VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
FACULTY OF MECHANICAL ENGINEERING

ÚSTAV AUTOMATIZACE A INFORMATIKY
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

PLÁNOVÁNÍ TRASY PRO AUTONOMNÍ ROBOTICKOU SEKAČKU
COVERAGE PATH PLANNING FOR AUTONOMOUS ROBOTIC LAWN MOWER

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR
Bc. Michal Moninec

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. Jakub Kůdela, Ph.D.

BRNO 2021

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání diplomové práce

Ústav:	Ústav automatizace a informatiky
Student:	**Bc. Michal Moninec**
Studijní program:	Strojní inženýrství
Studijní obor:	Aplikovaná informatika a řízení
Vedoucí práce:	**Ing. Jakub Kůdela, Ph.D.**
Akademický rok:	2020/21

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma diplomové práce:

Plánování trasy pro autonomní robotickou sekačku

Stručná charakteristika problematiky úkolu:

Diplomová práce se bude zabývat plánováním optimální trasy pro autonomní robotickou sekačku, s přihlédnutím na tvar oblasti, kterou má sekačka projet, a provozní charakteristiky sekačky (pracovní šířka a minimální poloměr otáčení).

Cíle diplomové práce:

Popsat problém plánování trasy.
Provést rešerši aktuálně používaných metod pro tento problém.
Pro vybranou metodu vytvořit software implementaci.

Seznam doporučené literatury:

HAMEED, Ibrahim A. Coverage path planning software for autonomous robotic lawn mower using Dubins' curve. In: IEEE International Conference on Real-time Computing and Robotics (RCAR). Okinawa, Japan, IEEE, 2017. DOI: 10.1109/RCAR.2017.8311915.

ABSTRAKT

Tato diplomová práce se zabývá problematikou optimalizace cesty autonomní robotické sekačky pro pokrytí celé oblasti, která je předem stanovena a nemění se. Dále ji tvoří rešerše aktuálně používaných metod a následuje implementace software s grafickým uživatelským rozhraním, který je schopen optimalizovanou trasu generovat.

ABSTRACT

This diploma thesis is covering the coverage path planning problem for autonomous robotic lawn mower in an area, which is fully defined before and is not changing. It contains a review of the currently used methods and an implementation of a software with a graphic user interface, which is capable of generating optimalized path.

KLÍČOVÁ SLOVA

Optimalizace sekací trasy, shluková analýza, dynamické programování, viditelnostní graf, genetické algoritmy.

KEYWORDS

Coverage path planning, cluster analysis, dynamic programming, visibility graph, genetic algorithms.

ÚSTAV AUTOMATIZACE A INFORMATIKY

2021

BIBLIOGRAFICKÁ CITACE

MONINEC, Michal. *Plánování trasy pro autonomní robotickou sekačku.* Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav automatizace a informatiky, 2021, 71 s. Diplomová práce. Vedoucí práce: Ing. Jakub Kůdela, Ph.D.

ČESTNÉ PROHLÁŠENÍ

Prohlašuji, že tato práce je mým původním dílem, vypracoval jsem ji samostatně pod vedením vedoucího práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury.

Jako autor uvedené práce dále prohlašuji, že v souvislosti s vytvořením této práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků.

V Brně dne 21. 5. 2021 .....

Michal Moninec

PODĚKOVÁNÍ

Tímto bych chtěl poděkovat Ing. Jakubovi Kůdelovi, Ph.D. za vedení a cennou pomoc při vypracování závěrečné práce. Zároveň bych chtěl velice poděkovat rodině a blízkým přátelům za podporu v průběhu celého studia.

OBSAH

1 ÚVOD

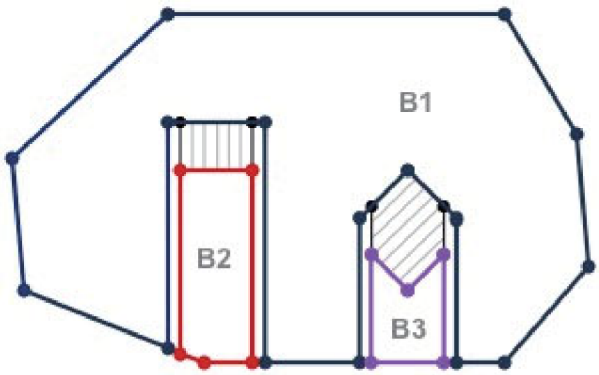
Robotické autonomní sekačky se v dnešní době stále více dostávají do povědomí věřejnosti, díky jejich dostupnosti a zvyšující se sofistikovanosti. S ohledem na tento trend se zvyšují nároky požadavků pro tyto stroje a to nejen z hlediska designu, výkonu, kvality, ale také z pohledu efektivního využití dané plochy. S tímto požadavkem nastává problém optimalizace trasy, která musí projít celou oblastí sekaní v nejkratší možné délce.

Tato práce se zabývá výše zmíněnou problematikou a hledáním optimální cesty s ohledem na možné překážky v předem definovaném prostoru, který se již v průběhu nemění, a softwarovou implementací nástroje pro řešení tohoto problému. Úvodní kapitola popisuje příklady používaných metod, zvolený přístup a specifikace tohoto problému. Následuje teoretický popis oblastí, které jsou využity v této práci. V praktické části je detailní popis softwarové implementace, výčet použitých technologií a prezentace výsledků. Výsledkem praktické části je software s grafickým uživatelským rozhraním, který je schopen pro oblast reprezentovanou GPS („global positioning system“) souřadnicemi najít optimalizovanou cestu, tak aby sekačka projela celou plochu. Závěrečná kapitola shrnuje dosažené výsledky a možné náměty na vylepšení. Z důvodu rozsahu práce jsou v teoretické části detailně popsané pouze použité metody a ostatní přístupy jsou jen zmíněny.

2 POUŽÍVANÉ METODY A KONCEPT ŘEŠENÍ

Problematikou optimalizace cesty pokrývající celou oblast se již zabývalo množství autorů. Z hlediska adekvátnosti k tématu této práce se jeví [9] jako velice dobře zpracovaná a dosahuje přijatelných výsledků. Z tohoto důvodu je tato práce inspirována prvky, které se ve zmiňovaném článku objevují s obohacením o některé funkcionality. Pro porovnání je vybráno další aplikační řešení [26], která se zabývá stejným problémem, ale s jiným praktickým využitím.

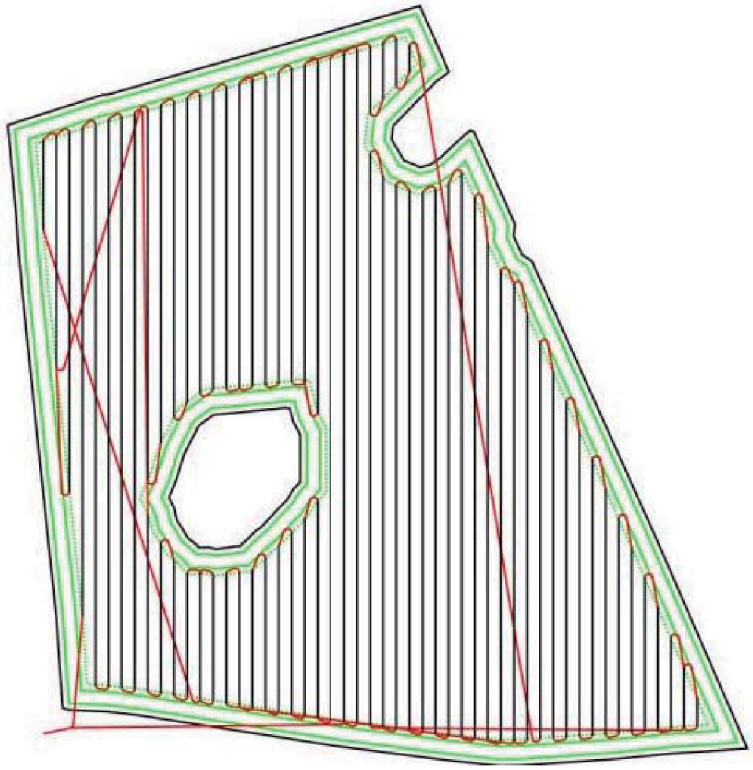
V [9] se autor Ibrahim A. Hameed zaobírá problémem optimalizace cesty pokrytí oblasti (CPP – „coverage path planning“) pro autonomní robotickou sekačku pomocí Dubinsových křivek. Vysoká úroveň poznání v dané oblasti problematiky je zjevná jak z tohoto článku, tak faktem, že se autor této oblasti věnuje i v několika dalších publikacích [10, 11, 8], nebo také v [7]. Autor zde zvolil přístup rozdělení oblasti na podoblasti s jednoduchou nejkratší vnitřní cestou. Generování těchto částí je na základě prolnutí plochy paralelními křivkami, příčemž přejezdy mezi jednotlivými čárami jsou vyhlazeny Dubinsovými křivkami s ohledem na poloměr otáčení sekačky. Dále jsou horní či spodní body těchto křivek následně roztříděny pomocí shlukové analýzy, kdy autor použil konkrétně metodu *k-means* a příklad výsledného rozdělení lze vidět na obr. 1. Na tomto obrázku můžeme vidět sekací plochu, tvořenou třemi podoblastmi B1, B2 a B3, které může sekačka projet aniž by se musela zjišťovat nejkratší cesta uvnitř těchto dílčích oblastí.



Obr. 1: Rozdělení na podoblasti pomocí k-means [9].

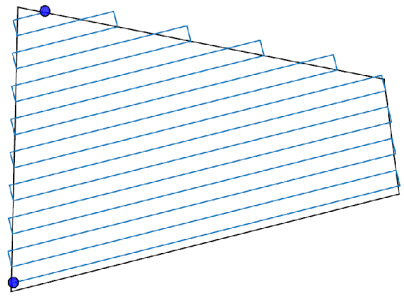
Následně probíhá hledání ideálního pořadí jednotlivých podoblastí z hlediska minimalizace ujeté cesty pomocí genetického algoritmu. Autor bohužel neuvádí jakým způsobem získává nejkratší cestu pro jednotlivé pořadí oblastí, ani strukturu evolučního algoritmu, nebo použité operátory. Dále není v práci zmíněna metoda hledání cesty mezi koncovým bodem předchozí oblasti a počátečním bodem následující oblasti s ohledem na překážky v prostoru.

Výsledkem této publikace je cesta kompletně pokrývající plochu určenou k sekání, kterou se může autonomní robotická sekačka řídit bez jakéhokoliv vnějšího zásahu člověka. Ukázku můžeme vidět na obr. 2, kde jsou hranice sekacího prostoru vyznačeny zelenou barvou a trajektorie robota barvou červenou. Implementace je zhotovena pomocí výpočetního prostředí `MatLab` a programovacího jazyka `Python` [9].



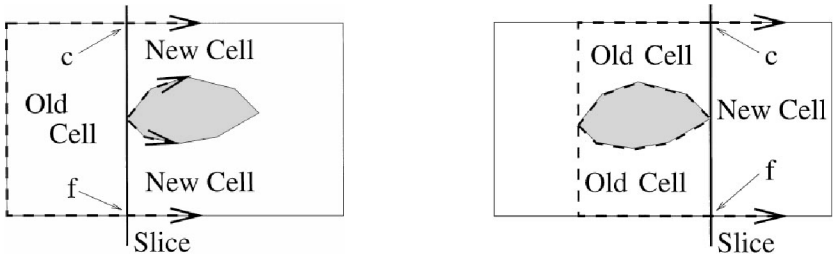
Obr. 2: Výsledná trajektorie autonomní robotické sekačky [9].

V článku [26] se autoři věnují koncepčně stejnému problému, ovšem v kontextu prohledávání prostoru pomocí automatických vzdušných vozidel, známých taktéž jako drony, zejména pro použití při pátrání či záchraně osob. Popisují prohledávání oddělených oblastí, kdy je důraz kladen na nalezení nejkratší cesty a také na rychlost hledání pro použití přímo za letu. Pohyb uvnitř jednotlivých oblastí je zde vyřešen vzorem „tam a zpět“, jak je popsáno na obr. 3, kdy pro pevně daný úhel dron letí až k hraně oblasti, kde provede obrat. Odsazení rovnoběžných cest mezi body obratu je dáno parametry kamery, která je připevněná zespod vozidla a snímá prostor pod sebou za účelem nalezení hledané osoby, případně objektu. Následuje nalezení ideální posloupnosti oddělených oblastí pro minimalizaci dráhy přejezdů mezi jednotlivými plochami, což popisují autoři jako problém obchodního cestujícího (TSP - „traveling salesman problem“). V tomto článku jsou uvedeny tři konkrétní možnosti řešení optimalizace nejkratší cesty.



Obr. 3: Vzor prohledávání „tam a zpět“ [26].

První možností je rozklad na buňky, které reprezentují jednotlivé mnohoúhelníky. Při inicializaci tvoří celková plocha jednu buňku. Rozdělení probíhá prohledáním prostoru čárou, která jej celý protíná, pohybem z jedné strany na druhou. Při přerušení celistvosti čáry o některou z vnitřních překážek dochází k vytváření nových buněk. Po opětovném spojení čáry, která prolíná prostor, dochází k uzavření buněk a vytvoření nové. Proces tvoření buněk je znázorněn na obr. 4. Takto se prohledá celá oblast a je rozdělena na buňky s jednoduchou vnitřní cestou, která se dá vytvořit vzorem „tam a zpět“, viz obr. 3. Dále je řešena úloha jako TSP pomocí genetického algoritmu.



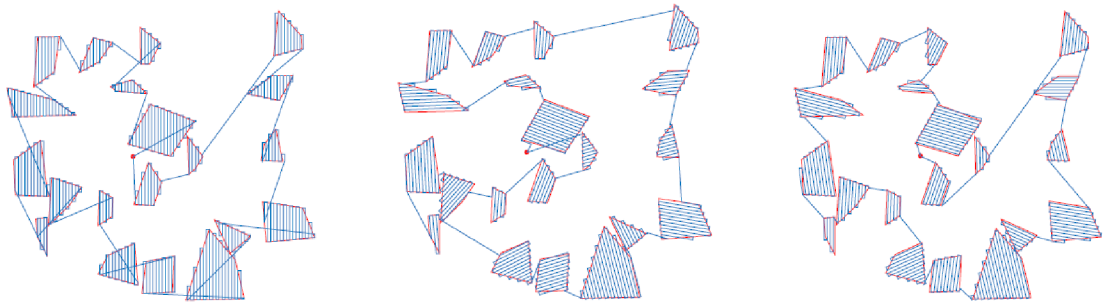
Obr. 4: Průběh buněčného dělení prostoru [26].

Další metodou je strategie dvou kroků, kterou uvádí autoři jako předmět jejich publikace. První krokem je určení pořadí navštívených oblastí. Tato část je řešena pomocí genetického algoritmu jako problém obchodního cestujícího, kdy ohodnocení pořadí je vzdálenost přejezdových bodů daných oblastí, pričemž pro usnadnění jsou použity místo konkrétních bodů vstupu a výstupu středy ploch objektů. Dalším krokem je vytvoření cesty uvnitř jednotlivých oblastí, kde autoři využívají algoritmu rotačního třmenu, který je výsledkem jejich předchozí práce. Detailní popis této metody je možné najít v [27]. Tato metoda plánování cesty pro jednotlivou oblast na základě vstupního a výstupního bodu nalezne optimální cestu uvnitř mnohoúhelníku a výstupem je tato nejkratší trasa a úhel natočení rovnoběžných čar.

Jako vstupní bod je použit střed předchozí oblasti, nebo počáteční bod, jedná-li se o první prohledávanou plochu. Výstupním bodem je pak střed oblasti následující.

Poslední použitou metodou je použití dvojice genetických algoritmů. Princip celkové optimalizace je zde rozdělen do dvou částí a to nalezení optimálního pořadí oblastí a jejich ideální úhel natočení. Obě tyto části jsou řešeny zvlášť pomocí genetického algoritmu, přičemž ta první je stejná jako u metody dvou kroků. Při určování pořadí je hodnocení kvality vzdálenost přejezdových bodů daných oblastí, kdy pro usnadnění používají místo konkrétních bodů vstupu a výstupu střed plochy objektu. Pro optimální nalezené pořadí následuje výpočet úhlu, který svírají paralelní čáry uvnitř oblastí s pomyslnou vertikálou.

Závěrem autoři porovnávají jednotlivé metody, kdy výsledky využitých přístupů můžeme vidět na obr. 5. Z jejich srovnání vychází, že z hlediska nejkratší nalezené cesty je metoda dvou kroků nejúspěšnější, pričemž rozdíl genetického algoritmu oproti tomuto řešení je průměrně 4,4% a metoda buněčného rozkladu se liší o 5,6%.



(a) Řešení pomocí BCD (b) Řešení pomocí GA (c) Řešení pomocí TSPP

Obr. 5: Výsledky jednotlivých metod [26].

Trochu jiný přístup byl zvolen v [15], kde oproti pevně zvolenému úhlu pro všechny podoblasti jako u předchozích metod je oblast rozdělena pomocí MSA(„minimal sum of altitudes“) na několik podoblastí s rozdílnými úhly. Rozdělení probíhá protnutím prostoru paralelními přímkami ve směrech, které jsou rovnoběžné s hranami prostoru. Místa protnutí vytvoří samostatné buňky, které reprezentují vrcholy grafu a pomocí dynamického programování je graf buď rozdělen na 2, nebo sloučí všechny buňky tak, aby tvořily jednu celistvou oblast. Tento proces probíhá za účelem minimalizovat součet nejdelších možných průřezů oblastí (MSA). V [6] jsou zmíněny další metody doplnující výčet předchozích přístupů rozdělení oblasti na podoblasti, mezi ně patří například trapézoidová metoda, nebo také metoda založena na Morseho funkcích.

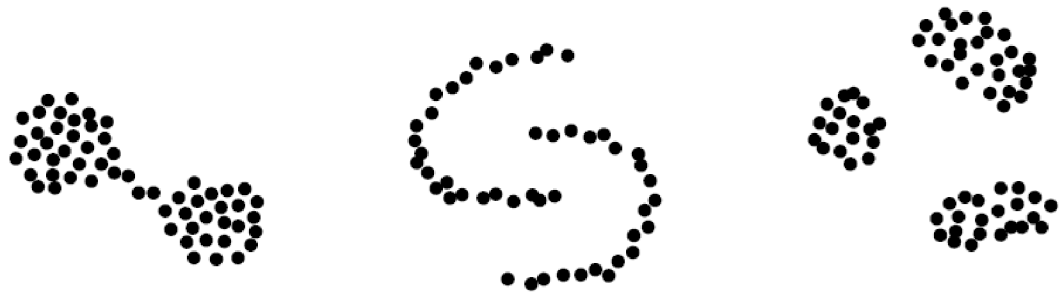
2.1 Zvolený přístup k problému

K problematice optimalizování cesty při pokrytí dané oblasti se dá přistoupit mnoha způsoby, jak zmiňují autoři v [9, 26]. Jelikož může být oblast sekací plochy pro problematiku této práce rozmanitá, je nutné rozdělit ji na několik podoblastí. Vhodnou metodou se jeví prolnutí oblasti paralelními křivkami a následné třídění jejich horních, či spodních bodů [9]. Tento proces je řešen shlukovou analýzou a následnou kontrolou, zda jsou výsledné podoblasti triviální z hlediska optimální cesty uvnitř jednotlivých částí. Úhel zůstává pro všechny segmenty stejný. Dále se tento problém transformuje na optimalizaci pořadí podoblastí, což se dá klasifikovat jako problém obchodního cestujícího [2]. U pořadí se zjistí nejlepší kombinace vjezdů a vyjezdů pomocí sestrojení grafu, ve kterém poté hledáme nejkratší cestu. Oproti aproximaci přejezdových bodů z publikace [26] jsou v této práci pro dosažení přesnějších výsledků použity reálné body vjezdu a výjezdu z oblastí. Z důvodu ošetření pohybu sekačky pouze v sekacím prostoru a možnosti vyhnout se vnitřním překážkám je tvorba cesty mezi jednotlivými oblastmi řešena viditelnostním grafem. Optimalizace pořadí je řešena genetickým algoritmem, kdy je ohodnocením dané sekvence délka celkové trasy potřebné pro projetí celého sekacího prostoru. Po získání vypočítaného pořadí se na trajektorii aplikuje vyhlazení pomocí Dubinsových křivek v souladu s poloměrem otáčení sekačky.

