

**VYSOKÁ ŠKOLA BÁŇSKÁ - TECHNICKÁ UNIVERZITA OSTRAVA**  
**HORNICKO - GEOLOGICKÁ FAKULTA**  
**Katedra geoinformatiky**

**APLIKACE PRO TVORBU PÁTRACÍCH SEKTORŮ  
NA ZÁKLADĚ PŘIROZENÝCH BARIÉR**

**DIPLOMOVÁ PRÁCE**

**Autor:**  
**Vedoucí diplomové práce:**

**Bc. Vendula Sládková**  
**Ing. Jan Růžička, Ph.D.**

**Ostrava 2019**

VŠB - Technická univerzita Ostrava  
Hornicko-geologická fakulta  
Katedra geoinformatiky

## Zadání diplomové práce

Student: **Bc. Vendula Sládková**

Studijní program: N3654 Geodézie, kartografie a geoinformatika

Studijní obor: 3608T002 Geoinformatika

Téma: Aplikace pro tvorbu pátracích sektorů na základě přirozených bariér  
Software for Search Sectors Creation Based on Natural Barriers

Jazyk vypracování: čeština

### Zásady pro vypracování:

Cílem projektu je vyvinout aplikaci, která by dokázala rozdělit prostor do pátracích sektorů pro Search and Rescue operaci se psy a rojnicemi. K rozdělení by měly být využity přirozené bariéry jako jsou silnice, železnice, vodní toky, apod.

### Úkoly:

1. Popište možnosti rozdělování prostoru dle přirozených bariér.
2. Navrhněte algoritmus realizující rozdělování prostoru dle konzultací se zadavatelem.
3. Realizujte pilotní aplikaci implementující navržený algoritmus.
4. Pilotní aplikaci otestujte na cvičných datech.
5. Zhodnoťte omezení aplikace.

### Rozsah grafických prací:

dle potřeby

### Rozsah původní zprávy:

50 - 70 normostran textu

Formální náležitosti diplomové práce stanoví směrnice děkana HGF HGF\_SME\_15\_001 Pokyny pro zpracování závěrečných prací, zveřejněné na webových stránkách fakulty - [https://www.hgf.vsb.cz/cs-old/portal-iso/interni/platne-dokumenty/sme/HGF\\_SME\\_15\\_001\\_A.pdf](https://www.hgf.vsb.cz/cs-old/portal-iso/interni/platne-dokumenty/sme/HGF_SME_15_001_A.pdf).

### Seznam doporučené odborné literatury:

- \* Ruas A. Map Generalization. Encyclopedia of GIS. Cham: Springer International Publishing, 2017, 2017-5-12, , 1167-1169. DOI: 10.1007/978-3-319-17885-1\_743. ISBN 978-3-319-17884-4.
- \* Jozífová K. Analýza činnosti záchranářských psů v rámci Integrovaného záchranného systému [online]. Praha, 2011. Dostupné z: <https://dspace.cuni.cz/handle/20.500.11956/48219>. Diplomová práce. Univerzita Karlova.
- \* Campbell J.E. Shin M. Geographic Information System Basics. 2012. Available at: <http://2012books.lardbucket.org/pdfs/geographic-information-system-basics.pdf>

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Jan Růžička, Ph.D.**

Datum zadání: 31.10.2018

Datum odevzdání: 30.04.2019



---

doc. Ing. Michal Kačmařík, Ph.D.  
*vedoucí katedry*



---


prof. Ing. Vladimír Slivka, CSc., dr.h.c.  
*děkan fakulty*

### ***Prohlášení autora diplomové práce***

- Celou diplomovou práci včetně příloh, jsem vypracovala samostatně a uvedla jsem všechny použité podklady a literaturu. Byla jsem seznámena s tím, že na moji diplomovou práci se plně vztahuje zákon č.121/2000 Sb. - autorský zákon, zejména § 35 – využití díla v rámci občanských a náboženských obřadů, v rámci školních představení a využití díla školního a § 60 – školní dílo.
- Beru na vědomí, že Vysoká škola báňská – Technická univerzita Ostrava (dále jen VŠB-TUO) má právo nevýdělečně, ke své vnitřní potřebě, diplomovou práci užít (§ 35 odst. 3).
- Souhlasím s tím, že jeden výtisk bude uložen u vedoucího diplomové práce. Souhlasím s tím, že údaje o diplomové práci, obsažené v Záznamu o závěrečné práci, umístěném v příloze mé diplomové práce, budou zveřejněny v informačním systému VŠB-TUO.
- Souhlasím s tím, že diplomová práce je licencována pod Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported licencí. Pro zobrazení kopie této licence, je možno navštívit <http://creativecommons.org/licenses/by-nc-sa/3.0/>
- Bylo sjednáno, že s VŠB-TUO, v případě zájmu o komerční využití z její strany, uzavřu licenční smlouvu s oprávněním užít dílo v rozsahu § 12 odst. 4 autorského zákona.
- Bylo sjednáno, že užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu komerčnímu využití mohu jen se souhlasem VŠB-TUO, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly VŠB-TUO na vytvoření díla vynaloženy (až do jejich skutečné výše).

V Ostravě dne 24.4.2019

Bc. Vendula Sládková



**Anotace práce:**

Cílem projektu je vyvinout aplikaci, která by dokázala rozdělit prostor do pátracích sektorů pro Search and Rescue operace se psy a rojnicemi. K rozdělení by měly být využity přirozené bariéry jako jsou silnice, železnice, vodní toky apod.

**Klíčová slova:**

Segmentace, pátrání po pohřešovaných osobách, polygonizace, algoritmus, Java, mapa, bariéra, přirozená překážka, sektor, generalizace, agregace.

**Annotation:**

A goal of this project is a developing an application for segmentation of space to searching areas for Search and Rescue operations with dogs and search teams. For dividing would be used natural barriers such as roads, railways, water streams and so on.

**Keywords:**

Segmentation, Search and Rescue operation, polygonization, algorithm, Java, map, barrier, natural border, sector, generalization, aggregation .

## **Poděkování**

Ráda bych poděkovala vedoucímu diplomové práce Ing. Janu Růžičkovi, Ph.D. za rady, díky kterým jsem zdárně dokončila tuto práci. Také děkuji doc. Ing. Daně Vrublové, Ph.D. za podporu a pomoc.

