target pro example skript.

4.3.1 Náhodné oblastí

Třída World poskytuje funkci `addRandomArea`, která přidá náhodnou oblast do plochy. Třídy, které lze předat této funkci jsou v jmenném prostoru `Logic::Areas`. Jejich použití je intuitivně zřejmé z parametrů konstruktorů. Jedinou výjimkou je `ComplexRandomArea`, která v konstruktoru dostává pouze výchozí oblast. Další oblasti, používané jako operandy množinových operací se následně přidávají funkcí `addArea`.

4.3.2 Implementace vlastní obálky

Program podporuje přidávání vlastních implementací počítadel obálek. Ty lze přidat do projektu ve formě C++ třídy do složky `src/logic/hulls/custom`. Každý vlastní kalkulátor je třída, která musí dědit z třídy `HullComputer`, která je v souboru `src/logic/hulls/HullComputer.h` a musí implementovat její rozhraní. Specificky to jsou funkce:

- `getName()`, která vrací řetězec s názvem obálky
- `computeHull()`, která vrací vektor indexů uzlů, které popořadě tvoří okraj obálky. Tato funkce přijímá jako argument vektor objektů `Vector2`, které reprezentují souřadnice uzlů.

Kód lze vyplnit do předpřipravené šablony, která je v souboru `src/logic/hulls/custom/CustomComputerTemplate.h` a nebo šablonu zduplikovat. V případě tvorby vlastního souboru s počítadlem obálky je nutné cestu k souboru přidat do souboru `src/logic/hulls/custom/CMakeLists.txt` do proměnné `CUSTOM_HULL_COMPUTERS`. Také je nezbytné přidat počítadlo obálky do listu v souboru `src/logic/hulls/custom/CustomHullComputers.h`.

Zde je třeba přidat direktivu `#include` pro nový soubor na začátek souboru a poté do funkce `constructHulls()` přidat příslušný řádek.

Po provedení těchto kroků by se mělo nově přidané počítadlo automaticky nabídnout ve výběru počítadel na horní liště aplikace.

Kapitola 5
Detaily implementace

V této kapitole popíšu použité nástroje a knihovny, strukturu aplikace a uvedu detailní popis použitých algoritmů.

5.1 Použité nástroje

Pro implementaci řešitele jsem zvolil jazyk C++, kvůli jeho relativní rychlosti a pro implementaci grafického rozhraní jsem zvolil knihovnu wxWidgets [21]. Zvažoval jsem použití kombinace programovacích jazyků, C++ na provádění výpočtů a Javu nebo Python pro implementaci uživatelského rozhraní, ovšem po nalezení knihovny wxWidgets jsem tento nápad opustil. Dalšími nástroji použitými pro vývoj aplikace jsou CMake [3], knihovna pro manipulaci s JSON formátem [12] a knihovna pro výpočetní geometrii CGAL [6].

5.1.1 wxWidgets

wxWidgets je multiplatformní open-source C++ knihovna pro tvorbu uživatelských rozhraní. Tato knihovna také umožňuje ovládání z Pythonu, Perlu, Ruby a dalších jazyků. wxWidgets používá celá řada známých aplikací, například Audacity, Code::Blocks či FileZilla. Styl oken a ovládacích prvků závisí na operačním systému, jelikož pro svou funkci používá nativní API platformy na které je spuštěna. Tuto knihovnu jsem vybral právě pro podporu více platforem a využití nativního API systému. Dalšími důvody jsou detailní dokumentace, která u některých jiných C++ GUI frameworků chybí a permisivní licence pod kterou je wxWidgets distribuován.

wxWidgets je relativně velká knihovna a proto jsem se rozhodl ji nepřidávat do projektu samotného. Pro úspěšnou kompilaci je tedy nutné mít knihovnu staženou a nainstalovanou na svém systému.

5.1.2 nlohmann/json

Pro ukládání a nahrávání přednastavení a testovacích případů jsem zvolil formát JSON. Hlavní výhodou tohoto formátu je, že vygenerované soubory jsou jednoduše čitelné pro člověka. Některé jiné programovací jazyky (například Python) mají zabudovanou podporu pro tento formát, ovšem C++ mezi

ně nepatří. Pro tyto účely jsem využil knihovny pro manipulaci s JSON formátem a soubory, kterou vytvořil Niels Lohmann.

Tuto knihovnu jsem zvolil, jelikož je stále udržována, komunita okolo ní je aktivní a je distribuována pod MIT licencí. Dalším faktorem bylo relativně jednoduché použití a detailní dokumentace se spoustou příkladů. Tato knihovna je napsána pro C++ a její implementace je v jediném souboru, což zjednodušuje její použití. Tuto knihovnu dokáže stáhnout nástroj CMake při kompilaci, takže není potřeba s tím zatěžovat uživatele.

5.1.3 Výpočetní geometrie

Aplikace musí poskytovat rozhraní pro definici komplexnějších tvarů náhodných oblastí než jsou obdélníky a elipsy. Proto jsem do aplikace přidal možnost definovat jednoduchý polygon. Generování náhodných dat do takového polygonu vyžaduje schopnost určit, zda je daný bod uvnitř, či vně polygonu. Pro tento účel existuje celá řada algoritmů, například algoritmus Ray casting, který popíšu v kapitole 5.3.2.

S možností definovat jednoduchý polygon přibyla do aplikace možnost definovat téměř jakýkoliv planární útvar. Ovšem z uživatelského hlediska není tento způsob definice některých útvarů příliš pohodlný. Proto jsem do aplikace přidal možnost provádět množinové (někdy také označované jako booleovské) operace. Implementace takových operací již není triviální a proto jsem začal hledat knihovnu, která by výsledky těchto operací počítala za mě.