3 SHLUKOVÁ ANALÝZA

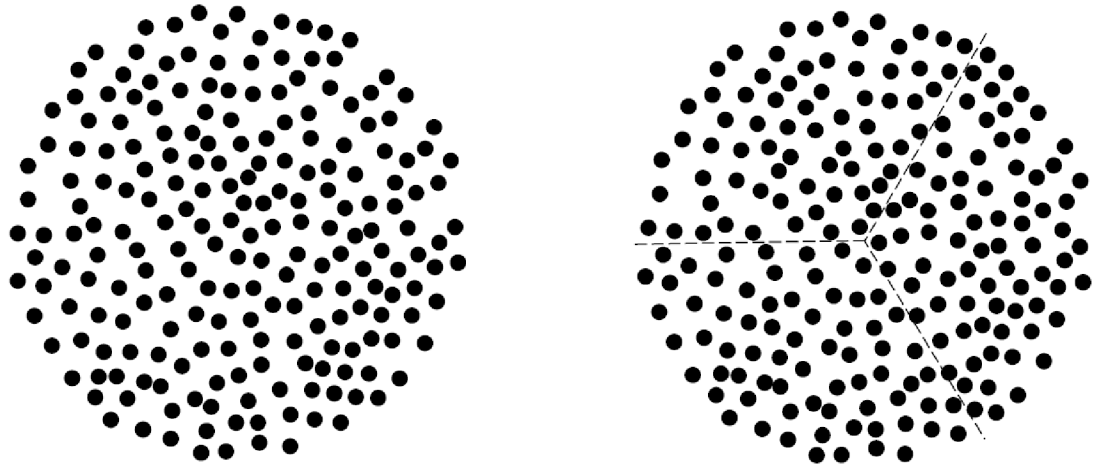
Schopnost třídit objekty do shluků dle podobností je jednou ze základních schopností důležitých pro život. Provází lidstvo již od počátků, kdy museli být lidé schopni rozdělit kupříkladu potravu na jedlou, nejedlou a jedovatou. V kontext jazyka se jedná například o vytváření synonym, nebo nadřazená slova k určitým skupinám slov. V biologii je klasifikace organismů obecně známá jako taxonomie, v chemii je zdárný příklad Mendělejevova periodická tabulka. Důvodem vytváření skupin na základě podobnosti může být například seřazení hodnot k lepšímu pochopení velkého množství dat, zjištění skrytých vzorců, nebo preferencí zákazníků. Se zvyšujícím se počtem využití obrovských databází je shluková analýza velice často vyhledávaným nástrojem.

Vzhledem k dosažení objektivních a stálých výsledků rozdělení do skupin se používají numerické metody. Jejichž výsledkem pro stejná data a použité metody musí být vždy totožné rozdělení [5]. Výhodou těchto metod je i možnost využití výpočetní techniky. Skupiny nejsou předem definované a vytvářejí se v průběhu utváření shluků [19]. Výsledné shluky vznikají na základě podobností, či rozdílností dat. Samotný shluk, či skupina, se dá popsat jako celek, který působí homogenně a izolovaně [19]. Takovéto vlastnosti se dají vyjádřit například podobně jako na obr. 6.



Obr. 6: Ukázka shluků [5].

Některé příklady použití shlukové analýzy pracují se souborem dat, který se na první pohled jeví jako jednotný a není zjevné případné rozdělení. Takovým se může zdát například obr. 7a. Kdyby jendnotlivé body měly například reprezentovat domy ve městě a cílem by bylo vytvořit podobně velké skupiny z důvodu roznášky novin, bylo by přípustné rozdělení například obdobně na obr. 7b. Převážná většina metod shlukové analýzy by zmíněné případy rozdělila také.



(a) Homogenně vypadají data

(b) Rozdělení do tři skupin

Obr. 7: Rozdělení domů z hlediska poštovní donášky [5].

Metod pro rozdělení dat do skupin je velké množství s širokou škálou možností využití. Jejich vhodnost se může lišit příklad od příkladu. Rozdělení metod používaných pro shlukovou analýzu může být velice rozsáhlý, proto je zde uveden krátký výčet metod, rozdělený dle jejich typických vlastností [5, 19]:

- grafické rozdělení (histogramy, bodový diagram, odhad hustoty, ...),
- měřením blízkosti (podobonosti, rozdílnosti dat, ...),
- hierarchické rozdělení (aglomerativní, divizivní, ...),
- optimalizační metody (***k-means***,***x-means***, *k-modes*),
- fuzzy shlukování,
- prohledávací metody shlukování.

3.1 Algoritmus *k-means*

Vytváření shluků pomocí *k-means* patří mezi nejjednodušší metody učení bez učitele pro rozdělení do skupin [9]. Řadí se mezi optimalizační metody, u kterých je snaha buď minimalizovat, či maximalizovat zvolené kritérium. Dále by se dal tento algoritmus kategorizovat jako shlukování pomocí hledání středů jednotlivých skupin [19].

Tento algoritmus byl poprvé představen MacQueenem v roce 1967 a dnešní formu tohoto algoritmu navrhli v roce 1975 Hartigan a Wong. Předpokládá se, že je možné data reprezentovat v euklidovském prostoru, dají se spočítat vzdálenosti těchto bodů a celkový počet shluků je pevně daný parametrem k. Odchylka jednotlivých bodů se bere vzdálenost dáného bodu a středu příslušného shluku. Tento střed se někdy může nazývat centroida, z čehož vychází samotný název algoritmu (z ang-

lického jazyka „means“, což znamená střed). Celková odchylka se dá matematicky zapsat následovně [19]:

$$E = \sum_{n=1}^{\infty} \sum_{n=1}^{\infty} d\left(x, \mu\left(C_i\right)\right) \tag{1}$$

Kde je $\mu(C_i)$ střed shluku C_i a $d(x, \mu(C_i))$ je vzdálenost bodu x od středu $\mu(C_i)$ a E je značí střední hodnotu. Tuto odchylku se následovně snažíme optimalizovat z hlediska hodnotícího kritéria přesunem středů $\mu(C_i)$.

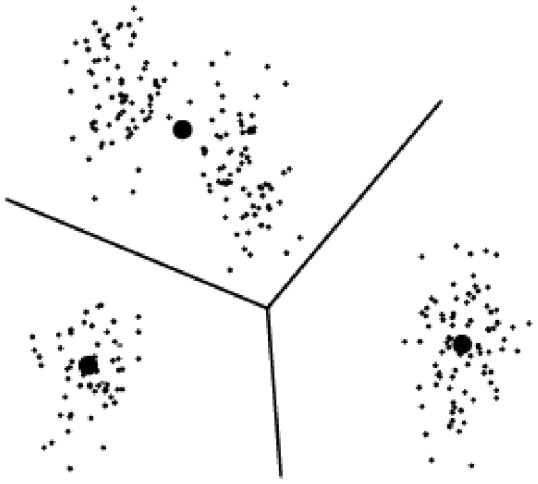
Algoritmus *k-means* se dá rozdělit na dvě fáze: inicializační část a iterační část. V první části se body přiřadí ke k náhodně zvoleným středům. V následující fázi se spočítá vzdálenost bodu k jednotlivým středům a vytvoří oblasti, pro které se spočítá těžiště a to je zvoleno jako nový střed této oblasti. Druhá část se opakuje do té doby, než změnou středu shluků nedosáhneme změny celkové odchylky. Pro lepší znázornění, je možné popsat průběh pomocí algoritmu 1 [19]:

Algorithm 1 *k-means*

1: Data set D, number of clusters k, dimensions d:
2: C_i is the ith cluster
3: $(C_1, C_2, ..., C_k)$ = Initial groups of D
4: **repeat**
5: d_{ij} = distance between point i and cluster j
6: $n_i = arg\, min_{1 \leq j \leq k} d_{ij}$
7: Assign case i to cluster n_i
8: Recompute the cluster means
9: **until** No further changes of cluster membership occur in complete iteration

Praktický výsledek přiřazení bodů k jednotlivým shlukům můžeme vidět na obr. 8, kde byl počet středů k nastaven na hodnotu 3. Výhodami *k-means* algoritmu může být například efektivnost pro velké množství dat, nebo časová závislost lineárně proporcionální k velikosti dat [19].

Jednou z nevýhod tohoto algoritmu je závislost na počátečním rozdělení středů shluků, které značně ovlivňuje dosažené výsledky. Další mohou být například nevhodnost pro vícedimenzionální data, nebo použití pouze pro numerická data, nebo nalezení lokálního minima. I přes tyto nedostatky je algoritmus velice oblíbený i po více než 50-ti letech od své formulace. Dále má *k-means* několik variant a vychází z něj řada algoritmů, jako jsou například: spojitý *k-means*, *sort-means*, *compare-means*, *k-d strom*, nebo také často používaný *x-means* [19].



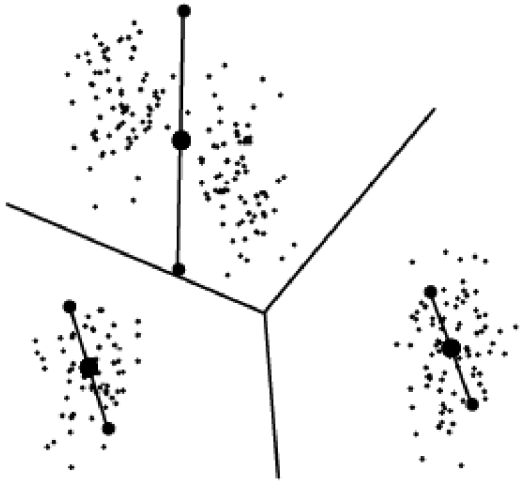
Obr. 8: Rozdělení bodů do tří skupin pomoci *k-means* [22].

3.2 Algoritmus *x-means*

Jelikož je pro algoritmus *k-means* počet skupin k definován jako vstupní parametr, není vhodný pro případy, u kterých je počet shluků předem neznámý. Z důvodu možnosti řešení právě takovýchto problémů a zlepšení výpočetní náročnosti algoritmu *k-means* přišli autoři Moore a Peleg (2000) s algoritmem *x-means*. Ve své práci uvádějí i částečné řešení pro zamezení náchylnosti k lokálnímu minimu [22].

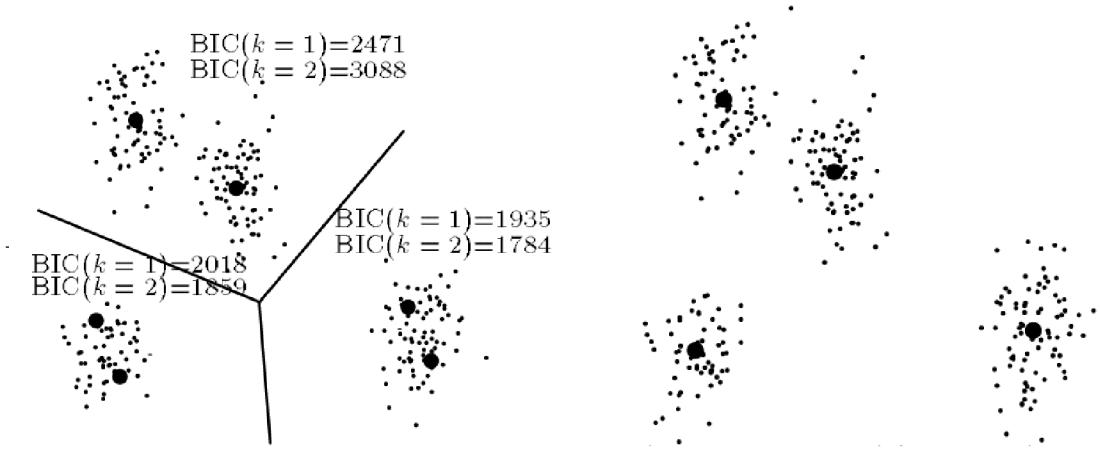
Namísto specifikování parametru k se udává rozmezí $[k_{min}, k_{max}]$, ve kterém se k má nacházet. Při inicializaci se parametr k nastaví na spodní hranici intervalu k_{min}. Následně se iterativně aplikuje algoritmus pro všechny hodnoty parametru k až do dosažení horní hranice intervalu k_{max}. V průběhu se zaznamenávají řešení a jako výsledné se vybere to, které má globálně nejlepší ohodnocení BIC („bayesian information criterion“). Část algoritmu *x-means*, která se aplikuje v každé iteraci, se dá rozdělit na dvě fáze a to: vylepšení parametrů a vylepšení struktury [22].

První fáze je pouze rozdělení do k skupin pomocí algoritmu *k-means*. Druhá fáze zjišťuje, jestli se některý shluk má rozdělit na více shluků, případně se taková skupina vybere a rozdělí se. Tento proces probíhá tak, že se pro každý shluk jeho střed rozdělí na dva potomky. Ty následně posuneme o vzdálenost proporcionální k velikosti příslušné oblasti předka v náhodně zvoleném vektoru směrem od sebe, viz obr. 9. Pro každou takovouto oblast se provede *k-means* s parametrem $k = 2$, za účelem rozdělení bodů mezi dva potomky. Tito potomci slouží jako iniciální centroidy oblasti a lokální *k-means* určí jejich finální polohu. Každá takto nově vytvořená oblast se ohodnotí pomocí BIC a porovná se s hodnotou svého předchůdce obr. 10a.



Obr. 9: Posunutí středů v náhodném směru [22].

U oblasti, která má největší rozdíl těchto hodnot se provede rozdělení a vytvoří se konečná podoba rozdělení do k skupin obr. 10b [22].



(a) BIC ohodnocení jednotlivých shluků (b) Následné rozdělení

Obr. 10: Průběh vylepšení struktury [22].

3.2.1 BIC ohodnocení

Pro vstupní data $D = \{x_1, x_2, ..., x_n\}$ a jejich případné modely $M_j = \{C_1, C_2, ..., C_k\}$ je použita pravděpodobnost po sledování $P(M_j|D)$ pro ohodnocení modelu. Pro průměrnou hodnotu je použito Schwarzovo kritérium, viz rov. (2), přičemž rovnice pro tuto podkapitolu jsou brány z [22]. Kde $\hat{l}_j(D)$ je pravděpodobnost D podle j-tého modelu v bodě největší pravděpodobnosti a p_j je počet parametrů v M_j.

Vybere se model s největším ohodnocením.

$$BIC(M_j) = \hat{l}_j(D) - \frac{p_j}{2}\log n \tag{2}$$

Při stejném gausově rozdělení je maximální odhad hodnoty pro rozptyl dán rovnicí (3). Kde $\mu_{(i)}$ je střed přiřazený k bodu x_i. Pravděpodobnost tohoto bodu je dána rovnicí (4).

$$\hat{\sigma}^2 = \frac{1}{n-k}\sum_{i=1}^{n}(x_i - \mu_{(i)}) \tag{3}$$

$$\hat{P}(x_i) = \frac{|C_{(i)}|}{n} \cdot \frac{1}{\sqrt{2\pi}\hat{\sigma}^d} exp\left(-\frac{1}{2\hat{\sigma}^2}||x_i - \mu_{(i)}||^2\right) \tag{4}$$

Celková pravděpodobnost je tedy dána rovnicí (5), přičemž počet volitelných parametrů $p_j = (d+1)k$. Výpočet BIC se používá v algoritmu *x-means* jak globálně pro výběr nejlepšího modelu, tak lokálně pro volbu oblasti, která se má rozdělit.

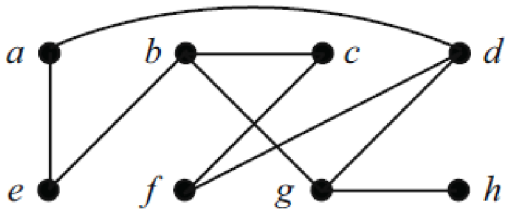
$$l(D) = log\prod_{i=1}^{n} P(x_i) = \sum\left(\log\frac{1}{\sqrt{2\pi}\hat{\sigma}^d} - \frac{1}{2\hat{\sigma}^2}||x_i - \mu_{(i)}||^2 + \log\frac{|C_{(i)}|}{n}\right) \tag{5}$$

3.2.2 Akcelerace algoritmu *x-means*

Pro zrychlení výpočtu *k-means* v jednotlivých iteracích se využívá toho, že zjištění příslušnosti samostatného bodu ke středu je stejné jako hledání příslušnosti skupiny bodů ke středu při dostatečných statistických informacích. Vzhledem k tomuto faktu autoři zvolili použití *k-d stromu* (viz 4.1) pro reprezentaci dat, kdy každý vrchol představuje podmnožinu dat. Ohraničení každého vrcholu je pomocí obdélníku tak, že jsou uvnitř všechny body podmnožiny. Dále pak obsahuje vrchol dva potomky, které představují rozdělení bodů, které jsou přiřazeny k jejich rodiči. Další vylepšení autoři popisují jako *blacklisting*, kdy zavrhnou ty středy, které nemůžou mít přiřazeny žádné body ze současného vrcholu. Obdobně je možné urychlit druhou fázi, kde dochází k lokálnímu rozdělení skupiny [22].

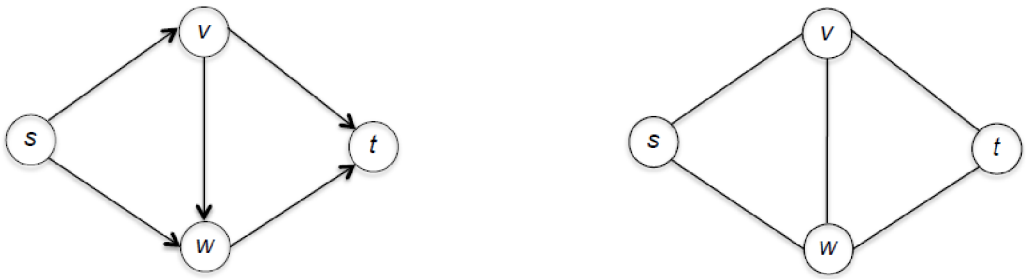
4 GRAFY

Pod pojmem graf si můžeme představit datovou strukturu, která reprezentuje vztahy mezi dvojicemi objektů, jenž se nazývají vrcholy V. Vztahy mezi jednotlivými vrcholy se nazývají hrany grafu E, přičemž může být mezi dvěmi vrcholy i více hran. Vizuální znázornění grafu může vypadat tak jako na obr. 11, kde jsou vrcholy $\{a, b, c, d, e, f, g\}$ vykresleny jako body a hrany grafu představují křivky spojující jednotlivé vrcholy [28].