## OBSAH

1	Úvod.....	1
2	PÁTRAC.....	3
3	Základní pojmy .....	5
3.1	Search and Rescue operace v České republice.....	5
3.2	Pátrání po pohřešovaných osobách .....	5
3.3	Hranice .....	7
3.4	Heuristika .....	8
4	Způsoby tvorby sektorů.....	9
4.1	Generalizace .....	9
4.2	Metody řešící podobnou problematiku .....	11
4.3	Shrnutí metod .....	14
5	Hodnocení tvaru .....	15
5.1	Tvarová analýza dat .....	15
6	Podmínky pro výstup dat.....	17
7	Vstupní data.....	19
7.1	Barevné schéma .....	21
7.2	Příprava vstupních dat.....	22
8	Využití konvenčních prostředků .....	25
8.1	ArcGIS Desktop .....	26
8.2	QGIS .....	28
9	Vlastní řešení.....	32
9.1	Použité technologie .....	32
9.2	Algoritmus vlastního řešení .....	33
9.3	Implementace .....	38
9.4	Struktura projektu.....	39
9.5	Výstupy .....	48
10	Hodnocení .....	52
10.1	Hodnocení polygonů .....	52
10.2	Hodnocení výstupů.....	53
10.3	Omezení programu.....	56
11	Závěr.....	58
	Seznam literatury	
	Seznam zkratk	

Seznam obrázků

Seznam tabulek

Seznam grafů

Seznam příloh na CD

Příloha 1: intravilán



## 1 ÚVOD

Nedílnou součástí přípravy pátrací akce po pohřešovaných osobách během plošného vyhledávání je určení neboli vytvoření sektorů, kdy každý sektor je přiřazen veliteli a ten následně řídí a koordinuje pátrací akci v rámci přiděleného sektoru. Plánování v rámci prohledávaného území začíná v kanceláři, kdy si operátor sedne před monitor počítače a musí vymezit sektory, které by neměly být ani moc velké a ani moc malé. Musí dbát na nebezpečné překážky a zhodnotit, jaká je pravděpodobnost, že se pohřešovaná osoba bude v dané oblasti nacházet. K tomu slouží další metody a to, jak předpovědět chování pohřešované osoby v terénu. Je rozdíl, když se ztratí osoba střízlivá nebo v podnapilém stavu.

Tvorba sektoru je aktuálně manuální prací. Proto se hranice výstupu nemusí shodovat s hranicemi podkladu, který vymezuje různé typy území ve zpracovávané oblasti. Není cílem, aby vznikala přesná návaznost na hranici, protože psovod nebo rojnice může z daného sektoru vyjít a následně se do něj vrátit. Hlavním cílem je většinu manuální práce zautomatizovat s tím, že hlavní podíl práce bude stále na operátorovi nebo uživateli. Jejich úkolem je vytvořit sektory patřící do území určeného pro pátrání. Následně operátor sám určí, zda je vybraný sektor vhodný, nebo jej rozdělí na dva menší anebo jej spojí s jiným sousedním sektorem. V takovém případě hovoříme o polo-automatizované tvorbě sektorů.

Geografické informační systémy jako je *ArcGIS Desktop* nebo *QGIS* pro zpracovávání prostorových dat nabízejí funkce, které jsou vhodné pro tvorbu sektorů. Musí se však dávat pozor, zda je uživatel ochoten zaplatit licenci za *ArcGIS Desktop*. *QGIS* není placený a jeho výhodou je, že je *open source* a mnoho uživatelů, respektive vývojářů, vytváří další funkcionalitu, která se může stáhnout jako *plugin*, popřípadě *modul*. Nevýhodou je však verze *pluginů* a často se stává, že daný *plugin* s vyhovující funkcionalitou je pro starší verzi aplikace. Nainstalování starší verze nemusí vyřešit problém, protože aplikace padá a není stabilní.

Problematika tvorby sektorů z hlediska generalizace není novinkou a na toto téma existuje mnoho řešení. Z hlediska tvorby sektorů, kdy je třeba přidat překážky, definovat typy ploch, které se mohou nebo nesmějí sloučit, neexistuje konkrétní řešení, a výsledkem by byly sektory určené pro pátrací akce. Dále se může do řešení zakomponovat ideální rozdělování ploch na menší, rozdělování a slučování sektorů se „špatným“ tvarem. Není možné definovat jednoznačné řešení, protože vždy bude záležet na tom, co od výsledků očekává uživatel a na jednoduchosti implementace a definici zadání. Příkladem může být, jestli je jednodušší spojovat nebo rozdělovat polygony, či definování, co všechno je bariérou.

Když se jeden velký problém, jak vytvořit sektory, rozdělí na menší, vznikne množství menších problémů a je nutné je rozdělit na více zpracování, například na základní tvorbu sektorů, úpravu a odstraňování sektorů. Výstupem této práce je aplikace, popřípadě program. Pokud se bude v rámci jednoho zpracování řešit více menších problémů, bude program

časově náročnější. To vyžaduje větší výkon počítače a větší kapacitu pevného disku, protože se musí neustále ukládat data a informace do pomocných proměnných. Na základě těchto skutečností bylo rozhodnuto, že se bude řešit hlavně spojování ploch a bariéry budou předem dané bez znalosti výšek v terénu, tj. mít k dispozici liniovou vrstvu.

V následujícím textu bude představen projekt PÁTRAC, kdy tato práce vzniká jako jedna jeho součást. Budou zmíněny Search and Rescue operace a dále budou popsány existující metody, které nějakým způsobem řeší spojování ploch. Nedílnou součástí této práce jsou vstupní data, která se nejdříve musí připravit, než se s nimi začne dál pracovat. Nejdříve se budou zkoušet konvenční prostředky pro spojování, popř. rozdělování ploch, pak nad daty bude spuštěna aplikace a výsledky budou zhodnoceny a srovnány.

## 2 PÁTRAČ

Tato práce vznikla ve spolupráci s policií České republiky. Z hlediska informací o dané problematice bylo čerpáno z elektronické publikace [12].

Pod projektem *Využití vyspělých technologií a čichových schopností psů pro zvýšení efektivity vyhledávání pohřešovaných osob v terénu*, zkráceně pouze *PÁTRAČ*, vznikají čtyři samostatné aplikace:

- zásuvný modul pro přípravu a řízení operace pro *QGIS*,
- Android aplikace pro komunikaci se štábem,
- **aplikace pro segmentaci prostoru na základě přirozených bariér** (téma této práce),
- aplikace pro generování datového skladu a projektu pro *QGIS*.

Později by se měla vyvíjet aplikace *Neuronová síť pro optimalizaci určení pravděpodobnosti místa výskytu pohřešované osoby*, která by se zabývala typickými znaky chování ztracených osob.

Cílem celého projektu je:

- Zvýšit efektivitu rozhodovacích a řídicích procesů při zahájení pátracích akcí pomocí nového softwaru spolupracujícího s navigační technikou.
- Optimalizace přípravy kynologických pátracích týmů (KPT) podle vlivu způsobu práce KPT na únavu, vyčerpání a stres a tím na spolehlivost psů při pátrání.
- Využitelnost výsledků pro složky IZS (tj. Integrovaný záchranný systém) zabývající se pátráním po pohřešovaných osobách v terénu (Policie České republiky, Horská služba, Městské policie, Hasičský záchranný sbor, občanská sdružení atd.).

Projekt se rozděluje na dvě části. Na technickou a biologickou. V technické části jsou tvořeny softwarové aplikace pro řízení pátracích akcí, tj. od plánování po zpětnou analýzu pátrací akce a propojení softwaru a hardwaru v terénu.

V biologické části jsou sbírána biologická data během pátrání u psů a psovodů, jsou to např.: srdeční tep, tělesná teplota, monitoring zátěže podle způsobu vyhledávání a vliv na spolehlivost pátrání.

Výstupy jsou v rámci obou částí. V technické jím je software, poloprovoz a certifikovaná metodika a v biologické certifikovaná metodika a tři vědecké publikace. Všechny tyto výsledky jsou zpracovávány a prezentovány na seminářích, konferencích a v diplomových nebo disertačních pracích.