Na internetu existuje celá řada knihoven, které obsahují takovouto funkcionalitu. Jako první jsem vyzkoušel knihovnu Boost [1], což je obsáhlá knihovna obsahující algoritmy z celé řady oblastí. Je vysoce zaměřená na výkon a obecnost dodávané funkcionality a je vydávána pod velmi permisivní licencí. Při implementaci řešení s Boostem jsem ovšem narazil na několik nepříjemných vlastností této knihovny. Dokumentace ke knihovně je velmi obecná a chybí v ní příklady použití. Obecnost funkcí nepřidává na jejich čitelnosti a různé části Boostu nejsou vzájemně kompatibilní. Například modul, který umožňuje definovat útvary v ploše a zjišťovat, zda je bod uvnitř, či vně útvaru není kompatibilní s modulem, který umožňuje provádět množinové operace.

Po mnoha nezdařených pokusech o poskládání kryptických funkcí do funkčního celku jsem snažení s touto knihovnou vzdal a přesunul se k dalšímu kandidátovi, knihovně CGAL.

CGAL

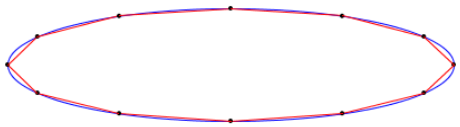
CGAL je open source knihovna algoritmů pro výpočetní geometrii, kterou společně vytváří celá řada evropských univerzitních institucí [6]. Protože **AAC** je open-source, tak je tato knihovna k disposici pod GPL licencí. CGAL poskytuje celou řadu modulů. Pro účely AAC jsem použil modul pro definici a zpracování polygonů a modul pro dvoudimenzionální regularizované booleovské operace.

S touto knihovnou se mi pracovalo výrazně lépe než s knihovnou Boost. Moduly, které jsem pro AAC využíval byly vzájemně kompatibilní a použití

funkcí bylo relativně jednoduché. Přiložená dokumentace obsahovala množství příkladů ze kterých bylo na první pohled jasné použití a význam knihovních funkcí.

Ovšem i tato knihovna má některé nedostatky, které bylo nutné obcházet. Příkladem může být nedostatečná flexibilita modulu pro booleovské operace vzhledem ke křivkám. Jedním z útvarů, které může uživatel v AAC definovat je i elipsa. Bohužel modul pro booleovské operace umožňuje práci pouze s polygony. Proto bylo nutné v aplikaci aproximovat elipsu polygonem, což v závislosti na detailu polygonu může mít fatální vliv na přesnost výsledné oblasti.

Aproximace rovnoměrným rozložením uzlů po obvodu elipsy není příliš výhodná. Ztráty rozlišení jsou velmi výrazné na ostřejších křivkách, jak je vidět na obrázku 5.1. Tento problém jsem vyřešil pomocí algoritmu popsaného v sekci 5.3.4.



Obrázek 5.1: Aproximace rovnoměrným rozložením bodů po obvodu elipsy

Jednou z komponent algoritmu konkávní obálky je i nalezení nejbližších sousedů. Knihovna CGAL tuto funkcionalitu neposkytuje. V průběhu implementace jsem narazil na knihovnu QGIS [17], která obsahuje jak algoritmus hledání nejbližších sousedů, tak množinové operace nad polygony. Bohužel jsem již měl velkou část knihovny hotovou a proto jsem knihovnu QGIS nepoužil, nicméně její použití by vše velmi zjednodušilo.

5.2 Struktura aplikace

Aplikaci jsem rozdělil na dvě části:

- Výpočty a logiku
- Zobrazení a uživatelské rozhraní

Hlavním účelem této separace bylo odstranit z logické části veškerou návaznost na část zobrazovací, aby bylo jednoduše možné vyměnit zobrazovací vrstvu za jinou. To otevírá možnost distribuovat logickou část jako knihovnu zdrojových souborů. Díky tomu by například mohlo být možné provádět výpočty z jiných programů, vyhodnocovat testovací případy pouze v konzoli, nebo přesunout práci nad výpočty do jiného vlákna pro zvýšení efektivity. Takovéto rozdělení také umožňuje implementovat automatické unit testy.

Chtěl jsem, aby byla aplikace rozšiřitelná o nové tvary oblastí a počítadla obálek. Proto bylo nutné vytvořit dostatečnou vrstvu abstrakce, aby přidání nových funkcionalit nevyžadovalo přepis velké části aplikace. Bohužel jsem

nepřišel na způsob jak dynamicky instancovat třídy bez úpravy řídícího kódu což značně komplikuje přidávání nových kalkulátorů pro běžného uživatele. Některé jiné programovací jazyky obsahují takovouto funkcionalitu (například reflexe v Javě).

Logická část aplikace je zcela ovladatelná pomocí třídy `World`. Ta poskytuje funkce pro přidávání nových antén a oblastí, generování instancí, výpočet řešení, ukládání a nahrávání oblastí atp. Data získaná z logické části pomocí těchto funkcí jsou chráněna proti neoprávněnému zapisování klíčovými slovy `const`, aby se minimalizovala šance nesprávného použití knihovny.

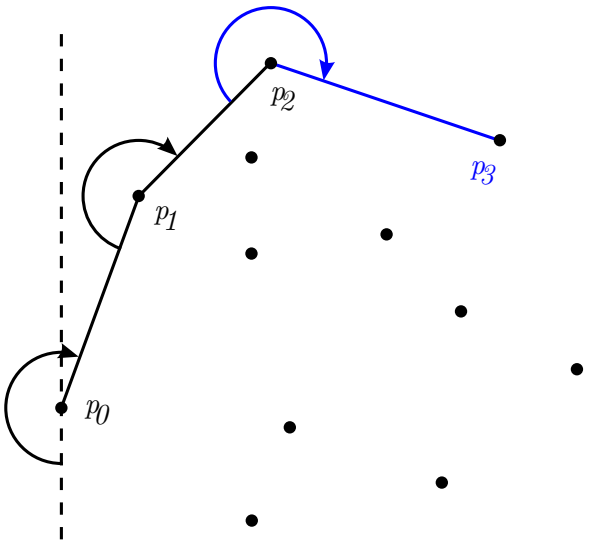
5.3 Použité algoritmy

Při implementaci knihovny jsem využil několika algoritmů pro získání dílčích výsledků. V této sekci se budu zabývat popisem jejich fungování.