Obr. 11: Vizualizace grafu [12].

K vrcholům, případně hranám se často přidává číselná hodnota, která má za úkol ohodnoti daný prvek. Příkladem může být velikost populace měst pro vrchol a délka trasy mezi městy pro ohodnocení hrany. Grafy se dají na základě orientace jejich hran rozdělit na orientované a neorientované. U prvně zmíněných (obr. 12a) je hrana definována seřazenou dvojicí vrcholů a má smysl pouze v daném směru. U neorientovaných na pořadí vrcholů nezáleží a je možné hranu projít oběma směry, viz. obr. 12b. Různých variant grafů je velké množství, například: multigraf, pseudograf, hypergraf. Dalším typem je strom, což neorientovaný graf, jehož vrcholy jsou spojeny právě jednou hranou [12].



(a) Orientovaný graf (b) Neorientovaný graf

Obr. 12: Typy grafů dle orientace hran [23].

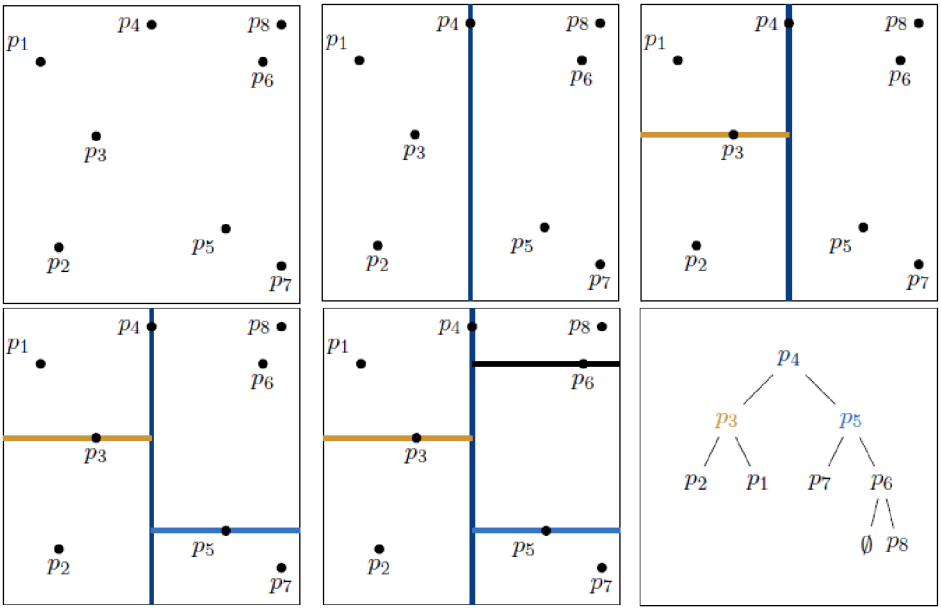
Využití grafů je velice rozmanité a rozsáhlé, používá se v odvětvích jako je například správa silniční sítě, kde může být využito hledání nejkratší cesty, či

dosažitelnost daného místa. Pro tyto aplikace je důležité zejména prohledávání grafu a nalezení nejkratší cesty v grafu [23].

4.1 *k-d strom*

Vzhledem k využití tohoto typu grafu v této práci (algoritmus *x-means*), je zde uveden jeho popis. Jak již bylo výše zmíněno stromem rozumíme graf, u kterého jsou vrcholy spojeny pouze jednou hranou. Vrcholy grafu tvoří body $P = \{p_1, ..., p_n\}$ v k-dimenzionálním prostoru, jehož roviny tvoří množinu d [25].

Konstrukce grafu je prováděna rekurzivně, přičemž pokud je P prázdná množina, nebo obsahuje pouze jeden bod, tak je výstupem prázdný graf, respektive graf s pouze jedním vrcholem obsahující jediný bod množiny. V ostatních případech je vybrána rovina d', ve které je množina bodů více rozptýlená. Následně vybereme bod, který je mediánem ve směru roviny d', a v tomto bodě vedeme rovinu H ve směru, který se s každou iterací mění v pevně daném pořadí. Tento vybraný bod tvoří první vrchol grafu. Pokud je například prostor tvořen dvěmi dimenzemi, může pořadí vytvářených rovin být nejprve v rovině rovnoběžné s osou X a následně v rovině rovnoběžné s osou Y. Toto pořadí se pak cyklicky opakuje [25].



Obr. 13: Konstrukce k-d stromu [25].

Takto vzniklá rovina H rozdělí prostor na dvě části, ve kterých dále potřebujeme najít nejbližší body k rovině H ve směru na ní kolmém. Získané body tvoří potomky předchozího vrcholu, kdy se bod, který je v kladném směru od dané roviny H, umístí doprava a druhý bod doleva. Tento proces pokračuje dále pro všechny další body p_i. Průběh konstrukce k-d stromu pro 8 bodů v dvoudimenzionálním prostoru

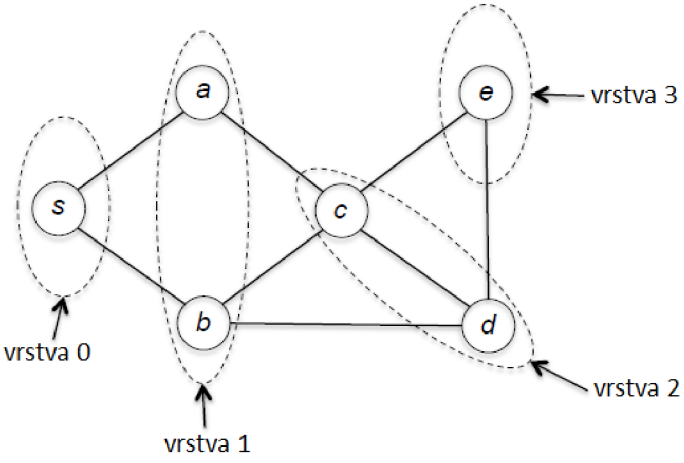
je znázorněn na obr. 13, kde v prvních pěti částech můžeme vidět vytváření rovin a na obrázku vpravo dole vidíme rekurzivně vytvořený strom [25].

4.2 Prohledávání grafu

Mezi metody prohledávání grafů patří prohledávání do šířky, prohledávání do hloubky, metoda větví a mezí a prohledávání s návratem. Nejčastěji používanými jsou právě první dvě zmíněné strategie, které jsou dále popsány [28].

4.2.1 Prohledávání do šířky

Tento přístup prostupuje grafem v pomyslných vrstvách, přičemž první vrstva obsahuje pouze startovací vrchol. Následující úroveň je tvořena vrcholy, které jsou vzdáleny od počátečního vrcholu pouze jednou hranou a nejsou již součástí některé existující vrstvy. V takto vytvořené vrstvě se ihned po jejím vytvoření prochází všechny její vrcholy. Zvyšováním možné vzdálenosti od vrcholu první úrovně se vytváří další vrstvy. Vrcholy, které nejsou dosažitelné z počátečního bodu, nejsou součástí žádné vrstvy. Znázornění příkladu, kdy je počáteční vrcholem s, lze vidět na obr. 14 [23].

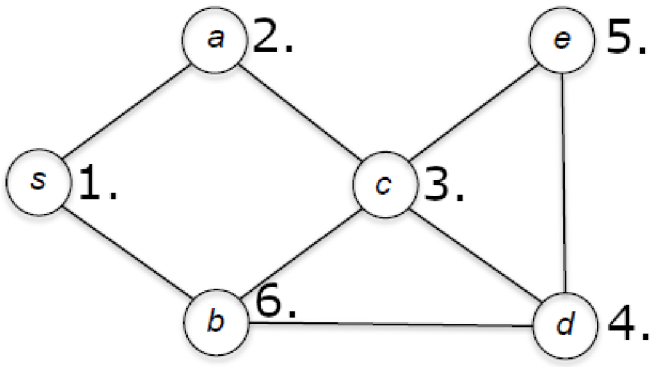


Obr. 14: Příklad prohledávání do šířky [23].

4.2.2 Prohledávání do hloubky

Tato metoda, obdobně jako prohledávání do šířky, označí vrchol jako prohlédnutý při jeho prvním navštívení. První vrchol je pevně daný a jako následující je vybrán jeden z vrcholů spojený hranou s počátkem. V každém dalším kroku se vybere některý ze sousedních vrcholů, který ještě není prohlédnut. Pokud se v určitém kroku není možné dostat dále z aktuálního vrcholu, přesouvá se algoritmus do předcházejícího

vrcholu do té doby, než přesun možný je. Ukončení nastává, pokud se tímto zpětným posunem dostaneme až do startovacího vrcholu a není možné se z něj dostat na některý další neprozkoumaný vrchol. Průběh této metody můžeme vidět na obr. 15, kde je vybrá jako počáteční vrchol s. Dále algoritmus může postupovat tak, že si vybere vrchol a jako následující, dále c, d a e, ve kterém již nemůže postupovat dále. Z tohoto důvodu se přesouvá algoritmus do vrcholu d, odkud je možné dostat se do vrcholu b, který je posledním neprohlédnutým bodem a algoritmus tímto krokem končí [23].



Obr. 15: Příklad prohledávání do hloubky [23].

4.3 Hledání nejkratší cesty grafem

Dalším velice důležitým procesem využívaný v teorii grafů je hledání nejkratší cesty. Cílem může být hledat nejkratší cestu mezi dvěma vrcholy grafu, cestu vrcholu ke zbytku vrcholů, nebo například nalezení nejkratších cest mezi všemi dvojicemi vrcholů [28]. Aplikačních použití je velké množství, přičemž příkladem používaných metody je Floyd-Warshallův, A*, nebo **Dijkstrův** algoritmus.

4.3.1 Dijkstrův algoritmus

Tento algoritmus dokáže pro graf s kladně ohodnocenými hranami najít nejkratší cestu z počátečního vrcholu do každého dosažitelného vrcholu grafu. Formuloval jej Edsger W. Dijkstra v roce 1956 a patří mezi velice často používáné prohledávací algoritmy z důvodu jeho efektivnosti a jednoduchosti.

Uvažujme graf, který má množinu vrcholů V a hran E, kde jednotlivé hrany e mají délku $l_e > 0$. Výslednou nejkratší cestu značíme množinou X a startující vrchol je označen jako s. Při inicializaci je množina tvořena pouze počátečním vrcholem, tedy $X = \{s\}$. Pro každý vrchol se počítá jeho vzdálenost d od počátečního vrcholu, přičemž vzdálenost v počátku je $d[s] = 0$, ostatním vrcholům se přiřadí

nekonečná vzálenost, která značí ještě neprohlédnutou cestu. Pro každou hranu spojující vrchol w, který není obsažen v množině X, s vrcholem v z množiny X zjišťujeme zda je cesta po této hraně (v, w) kratší, než aktuální ohodnocení vrcholu. Pokud je nově zjištěná hodnota nižší, než hodnota vrcholu, tak je ohodnocení aktualizováno. Není-li hodnota nižší, tak algoritmus postupuje k další hraně. Takto se postupuje pro všechny následující neprobádané vrcholy grafu. Ukončení nastává, pokud jsou již ohodnoceny všechny dosažitelné vrcholy grafu. Tento přístup se dá popsat algoritmem 2 [25].

Algorithm 2 Dijksktrův algoritmus

1: Graph $G = (V, E)$
2: Start node $s \in V$
3: Length $l_e \geq 0$ for each e inE
4: C_i is the ith cluster
5: $X := \{s\}$
6: $len(s) = 0, len(v) := +\infty$ for every $v \neq s$
7: **while** there is an edge (v, w) with $v \in X$, $w \notin X$ **do**
8: $\quad (v^*, w^*) :=$ such an edge minimizing $len(v) + l(v, w)$
9: $\quad$ add w^* to X
10: $\quad len(w^*) := len(v^*) + l_{v^*,w^*}$
11: **end while**

5 DYNAMICKÉ PROGRAMOVÁNÍ

Pro optimalizaci úloh, které se dají rozdělit do více kroků, ve kterých dochází k určitému procesu rozhodování, se řešení může získávat využitím rekurentních vztahů. Taková metoda je založena na principu optimality a nazývá se dynamické programování, jehož zakladatalem je Richard Bellman [1]. Jednotlivé etapy často následují za sebou v čase, ale tento přístup lze použít i v případech, kde čas nevystupuje a je uměle přidán. Cílem této metody je následně minimalizovat cenu za vybraná rozhodnutí, které nemůžeme brát v potaz pouze lokálně, ale jako celek voleb, které se mohou navzájem ovlivňovat. Rovnice v této kapitole jsou brány z [2]. První částí modelu tohoto problému je diskrétní dynamický systém rov. (6), kde k je index časové posloupnosti, x_k je stav systému s předchazejícími informacemi důležitými pro následující optimalizaci, u_k značí vybrané rozhodnutí, w_k je prvkem náhody, N je počet etap a f_k je funkce, která popisuje chování systému a aktualizaci jeho stavu [2].

$$x_{k+1} = f_k(x_k, u_k, w_k),\ k = 0, 1, ..., N-1 \tag{6}$$

Druhou částí modelu je jeho cenová funkce, která je v průběhu času k roustoucí. Formulovat se dá pomocí rovnice (7), kde $g_N(x_N)$ je ohodnocení konečného stavu.

$$g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k, w_k) \tag{7}$$

5.1 Základní problém

Základním problémem se může rozumět výběr za určité nejistoty pro konečný počet etap. Kde pro diskrétní systém popsaný rovnicí (6) platí $x_k \in S_k, u_k \in C_k, w_k \in D_k$. Řízení u_k je omezeno na hodnoty nenulové množiny $U(x_k) \subset C_k$, které závisí na aktuálním stavu x_k, tedy $u_k \in U_k(x_k)$ pro všechny $x_k \in S_k$ a k. Náhodná nejistota w_k je dána pravděpodobnostním rozdělením $P_k(\cdot\,|x_k, u_k)$ a dle Markovovy vlastnosti je závislá pouze na x_k a u_k, přičemž nesmí být závislá na předchozích nejistotách $w_{k-1}, ..., w_0$. Dále máme posloupnost funkcí $\pi = \{\mu_0, ..., \mu_N - 1\}$, kdy na základě μ_k a stavu x_k získáme řízení $u_k = \mu_k(x_k)$ a E, což značí průměrnou hodnotu. Po těchto formulacích je následující stav dán rovnicí (8) a cenová funkce rovnicí (9).

$$x_{k+1} = f_k(x_k, \mu(x_k), w_k),\ k = 0, 1, ..., N-1, \tag{8}$$

$$J_\pi(x_0) = E\left\{g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, \mu(x_k), w_k)\right\}. \tag{9}$$

Optimální řízení π^* je to, které minimalizuje cenovou funkci, tedy

$$J_{\pi^*}(x_0) = \min_{\pi \in \Pi} J_\pi(x_0). \tag{10}$$

5.2 Algoritmus dynamického programování

Tento algoritmus je založen na principu optimality, kdy pro optimální množinu řídicích rozhodnutí celého problému $\{\mu_0^*, ..., \mu_{N-1}^*\}$ je každá jeho podmnožina, začínající v okamžiku i a končící v čase N, $\{\mu_i^*, ..., \mu_{N-1}^*\}$ taktéž optimální. Algoritmus dynamického programování využívá tuto vlastnost a řeší daný problém po částech zpětně od časového okamžiku $N-1$ do 0. Optimální cena za poslední krok je dána vztahem (11).

$$J_N(x_N) = g_N(x_N) \tag{11}$$

$$J_k(x_k) = \min_{u_k \in U_k(x_k)} E_{w_k} \left\{ g_k(x_k, u_k, w_k) + J_{k+1}(f_k(x_k, u_k, w_k)) \right\}, \ k = 0, ..., N-1 \tag{12}$$

Pokud tedy $u_k^* = \mu_k^*(x_k)$ minimalizuje pravou stranu rovnice (12), pro každé x_k a k, je řízení $\pi^* = \{\mu_0^*, ..., \mu_{N-1}^*\}$ optimální.

5.3 Deterministické systémy a hledání nejkratší cesty

Pro deterministické systémy může každá odchylka w_k nabývat pouze jedné hodnoty, což je využíváno v širokém spektru aplikací a používá se i pro zjednodušení stochastických problémů. Jelikož není pro deterministické systémy použití zpětné vazby z hlediska snížení ceny žádným zlepšením, je minimalizace ceny přes přípustná řízení $\{\mu_0, ..., \mu_{N-1}\}$ stejná jako minimalizace přes vektory řízení $\{u_0, ..., u_{N-1}\}$. Předchozí tvrzení je pravdivé, jelikož pro sekvenci řízení $\mu_0, ..., \mu_{N-1}$ a počáteční stav x_0 jsou následující stavy přesně předpověditelné pomocí rovnice (13), stejně jako řízení je dáno vztahem (14).

$$x_{k+1} = f_k(x_k, \mu(x_k)), \ k = 0, 1, ..., N-1 \tag{13}$$

$$u_k = \mu_k(x_k), \ k = 0, 1, ..., N-1 \tag{14}$$

Za předpokladu deterministického problému, kdy je S_k konečná množina pro všechny k, je pro kterýkoli stav x_k možné řízení u_k brát jako přechod ze stavu x_k do stavu $f(x_k, u_k)$ za cenu $g_k(x_k, u_k)$. Pro takovýto problém je možné reprezentovat jej jako graf, jehož hrany značí přechody mezi jednotlivými stavy ohodnoceny cenou přechodu. Vrcholy tvoří stavy, přičemž první vrchol se značí s a konečný vrchol t je umělě přidán a hrany které do něj vstupují mají hodnotu $g_N(x_N)$. Řídicí posloupnost je pak cesta z vrcholu s do některého z vrcholů poslední etapy N. Úprava je pak následující: $a_{i,j}^k$ je cena za přechod ze stavu $i \in S_k$ do stavu $j \in S_{k+1}$ ve fázi k, $a_{i,t}^N$ je terminální cena. Řízení, které nemůže přejít z i do j je ohodnoceno $a_{i,j}^k = \infty$.