Harmonogram práce je navržen od 1. 1. 2017 do 31. 7. 2020 ve spolupráci s dalšími subjekty zapojených do projektu, např. všechna krajská ředitelství policie.

### 3 ZÁKLADNÍ POJMY

V první řadě je dobré vysvětlit pojmy pátrání a pohřešovaná osoba. V práci se bude také často vyskytovat pojem sektor či bariéra.

#### 3.1 Search and Rescue operace v České republice

Search and Rescue (zkratka SAR) v českém jazyce znamená služba pátrání a záchrany. Význam SAR se může měnit napříč zeměmi. V České republice je SAR operace chápána jako letecká služba pátrání a záchrany při leteckém neštěstí. Jiří Zeman ve své bakalářské práci [25] uvádí, že *SAR operace je jednorázové a časově omezené nasazení dostupných sil a prostředků směřující k vyhledání a záchraně pohřešované osoby, které hrozí nebezpečí ohrožení života nebo zdraví.*

Podle charakteru prostředí vyskytujícího se v České republice se pátrací akce podle [25] dělí na:

- sutinové vyhledávání,
- **plošné vyhledávání**,
- vyhledávání na vodních hladinách,
- vyhledávání v horském terénu a lavinách.

#### 3.2 Pátrání po pohřešovaných osobách

**Pátráním** lze rozumět činnost s cílem nalezení konkrétního objektu. Objektem je pak osoba hledaná, pohřešovaná či neznámé totožnosti. Do kategorie osob spadají i osoby usmrcené. Pátrání se netýká jenom osob, ale taky odcizených věcí jako jsou motorová vozidla, střelné zbraně apod. Jedná se o formu policejní činnosti. [15]

**Pohřešovaná osoba** je chápána jako osoba, po níž bylo vyhlášeno pátrání skrze oznámení o jejím pohřešování. Není známo, kde se osoba nachází a není podezřelá ze spáchání trestného činu. [15]

V rámci pátrání po pohřešovaných osobách se provádí několik opatření. Mezi ně patří systém organizace, operativní pátrání, administrativně evidenční opatření a další. Cílem je provést takové úkony prováděných policií, aby se dosáhlo nalezení pohřešované osoby. Pokud je pohřešovaná osoba v ohrožení života, jedná se o mimořádnou událost. [15]

**Sektor** je podle [24] část místa zásahu, kterou řídí a koordinuje velitel sektoru. Sektor dělí vytyčenou oblast určenou pro pátrání na menší plochy. Často se k dělení plochy využívají přirozené přírodní hranice, např. řeka, okraj lesa či hranice vytvořené člověkem např. cesta, plot, zakázaný vstup apod.

**Úsek** je pak místo zásahu, na kterém složky integrovaného záchranného sboru provádějí záchranné práce, které koordinuje velitel úseku. Ten je určen velitelem zásahu. [24]

**Bariérou** může být jak překážka vytvořená přírodou, tak překážka vytvořená člověkem. Jedná se o místa, která není možné překročit nebo jejich překročení je nebezpečné. Bariéra je kombinací přirozených a umělých hranic (viz kapitola 3. 3).

### **3. 2. 1 Postup přípravy pátrání**

Pátrání je velmi individuální a závisí na zdrojích, které budou využity pro samotné pátrání. Určitě se bude využívat jiný postup při pátrání v rojnicích než u pátrání pomocí vrtulníku. Nicméně i když se pátrání od sebe liší, mají podobné etapy, které je třeba provést s cílem úspěšného ukončení pátrání.

#### **Příprava pátrání a jeho vyhlášení a zahájení**

Prvním krokem je přípravná část, která bývá nejdůležitější u jakéhokoliv projektu, ať už se jedná o projektové plánování nebo přípravy v rámci pátrání. Předpokladem pro úspěšnou pátrací akci je úplná, správná a také včasná příprava. V podstatě bez nějaké přípravy by pátrací tým nebyl schopen efektivně prohledat oblast, kde se pohřešovaná osoba ztratila nebo se s největší pravděpodobností nachází.

Pro pátrání je důležité shromáždit co nejvíce údajů a informací, aby bylo možné předpokládat, kde se bude pohřešovaná osoba nacházet, jestli je ohrožena na životě a popřípadě predikovat, jak se bude dále pohybovat. Po sběru informací je možné vyhlásit a zahájit pátrání.

#### **Realizace pátrání**

V této etapě se realizuje vlastní pátrání. Jedná se o využití všech možných forem a prostředků pátrání s cílem úspěšného dokončení, tzn. nalezení pohřešované osoby, popř. objektu. Je to nepřetržitá pátrací činnost získávání, shromažďování a vyhodnocování nových informací.

Pátrání je forma osobního pátrání. Tento pojem lze vyložit i také jako každodenní cílevědomou činnost policisty, využívající pátrací pomůcky a prostředky pátrání. Dalším typem pátrání je pátrací akce, která je jednorázová a časoprostorově omezená, kdy se využívá více zdrojů k dosažení cíle.

K prostředkům pátrání patří kriminalisticko-technické prostředky, služební psi, informační systémy, kde se zařazují i geoinformační systémy, služební vrtulníky, zveřejnění v hromadných prostředcích, pátrací pomůcky apod.

#### **Ukončení pátrání**

Pátrání je ukončeno ve chvíli, kdy je pohřešovaná osoba nalezena, při jejím úmrtí nebo když dále neexistuje důvod pátrání. Příkladem může být přihlášení pohřešované osoby nebo se z pohřešované osoby stane osoba podezřelá ze spáchání trestného činu. Také k tomu může dojít při ztotožnění pohřešované osoby s mrtvolou, částí lidského těla či kosterním pozůstatkem. Pátrání se ukončuje i v případě, kdy uběhne 20 let od vyhlášení pátrání.

Podkapitola 3. 2. 1 *Postup přípravy pátrání* čerpá informace z akademické publikace [17].

### 3.3 Hranice

Hranici si lze představit jako myšlenou čáru, která vymezuje nějaký přírodní nebo společenský objekt. Nejčastěji je v praxi vedena po zemském povrchu. Pod tímto pojmem si lze představit mnoho možností a nemusí jít jenom o hranice státní. Příkladem může být břeh okolo jezera, který se označuje jako přirozená hranice. [11]

Následující odstavce jsou čerpány z článku zabývajícího se aspekty znázorňování hranic na mapách [8].

Stanovení hranice je výstupem regionalizace, což je vymezení územní jednotky či areálů splňujících nějaké kritérium. V této práci je tímto kritériem typ využití půdy (např. lesy, vodní plochy, louky, pole apod.).

V odborné literatuře se hranice definují jako linie oddělující plochy jednotlivých administrativních nebo přírodních celků a plochy různých typů využití půdy. Existuje několik způsobů, jak klasifikovat hranice a to:

- **Podle způsobu vymezení:**

Dělí se na hranice **přirozené** (orografické) nebo **umělé** (geometrické). Přirozenou hranicí může být pobřežní čára, vodní tok či horský hřeben. Umělá hranice je definovaná člověkem. Jde o hranici sledující například rovnoběžky nebo poledníky nebo také silnice, ploty apod.