5.3.1 Jarvisův algoritmus výpočtu konvexní obálky

Algoritmus, který je běžně označován jako Jarvis Wrapping. Jedná se o jeden ze základních algoritmů pro výpočet konvexní obálky ve dvoudimenzionálním prostoru. Tento algoritmus má v obecném případě asymptotickou složitost $O(n^2)$. Pro dvourozměrný prostor existuje algoritmus Graham Scan [8], který má asymptotickou složitost $O(n \log n)$, nicméně ten je výrazně složitější a pro tuto aplikaci relativně nadbytečný, jelikož jiné použité algoritmy mají výrazně vyšší asymptotickou složitost. Dalším důvodem, proč zde popisuji tento algoritmus je fakt, že se na něm zakládá algoritmus pro tvorbu konkávní obálky, který uvedu v sekci 5.3.7

Jarvisův algoritmus pro sadu bodů B s délkou n hledá nejmenší konvexní polygon H, který je všechny obsahuje. Začne v uzlu, který se nachází v nějakém směrovém extrému, například uzel s minimální souřadnicí na ose y. Při každé iteraci algoritmu hledá další bod, který minimalizuje úhel mezi posledním a dalším segmentem, jak je vidět na ilustraci 5.2.



Obrázek 5.2: Jarvis Wrapping - ilustrace převzata z [10]

Algoritmus postupuje takto:

Algorithm 1: Jarvis Wrapping

Result: List H bodů, které definují obálku
Definujeme $p_{\min}$ jako index bodu s minimální souřadnicí na ose y;
Definujeme $H = \{p_{\min}\}$ a nastavíme $p_{curr} = p_{\min}$;
while *True* **do**
  for $q \in B, q \neq p_{curr}$ **do**
    for $r \in B, r \neq p_{curr}, r \neq q$ **do**
      Definujeme $\vec{u} = r - p$;
      Definujeme $\vec{v} = q - r$;
      // Pokud je úhel (p, q, r) po směru hodinových ručiček
      if $\vec{u} \times \vec{v} < 0$ **then**
        continue next q;
      end
    end
    $p_{curr} = q$;
    break for;
  end
  if $p_{curr} = p_{\min}$ **then**
    return H;
  end
  Nastavíme $H = H \cup \{p_{curr}\}$;
end

5.3.2 Ray casting

Ray casting [9] je algoritmus, který určí, zda je bod B uvnitř jednoduchého polygonu P. Pracuje na jednoduchém principu. Pokud je polygon jednoduchý (tedy nemá žádné dvě hrany, které by se protínaly) tak můžeme využít faktu, že při přechodu mezi vnitřkem a vnějškem polygonu musíme přejít lichý počet hran.

Algoritmus tedy najde jakoukoliv přímku L, která prochází bodem B a poté hledá parametrická vyjádření průsečíků přímky L s hranami polygonu P ve formě:

$$\vec{P_i} = \vec{B} + t \cdot \vec{L_u} \tag{5.1}$$

kde $\vec{P_i}$ je i-tý průsečík přímky L s hranou polygonu P, $\vec{L_u}$ je směrový vektor přímky L a t je hodnota parametru. Označme množinu T všech t, které jsou řešeními rovnice 5.1. Pokud platí:

$$|\forall t \in T, t > 0| = 2k + 1, k \in \mathbb{Z}$$

pak je bod B uvnitř polygonu P.

Zápis algoritmu Ray casting v pseudokódu:

Algorithm 2: Ray casting algoritmus

Result: *true* pokud je bod B uvnitř polygonu P, jinak $false$
$c = 0$;
for $s \in P$ **do**
  if *intersects(B, s)* **then**
    $c = c + 1$;
  end
end
if $(c \mod 2) = 0$ **then**
  return $false$;
else
  return *true*;
end

Methoda intersects se dá zapsat jako:

Algorithm 3: Methoda intersects algoritmu Ray casting
Zdroj: Pseudokód inspirován [19] a [9]

Result: *true* pokud se paprsek z bodu P protíná s úsečkou $|AB|$,
jinak $false$
if $A_y > B_y$ **then**
  swap(A, B);
end
if $P_x >= \max(A_x, B_x) \vee P_y < A_y \vee P_y > B_y$ **then**
  return $false$;
end
if $P_x < \min(A_x, B_x)$ **then**
  return *true*;
end
if $A_x \neq B_x$ **then**
  $r = \frac{B_y - A_y}{B_x - A_x}$;
else
  $r = \infty$;
end
if $A_x \neq P_x$ **then**
  $b = \frac{P_y - A_y}{P_x - A_x}$;
else
  $b = \infty$;
end
return $b \geq r$;

5.3.3 Hledání řešení

Řešení úlohy obnáší pro dvojici vysílačů $a, b \in S$ nalézt $c \in R$ takové, že bude $E(a, b, c)$ maximální. Prohledávání provádím hrubou silou, nicméně pro zrychlení výpočtu odřezávám některé kombinace uzlů, které budou zcela jistě

suboptimální:

Algorithm 4: Antenna/shift selection

```
Result: Množina U všech řešení úlohy
for a ∈ S do
    for b ∈ S, b ≠ a do
        if ‖a − b‖ ≥ d_max then
            continue;
        end
        for x ∈ R do
            if ‖a − x‖ > 2d_max ∧ ‖b − x‖ > 2d_max then
                continue;
            end
            β̃ = argmax_β E(a, b, x, β);
            if E(a, b, x, β̃) ≥ E_min then
                U = U ∪ {(a, b, x)};
            end
        end
    end
end
```