Algoritmus dynamického programování následně vypadá dle rovnice (15) a (16).

$$J_N(i) = a_{it}^N \; i \in S_N, \tag{15}$$

$$J_k(j) = \min_{i \in S_{k+1}} [a_{ij}^k + J_{k+1}(i)], \; i \in S_k \; k = 0, 1, ..., N-1 \tag{16}$$

5.4 Dopředný algoritmus dynamického programování

Předchozí algoritmus (5.3) postupuje časem zpětně a na základě jednoduchých předpokladů se dá formulovat algoritmus, který postupuje dopředu. Tím je fakt, že optimální cesta z s do t musí být optimální i v obráceném pořadí z t do s, kdy pouze obrátíme směry hran grafu a zbytek ponecháme jak je. Dopředný algoritmus začíná ve stavu $x_1 \in S_1$ a pokračuje až do stavů $x_N \in S_N$ a je dán vztahy (17), (18).

$$\tilde{J}_N(j) = a_{sj}^0, \; j \in S_1 \tag{17}$$

$$\tilde{J}_k(j) = \min_{i \in S_{N-k}} [a_{ij}^{N-k} + \tilde{J}_{k+1}(i)], \; j \in S_{N-k+1}, \; k = 1, 2, ..., N-1 \tag{18}$$

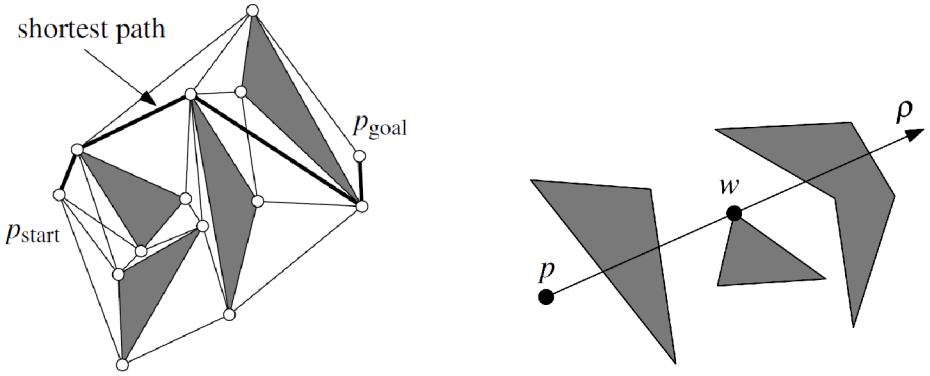
Optimální cena je dána vztahem (19) a výsledek zpětného algoritmu je stejný jako výsledek dopředného, což je znázorněnoho vztahem (20). Využití tohoto algoritmu je například pro aplikace pracující v reálném čase, kdy informace předcházející etapě k jsou neznámy a jsou dostupné až těsně před začátkem etapy k [2].

$$\tilde{J}_0(t) = \min_{i \in S_N} [a_{it}^N + \tilde{J}_1(i)] \tag{19}$$

$$J_0(s) = \tilde{J}_0(t) \tag{20}$$

6 VIDITELNOSTNÍ GRAF

Jednou z možností hledání nejkratší cesty mezi dvěma body prostoru, s ohledem na možné překážky v něm, je vytvoření viditelnostního grafu G a jeho následné prohledání. Vrcholy tohoto grafu tvoří body polygonů objektů prostředí a jeho hrany jsou tvořeny přímkami mezi jednotlivými vrcholy, které neprotínají žádný z objektů. Pokud nejsou body, pro které hledáme nejkratší cestu, součástí některého polygonu, tak jsou do grafu přidány. Při inicializaci jsou již všechny vrcholy grafu vytvořeny, dále se přiřadí viditelné hrany grafu ohodnocené euklidovskou vzdáleností mezi jednotlivými body. Posledním krokem je nalezení nejkratší cesty, kde je použit Dijsktrův algoritmus (viz. 4.3.1). Výsledný viditelnostní graf s nejkratší cestou mezi body p_{start} a p_{goal} můžeme vidět na obr. 16a [4]. Jelikož by zjištění viditelnosti jednotlivých



(a) Viditelnostní graf s nejkratší cestou mezi dvěma body .

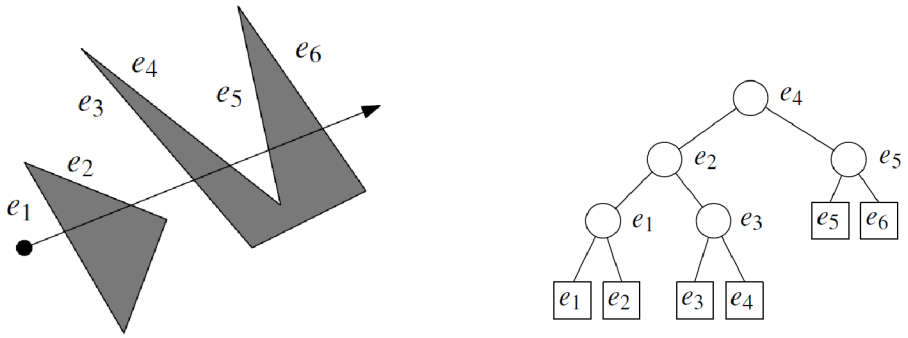
(b) Protínající polopřímka.

Obr. 16: Viditelnostní graf [4].

hran procházením všech vrcholů a následné kontroly protnutí bylo výpočetně velice náročné, používá se rotační pročesávání polopřímkou ρ. Ta začíná v bodě p a prochází bodem w některého z objetků. Principem této metody je vytvoření binárního stromu S, jehož listy jsou hrany protnuté polopřímkou ρ, jak lze vidět na obr. 16b. Tato křivka je postupně natáčena pomocí úhlů, které svírají přímky s rovinou x, přičemž jsou úhly seřazeny tak jak jdou ve směru hodinových ručiček [4].

Pro jednotlivou polopřímku ρ, protínající určitý počet hran, se listy tvoří ve stejném pořadí jako dochází k protnutí polopřímky a vkládají se do grafu zleva. Nadřazený vrchol tvoří vždy list, který je nejvíc vlevo v podstromu tohoto vrcholu. Příklad vytváření takového binárního stromu pro hrany e_1 až e_6 můžeme vidět na obr. 17 [4].

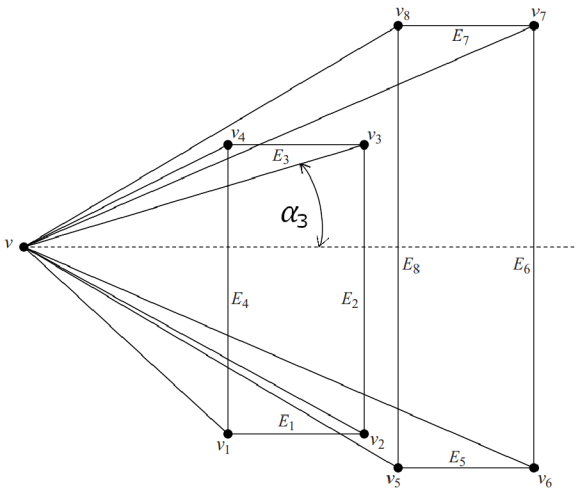
Počáteční poloha polopřímky je rovnoběžná s osou x a do binárního stromu se vloží hrany, které ρ protínají. Dále se polopřímka natočí dle dalšího úhlu, kdy prochází daným bodem w prostoru. Následně se v tomto procesu do binárního stromu vkládají hrany, které jsou proti směru hodinových ručiček a odebírají se hrany, které



(a) Polopřímka protínající hrany (b) Vytvořený binární strom

Obr. 17: Vytváření binárního stromu při pročesávání [4].

jsou ve směru. V takto upraveném grafu zjišťujeme viditelnost dvou bodů, přičemž se prohledává pouze oblast obsažená v tomto stromu, což značně snižuje časovou závislost. Tato smyčka se provádí pro všechny vrcholu grafu, tak jako je vidět na obr. 18. Počáteční poloha polopřímky je vyznačena čárkovanou čárou a listy stromu tvoří hrany $\{E_4, E_2, E_8, E_6\}$, které tato polopřímka protíná. Dále pro natočení pod úhlem α_3 upravíme strom tak, že odebereme hranu E_2 a přídáme hranu E_3 jelikož jsou ve směru, respektive proti směru hodinových ručiček od polopřímky ρ. Následuje zjištění viditelnosti, kdy se porovnávají pouze hrany obsažené ve vytvořeném binárním stromě a pokud polopřímka z bodu p do bodu w neprotíná žádnou z hran, přidáme hranu (p, w) do viditelnostního grafu. Tento proces opakujeme pro všechny úhly ve výše definovaném pořadí a výsledkem je zjištění viditelnosti pro hrany (v, v_4), (v, v_8) a (v, v_1) [3].



Obr. 18: Průběh rotačního pročesávání [3].

7 EVOLUČNÍ ALGORTIMY

Metody využívající principy evoluce a dědičnosti jsou v současné době velice často využívané pro optimalizační úlohy plánování cesty, vytváření časových plánů, nelinárních dynamických systémů, TSP („traveling salesman problem“) a mnoho dalších typů úloh. Základem těchto přístupů je populace potenciálních řešení, kdy se pomocí výběru s ohledem na vhodnost daného řešení aplikují genetické operátory inspirované přírodní evolucí a následná generace se tvoří přežitím nejsilnějších jedinců. Pro každou populaci existuje její ohodnocení kvality (fitness). Postupem času a úpravou populace se optimalizuje daný problém, který většinou nelze klasickými přístupy dostatečně rychle vyřešit. Mezi tyto algoritmy patří například evoluční programování, genetické programování, evoluční strategie nebo **genetické algoritmy** [20].

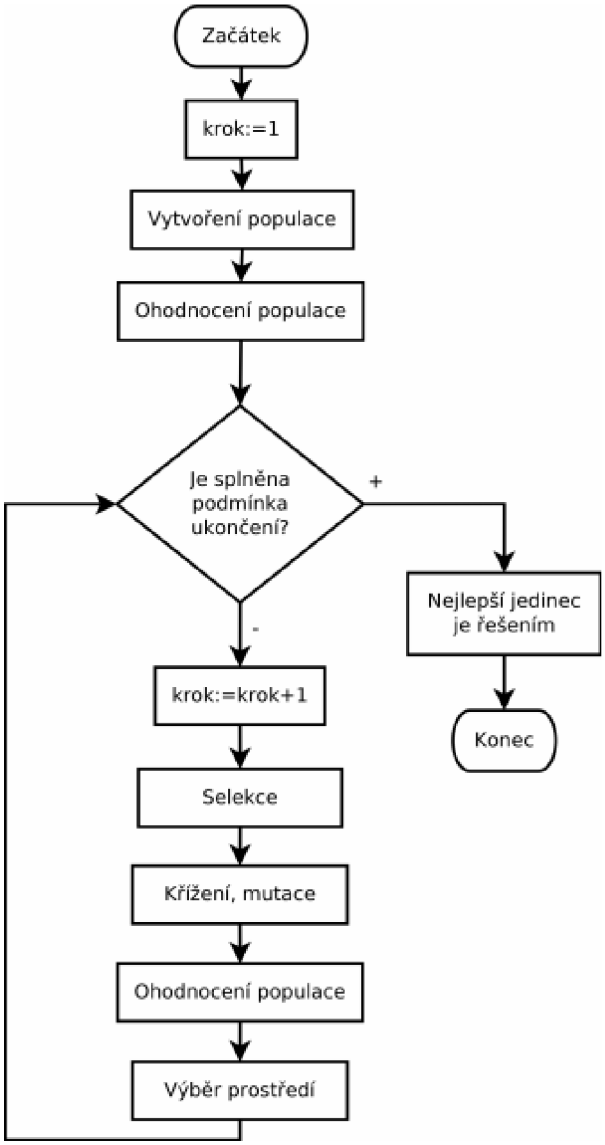
7.1 Genetické algoritmy

Historie nejčastěji používané metody evolučních algoritmů sahá do roku 1975, kdy ji poprvé formuloval americký vědec J. H. Holland [13]. Na základě principu přírodní evoluce popsal první optimalizační algoritmus, ze kterého vycházejí genetické algoritmy v podobě, kterou známe nyní [24].

V biologii se věda, která se zabývá mechanismy pro rozpoznání společných, či rozdílných prvků u jedinců, nazývá genetika. Z té taky vychází terminologie jednotlivý částí jako je chromozom, který obsahuje DNA (deoxyribonukleová kyselina) informace. Ty pak jsou rozděleny do genů, které kódují vlastnosti druhů a jedince. Různé varianty genů jsou zvány alely, které dohromady tvoří genofond. Všechny geny jednoho druhu pak tvoří genom, přičemž pro genetické algoritmy je chromozom a genom synonymem. K reprodukci dochází buď pomocí mitózy, kdy dochází ke kopírování genetické informace, nebo pomocí meiózy, kdy dochází ke sdílení genetické informace. Přežití jednotlivců je pak dáno přirozeným výběrem, kdy déle přetrvavá silnější jedinec a je pravděpodobnějším adeptem pro reprodukci [20].

7.1.1 Struktura obecného genetického algoritmu

Základní struktura GA (genetický algoritmus) je tvořena selekcí, reprodukcí, ohodnocením a nahrazením jedinců. Při inicializaci se vytvoří náhodná populace, ze které se poté vyberou statnější jedinci, kteří jsou následně podrobení operátorům reprodukce. Na základě jejich následného ohodnocení probíhá utváření nové generace s cílem vytvoření kvalitnějšího ohodnocení. Tento proces pokračuje dokud není splněna některá z podmínek, jako je nalezení dostatečně dobré hodnoty, nebo určitý počet iterací. Průběh obecného GA se dá popsat vývojovým diagramem (19) [24].



Obr. 19: Vývojový diagram GA [17].

Populace

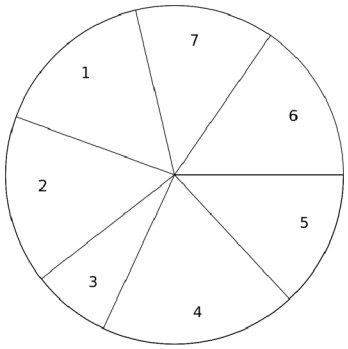
Populace je souhrn všech jedinců v aktuální generaci, kdy vlastnosti jedince předsatvuje genom. Ten může být kódován několika způsoby a to zejména binárním (0101), permutačním (1234), nebo hodnotovým řetězcem (ABCD). Použité typy se mohou lišit pro jiná konkrétní řešení a využívá se ten, který nejlépe popisuje daný systém [20].

Selekce

Tento proces vybírá dvojici rodičů, která je vhodná pro vytvoření potomků. Děje se tak dle fitness jednotlivých jedinců za účelem vytvoření potomka, jehož ohodnocení bude mít vyšší hodnotu. Výběr probíhá s pravděpodobností, která je úměrná fitness hodnotě jedince tak, že lepší jedinci mají větší šanci být vybrání. Konkrétních

metod selekce je více a to například čistě náhodná, turnajový výběr, stochastická univerzální metoda, nebo ruletová metoda, která je z důvodu využití v této práci dále stručně popsaná [20].

Jedná se o velice tradiční metodu selekce využívanou v GA, kde se pro každého jedince přiřadí pravděpodobnost úměrná jeho fitness. Tyto hodnoty pak tvoří rozdělení ruletového kola, kdy plocha dané částí odpovídá hodnotě, čímž se zvyšuje pravděpodobnost při simulaci dopadu kuličky. Provádí se množství simulací s předpokladem, že vhodnější jedinci jsou vybírání častěji. Grafické znázornění přiřazení pravděpodobností připomínající ruletu lze vidět na obr. 20 [20].



Obr. 20: Ruletový výběr [17].

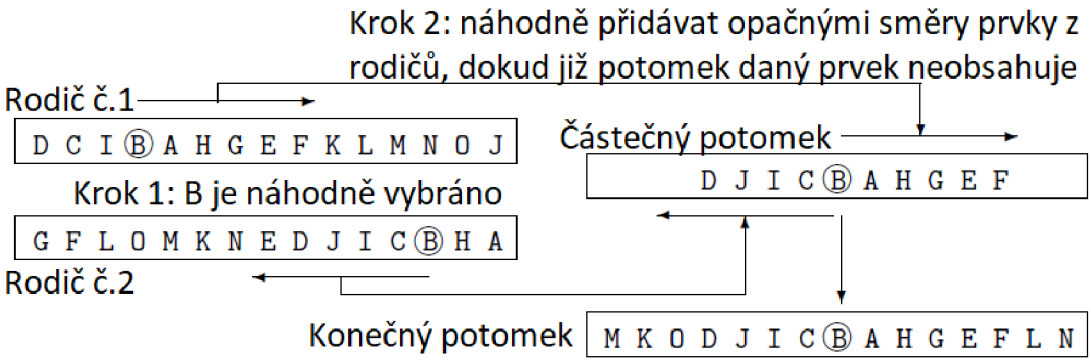
Křížení

Po výběru rodičů přichází na řadu reprodukční proces, přičemž potomci vznikají kombinací svých rodičů s cílem obohatit populaci o nové silnější jedince. Jedním ze základních a nejjednodušších přístup je náhodě zvolit pozici v genomu a část předcházející této pozici se veme z prvního rodiče a nadcházející část se bere z druhého rodiče. Na základě počtu těchto přerušení může dojít k n-bodovému křížení [20].

Pro použití křížení u TSP nemusí být vždy n-bodové křížení vhodné z možného důvodu vytváření duplikací v genomu. Proto se velice často používá metoda GSX („greedy subtour crossover“), kdy se náhodně vybere prvek genomu, který při inicializaci tvoří potomka. Dále se v obou rodičích vyhledá tento prvek a v každém z rodičů se postupuje opačným směrem, přičemž se náhodně vybere začínající rodič ze kterého se berou prvky v jeho vybraném směru a pokud již nejsou obsaženy v potomkovi, tak se ve stejném směru přiřadí i v něm. Takto se alternují prvky dosazované z rodičů a pokud dojde k situaci, kdy je již prvek obsažen v potomkovi, tak se náhodně dosadí zbývající chybějící prvky s alternujícím přiřazením ve směrech rodičů, viz obr. 21 [20].

Mutace

Tento proces se s určitou malou pravděpodobností aplikuje na vytvořené potomky z důvodu diverzifikace nalezených řešení, které mohou omezit setrvání v lokálním

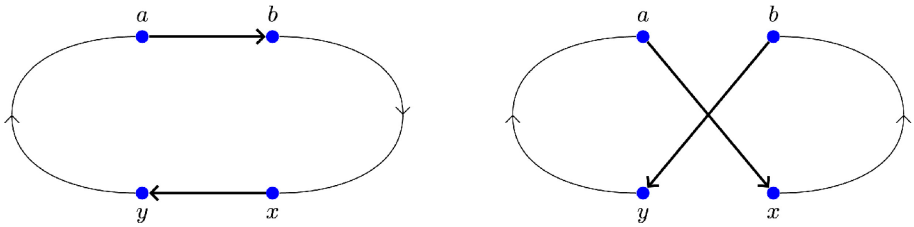


Obr. 21: GSX křížení [16].

extrému funkce. Pro binární řetězce se může jednat například o změnu hodnoty některého z bitů a pro permutační řetězec je možné prohodit 2 hodnoty mezi sebou [20].

Lokální optimalizace

Hledání nejlepší možné varianty daného jedince by se dalo popsat jako metoda lokální optimalizace, kdy pracujeme pouze s jedním řešením a to patřičně upravujeme. Velice častým jevem při řešení TSP je křížení generované cesty (viz obr. 22), což je velice nežádoucí a je vhodné použít lokální optimalizaci, která těmto křížením zamezí. Konkrétních typů lokální optimalizace použitelných pro TSP je několik a příkladem mohou být *2-opt*, *3-opt*, nebo také *4-opt*. Z důvodu použití v praktické části této práce a pro přiblížení je dále popsána metoda *2-opt* [16].



Obr. 22: Křížení cesty mezi jednotlivými body [14].

Principem této metody je postupné otáčení pořadí všech kombinací jednotlivých podoblastí. Prohledáváním těchto možností se zjistí nejlepší kombinace a pokud je kvalitnější, než původní řešení, tak toto řešení nahrazuje (viz. algoritmus 3, 4). I přes značnou časovou náročnost výpočtu je tato metoda velice často používaná, jelikož přináší značné zlepšení jednotlivých řešení [16].