- **Podle stálosti hranice:**

Existují dva typy stálosti hranice, a to **pevné** a **pohyblivé**. Jak už název napovídá, tak pevné hranice nemění svou polohu ani v závislosti na vývoji okolí. Naopak pohyblivé hranice se přizpůsobují. Jako příklad pohyblivé hranice může být hraniční tok, kde linie sleduje jeho střednici. Nicméně změna vodního toku musí být přirozená, ne umělá.

- **Podle jednoznačnosti průběhu:**

Hranice může být buď jednoznačná, tj. určitá nebo neurčitá. Určitá hranice je jednoznačně definována a sledovatelná v terénu, např. administrativní hranice. Co se týče hranice neurčité, tak její vymezení v terénu je více obtížné. Např. není jasné, kde hranice mezi lesem a loukou začíná a končí. Zda začíná první větví nebo až kmenem. Podobně tomu je i u klimatických pásů. V geoinformatice se můžeme setkat s pojmem *fuzzy border*.

### 3.4 Heuristika

Heuristika je podle [9] shromažďování pramenů a literatury pro zvolené téma. Je to soubor metod, kritérií a dalších principů, které by měly pomoci při rozhodování, kterým alternativním řešením bude co nejefektivněji dosaženo požadovaného cíle. [16]

Jako nejjednodušší heuristická metoda je *pokus a omyl*. Prvním krokem je podívat se na problém, popřípadě si jej rozkreslit. V podstatě jde o zkoušení všech možných řešení s výsledkem zjištění, zda tato jednotlivá řešení jsou správná, tzn. následuje úspěšný cíl. Také může nastat situace, že řešení není známé. V tomto případě se předpokládá, že řešení existuje a získává se jeho postup, tj. práce dozadu. Heuristika v informatice nemusí dávat přesné řešení konkrétního problému. Může nastat situace, že se sice nalezne vhodné řešení, ale to už není aplikovatelné na různé vstupy. Příkladem může být vstup polygonové vrstvy s odlišnými názvy atributů a to, i když se jedná o stejné. [10]

V práci je využit hlavně princip *pokus a omyl*, při němž další řešení byla konzultována s uživatelem aplikace. Postupně je řešen problém se vstupními daty a hledají se možnosti, jakým způsobem data připravit ke zpracování pro co nejefektivnější výsledek.



## 4 ZPŮSOBY TVORBY SEKTORŮ

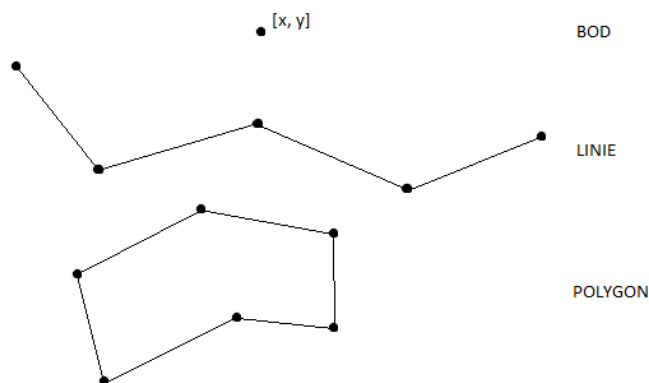
Pojem sektor byl vysvětlen v kapitole 3. 1. Neexistuje přímo problematika týkající se tvorby sektorů v rámci pátrání po pohřešovaných osobách, a proto budou vyjmenovány metody, které by bylo možné aplikovat a získat tak nějaký výsledek, popřípadě se inspirovat k získání vlastního řešení. Tvorba sektorů patří do částí přípravy pátrání.

Ještě před popisem některých metod bude vysvětleno několik pojmů, které budou v této práci používány a vysvětleno, proč tomu tak je.

**Geodata** jsou geografická nebo geoprostorová data. Geodata, která mají vztah k místu na Zemi a mají prostorovou nebo atributovou složku. Prostorová složka je vyjádřena geometrickým prvkem či databázovým záznamem. Atributová obsahuje popisné informace prostorové složky. [6]

**Geoprvek** (angl. feature) se rozumí modelový obraz lokalizovatelného objektu reálného světa, který je dále nedělitelný a je popsán geodaty. [20]

**Vektory** neboli **lineární geometrické prvky** se používají pro popis geometrických vlastností geoprvků. Vektor je z hlediska geoinformačních systémů orientovaná úsečka definovaná souřadnicemi počátečního a koncového bodu. Z vektorů jsou skládány tři základní geometrické prvky. **Bod** nulové délky, kde počáteční a koncový bod je to samé. **Linie** je otevřenou posloupností vektorů s počátečním a koncovým bodem, které se označují jako uzly a mezilehlé body jsou vrcholy. Posledním geometrickým prvkem je **polygon** reprezentovaného hranicí, která je popsána uzavřenou posloupností linií (viz Obr. 1). [20]



Obr. 1: Základní geometrické prvky.

### 4.1 Generalizace

Jednou z možností, jak tvořit sektory, je využití generalizace. Pokud se pracuje s polygony vygenerovaných z linií, je výstupem velmi členitá oblast zkoumání. Generalizaci je pak možné využít například ke spojování či redukování počtu polygonů.

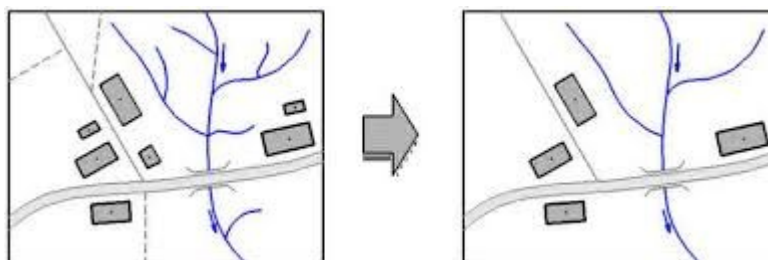
Co to vlastně generalizace je? Obecně lze říct, že je to zvolená a zjednodušená reprezentace skutečného světa, která odpovídá měřítku a účelu mapy. Mapa, z níž jsou informace velmi snadno čitelné, je určitě lepší než mapa, kde je příliš velké množství informací a uživateli trvá déle, než se v ní zorientuje. Generalizaci také jde popsat jako proces vytváření mapy s menší úrovní detailu se zachováním základní geografické informace. [23]

Důvodů, proč používat generalizaci, je několik. Mezi ně patří například redukce objemu dat, změna měřítka mapy, změna účelu mapy nebo zlepšení grafické stránky mapy. [4]

#### 4. 1. 1 Metody generalizace

Pod generalizací je ukryto několik metod. Nejedná se jenom o metody grafické, ale také o textové, kdy se generalizují textové popisky a doplňky nebo o generalizaci atributové složky.

Metodou **výběr** se vybírají prvky, které mají být vizuálně potlačeny, aby tvořila jakési pozadí mapy, nebo jsou úplně vypuštěny. Naopak takové prvky, které jsou důležité pro daný účel mapy, jsou naopak zvýrazněny (viz Obr. 2).