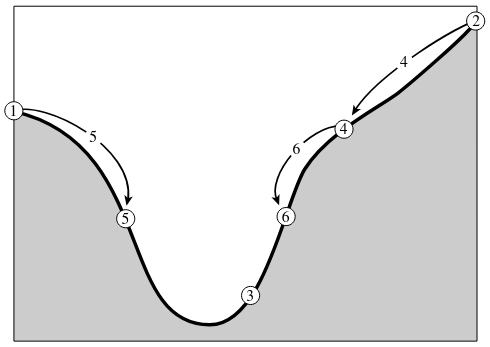
Zde $d_{\max}$ je maximální dosah jedné antény. Pokud jsou od sebe antény vzdáleny více než $d_{\max}$ je komplikované synchronizovat časy vysílání a fázový posun (zmíněno v kapitole 2.4), proto tyto případy pro jednoduchost nepočítám. Dalším zjednodušením je odřezávání přijímačů, které jsou příliš daleko od dvojice vysílačů (zmíněno v kapitole 2.1.3).

Zmíněnou funkci argmax jsem implementoval pomocí **Golden-section search** [15] algoritmu.

Golden-section search algoritmus

Tento numerický algoritmus pro hledání extrému funkce vyvinul v roce 1953 Jack Carl Kiefer. Zvolil jsem jej, jelikož nevyžaduje znalost první derivace kriteriální funkce. Algoritmus funguje na podobném principu jako algoritmus **binárního půlení** [16], který se používá pro nalezení nulového bodu. Binární půlení zmenšuje interval, na kterém se nachází nulový bod dokud interval nedosáhne požadované přesnosti. Zmenšování intervalu dociluje tím, že rekurzivně porovnává hodnoty na krajích dvou podintervalů, přičemž využívá podmínky spojitosti a monotonity na vstupním intervalu.

Golden-section search algoritmus hledá maximum spojité funkce f na zadaném intervalu. Na rozdíl od binárního půlení, tento algoritmus funkci vyhodnocuje ve třech bodech: dvou krajních bodech a, c a jedním vnitřním bodem b. Poté vyhodnotí ještě jeden bod x, který zvolí z intervalu (b, c). Jestliže $f(x) > f(b)$, pak musí být maximum funkce v intervalu (b, c), jinak je maximum zcela jistě na intervalu (a, x). Pokud tento algoritmus opakujeme rekurzivně, lze nalézt maximum relativně rychle.



Obrázek 5.3: Obrázek pro verzi algoritmu, hledající minimum funkce. Na začátku je minimum v intervalu $(1, 2)$. Jako body b, x byly zvoleny 3 a 4. Jelikož $f(3) < f(4)$, tak je minimum v intervalu $(1, 4)$. Po provedení dalších kroků algoritmu zaznamenaných na obrázku je minimum v intervalu $5, 6$.
Zdroj: Obrázek převzat z [15]

Volba bodů b a x není náhodná. Z důkazu uvedeném v [15] plyne, že pro obecný případ je optimální volit pozice bodů b, x tak, aby:

$$\frac{b-a}{c-a} = \frac{3-\sqrt{5}}{2} \approx 0.38197$$

Jinými slovy volit bod b tak, aby byl v poměrné vzdálenosti 0.38197 od bodu a. Tím je zaručeno, že každý krok algoritmu zmenší velikost prohledávaného intervalu na 0.61803-násobek velikosti původního intervalu. Tato hodnota se nazývá zlatý řez a pro její estetické vlastnosti ji využívali již staří řekové.

Jako ukočnovací podmínku jsem zvolil hodnotu tolerance $1 \cdot 10^{-6}$, která docílí přesnějších výsledků, než dříve používaný optimalizátor ve Wolfram Mathematice. Pseudokód algoritmu Golden-section search:

Algorithm 5: Golden-search algorithm pro hledání optimálního úhlu β

Result: Odhad optimální hodnoty β
$gr = \frac{1+\sqrt{5}}{2}, a = 0, b = 2\pi;$
$c = b - \frac{b-a}{gr}, d = a + \frac{b-a}{gr};$
while $|b - a| > 1 \times 10^{-6}$ **do**
  if $f(c) > f(d)$ **then**
    $b = d;$
  else
    $a = c;$
  end
  $c = b - \frac{b-a}{gr};$
  $d = a + \frac{b-a}{gr};$
end
$\beta = \frac{a+b}{2};$

5.3.4 Adaptivní distribuce uzlů na obvodu elipsy

Jak je vidět z obrázku 5.1, rovnoměrné rozložení uzlů po obvodu elipsy neposkytuje příliš dobrou aproximaci elipsy. Výsledek takového algoritmu je příliš detailní na částech, které takové rozlišení nevyžadují a naopak. Mnohem lepších výsledků lze dosáhnout pomocí aproximace polygonem, který má všechny vnitřní úhly stejné. Jinými slovy polygon, který má více uzlů na prudkých křivkách elipsy a menší množství bodů na rovnějších částech elipsy.