Algorithm 3 2-opt swap

```
function swap_2_opt(route, i k)
    new_route = empty
    Add route[0 : i − 1] to new_route
    Add route[i : k] swapped to new_route
    Add route[k + 1] to new_route
    return new_route
end function
```

Algorithm 4 Algoritmus 2-opt

```
best_val = fitness(route)
repeat
    for (i = 0; len(route) − 1; i + +) do
        for (k = i + 1; len(route) − 1; k + +) do
            new_route = swap_2_opt(router, i, k)
            new_val = fitness(new_route)
            if new_val < best_val then
                route = new_route
                best_val = new_val
            end if
        end for
    end for
until No improvement is made
```

8 DUBINSOVY KŘIVKY

Pro generování vyhlazené spojité cesty při obratech autonomní robotické sekačky se dá využít Dubinsových křivek. Aplikují se v oblasti přejezdu mezi dvěma rovnoběžnými čarami s ohledem na možný poloměr otáčení robota. Cílem této metody je najít nejkratší možnou cestu mezi dvěma body, která je proveditelná pomocí jednoduchých geometrických obrazců. Takový přístup poprvé kompletně formuloval Dubins v roce 1957 pomocí jednoduchého vozidla, které má konstantní rychlost, omezený úhel otáčení a pohybuje se pouze vpřed [18].

Pro takto stanovený problém je účelem optimalizovat délku dráhy mezi počátečním bodem q_I a koncovým bodem q_G při omezeném maximálním úhlu otáčení ϕ_{max}. Délka této dráhy se dá zapsat rovnicí (21), kde t_F je čas kdy byl dosažen bod q_G, který je dán jako $q = (x, y, \theta)$. Jelikož je rychlost konstantní, dá se tento systém zjednodušit, viz. rovnice (22)-(24) [18].

$$L(\tilde{q}, \tilde{u}) = \int_0^{t_F} \sqrt{\dot{x}(t)^2 + \dot{y}(t)^2 dt} \tag{21}$$

$$\dot{x} = \cos\theta \tag{22}$$

$$\dot{y} = \sin\theta \tag{23}$$

$$\dot{\theta} = u \tag{24}$$

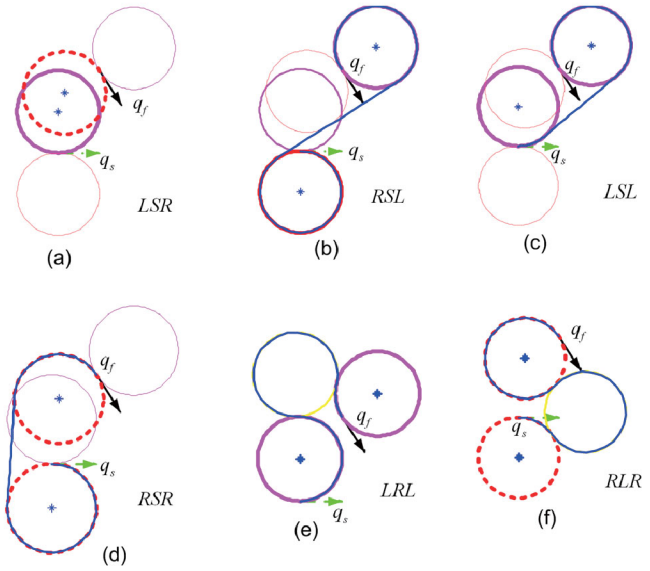
V rovnici (24) je u vybráno z intervalu $U = [-\tan\phi_{max}, \tan\phi_{max}]$, což mění problém na optimalizace času, jelikož se integrand rovná 1 a pro jednoduchost předpokládáme, že $\tan\phi = 1$.

Za těchto předpokladů prokázal Dubins, že mezi jakýmikoliv dvěma body je možné zjistit nejkratší cestu pomocí kombinace tří jednoduchých pohybů. Každá z těchto primitivních trajektorií provádí konstantí pohyb za určitý časový interval a dají se vyjádřit $u \in \{-1, 0, 1\}$. Ke každému z těchto úhlů je přiřazen symbol, který charakterizuje daný pohyb, kdy pojmenování lze vidět v tabulce 2, kde symbol S, L, R znamenají jízdu rovně, maximální otočení doleva a maximální otočení doprava [18]. Pomocí těchto tří triviálních pohybů lze jejich kombinací dosáhnout jakéhokoliv složeného pohybu, kdy se takový pohyb reprezentuje pořadím symbolů, seřazených tak, jak jdou při vytváření dráhy po sobě. Trajektorií složených z těchto

Symbol	Otočení: u
S	0
L	1
R	-1

Tab. 1: Přiřazení symbolů otáčení [18].

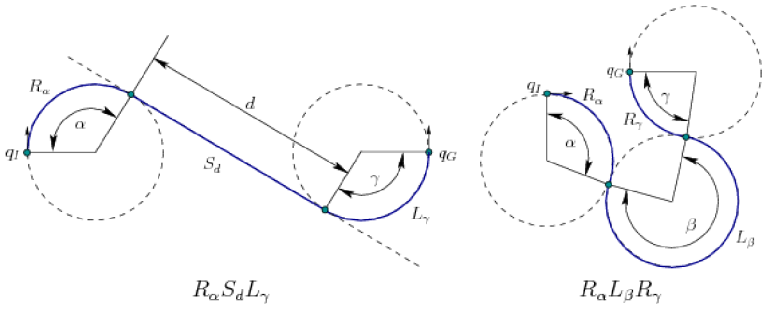
tří jednoduchých možností je celkově 10, přičemž Dubins prokázal, že použitelných je pouze 6 a ty jsou $\{LRL, RLR, LSL, LSR, RSL, RSR\}$ a jejich výsledné pohyby jsou znázorněné na obr. 23 [18]. Optimální cesty nalezeny pomocí těchto kombi-



Obr. 23: Použitelné kombinace primitivních pohybů [9].

nací se nazývají Dubinsovy křivky. Pro úplnost musí být ještě doplněny doby trvání jednotlivých pohybů, kdy pro přímočarý pohyb definujeme délku d, kterou má v daném směru vozidlo urazit. Pro otočení vlevo a vpravo se uvádí úhel, který vytvořená křivka svírá od svého začátku do konce. Grafické znázornění doplnění pohybů o jejich dobu trvání lze vidět na obr. 24 a použitelné kombinace vypadají následovně takto [18]:

$$\{L_\alpha R_\beta L_\gamma, R_\alpha L_\beta R_\gamma, L_\alpha S_d L_\gamma, L_\alpha S_d R_\gamma, R_\alpha S_d L_\gamma, R_\alpha S_d R_\gamma\} \tag{25}$$



Obr. 24: Doba trvání jednotlivých pohybů [4].

9 SOFTWAROVÁ IMPLEMENTACE

V této kapitole je popsána softwarová implementace nástroje s grafickým rozhraním, který je pro zadanou oblast schopen optimalizovat cestu sekačky. Struktura této kapitoly kopíruje strukturu zvoleného přístupu k řešení (kapitola 2.1). Implementace proběhla v jazyce `Python` a jeho knihovnách (viz. tabulka 2). Pro vytvoření GUI („graphic user interface“) je využita knihovna `PyQt5`, kdy se zejména využívá třída `QtThread` pro plynulý chod rozhraní, tak aby se jednotlivé procesy nemohly překrývat.

Po spuštění aplikace se zobrazí hlavní obrazovka s možností úpravy šířky sekačky a natočení. Po navolení parametrů a spuštění se zpracuje vstupní soubor, vytvoří se pracovní prostředí, dojde k rozdělení do podoblastí a následně se pomocí genetického algoritmu hledá optimální pořadí těchto podoblastí a výstupem je tato optimalizovaná cesta sekačky. Dílčí procesy jsou tvořeny vlákny takovým způsobem, že po skončení aktuálního kroku následuje další, čili se jedná o serializaci výpočtů, které nejsou časově triviální. Popis jednotlivých procesů se nachází dále v této kapitole a je úzce spjat s teoretickou částí této práce.

Knihovna	Použití	Instalace
`PyQt5`	grafické rozhraní	`pip`
`shapely`	operace s geometrickými tvary	`pip`
`pyvisgraph`	viditelnostní graf	`pip`
`numpy`	matematické operace	`pip`
`pyclustering`	shluková analýza	`pip`
`utm`	transformace zeměpisných dat	`pip`
`pykml`	zpracovná KML souboru	`pip`
`random`	generování náhodných čísel	`standard`

Tab. 2: Nejdůležitější používané knihovny.

9.1 Grafické rozhraní a vstupní parametry

Vstupními parametry jsou mimo soubor obsahující souřadnice dané oblasti (více kapitola 9.2) také specifikace technických parametrů sekačky a v případě rozšířeného nastavení i některé parametry genetického algoritmu. Grafické rozhraní je vytvořeno v jednoduchém a intuitivním duchu, kdy hlavní obrazovku můžeme vidět na obr. 25 s následně popsanými prvky.

Obr. 25: Hlavní obrazovka.

1 – Nastavení parametrů sekačky

Tento formulář slouží pro nastavení parametrů sekačky a to její šířky, která je dána v metrech a ovlivňuje počet jednotlivých paralelních čar a také radius otáčení sekačky, přičemž se bere v potaz nejhorší možná situace a tedy to, že je radius roven polovině šířky sekačky. Šířka sekačky se obvykle pohybuje mezi 20-50 centimetry. Další z parametrů je úhel, který svírají rovnoběžné čáry s pomyslnou vodorovnou osou. Tento úhel je pro všechny oblasti stejný a vychází z něj celé rozdělení sekací oblasti.

2 – Tlačítko pro start výpočtu

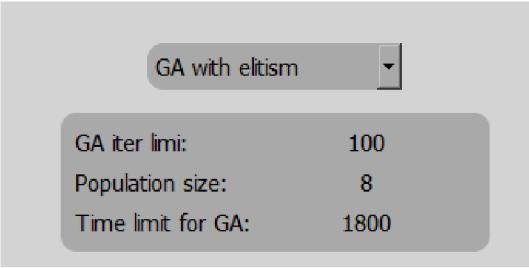
Tímto tlačítkem se zobrazí okno pro výběr souboru a po jeho potvrzení začínají veškeré výpočty a také přechod na načítací obrazovku.

3 – Tlačítko pro návrat na zobrazení mapy

Toto tlačítko slouží pro přepnutí na obrazovku s vykreslenou mapou, přičemž je aktivní pouze poté co již proběhl výpočet a je aktivní vykreslovací obrazovka.

4 – Tlačítko pro zobrazení rozšířeného nastavení

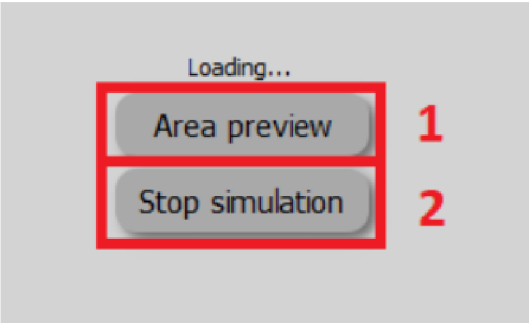
Stisknutí tohoto tlačítka zobrazí/skryje rozšířená nastavení týkající se převážně nastavení genetického algoritmu. Rozšiřující parametry lze vidět na obr. 26, kde je možnost nastavit počet iterací, velikost populace a maximální časový limit pro výpočet genetického algoritmu. Dále je také možné zvolit jeden ze dvou typů aplikovaných genetických algortimů, tedy variantu s elitismem, či lokální optimalizací. Po spuštění jsou parametry automaticky nastaveny na hodnoty vycházející ze simulací (viz. 9.9).



Obr. 26: Rozšířená nastavení.

9.1.1 Popis načítací obrazovky

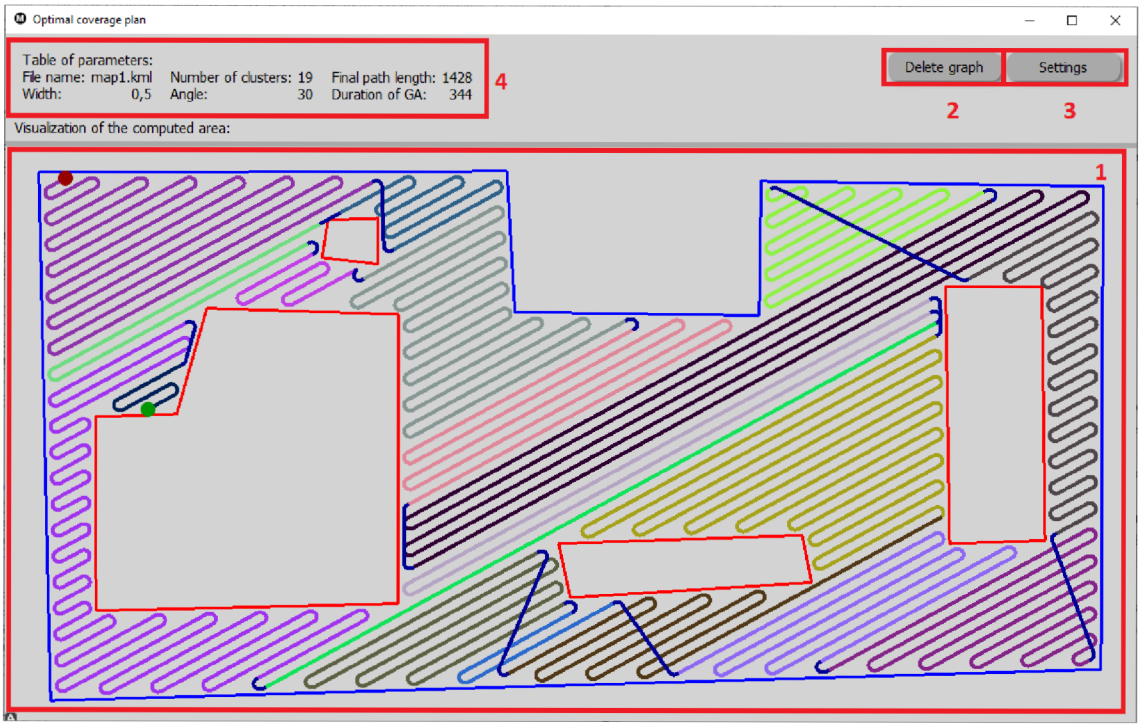
Po spuštění serializovaného výpočtu, je aktivní načítací obrazovka, která je tvořena statickým loaderem a ovládacími prvky. Možnost zapnutí rozšířeného nastavení společně s tlačítkem návratu na vykreslenou oblast jsou neaktivní. Prvek označený číslicí 1 na obr. 27 je tlačítkem přechodu na nahlédnutí rozdělení oblasti a prvek č. 2 ukončuje veškerou simulaci a vykreslí pouze ohraničení oblasti.



Obr. 27: Načítací obrazovka.

9.1.2 Popis vykreslovací obrazovky

Po dokončení výpočtů a optimalizaci výsledné cesty se ukončí serializovaný výpočet a aktivuje se obrazovka pro zobrazení výsledné cesty (viz. obr. 28), přičemž popis jednotlivých prvků se nachází níže.



Obr. 28: Vykreslená optimalizovaná cesta.

1 – Komponenta pro zobrazení oblasti a cesty

V této části je zobrazena pracovní oblast a výsledná optimalizovaná cesta, přičemž je pro přehlednost cesta v každé oblasti vyznačena jinou barvou. Přejezdy mezi jednotlivými oblastmi jsou značeny modrou barvou a počáteční bod tvoří zelený bod, konec potom bod červený. Vnější ohraničení je vyznačeno modrou barvou a vnitřní překážky barvou červenou.

2 – Tlačítko smazání výsledku

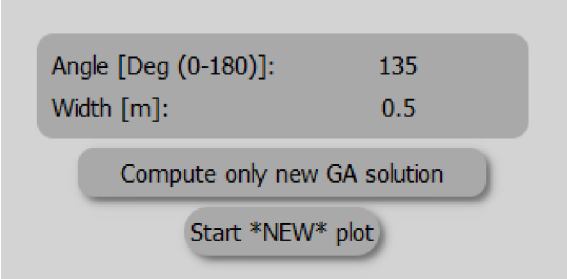
Toto tlačítkou slouží pro vymazání výsledku, vykreslené oblasti a po jeho stisknutí je aktivní hlavní obrazovka.

3 – Tlačítko pro přechod na nastavení

Stisknutím tlačítka se aktivuje hlavní obrazovka pro možnou kontrolu nastavených parametrů, přičemž je výsledek stále zachován. Změna nastává pro oblast spuštění výpočtu, kde je zobrazeno nové tlačítko, které slouží pouze pro přepočítání GA se pro stávající prostor, viz obr. 29.

4 – Informační tabulka

Tato komponenta slouží pro zobrazení různých informací, které byly buď zadány uživatelem, či dosaženy v průběhu výpočtu.



Obr. 29: Tlačítka pro přepočítání GA.

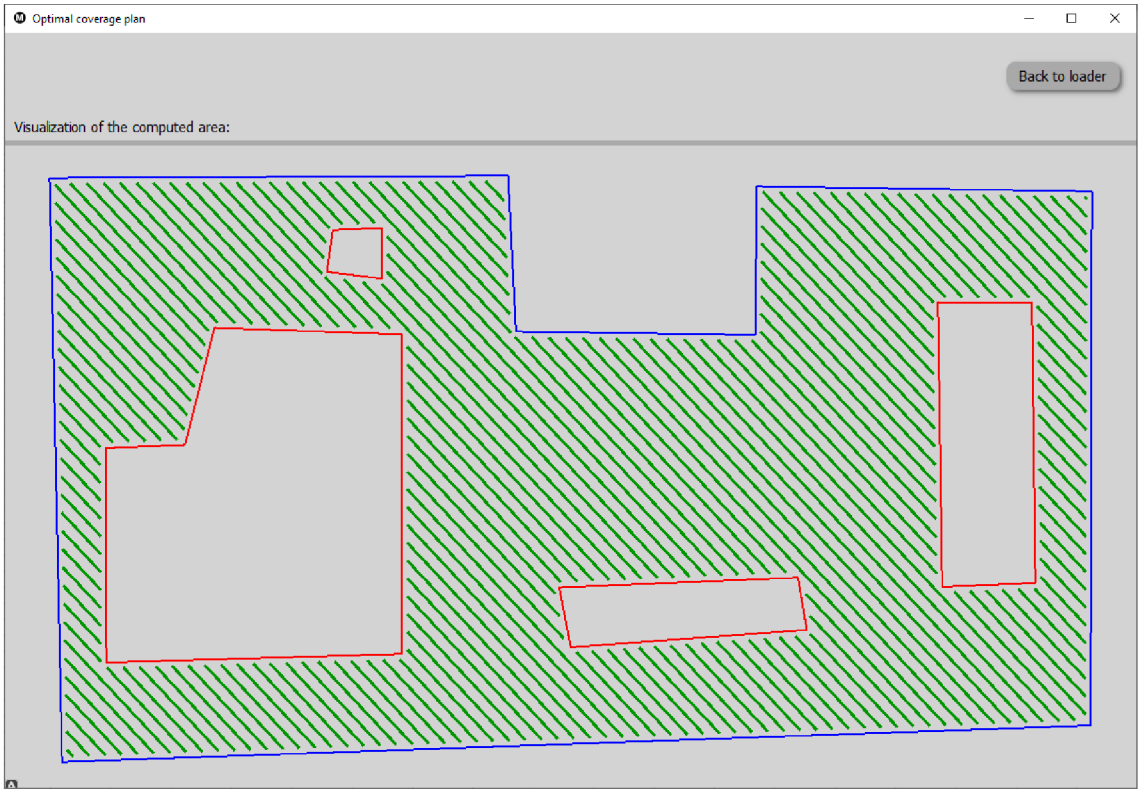
9.2 Reprezentace oblasti

Pro reprezentaci oblasti byl vybrán formát KML („keyhole markup language“), který je standardem OGC („open geospatial consortium“). Vytvořené objekty, ať už vnitřní či vnější, musí být posloupnosti bodů takové, kdy je poslední bod polygonu stejný jako první, nebo se tento polygon v některém místě protíná. Dále se předpokládá, že objekty nezasahují jeden do druhého a to z důvodu náročnosti ošetření těchto případů a také tím, že tato část není hlavním motivem této práce. Ukázku KML souboru lze vidět dále, kdy se jedná o záznam měření na portálu Mapy.cz, přičemž tyto záznamy exportované do KML formátu byly použity pro generování konkrétních map použitých v této práci.

```
<?xml version="1.0" encoding="utf-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
<Document>
    <name>Mereni z Mapy.cz</name>
    ...
    <Folder>
        <name>1. cast</name>
        <visibility>1</visibility>
        <open>1</open>
        <Placemark>
        <name>Mereni z Mapy.cz</name>
        <styleUrl>#path</styleUrl>
        <LineString>
            <tessellate>1</tessellate>
            <coordinates>
                17.8770048916,49.8769171536,269.0
                ...
            </coordinates>
        </LineString>
        </Placemark>
    </Folder>
</Document>
</kml>
```

9.3 Vytvoření pracovní oblasti

Jelikož jsou vstupní data ve formátu zeměpisné šířky a délky, je nutné je prvně převést pomocí knihovny `utm` na kartézské souřadnice. Z takto transformovaných dat se následně získají uzavřené objekty a zjistí se, který z nich je vnější ohraničení a které jsou vnitřními překážkami. Od všech ohraničení se ve vzdálenosti poloviny šířky sekačky vytvoří okraj sekací plochy z důvodu bezpečného obratu sekačky. Následně se celá oblast protne rovnoběžnými čárami, které jsou od sebe vzdáleny o šířku sekačky, což je vstupní parametr volený uživatelem. Ty se následně ořežou o vnitřní překážky a vzniknou tak jednotlivé cesty, po kterých se může sekačka bez přerušení volně pohybovat. Takto vytvořené prostředí lze vidět na obr. 30.