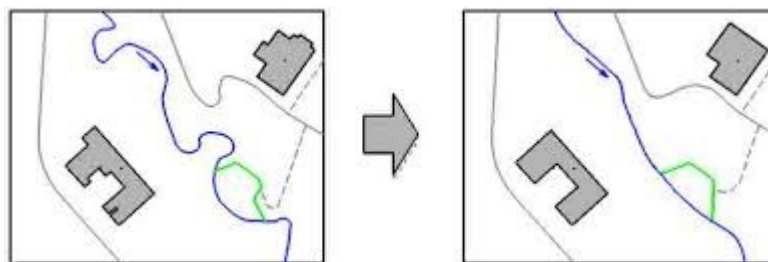


Obr. 2: Metoda výběru prvků.

Výběr se dělí na cenzální, kdy jsou vybrány prvky na základě splnění podmínky (např. velikost, rozměry, význam apod.) a na normativní, kdy se vypočítá maximální počet prvků v mapě.

Dalším typem je **změna klasifikace**. Ta se většinou používá při převodu z jednoho datového modelu na jiný. Jde o rozdělování nebo slučování tříd.

**Geometrická generalizace** upravuje tvar linií buď samotných linií nebo hranic plošných prvků. Patří zde zjednodušení, zlepšení, vyhlazení, posun nebo pootočení. V případě zjednodušení se snižuje detailnost prvků, což je vhodné například pro změny měřítek nebo vrstevnic (viz Obr. 3).



Obr. 3: Zjednodušení linií a polygonů.

V případě maximální generalizace se polygon nahradí obdélníkem ve směru nejdelší hrany polygonu. Dále existuje metoda vyhlazování, vylepšení, posunutí, pootočení apod. Této práci se nejvíce dotýkají operace s plochami, mezi které se řadí sjednocení, zrušení nebo rozdělení ploch.

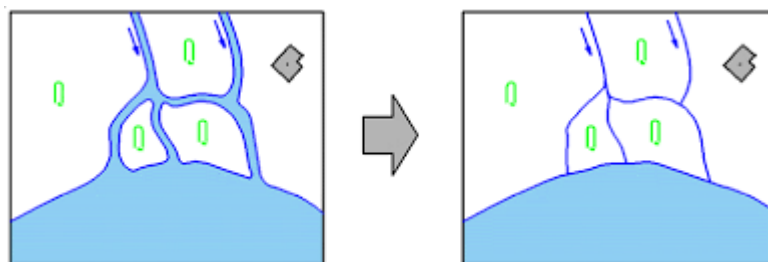
Tato kapitola 4. 1. 1 čerpala z literatury [4], která se zabývá popisem různých metod generalizace.

#### 4. 1. 2 Agregace

Slučování více ploch do jedné. Často se jedná o malé nebo izolované plochy či linie. Pro danou menší plochu se vybírá sousední plocha, která splňuje podmínky sloučení. Například je stejného typu, maximální velikost nepřekročí velikost apod. Sloučením se vypustí hranice mezi oběma plochami.

#### 4. 1. 3 Prostorová redukce

Jednoduše řečeno, prostorová redukce je změna dimenze geometrického prvku. Plocha se může převést na linii nebo na bod, linie na bod a bod na plochu (viz Obr. 4).



Obr. 4: Prostorová redukce vodního toku.

### 4. 2 Metody řešící podobnou problematiku

Následující metody neřeší přímo problematiku dělení prostoru do segmentů, ale podobným postupem nebo jejich úpravou by se také dalo dosáhnout cíle.

#### 4. 2. 1 Boundary-based algoritmy

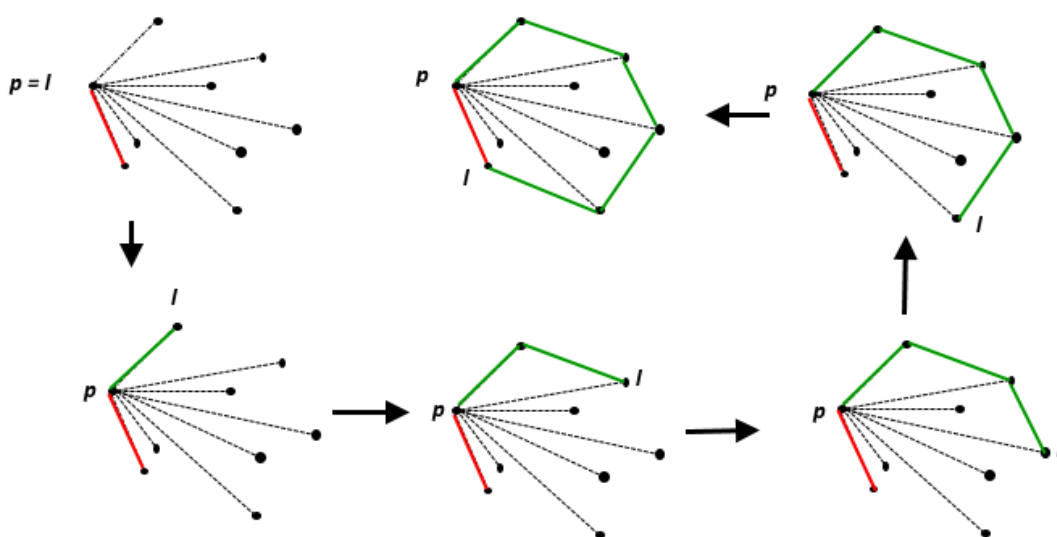
Tyto algoritmy pracují s hranicí polygonu a jedná se o metodu agregace, kdy se z více polygonů stává jeden s využitím právě daného algoritmu. Dále se dají metody pro agregaci

rozdělit na skupinu pracující s posunutím prvků a druhá s vyplněním místa mezi objekty. Do první se řadí například algoritmus, který využívá konvexní obálku a k druhé se řadí algoritmy pracující s morfologickými operátory. [22]

Nejjednodušším způsobem je obalit objekty konvexní obálkou a získat tak agregovaný prvek. Mezi takové algoritmy se řadí například *Gift Wrapping* nebo *Quick Hull* algoritmus. Princip algoritmů bude vysvětlen na množině bodů se souřadnicemi  $x$  a  $y$ .

*Gift Wrapping* algoritmus (také *Jarvis's March*) funguje tak, že začíná od bodu umístěného nejvíce nalevo a následně obaluje další body v protisměru hodinových ručiček. Další bod je nalezen tak, že jeho úhel v protisměru hodinových ručiček je největší ze všech bodů. [2]

Nevýhodou *Jarvisova* algoritmu je čas potřebný pro výpočet. V nejhorším možném případě je délka skenování  $O(n^2)$ . S využitím *Grahamova* skenovacího algoritmu je snížení času na  $O(n \log n)$ . Na rozdíl od *Gift Wrapping* algoritmu jako první hledá bod s nejnižší  $y$ -souřadnicí. Pokud nalezne více než jeden, vybere ten, který má nejnižší  $x$ -souřadnici. Dále hledá všechny body, které mají stejný úhel a vynechá všechny kromě nejvzdálenějšího bodu (viz Obr. 5). [3]