Naštěstí je elipsa jednoduše definovaný útvar, který má dvě osy souměrnosti. Výpočet se tedy dá redukovat pouze na jeden kvadrant elipsy a ten poté pouze ozrcadlit podle os. Vstupy tohoto algoritmu jsou:

- $N \in \mathbb{N}$ - počet bodů na kvadrantu elipsy
- C_x, C_y - souřadnice středu elipsy
- a - délka hlavní poloosy
- b - délka vedlejší poloosy

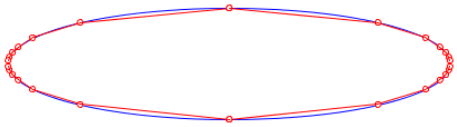
Pseudokód algoritmu, inspirovaný [4]:

Algorithm 6: Algoritmus generující body na kvadrantu elipsy

Result: Množina bodů M, uzlů aproximačního polygonu kvadrantu elipsy

$M = \emptyset$;
$i = 0$;
while $i < N$ **do**
  $\theta = \frac{\pi i}{2N}$;
  $\phi = \frac{\pi}{2} - \arctan\left(\tan\left(\theta\right) \cdot \frac{a}{b}\right)$;
  $x = C_x + a \cdot \cos\phi$;
  $y = C_y + b \cdot \sin\phi$;
  $M = M \cup [x, y]$;
  $i = i + 1$;
end

Výsledná aproximace elipsy může mít mnohem méně uzlů při zachování detailu elipsy oproti rovnoměrnému rozložení.



Obrázek 5.4: Výsledek běhu algoritmu pro adaptivní distribuci bodů na elipse

5.3.5 Generování antén

Problém generování antén v oblastech je vlastně problémem generování náhodých bodů v útvarech v ploše. Kvůli již zmíněným plánům na rozšíření o další tvary oblastí jsem zvolil obecný algoritmus pro generování bodů, který

bude fungovat pro libovolný útvar v s nenulovou plochou. Volba padla na algoritmus rejection sampling [2]. Pro zjednodušení jsem zvolil uniformní náhodné rozložení, které nepotřebuje žádnou korekci. Tento algoritmus pro své fungování potřebuje obdélníkové ohraničení útvaru L a funkci $contains(x)$, která pro bod x rozhodne zda je uvnitř útvaru či nikoliv.

Algorithm 7: Generating antennas

Result: Množina Q náhodných bodů v útvaru
$Q = \emptyset$;
while $|Q| < count$ **do**
  while *True* **do**
    $P_x = L_x + random(0,1) \cdot L_w$;
    $P_y = L_y + random(0,1) \cdot L_h$;
    if $contains(P)$ **then**
      $Q = Q \cup \{P\}$;
    end
  end
end

Jak je vidět z pseudokódu, tento algoritmus může v závislosti na náhodně vygenerovaných datech běžet velmi (potenciálně nekonečně) dlouho. Nedeterminističnost tohoto algoritmu vyvažuje výhoda jeho obecnosti. Ovšem v celé řadě útvarů v ploše lze generovat náhodné body deterministicky. Z tohoto důvodu může třída implementující oblast přepsat funkci pro generování náhodných bodů svojí vlastní implementací. Obě existující třídy pro výpočet oblastí tak činí, jelikož generování náhodných bodů v obdélníkové oblasti je triviální a generovat body do elipsy lze také v determinisitickém čase pomocí polárních souřadnic.

5.3.6 QuadTree

QuadTree je datová struktura, která umožňuje efektivně ukládat a vyhledávat body v ploše. Používá se například pro kompresi obrazu a umožňuje rychle vyhledávat nejbližší sousedy.

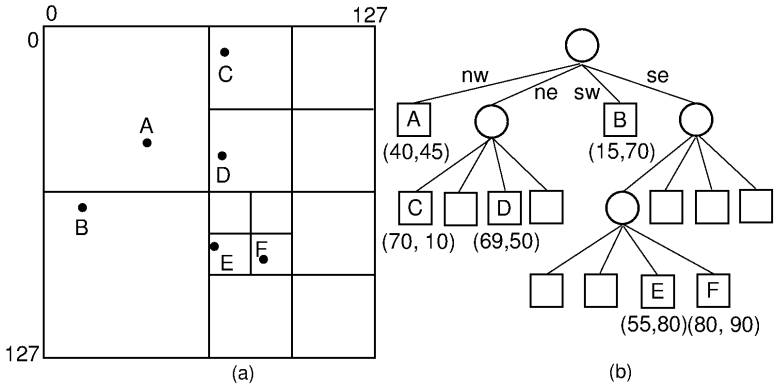
Jedná se o rekurzivní stromovou strukturu, kde každý uzel reprezentuje vymezenou část plochy a obsahuje následující data:

- Obdélník A, který uzel reprezentuje.
- Množinu D, která obsahuje reference na data, která uzel obsahuje.
- Množinu C, referencí na uzly, které jsou dětmi tohoto uzlu.

Množina C má pouze dvě možné velikosti: $|C| \in \{0, 4\}$ a všechny uzly zpravidla mají stejnou velikost množiny $|D| \leq q$, kde q nazveme kapacitou uzlu. Uzly v množině C reprezentují podmnožiny oblasti reprezentované jejich rodičem a data v množině D musí být z obdélníka A.

Prázdný QuadTree se skládá pouze z jednoho kořenového uzlu, jehož $C = D = \emptyset$ a jehož obdélník A je celá oblast na které se budou vstupní data

nacházet. Nad QuadTree používám operace **Insert** a **Search**, jejichž časová složitost je logaritmická vůči počtu uzlů v QuadTree.



Obrázek 5.5: Vizualizace struktury QuadTree
Zdroj: Stránky univerzity Virginia Tech [18]

Při operaci vkládání se rekurzivně najde první uzel, který má $|D| < q$ a jehož obdélník A obsahuje i pozici vkládaného datového bodu a uzel se přidá do jeho množiny D. Operaci vkládání popisuje algoritmus 8. Funkce **CreateNodes**(R) vrátí množinu se čtyřmi prázdnými uzly, jejichž obdélníky rozdělují oblast R_A na čtyři stejně velké části.