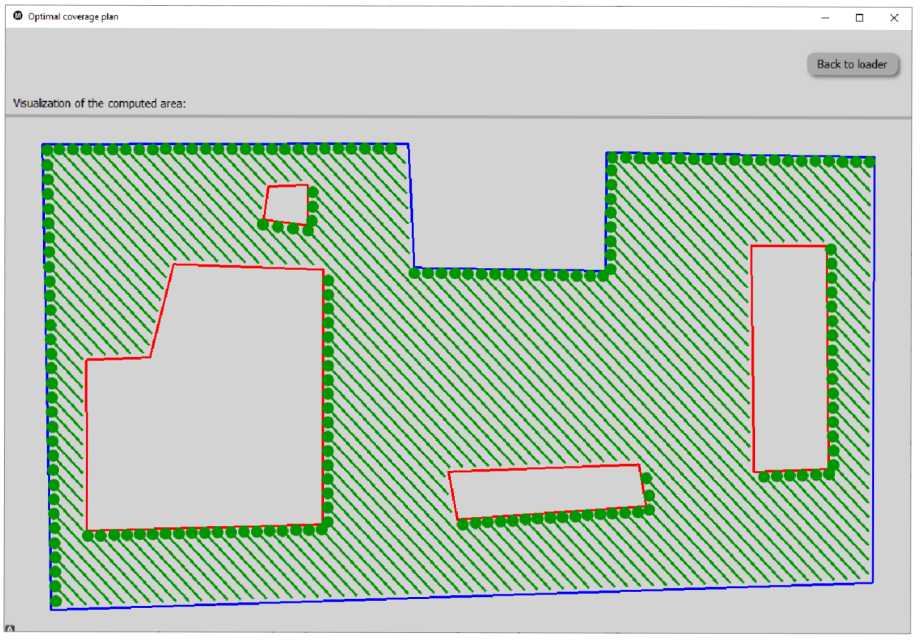


Obr. 30: Vytvořená oblast s rovnoběžnými čárami.

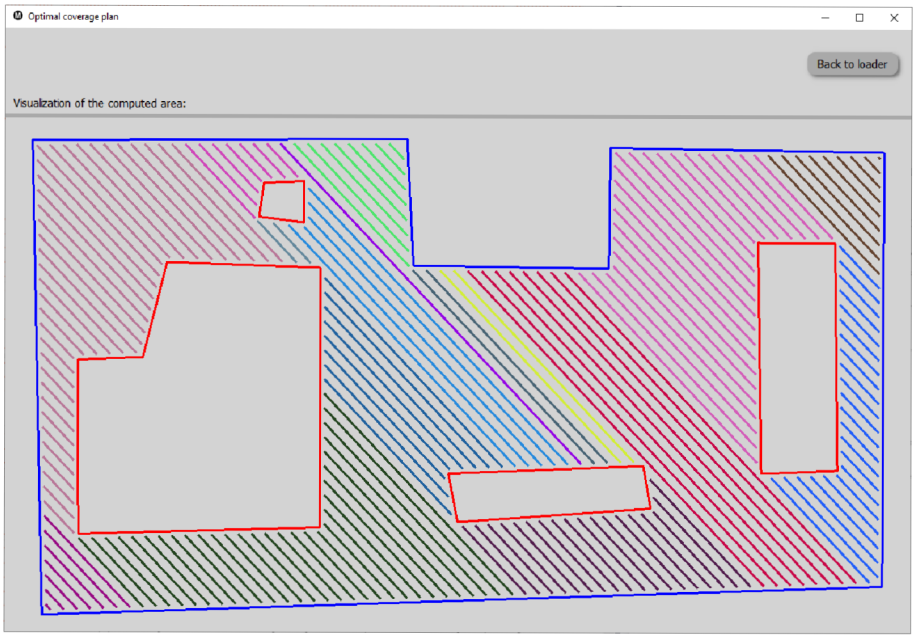
9.4 Rozdělení do podoblastí

Pro získání jednotlivých podoblastí, ve kterých má sekačka optimální dráhu danou pouze jako sekvenci rovnoběžných čar se paralelní čáry již vytvořené oblasti roztřídí shlukovou analýzou pomocí jejich horních bodů (viz. obr. 31). Pro tento proces se využívá knihovna `pyclustering`, konkrétně její metoda `xmeans`. Princip této metody je již popsán v kapitole (3.2). Z praktického hlediska se na základě vstupních

parametrů, což jsou vstupní body, počáteční počet středů, maximální počet středů a typ kritéria vygenerují požadované shluky. V této práci je jako kritérium pro přidání dalšího středu vybráno BIC ohodnocení a z důvodu snížení výpočetní závislosti je maximální počet shluků razantně omezen. Takto vytvořené shluky musí projít ještě kontrolou a případným rozdělením tak, aby vnitřní cesta byla opravdu triviální. Rozdělení do takovýchto oblastí lze vidět na obr. 32.



Obr. 31: Horní body souřadnic pro vytvoření shluků.



Obr. 32: Rozdělení oblasti pomocí algoritmu x-means.

9.5 Volba optimálního pořadí podoblastí

Optimalizace pořadí podoblastí probíhá pomocí genetického algoritmu, kdy je genomem posloupnost podoblastí (permutační repzrezentace). Populaci potom tvoří počet genomů, který je definován uživatelem. Fitness ohodnocení je délka optimální cesty pro pevně zvolené pořadí (více v 9.6). Počáteční populace je volena náhodně a jelikož je tento problém možné kategorizovat jako TSP, je jako operátor křížení zvoleno GSX (viz. 7.1.1), který vždy vytvoří validního potomka. Mutace probíhá prohozením náhodně dvou zvolených podoblastí mezi sebou a je aplikovana s nízkou pravděpodobností.

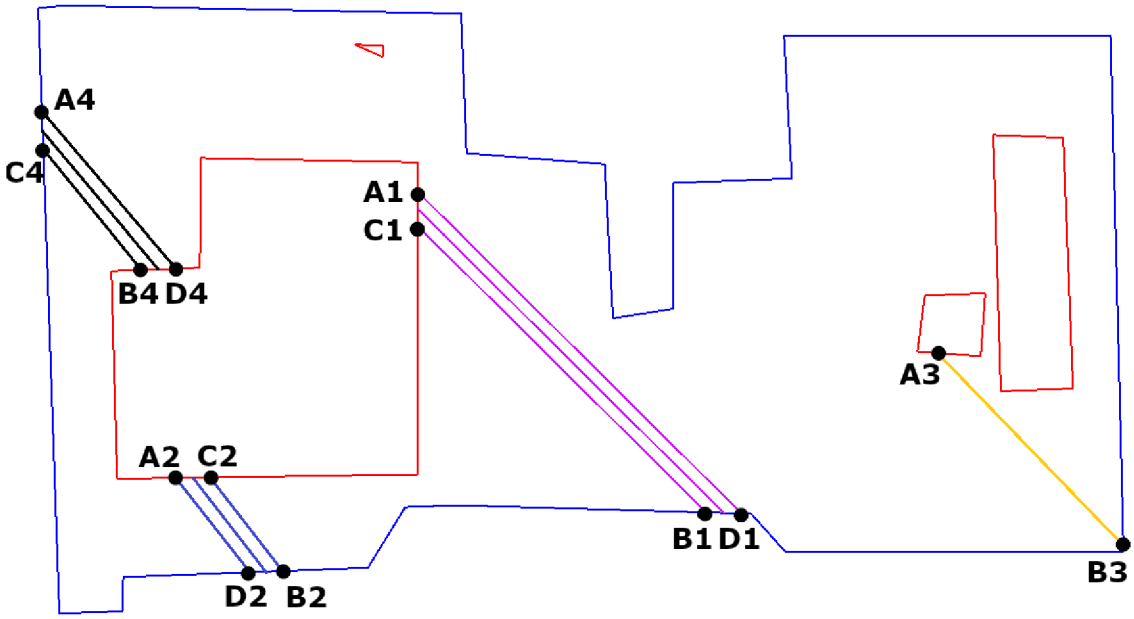
Nová generace první varianty se tvoří obměnou té stávající, kdy se vyberou pomocí ruletové selekce 2 rodiče, kteří následně pomocí křížení vytvoří dva potomky, lepšího z těchto potomků podrobíme případné mutaci a následně lokální optimalizaci pomocí metody 2-opt. Pokud je takto vytvořený potomek lepší, než horší z rodičů, tak jej místo tohoto rodiče nahradíme. Tento proces se opakuje ve zvoleném počtu iterací, nebo je ukončen časovým omezením. Druhá varianta slouží zejména k porovnání a využívá prvky elitismu, kdy se mění větší část stávající populace a novou generaci tvoří zejména nejstatnější jedinci generace předchozí. U této varianty není využito lokalní optimalizace a celkové srovnání těchto dvou metod se nachází v kapitole 9.9.

9.6 Optimální cesta pro pevné pořadí

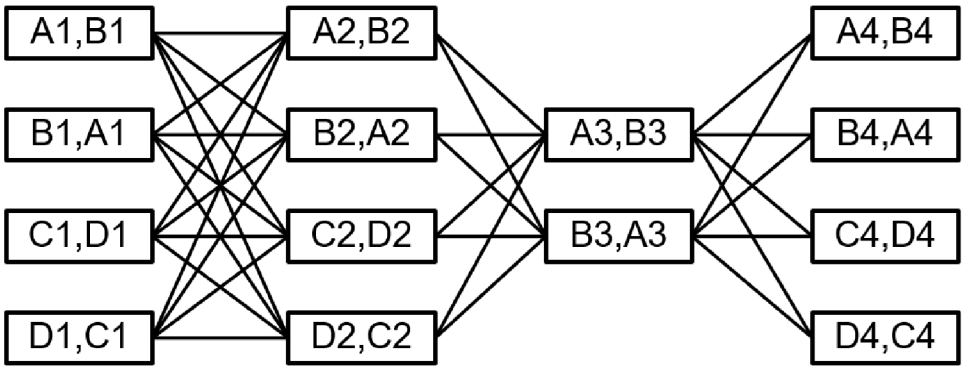
Následně je potřeba zjistit optimální cestu pro pevně stanovené pořadí mezi jednotlivými podoblastmi. Každá oblast, která je tvořena více než jednou čárou, má 4 možné body vjezdu, respektive výjezdu, značeny A až D. Oblasti tvořené pouze jednou čárou mají tyto body pouze 2 (A až B). Jelikož je cesta uvnitř podoblasti pevně stanovena jako pořadí jednotlivých paralelních čar, tudíž počáteční bod definuje i bod koncový. Tímto si můžeme pro každý vstupní bod vytvořit stav oblasti, který je tvořen dvojicí bodů a to vstupního a výstupního.

Pro nalezení nejlepší kombinace stavů oblastí je vhodné vytvořit pomocí dynamického programování graf, jehož každá vrstva představuje jednu oblast a vrcholy této vrstvy jsou její stavy (indexy 1 až N). Každý vrchol aktuální vrstvy je spojen s každým vrcholem následující etapy, nikoliv však s ostatními vrcholy aktuální vrstvy. Ohodnocení vrcholů je dáno délkou trasy uvnitř oblasti, přičemž pokud je pouze opačný směr tak je délka trasy stejná. Hrany mají hodnotu vzdálenosti výstupního bodu předchozí oblasti a vstupního bodu náledující oblasti, kdy je pro zjištění této vzdálenosti v prostoru s překažkami využit viditelnostní graf (více v kapitole 9.7). Příklad sestrojeného grafu 4 podoblastí (viz. obr. 33) lze vidět na obr. 34.

Pro takovýto graf zbývá už pouze najít nejkratší cestu mezi první a poslední vrstvou, na jejíž základě se vytvoří optimální cesta pro pevně stanovené pořadí pomocí dopředného algoritmu dynamického programování (viz. 5.4).



Obr. 33: Podoblasti se vstupními a výstupními body.



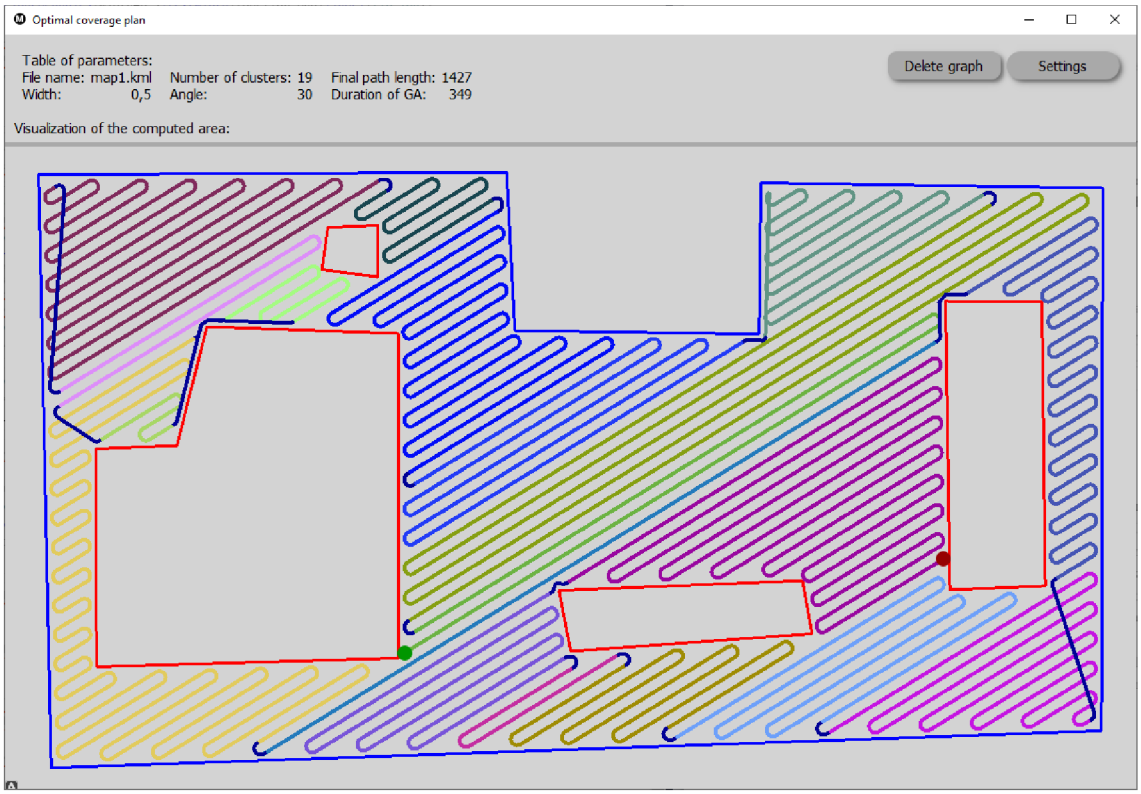
Obr. 34: Graf pro nalezení optimální cesty.

9.7 Viditelnostní graf

S ohledem na ohraničení sekacího prostoru nelze použít cestu mezi jednotlivými stavy (viz. 9.6) jako euklidovskou vzdálenost, jelikož sekačka nesmí jet mimo oblast určenou pro sekání. Z toho důvodu se využívá viditelnostního grafu (viz. kapitola 6), který je vytvořen pro celou oblast hned po získaní shluků. Při ohodnocování grafu pevně stanového pořadí oblastí se pak vždy požadované body přidají do grafu a zjistí se jejich nejkratší cesta. Vytvoření tohoto viditelnostního grafu proběhlo pomocí knihovny `pyvisgraf`, kdy se na základě okrajových objektů vytvoří graf a pro zjištění nejraktší cesty mezi dvěma body se využívá její metoda `shortest_path`.

9.8 Vyhlazení výsledné cesty

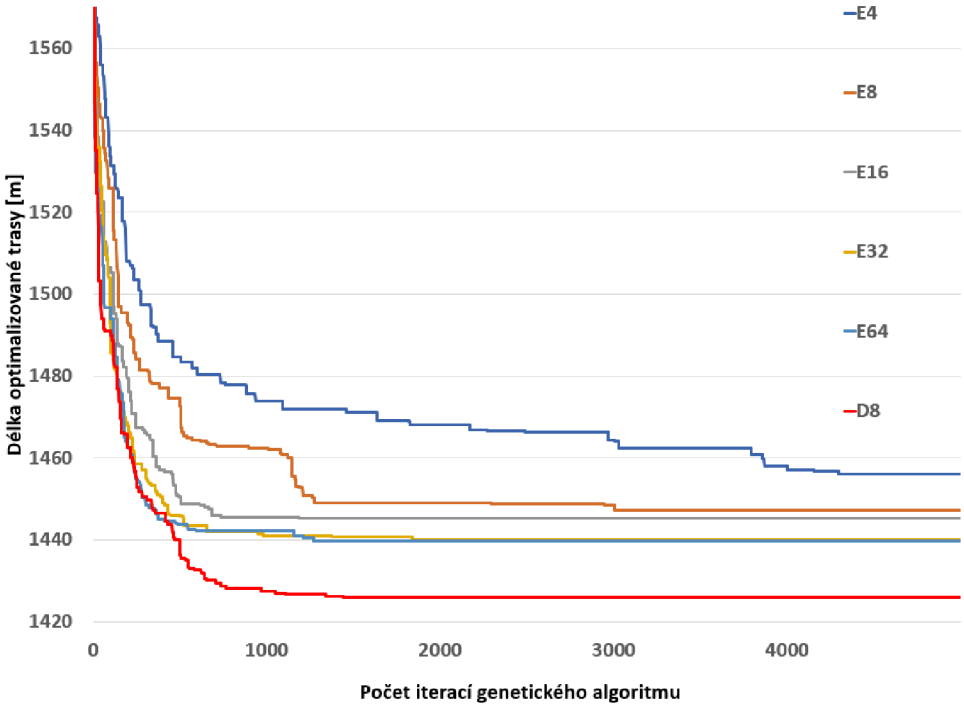
Pro plynulý pohyb sekačky je nutné vyhladit ostré hrany cesty a to pomocí maximálního radiusu otáčení, který je zvolen jako polovina šířky sekačky. Vyhlazení se aplikuje na jednotlivé podoblasti i přejezdy mezi jednotlivými oblastmi, přičemž výsledná cesta lze vidět na obr. 35, kdy je počátek cesty znázorněn zeleným bodem, dále jsou jednotlivé segmenty odděleny barevně, kdy přejezdy jsou vyznačeny modrou čarou a konec cesty červeným bodem.