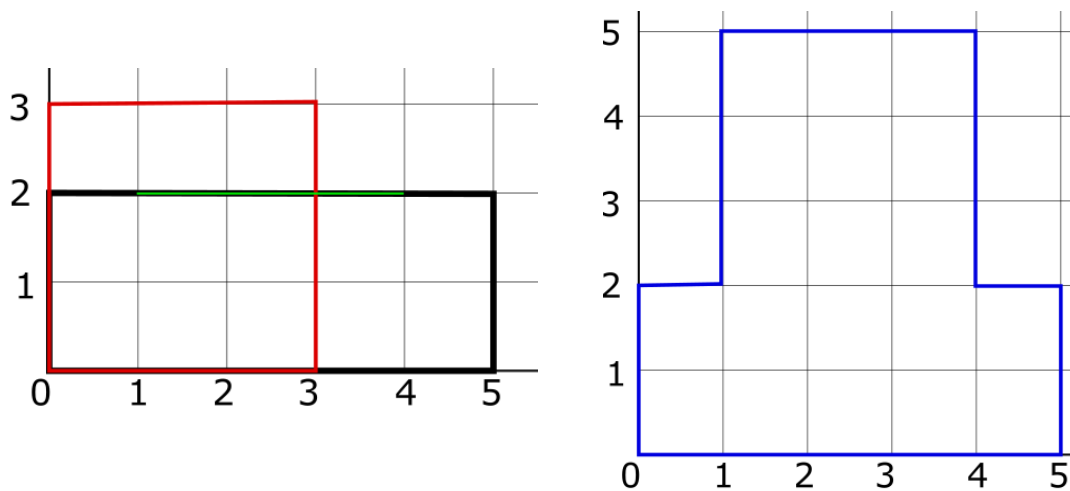
Obr. 5: Ukázka Gift Wrapping algoritmu. [2]

Dalším zmíněným algoritmem je *Quick Hull* algoritmus založený na vzestupném seřazení x-souřadnic. Podle [19] funguje následovně:

1. Najdi bod s nejmenší x-souřadnicí a označ ji jako  $min\_x$ , najdi bod s největší x-souřadnicí a označ ji jako  $max\_x$ .
2. Spoj tyto body linií  $L$ . Vzniknou dvě části. Tyto části procházej jednu po druhé.
3. Pro část najdi bod  $P$  s největší vzdáleností od linie  $L$ . Vytvoří se trojúhelník  $P, min\_x, max\_x$ . U bodů uvnitř trojúhelníka nemůže nastat, že by byly součástí konvexní obálky.
4. Nyní spoj body  $P$  s  $min\_x$  a  $P$  s  $max\_x$ . Body ležící mimo trojúhelník jsou vstupní množinou bodů. Opakuj bod 3 dokud nezůstane bod s linií. Přidej koncové body tohoto bodu do konvexní obálky.

#### 4. 2. 2 Rostoucí náhodné polygony

Podle literatury [21] nejjednodušší možností, jak vytvořit náhodný polygon, je vygenerovat seznam souřadnic představující daný polygon, např. obdélník  $[(0, 0), (5, 0), (5, 2), (0, 2)]$ . Pokud by byla potřeba vytvořit něco složitějšího, může se zvolit připojení dalšího obdélníka k již stávajícímu. Příkladem může být obdélník o souřadnicích  $[(0, 0), (3, 0), (3, 3), (0, 3)]$ , který se bude připojovat k segmentu  $[(4, 2), (1, 2)]$  na hraně  $[(5, 2), (0, 2)]$  (viz Obr. 6).



Obr. 6: Vlevo: původní polygon (černý), připojovaný polygon (červený), segment (zelený); vpravo: výsledný polygon po spojení (modrý).

Tento proces se dá popsat jako (i) *výběr* vhodného polygonu pro spojení, (ii) *lokalizace* hrany původního polygonu, ke které se bude připojovat vhodný polygon a (iii) *připojení* vhodného polygonu, a to jeho transformací, tj. posunutí, otočení nebo změna měřítka. Také se (iv) *kontroluje*, zda nedošlo k průniku vlastních hran. Pokud tato možnost nastane, opakují se kroky (i) – (iii). Tato metoda je nazvána jako *Pick-Locate-Attach (PLA)*. [21]

### 4.3 Shrnutí metod

Existuje několik postupů a metod umožňující spojování polygonů. Určitě největší část zabírají právě metody generalizace, které buď vyplňují prázdný prostor mezi polygony nebo používají další operátory pro spojení. V této práci se pracuje převážně s polygony, které mají vždy společnou hranici s jiným polygonem a nebude se nad jejich původním tvarem provádět žádné zjednodušení či vyhlazení hrany. V tomto případě se použije kombinace agregace a metody rostoucích polygonů, kdy není cílem najít nejvhodnější umístění, velikost či orientaci polygonu, nýbrž je spojovat tak, že daný vstupní polygon bude růst připojením všech vhodných polygonů splňující vstupní podmínky.

## 5 HODNOCENÍ TVARU

V závěru práce bude zhodnocen celý výstup na základě několika parametrů, které by měly uživateli poskytnout představu, jak kvalitní výsledek získal. Zde je dobré uvést několik parametrů, které budou využity, popř. budou popsány další, které by se mohly použít také, ale z důvodu složitosti implementace na ně nepřišla řada.

### 5.1 Tvarová analýza dat

Následující kapitola čerpá informace z literatury [14].

Pro popis objektu na základě jeho vlastností (př. plocha, obvod, kompaktnost apod.) se používají tzv. deskriptory. Existují také Fourierovy deskriptory a jiné, které umožní popsat tvar v obrazových datech.

#### 5.1.1 Geometrické deskriptory

**Plocha** neboli **obsah** je jedním z geometrických deskriptorů. Lze jej získat jako součet všech pixelů v objektu o velikosti 1 a slouží pro odhadnutí velikosti objektu. Nesmí se zapomínat ani na **obvod**, který taktéž může sloužit k určení velikosti objektu. Obvodem se rozumí linie tvořící hranici objektu.

**Kompaktnost** je geometrický deskriptor určující kompaktnost tvaru. Jeho vztah je definován jako  $C = \frac{O^2}{4\pi S}$ , kde  $O$  je obvod a  $S$  obsah objektu. Pokud se  $C = 1$ , pak objekt má nejkompaktnější tvar, tzn. tvarem je kruh. Čím je objekt méně kompaktní, tak hodnota  $C$  stoupá.

**Kulatost** je definována jako vzdálenost mezi centroidem a body ležících na obvodu objektu. Vzdálenosti jsou počítány v úhlech:

$$Kulatost = \sum_{\theta=0}^{\theta=359} \frac{|R_{\theta} - R_p|}{R_p} \quad (1)$$

kde  $\theta$  je konkrétní úhel v daném směru,  $R_p$  je průměrná vzdálenost k okrajovým bodům od centroidu ve všech směrech a  $R_{\theta}$  je konkrétní vzdálenost. Pokud je výsledkem 0, mluvíme o kulatém tvaru. Neplést si kulatost s plošnou zakulaceností.

#### 5.1.2 Topologické prvky

V **analýze momentu** se nejčastěji počítají tři momenty s nejnižším řádem. Obsahu objektu odpovídá 0. řád. Centroid je dán I. řádem a rovná se těžišti objektu. Hlavní osy dané momentem II. řádu reprezentují orientaci nekulatého tvaru.