Algorithm 8: *Insert*(R, p)

```
Data: Kořen QuadTree R, vkládaný datový bod p
Result: true pokud byl p úspěšně vložen do R
if Contains(R_A, p) = false then
    return false;
end
if |R_D| < q then
    R_D = R_D ∪ {p};
    return true;
end
if |R_C| = 0 then
    R_C = CreateNodes(R_A);
end
for c ∈ R_C do
    if Insert(c, p) = true then
        return true;
    end
end
return false;
```

Operace **Search** vyhledává nejbližší sousedy dané souřadnice a přijímá jako argument strukturu **Query**. Tato struktura obsahuje následující informace:

- Bod X, jehož sousedy hledáme.
- Počet sousedů k, který chceme najít.

- Maximální vzdálenost r ve které chceme hledat.

Algoritmus 9 opět rekurzivně prochází uzly QuadTree, které mohou obsahovat nějaký relevantní datový bod.

Algorithm 9: *Search*(R, Q)

```
Data: Kořen QuadTree R, Query struktura Q
Result: Uspořádaná množina nejbližších sousedů
b = PriorityQueue();
if Overlaps(R_A, X, r) = false then
    return b;
end
for p ∈ R_D do
    d = Distance(p, Q_X);
    if d ≤ Q_r) then
        InsertInto(b, p, d);
    end
end
for c ∈ R_C do
    for p ∈ Search(c, Q) do
        d = Distance(p, Q_X);
        InsertInto(b, p, d);
    end
end
if Size(b) > Q_k then
    b = ResizeTo(b, Q_k);
end
return b;
```

V zápisu tohoto algoritmu používám několik funkcí a datovou strukturu *PriorityQueue*, která udržuje prvky seřazené podle hodnot kritéria. S touto strukturou souvisí tyto funkce:

- *InsertInto*(b, p, d), která do prioritní fronty b vloží prvek p s hodnotou kritéria d.
- *Size*(b), která vrací počet prvků v prioritní frontě b.
- *ResizeTo*(b, k), která odstraní prvky s nejvyšší hodnotou kritéria tak, aby *Size*(b) byl k.

Dalšími použitými funkcemi jsou *Distance*(p, X), která vrátí vzdálenost pozice datového bodu p od bodu X a *Overlaps*(A, X, r), která vrací *false* právě tehdy když $A \cap K(X, r) = \emptyset$, kde $K(X, r)$ je kružnice daná bodem X a poloměrem r.

5.3.7 Konkávní obálka

Algoritmus pro konkávní obálku jsem založil na algoritmu který vyvinuli Adriano Moreira a Maribel Yasmina Santos v roce 2007 [14]. Tento algoritmus je založen na algoritmu Jarvis Wrapping, který počítá konvexní obálku, nicméně pro výběr dalšího bodu na obálce vyhodnocuje pouze body, které jsou mezi k nejbližšími sousedy současného bodu.

Autoři ve své verzi algoritmu nejprve spouštějí nad daty clusterovou analýzu SNN [5], která od sebe oddělí shluky (clustery) uzlů a filtruje šum v datech. Tento krok je dle mého názoru nadbytečný, jelikož outlieři (data, která jsou pravděpodobně šum a nepatří do žádného clusteru) nebudou součástí senzorových sítí, kterými se zabývám. Je možné, že v praxi může být clusterová analýza prospěšná, nicméně v simulaci jsem pro zjednodušení algoritmu a zvýšení výkonu tento krok vynechal. Nad detekovanými clustery spouští autoři samotný algoritmus hledání obálky.

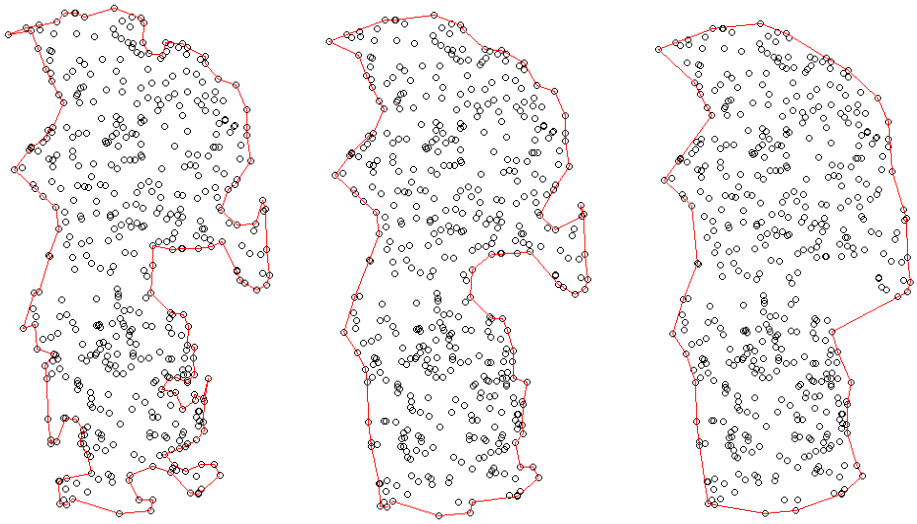
Algoritmus postupuje tak, že se pro počáteční nastavení parametru k pokusí sestavit obálku. Pokud výsledný útvar není spojitý, nebo pokud výsledný útvar není obálkou všech bodů (tj. existuje bod, který není uvnitř polygonu definovaného obálkou), pak se algoritmus spouští znovu s hodnotou parametru k o jedničku vyšší. Z tohoto popisu je patrné, že nastavení parametru k je pouze minimem počtu sousedů, které algoritmus zvažuje. Proto může pro některé intervaly k vycházet stejná obálka.

Algoritmus je rekurzivní a počet jeho volání závisí i na parametru k.