Obr. 35: Vyhlazení výsledné optimalizované cesty.

9.9 Numerické simulace

Z důvodu zjištění vhodnosti a porovnání dvou aplikovaných genetických algortimů byla provedena řada simulací konkrétních oblastí určených pro sekání pomocí robotické sekačky. Důvodem prvního souboru simulací bylo porovnání kvality obou metod a závislost dosažené kvality na počtu iterací a velikosti populace. Další sada již zkoumala lepší z uvedených metod, kdy proběhlo testování několika oblastí pro více úhlů a na tomto základě se pozorovala závislost počtu iterací na množství rozdělených podoblastí. Cílem těchto simulací bylo také zjištění vhodných parametrů, které byly následně zvoleny jako defaultní nastavení genetického algoritmu v softwarové implementaci.



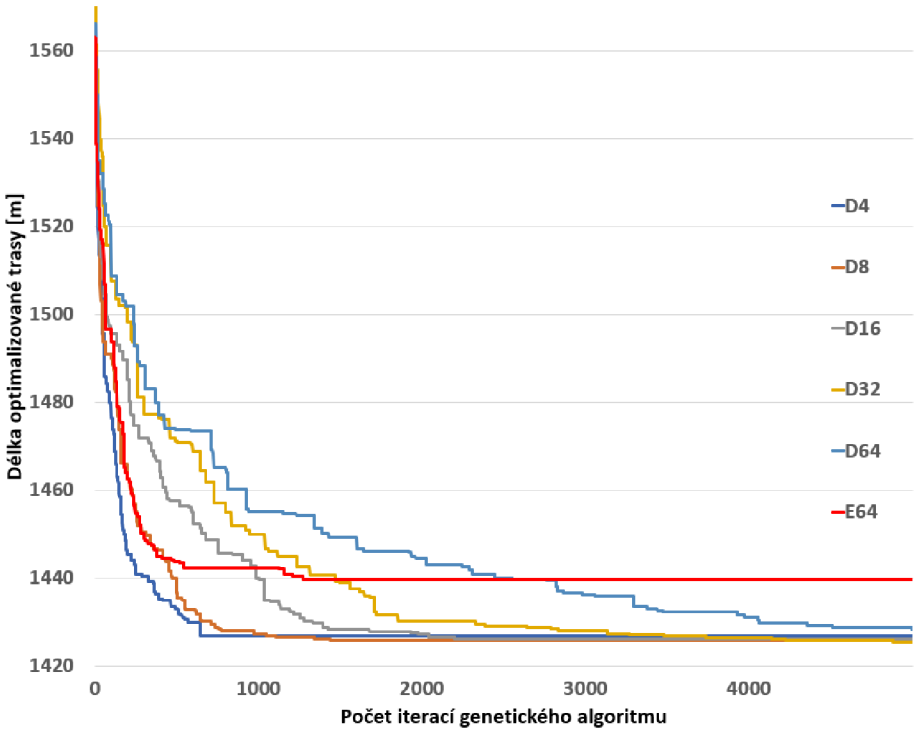
Obr. 36: Výsledek simulace GA využívající elitismu.

9.9.1 Porovnání metod GA

Jak již bylo zmíněno, tato část zkoumá porovnání genetického algoritmu využívajícího elitsmus a GA s lokální optimalizací. Pro testování byla vybrána jednotná oblast (rozměry 30x20 metrů), úhel 45° a každá metoda proběhla simulací za měnící se velikosti populace, konkrétně tedy pro hodnoty 4, 8, 16, 32 a 64. Jelikož se v GA objevují prvky náhody, každý proces byl proveden pětkrát, přičemž následně využíváme jejich průměrnou hodnotu. Počet iterací genetického algoritmu byl nastaven na 5000 a časový limit na 45 minut. Vytvoření viditelnostního grafu závisí

pouze na charakteru oblasti a to hlavně celkovém počtu překážek, pro představu pro tuto oblast trvalo vygenerování tohoto grafu 72s. Výsledky jednotlivých metod jsou uvedeny dále na obrázcích 36, 37.

Na obrázku 36 vidíme výslednou závislost délky optimalizované trasy (osa Y) v průběhu iterací (osa X) na velikosti populace. Křivky E4 až E64 (číslo znační velikost populace) zobrazují GA s využitím elitismu, přičemž je z grafu patrné, že pro rostoucí velikost populace klesá délka celkové trasy (dochází ke zvyšování fitness ohodnocení). Dále z grafu vyplývá dosažení lepšího ohodnocení za menší počet iterací pro simulace s větší populací. Pro porovnání s druhou metodou je zde vykreslena křivka s označením D8 (lokální optimalizace využívající 2-opt s velikostí populace 8), která v porovnání s ostatními značně rychleji nabývá kratších délek trasy a celkově dosahuje lepší výsledné hodnoty. Časová náročnost jednoho procesu je taktéž úměrná k velikosti populace, přičemž pro jednotlivé populace 4 až 64 byly průměrné časy potřebné k dosažení 5000 iterací 65s, 174, 683s, 2120s a pro populaci 64 byl dosažen časový limit namísto limitu iteračního (předpokládaná doba pro dosažení počtu iterací je okolo 6700s).

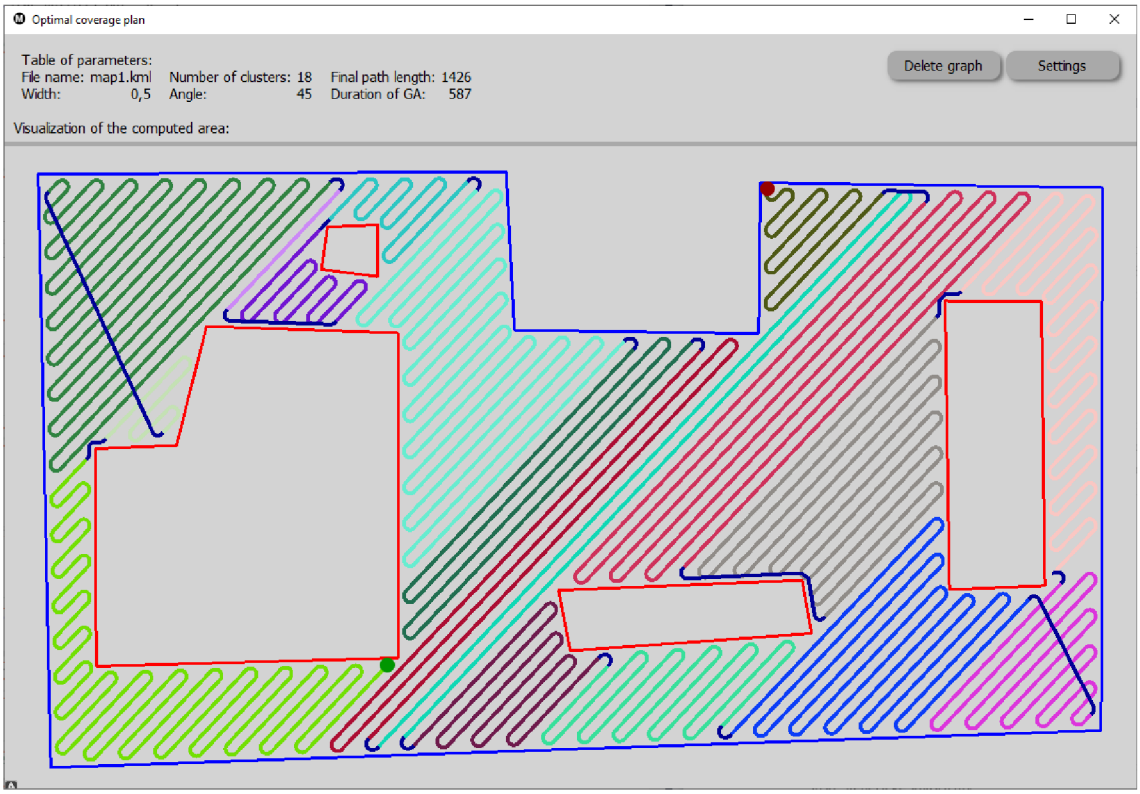


Obr. 37: Výsledek simulace GA s lokální optimalizací.

Graf na obr. 37 znázorňuje stejnou závislost jako v předchozím grafu, jen pro druhou variantu genetického algoritmu. Jednotlivé křivky populací jsou značeny D4 až D64 a pro porovnání je zde vykreslena také křivka E64 (nejvhodnější metoda prvního typu GA). Z tohoto grafu lze pozorovat menší rozptyl pro jednotlivé ve-

likosti populací a také celkově lepší hodnoty délky trasy oproti předchozí metodě (D64), přičemž simulace s menší velikostí populace mají tendenci rychlejšího dosažení lepších hodnot celkové délky optimalizované trasy. Jako vhodná velikost se jeví nejrychleji klesající křiva D4, která ovšem ve výsledku nedosahuje tak dobrých hodnot a proto je jako výsledný parametr zvolena velikost populace 8. Tato volba je výhodná z důvodu rychlého dosažení velice dobrých hodnot délky trasy (okolo 1000 iterací) a také ustálení na druhé nejnižší průměrné hodnotě (1425,91), hned po D32 (1425,49). Z hlediska časové náročnosti není tato metoda náchylná na velikost populace a průměrně trvalo dokončení 5000 iterací 500 až 1400 sekund.

Na základě těchto simulací je zjevné, že je metoda s lokální optimalizací vhodnější a jako přednastavený parametr velikosti populace je zvolena hodnota 8. Dále je zjištěna přibližná hodnota iterací potřebná pro ustálení optimalizované délky trasy, která se pohybuje pro zvolenou velikost populace okolo 2000. Tyto parametry jsou dále použity v následujících simulacích. Pro představu je na obr. 38 zobrazena výsledná optimalizovaná trasa.

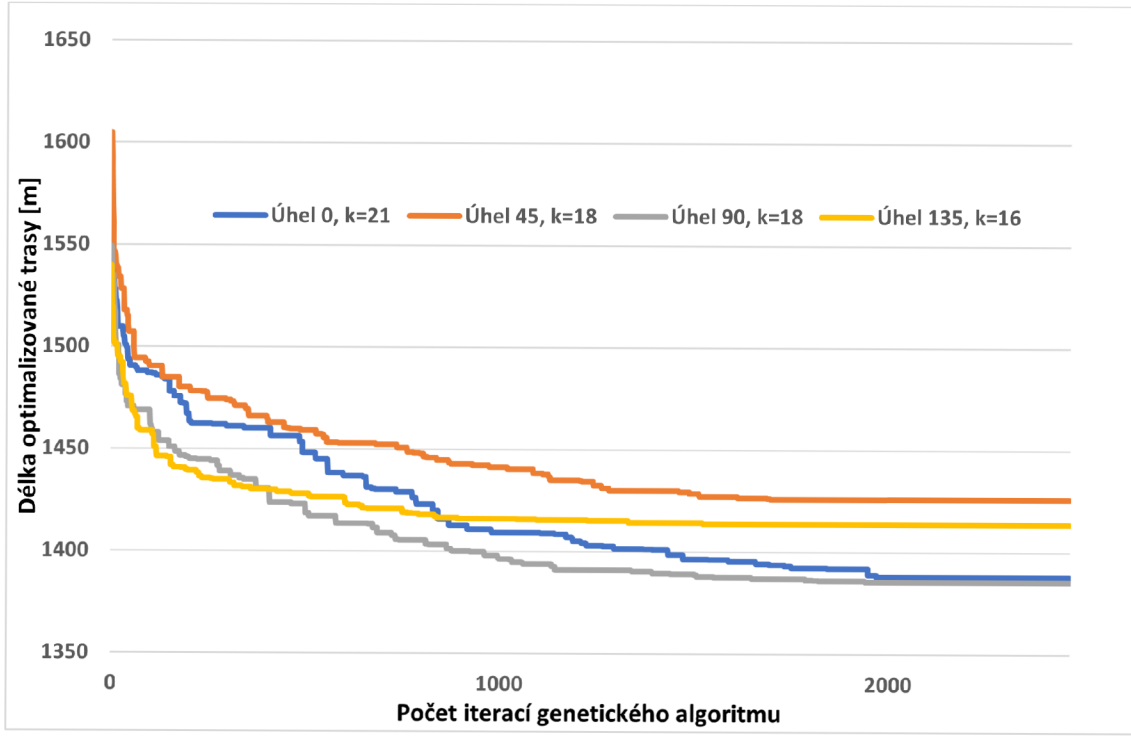


Obr. 38: Výsledná optimalizovaná cesta pro populaci velikosti 8.

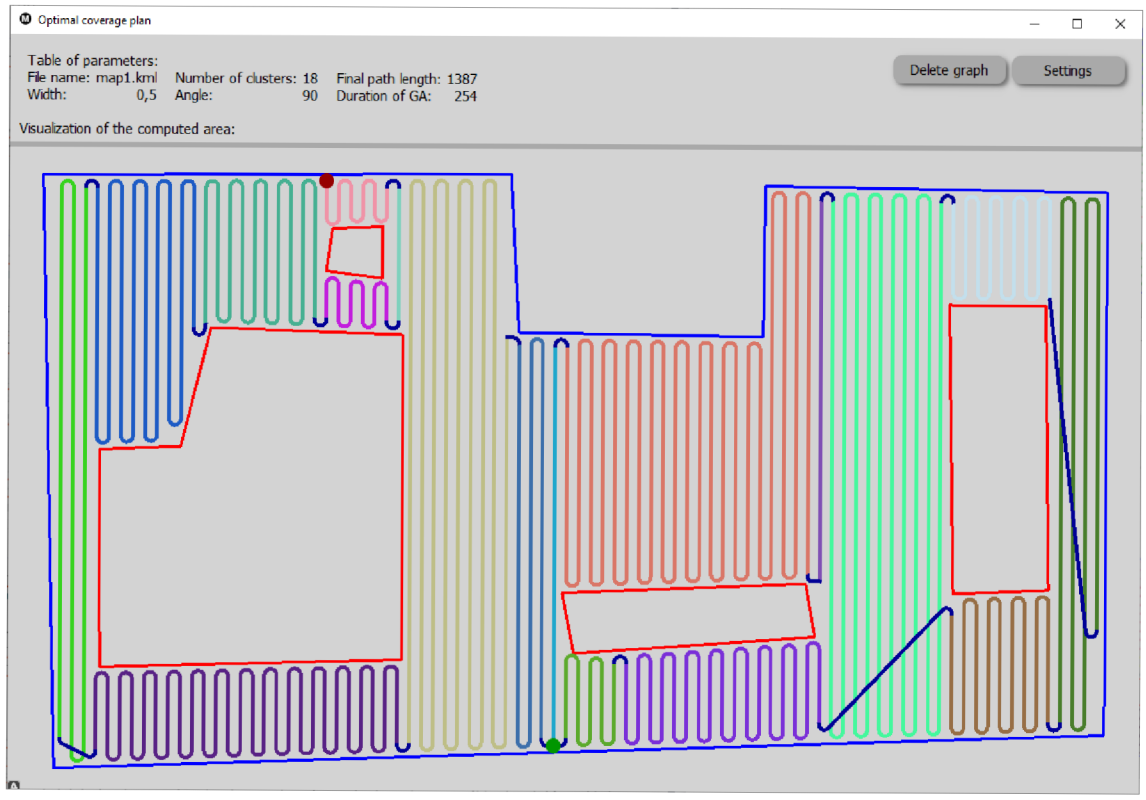
9.9.2 Závislost počtu iterací na počtu podoblastí

Pro potvrzení předchozího nastavení a také z důvodu prokázání důležitosti vybraného úhlu je provedena další skupina simulací. Tentokrát jsou přidány další dvě oblasti, přičemž použitá je pouze druhá varianta GA a simulace jsou provedeny pro úhly 0, 45, 90 a 135°. Obdobně jako v předchozích testech je proveden každý proces pětkrát. Rozměry druhé mapy jsou 40x40 metrů a poslední z oblastí má rozměry 50x40 metrů.

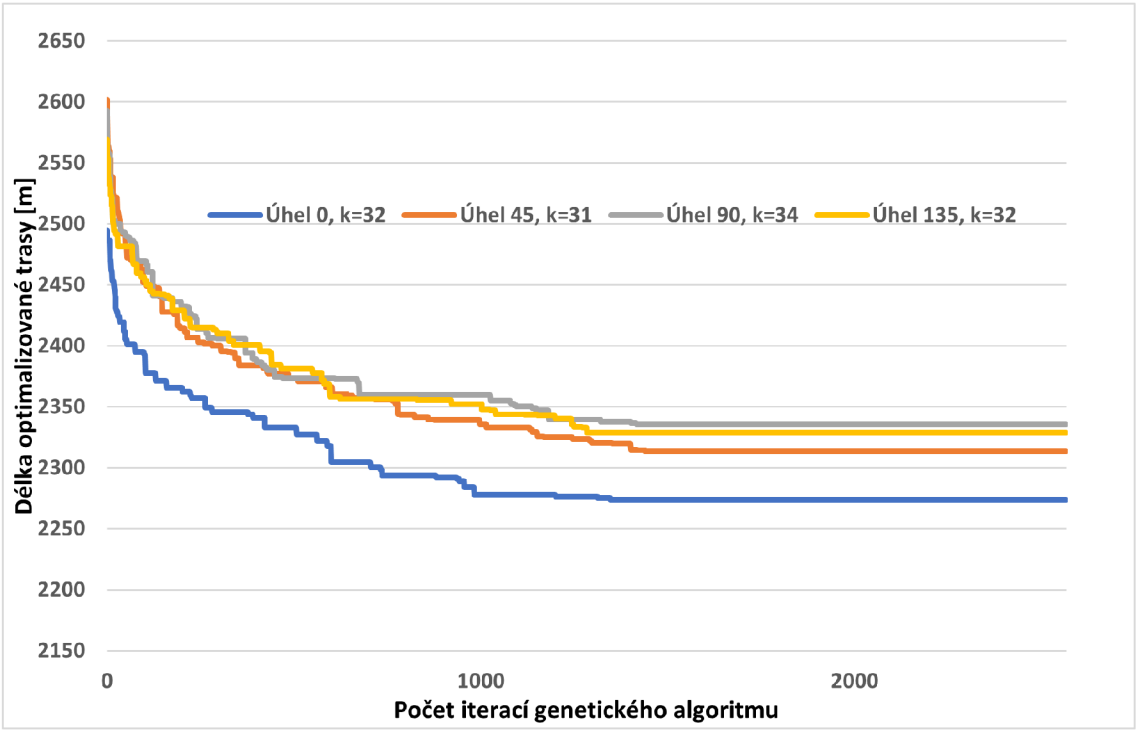
Výsledky jednotlivých simulací lze vidět na obrázcích (39 až 44), kde je zobrazena závislost počtu iterací na počtu rozdělených podoblastí k. Z těchto grafů je zřejmé, že mezi k a počtem iterací není přímá závislost a záleží na mnoha dalších aspektech prostředí. Z výsledků pro oblast č. 2 je ale zřejmé, že počet iterací přednastavený z předešlých simulací nemusí být pro oblasti s větším počtem podoblastí dostačující (stejně jako časový limit). Tento jev je ale pro úlohy TSP typický, jelikož se výpočetní náročnost zvyšuje s roustoucím počtem oblastí. Pro tuto oblast trvalo vygenerování viditelnostního grafu 353s, což je dáno větším počtem celkových překážek v oblasti. Poslední oblast dle výsledné optimalizované cesty působí stejně přijatelně jako oblast č. 1 (minimalizované délky přejezdových čar). Tato oblast je o něco méně členitá než předchozí a vytvoření viditelnostního grafu zde zabralo průměrně 140s. Pro oblati 1 a 3, které nebyly ukončeny časovým limitem, trval výpočet genetického algoritmu v průměru 244s, respektive 520s. Dále je z výsledných úhlů nejlepších simulací zjevné, že je velice vhodné volit úhel tak, aby byly čáry paralelní s některou stranou okrajové hranice.