Kylián v literatuře [14] říká: *Informace získané z takové analýzy je možné využít k vyhodnocení centrálních momentů, normalizovaných centrálních momentů a momentových invariant.*

Výstupem jsou tvarové příznaky nezávislé na poloze objektu, velikosti či orientaci.

**Geometrické momenty** se nejčastěji používají k tvarové analýze a jsou definovány jako součin souřadnic pixelů  $x^p y^p$ .

$$m_{pq} = \sum_{y=1}^M \sum_{x=1}^N x^p y^q f(x, y) \quad (2)$$

kde  $p, q = 0, 1, 2 \dots$  a  $M \times N$  představují velikost obrazu.

**Deskriptor**, který používá **geometrický moment**, pracuje s momentem nejnižšího řádu  $m_{00}$ , reprezentující plochu objektu. I. řád momentu je normalizován podle obsahu a jeho výsledkem je přibližné umístění objektu v obraze. Centroidy se pak počítají podle následující rovnice:

$$x_c = \frac{m_{10}}{m_{00}} \quad y_c = \frac{m_{01}}{m_{00}} \quad (3)$$

kde  $m_{xy}$  je moment řádu  $xy$ .



## 6 PODMÍNKY PRO VÝSTUP DAT

Celý proces zpracování vstupních dat by se měl řídit několika podmínkami a na výstupu by měly být odpovídající sektory. Než přijde řada na konvenční metody a vlastní řešení, bude zde popsáno, co je od výstupu požadováno a jakým způsobem se budou zadávat parametry.

Jednou z hlavních podmínek je **maximální a minimální plocha** výsledného sektoru. Nejvíce se pracuje s maximální plochou, protože je jednodušší polygony spojovat, než je nějakým způsobem dělit na menší části. Může však nastat případ, kdy se na výstupu objeví polygony s větší plochou, protože jejich původní velikost byla větší než maximální práh.

S minimální plochou se pracuje až v závěrečném kroku, kdy se zpětně procházejí vytvořené sektory a hledají se takové, které mají menší obsah (viz kapitola 9. 2. 5). Nicméně při rozhovoru s policejním operátorem bylo řečeno, že už 20 ha je hodně, takže je otázkou, jestli je parametr minimální plochy třeba užívat pro definici spodní hranice nebo jenom pro odstranění miniaturních polygonů, které se uložily do výsledku z důvodu použité logiky.

Nastavení parametrů plochy není přesně definováno a je na uživateli, jak si čísla zvolí. Pokud je pro něj vhodná rozloha kolem deseti hektarů, pak není důvod, proč takový sektor negenerovat. Pro někoho je to malá plocha, pro někoho velká.

Je také **omezeno spojování různých typů využití půdy**. V tomto kroku se ověřuje, zda dva polygony určené pro sloučení jsou stejného typu. Jelikož oblastí zájmu zadavatele je oblast mimo zastavěná území a zároveň se tato práce týká plošného hledání. Nejvíce dominantním typem jsou louky a pastviny se stromy (*LPSTROM*) s 7 270 prvky ze vzorku 38 800, tj. cca 19 %. Dalším dominantními typy je travnatý povrch (*TRTRPO*) s 4 946 (cca 13 %) prvky a orná půda (*ORNAPU*) s 4 221 (cca 11 %) prvky. Je však důležité podotknout, že travnatá plocha se řadí do zastavěného území a do výpočtu přímo nevstupuje. Nejvíce četným typem, který zde není zařazen z důvodu jeho pozdější eliminace, jsou budovy (*BUBLBU*) zaujímající cca 27 % (tj. 10 614) z celkového počtu prvků.

Jednodušším přístupem k této problematice je nastavit jako výchozí omezení spojování polygonů různých typů využití půdy a vytvořit seznam výjimek, který si tvoří sám uživatel, a říct, které typy se mohou sloučit.

```

1 LPSTROM;Louky, pastvina se stromy
2 ORNAPU;Orná půda
3 TRTRPO;Travnatý povrch
4 SADZAH;Sad, zahrada
5 LPKROV;Louka, pastvina s křovinami
6 ZAHPAR;Zahrady, parky
7 BAZMOC;Bažiny, mokřady
8 OSPLSI;Nějaké plochy
9 VODPLO;Vodní plochy
10 VODZEM;Vodojem zemní

```

Obr. 7: Příklad výstupu typů využití půdy vyskytujících se v oblasti zájmu.

Přístup vyřazení **intravilánu** z výpočtu a vůbec jej nevkládat do výstupního souboru *Shapefile ESRI* by způsobil, že by se ve výsledné vrstvě vyskytovaly díry, což je místo, kde není žádný polygon. Proto byl všechny intravilán, na základě rozhodnutí, sloučen dohromady a vložen do výstupu jako jeden jediný polygon. Pro stanovení, co je intravilán, musí uživatel vytvořit seznam typů. Pro zjednodušení práce nabízí aplikace možnost vygenerovat všechny typy využití půdy v dané oblasti ve tvaru například LPSTROM;Louky, pastvina se stromy (viz Obr. 7). Výstupem je textový soubor, ze kterého uživatel vymaže všechny řádky typů, které nejsou intravilánem.

Poslední a neméně důležitou podmínkou je **omezení spojování přes bariéru**. Bariéry představuje liniová vrstva, která vznikla sloučením různých typů liniových vrstev představující jednotlivé typy bariér. V předchozích kapitolách bylo řečeno, že bariérou je taková překážka, kterou nelze překročit nebo je její překročení nebezpečné. Dále to mohou být náhle změny výšky v terénu, příkopy, prudké svahy, strže apod. Záleží na tom, jaká data má uživatel k dispozici, a která se rozhodne využít. V rámci práce se neřeší, jak bariéry vytvořit.