> "These results show that the time to compute the polygons increases approximately linearly with the number of points . . . The other result is that the computing time is smaller for higher values of k.[14, Kap. 4.2, str.67]

Intuitivně parametr k ovlivňuje rozlišení obálky (počet bodů na obálce), jak je popsáno v kapitole 2.4.1. Vyšší počty uzlů na obálce nejsou vhodné pro použití v senzorových sítích, nicméně velmi vysoká nastavení k neposkytují signifikantní výhody oproti konvexní obálce. Správná volba hodnoty k je tedy kritická jak z pohledu rychlosti konvergence algoritmu tak i z pohledu kvality výsledné obálky. Příklady vlivu parametru k na tvar obálky jsou na obrázcích 5.6.

Pro výpočet konkávní obálky tímto algoritmem potřebujeme efektivně hledat k nejbližších sousedů. Pro tento účel jsem zvolil datovou strukturu QuadTree, kterou jsem popsal v kapitole 5.3.6. Algoritmus prohledávání jsem upravil tak, aby přijímal dva parametry k a r, což je poloměr ve kterém se sousedi vyhledávají. V původní verzi algoritmu je $r = \infty$.



$k = 3$ $k = 10$ $k = 20$

Obrázek 5.6: Výsledky konkávní obálky pro různé hodnoty parametru k

$r = 60$ $r = 120$

Obrázek 5.7: Výsledky konkávní obálky pro různé hodnoty parametru r

Parametr r jsem do algoritmu konkávní obálky přidal na popud vedoucího práce a stručná analýza jeho efektu na výslednou obálku je v kapitole 2.4.1. Jednou z nevýhod implementace tohoto parametru je jeho chování pro nízké hodnoty. Parametr k specifikuje pouze počáteční nastavení a pro všechny jeho hodnoty je výstupem algoritmu validní obálka. Pro parametr r je tento postup nevhodný, jelikož $r \in \mathbb{R}^+$ a jeho smysluplné hodnoty se mohou řádově lišit pro různá vstupní data. Z těchto důvodů pro některé hodnoty parametru r obálka není definována. Příklady vlivu parametru r na tvar obálky jsou na

obrázcích 5.7.

Příkladem hodnot parametru r, pro které obálka množiny bodů M není definována je:

$$r < \max_{a \in M} \min_{b \in M, a \neq b} \|a - b\|$$

Nicméně obálka nemusí existovat ani pro hodnoty r, které nesplňují výše uvedenou nerovnici. Ve výstupu algoritmu se uzly na obálce neopakují a tedy se může stát, že si přesunem do odlehlého bodu odřízne cestu zpět.

Pseudokód 10 popisuje metodu **Concave**(S, r, k). V zápisu algoritmu je použito několik funkcí, které zkracují zápis:

- **LinesIntersect**(a, b, c, d) vrací *true*, pokud úsečky $|ab| \cap |cd| \neq \emptyset$.
- **AllIniside**(M, N) vrací *false*, pokud $\exists a \in M$, který není uvnitř polygonu daného N.
- **QueryPoints** využívá strukturu QuadTree a vrací množinu bodů vygenerovanou algoritmem prohledávání zmíněného v sekci 5.3.6.
- **SortByAngle** seřadí vstupy sestupně podle pravotočivého úhlu vůči poslednímu segmentu.
- **LastAngle** vrátí velikost úhlu svíraného posledními segmenty uspořádané množiny na vstuu.

Algorithm 10: Konkávní obálka

Data: Množina bodů $S_o, r \in \mathbb{R}^+, k \in \mathbb{N}$
Result: Uspořádaná množina $M \subseteq S$ bodů na konkávní obálce
if $|S| < 3$ **then**
  return *null*;
else if $|S| = 3$ **then**
  return S;
else if *We came from recursion* $\wedge\, r \neq \infty \wedge k \geq |S|$ **then**
  return *null*;
end
$k_c = \textbf{Clamp}(k,\, 3,\, |S| - 1)$;
$f = \textbf{PointWithMinY}(S)$;
$M = \{f\}, c = f, S = S_o \setminus \{f\}, \alpha = 0, s = 2$;
while $(c \neq f \vee s = 2) \wedge |S| > 0$ **do**
  if $s = 5$ **then**
    $S = S \cup \{f\}$;
  end
  $n = \textbf{QueryPoints}(S, c, k_c, r)$;
  if $n = \emptyset \wedge r \neq \infty$ **then**
    return *null*;
  end
  $n = \textbf{SortByAngle}(n, c, \alpha)$;
  $i = 0, w = \mathit{true}$;
  while $w = \mathit{true} \wedge i < |n|$ **do**
    $i = i + 1, j = 2, w = \mathit{false}$;
    if $n_i = f$ **then**
      $e = 1$;
    else
      $e = 0$;
    end
    while $w = \mathit{false} \wedge j < |M| - e$ **do**
      $w = \textbf{LinesIntersect}(M_{s-1}, n_i, M_{s-1-j}, M_{s-j})$;
      $j = j + 1$;
    end
  end
  if $w = \mathit{true}$ **then**
    return $\textbf{Concave}(S_o, r, k + 1)$;
  end
  $c = n_i, M = M \cup \{c\}, S = S \setminus \{c\}$;
  $\alpha = \textbf{LastAngle}(M)$;
  $s = s + 1$;
end
if ***AllInside***$(S_o, M) = \mathit{false}$ **then**
  return $\textbf{Concave}(S_o, r, k + 1)$;
end
return M;

Kapitola 6

Závěr

V této práci jsem stručně popsal relevantní část problematiky komunikace mezi senzorovými sítěmi pomocí beamformingu a anténních řad. Implementoval jsem simulátor **AAC**, který umožňuje rychle vygenerovat náhodná data, spočítat síly signálu a vyhodnotit algoritmy obálek.