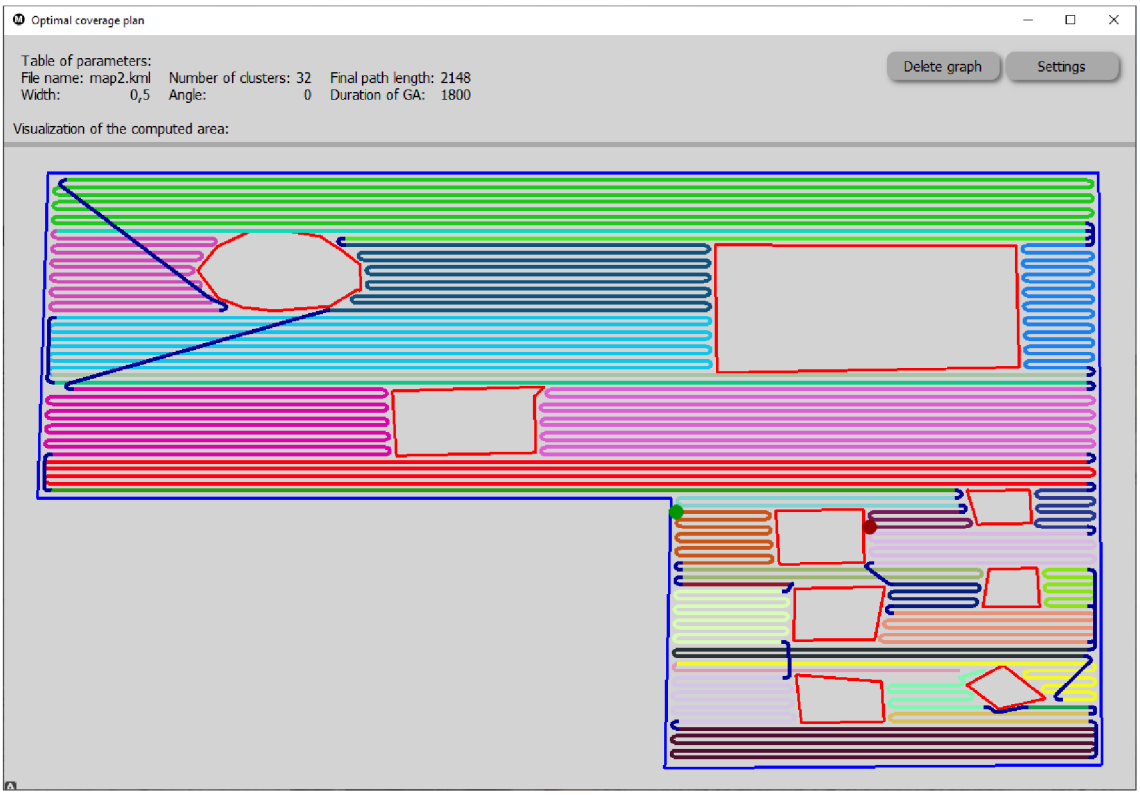
Obr. 39: Graf závislosti úhlu na počtu podoblastí oblasti č. 1.



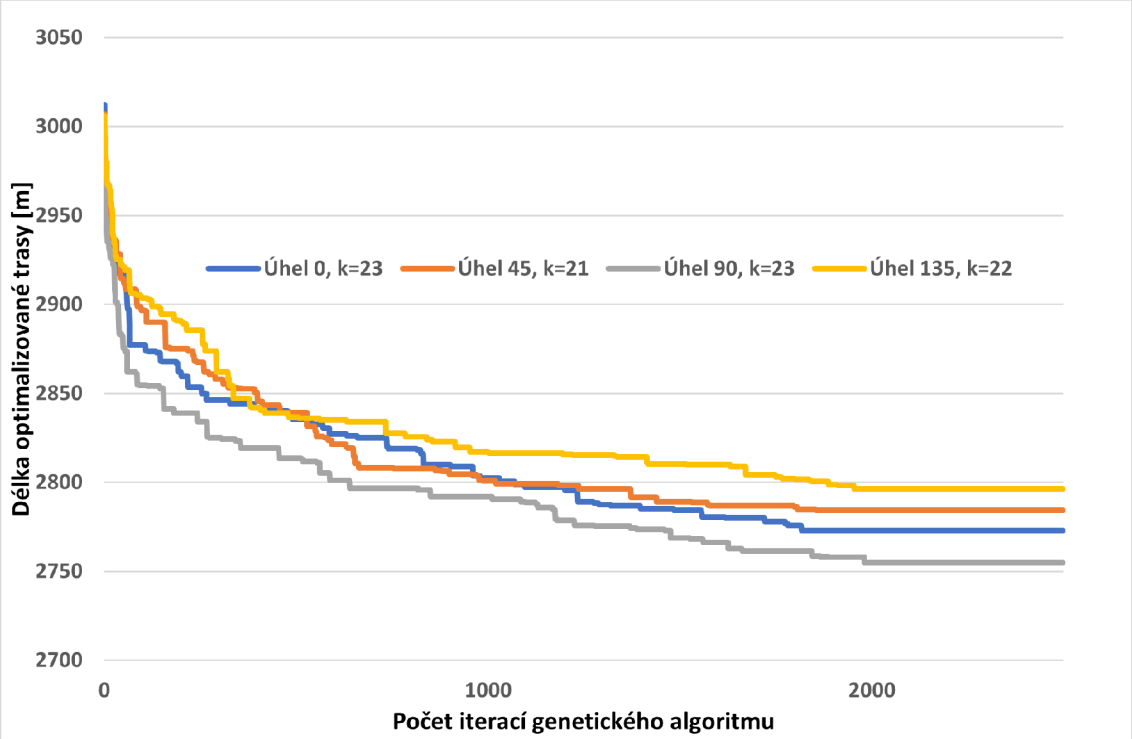
Obr. 40: Výsledná optimalizovaná cesta oblasti č. 1.



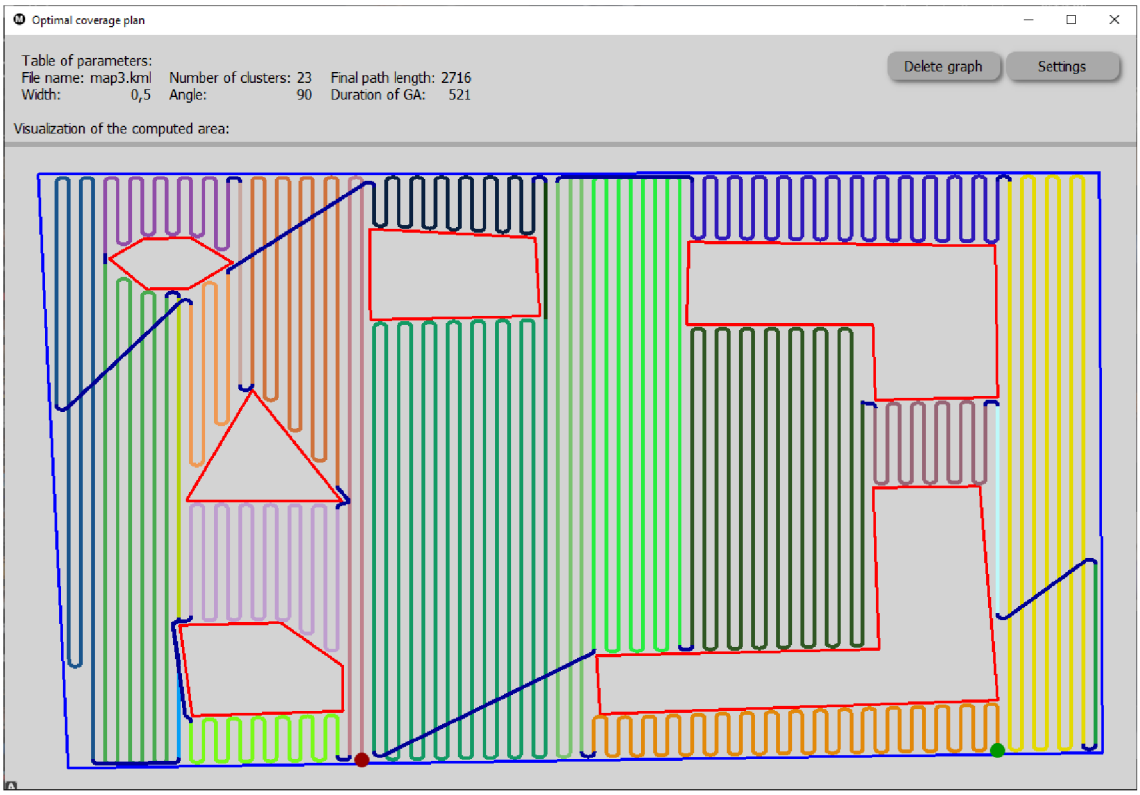
Obr. 41: Graf závislosti úhlu na počtu podoblastí oblasti č. 2.



Obr. 42: Výsledná optimalizovaná cesta oblasti č. 2.



Obr. 43: Graf závislosti úhlu na počtu podoblastí oblasti č. 3.



Obr. 44: Výsledná optimalizovaná cesta oblasti č. 3.

10 ZÁVĚR

Tato práce se zabývala problémem optimalizace cesty pro autonomní robotickou sekačku a v jejím rámci byl vytvořen nástroj, který pro soubor s GPS souřadnicemi dané oblasti a vstupní parametry vytvoří a optimalizuje cestu, která pokrývá celou plochu s ohledem na ohraničení prostoru a pevně stanovené vnitřní překážky.

V první části se nachází rešerše několika aktuálně používaných metod, kdy po stručném výčtu těchto metod, jejich porovnání a kontrole vhodnosti využití následuje formulace přístupu k danému problému, kterým se tato práce zabývá. Tento přístup využívá některé prvky z metod rešerní části, další jsou pak přidány z důvodu reálnější výsledné cesty. Konceptem tohoto přístupu je rozdělení oblasti na několik podoblastí pomocí shlukové analýzy, přičemž je kladen důraz na to, aby byla každá podoblast z hlediska optimální dráhy uvnitř triviální. Tomuto rozdělení předchází prolnutí celé oblasti rovnoběžnými čárami, které s vodorovnou osou svírají pevně stanovený úhel definovaný jako vstupní parametr. Dále následuje optimalizace pořadí podoblastí, která je řešena genetickým algoritmem a nejlepší kombinaci vstupních a výstupních bodů oblastí získává dopředný algoritmus dynamického programování. Tuto výslednou optimalizovanou trasu vyhlazují Dubinsovy křivky.

Následující část obsahuje teoretické základy oblastí, které jsou využity pro řešení daného problému, přičemž je kladen důraz na konkrétní využité metody a další jsou pouze zmíněny. Struktura této kapitoly je dána chronologicky tak, jak jsou postupně metody aplikovány při řešení problému.

V další kapitole je popsána softwarová implementace, kdy byl použit programovací jazyk `Python` a jeho knihovny. Po výčtu funkcionalit a možností rozhraní je obdobně jako v teoretické části detailně popsáno řešení jednotlivých částí problému z hlediska implementace. Pro prezentaci dosažených výsledků bylo provedeno několik souhrnných simulací, ze kterých vycházejí patřičné závislosti na počtu podoblastí, členistosti prostoru a zvoleném úhlu natočení.

Výsledné optimalizované cesty při vhodně nastavených parametrech dosahují velice dobrých výsledků co se týče délky celkové trasy. Jako námět pro další činnost se určitě jeví vyhlazení výsledné cesty pomocí Dubinsových křivek, které v určitých případech nevyhladí trasu úplně správně a dochází k nepřirozeným pohybům sekačky v některých okrajových situacích.

11 SEZNAM POUŽITÉ LITERATURY

[1] Bellman, R.: *Dynamic programming and stochastic control processes. Information and Control*, ročník 1, č. 3, 1958: s. 228–239, ISSN 0019-9958, doi: 10.1016/S0019-9958(58)80003-0.

[2] Bertsekas, D. P.: *Dynamic programming and optimal control: Vol. 1.* Athena scientific Belmont, 2000.

[3] Choset, H.; Lynch, K.; Hutchinson, S.; aj.: *Principles of Robot Motion: Theory, Algorithms, and Implementation.* A Bradford Book, 2005, ISBN 9780262033275.

[4] De Berg, M.; Cheong, O.; Van Kreveld, M.; aj.: *Computational geometry: introduction. Computational geometry: algorithms and applications*, 2008: s. 1–17.

[5] Everitt, B.; Landau, S.; Leese, M.; aj.: *Cluster Analysis.* Wiley Series in Probability and Statistics, Wiley, 2011, ISBN 9780470978443.

[6] Galceran, E.; Carreras, M.: *A survey on coverage path planning for robotics. Robotics and Autonomous Systems*, ročník 61, č. 12, 2013: s. 1258–1276, doi: 10.1016/j.robot.2013.09.004.

[7] Hameed, I. A.: *Intelligent behavior of autonomous vehicles in outdoor environment. Technical Report Mechanical Engineering*, ročník 1, č. 2, 2012.

[8] Hameed, I. A.: *Intelligent Coverage Path Planning for Agricultural Robots and Autonomous Machines on Three-Dimensional Terrain. Journal of Intelligent & Robotic Systems*, ročník 74, 06 2013, doi:10.1007/s10846-013-9834-6.

[9] Hameed, I. A.: *Coverage path planning software for autonomous robotic lawn mower using Dubins' curve. 2017 IEEE International Conference on Real-time Computing and Robotics (RCAR)*, 2017: s. 517–522, doi:10.1109/RCAR.2017.8311915.

[10] Hameed, I. A.; Bochtis, D.; Sørensen, C. A.: *Driving Angle and Track Sequence Optimization for Operational Path Planning Using Genetic Algorithms. Applied Engineering in Agriculture*, ročník 27, 2011: s. 1077–1086, doi:10.13031/2013.40615.

[11] Hameed, I. A.; Bochtis, D.; Sørensen, C. A.: *An Optimized Field Coverage Planning Approach for Navigation of Agricultural Robots in Fields Involving Obstacle Areas. International Journal of Advanced Robotic Systems*, ročník 10, č. 5, 2013: str. 231, doi:10.5772/56248.

[12] Harris, J.; Hirst, J.; Mossinghoff, M.: *Combinatorics and Graph Theory*. Undergraduate Texts in Mathematics, Springer New York, 2009, ISBN 9780387797113.

[13] Holland, J.: *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. University of Michigan Press, 1975, ISBN 9780472084609.

[14] Hougardy, S.; Zaiser, F.; Zhong, X.: *The approximation ratio of the 2-Opt Heuristic for the metric Traveling Salesman Problem. Operations Research Letters*, ročník 48, č. 4, 2020: s. 401–404, ISSN 0167-6377, doi:10.1016/j.orl.2020.05.007.

[15] Huang, W.: *Optimal line-sweep-based decompositions for coverage algorithms. Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, ročník 1, 2001: s. 27–32 vol.1, doi:10.1109/ROBOT.2001.932525.

[16] Ikuo, Y.; Kunihito, Y.; Moritoshi, Y.; aj.: *Greedy Genetic Algorithms for Symmetric and Asymmetric TSPs. Greedy Genetic Algorithms for Symmetric and Asymmetric TSPs*, ročník 43, č. 10, 2002: s. 165–175.

[17] Kočí, J.: *Optimalizace PID regulátoru pomocí evolučních výpočetních technik.* Diplomová práce, Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2018, 63 s., vedoucí práce: Ing. et Ing. Stanislav Lang.

[18] LaValle, S. M.: *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006, available at http://planning.cs.uiuc.edu/.

[19] Ma, C.; Wu, J.: *Data Clustering: Theory, Algorithms, and Applications. ASASIAM Series on Statistics and Applied Probability*, 2007, doi:10.1137/1.9780898718348.

[20] Michalewicz, Z.; Michalewicz, Z.: *Genetic Algorithms + Data Structures = Evolution Programs*. Artificial intelligence, Springer, 1996, ISBN 9783540606765.

[21] Nguyen, H. D.; Yoshihara, I.; Yamamori, K.; aj.: *Implementation of an Effective Hybrid GA for Large-Scale Traveling Salesman Problems. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, ročník 37, č. 1, 2007: s. 92–99, doi:10.1109/TSMCB.2006.880136.

[22] Pelleg, D.; Moore, A. W.: *X-Means: Extending K-Means with Efficient Estimation of the Number of Clusters*. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000, ISBN 1558607072, str. 727–734.

[23] Roughgarden, T.: *Algorithms Illuminated: Graph algorithms and data structures. Part 2*. Algorithms Illuminated, Soundlikeyourself Publishing LLC, 2018, ISBN 9780999282922.

[24] Sivanandam, S.; Deepa, S.: *Introduction to Genetic Algorithms*. Springer Berlin Heidelberg, 2007, ISBN 9783540731900.

[25] Skrodzki, M.: *The k-d tree data structure and a proof for neighborhood computation in expected logarithmic time*. *arXiv:1903.04936*, 2019.

[26] Vasquez-Gomez, J. I.; Herrera-Lozada, J.-C.; Olguin-Carbajal, M.: *Coverage Path Planning for Surveying Disjoint Areas*. 2018: s. 899–904, doi:10.1109/ICUAS.2018.8453386.

[27] Vasquez-Gomez, J. I.; Melchor, M. M.; Herrera Lozada, J. C.: *Optimal Coverage Path Planning Based on the Rotating Calipers Algorithm*. 2017: s. 140–144, doi:10.1109/ICMEAE.2017.11.

[28] Šeda, M.: *Teorie grafů*. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav automatizace a informatiky, listopad 2003.