Také jsem rozebral a otestoval nevýhody konvexní obálky pro výběr dvojic antén pro anténní řady a navrhl alternativní řešení pomocí konkávní obálky. Oba algoritmy pro tvorbu obálek jsem porovnal pomocí empirických testů za použití knihovny **AAC**, kterou jsem v rámci této práce vyvinul. Analýza ukázala, že algoritmus konvexní obálky poskytuje velmi dobrý odhad pro konvexní útvary v ploše.

Odhad pomocí konvexní obálky je však pro některé konkávní útvary nedostatečný. Nový algoritmus konkávní obálky poskytuje mnohem lepší odhad kompletního řešení, kdy jsou oba uzly na obálce. Také poskytuje lepší pravděpodobnost pro nalezení optimálního řešení. Avšak tato zlepšení jsou vyvážena výrazně vyšším podílem uzlů na obálce a tedy vyšší složitostí následného prohledávání.

Výsledky konkávní obálky lze výrazně ovlivnit úpravami jejích parametrů. Jak tyto parametry optimálně využít pro zmírnění zmíněné nevýhody může být předmětem dalšího výzkumu.

V dalších částech práce jsem stručně popsal instalaci a způsob ovládání **AAC** a možnosti, jak implementovat vlastní algoritmus pro tvorbu obálky. Následně jsem dopodrobna rozepsal strukturu, vnitřní fungování aplikace a použité algoritmy.

Aplikace a knihovna **AAC** je určena pro budoucí výzkum a proto je strukturována tak, aby šla jednoduše rozšířit o další funkcionalitu. Knihovní část aplikace byla dostatečně separována od grafické části a lze ji použít i v zcela jiném kontextu (například ze skriptovacího rozhraní). Grafická část aplikace lze jednoduše využít pro sestavení testovacích případů a skriptovací část pro vygenerování instancí, vyhodnocení a export výsledků. Tyto vlastnosti mohou být důležité pro výzkumníky, kteří budou v budoucnu vyvíjet nové varianty algoritmů pro výběr uzlů.

Bibliografie

[1] Boost. *Boost C++ Libraries*. http://www.boost.org/. 14.12.2021. 2021.

[2] George Casella, Christian P. Robert a Martin T. Wells. "Generalized accept-reject sampling schemes". English. In: *A Festschrift for Herman Rubin*. Beachwood, OH: IMS, Institute of Mathematical Statistics, 2004, s. 342–347. ISBN: 0-940600-61-7.

[3] *Program CMake*. https://cmake.org/. 6.12.2021.

[4] *Aproximační algoritmus pro kvadrant elipsy*. https://stackoverflow.com/a/22707098/17071788. 28.12.2021.

[5] Levent Ertöz, Michael Steinbach a Vipin Kumar. "Finding Clusters of Different Sizes, Shapes, and Densities in Noisy, High Dimensional Data". In: květ. 2003. DOI: 10.1137/1.9781611972733.5.

[6] Efi Fogel et al. "2D Regularized Boolean Set-Operations". In: *CGAL User and Reference Manual*. 5.3.1. CGAL Editorial Board, 2021. URL: https://doc.cgal.org/5.3.1/Manual/packages.html#PkgBooleanSetOperations2.

[7] *Repozitář s kódem na GitLabu*. https://gitlab.fel.cvut.cz/horovtom/dtt. 2.1.2022.

[8] R.L. Graham. "An efficient algorith for determining the convex hull of a finite planar set". In: *Information Processing Letters* 1.4 (1972), s. 132–133. ISSN: 0020-0190. DOI: https://doi.org/10.1016/0020-0190(72)90045-2. URL: https://www.sciencedirect.com/science/article/pii/0020019072900452.

[9] Hanjoo Cho a Young Hwan Kim. "Ray-casting algorithm and its considerations for parallel processing optimization techniques for parallel ray-casting algorithm". In: *2014 International Symposium on Integrated Circuits (ISIC)*. 2014, s. 107–110. DOI: 10.1109/ISICIR.2014.7029497.

[10] *Jarvis Wrapping ilustrace*. https://en.wikipedia.org/wiki/Gift_wrapping_algorithm#/media/File:Jarvis_march_convex_hull_algorithm_diagram.svg. 6.12.2021.

[11] J. Kubr. "Směrování v bezdrátových Ad-Hoc a senzorových sítích". Dis. pr. Praha: České vysoké učení technické v Praze, Fakulta elektrotechnická, Katedra telekomunikační techniky, 2017.

[12] Niels Lohmann. *JSON for Modern C++*. Ver. 3.10.4. URL: https://github.com/nlohmann.

[13] K. Lorincz et al. "Sensor networks for emergency response: challenges and opportunities". In: *IEEE Pervasive Computing* 3.4 (2004), s. 16–23. DOI: 10.1109/MPRV.2004.18.

[14] Adriano Moreira a Maribel Santos. "Concave hull: A k-nearest neighbours approach for the computation of the region occupied by a set of points." In: led. 2007, s. 61–68.

[15] William Press. *Numerical recipes : the art of scientific computing*. Cambridge, UK New York: Cambridge University Press, 2007, s. 492–496. ISBN: 9780521880688.

[16] William Press. *Numerical recipes : the art of scientific computing*. Cambridge, UK New York: Cambridge University Press, 2007, s. 447–449. ISBN: 9780521880688.

[17] *Knihovna QGIS*. https://qgis.org/en/site/. 28.12.2021.

[18] *Vizualizace QuadTree*. https://opendsa-server.cs.vt.edu/ODSA/Books/Everything/html/_images/PRexamp.png. 28.12.2021.

[19] *Pseudokód algoritmu ray-casting*. https://rosettacode.org/wiki/Ray-casting_algorithm. 28.12.2021.

[20] *WSNA '02: Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications*. Atlanta, Georgia, USA: Association for Computing Machinery, 2002. ISBN: 1581135890.

[21] *Knihovna wxWidgets*. https://www.wxwidgets.org/. 6.12.2021.