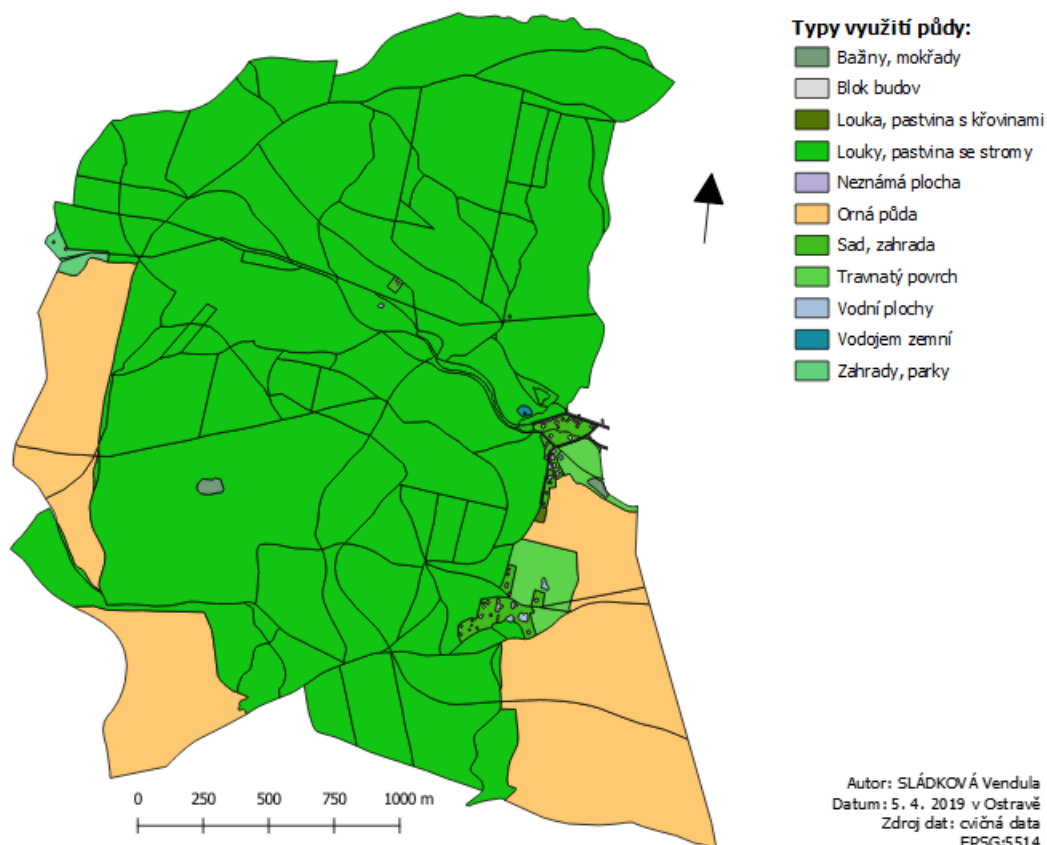
Obr. 8: Příklad nového sektoru ohraničeného bariérami ze všech stran. (Žlutá barva označuje vybraný prvek pro lepší přehlednost.)

Na vstupu je potřeba mít liniovou vrstvu, která vznikla sjednocením několika jiných liniových vrstev. Není potřeba znát jakého typu jsou, ale pomůže to při vizualizaci a prezentaci prostorových dat. Je otázkou, co se sektory, které nejsou během výpočtu sloučeny s jiným polygonem, protože jsou ohraničeny bariérami (viz Obr. 8). Rozhodnutí zůstane na operátorovi, jehož úkolem je rozhodnout, které sektory budou použity pro samotné plánování a vymezení oblasti pro pátrání po pohřešované osobě. Pokud zná místní terén dobře, může usoudit, že například řeka omezující spojení je překročitelná a ručně provede sjednocení sektoru se sousedním sektorem, i když mezi nimi vede bariéra.

## 7 VSTUPNÍ DATA

Pro testování jsou k dispozici **cvičná data**. Vrstva s co nejpodrobnějšími polygony byla vygenerována spojením liniových vrstev a následně se pro každý polygon určil jeho typ využití půdy. Původní cvičná data obsahují 38 800 prvků a z důvodu opakovaného spouštění aplikace pro tvorbu sektorů a ušetření času se použila jen malá část, která se bude nazývat **cvičná oblast** (viz Obr. 9).

### Typy využití půd ve cvičné oblasti



Obr. 9: Mapa typů využití půd vyskytujících se ve cvičné oblasti určené pro zpracování.

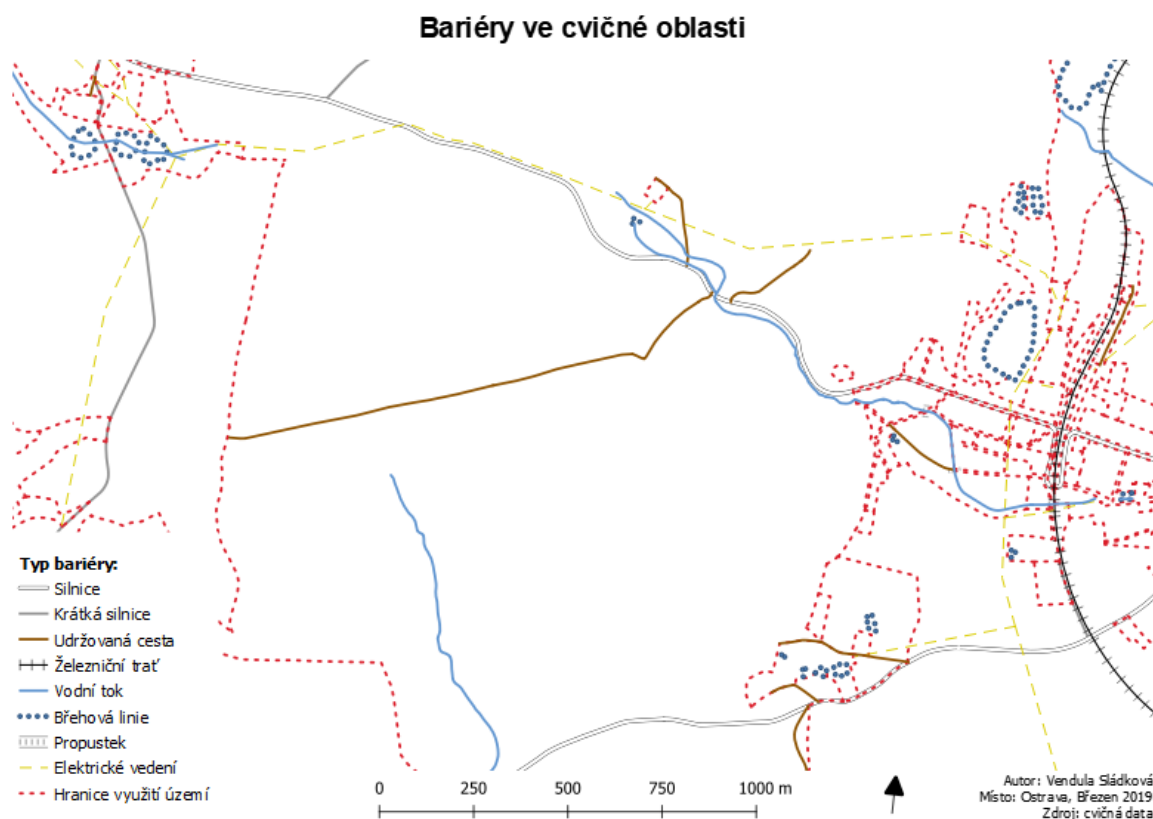
### Základní typy využití půdy (též typy ploch):

- plocha u nádraží,
- průmyslové plochy,
- bažiny, mokřady,
- bloky budov,
- chmelnice,
- elektrárna,
- halda, odval,
- hřbitov,
- kolejiště,
- letiště,
- louky, pastviny s křovinami,
- louky, pastviny se stromy,

- odpočívadla,
- orná půda,
- povrchový důl,
- rašeliniště,
- rozvalina, zřícenina,
- sad, zahrada,
- sesuv půdy,
- silo,
- stromová školka,
- skládka,
- travnatý povrch,
- usazovací nádrž,
- budovy na nádraží,
- vinice,
- vodní plochy,
- zahrady, parky.

Z těchto dat se také generovaly bariéry, které budou využity pro zabránění spojení ploch stejného typu. Vstupními daty jsou (viz Obr. 10):











- silnice,
- krátká silnice,
- silnice,
- udržovaná cesta,
- železniční trať,
- vodní tok,
- břehová linie,
- propustek,
- elektrické vedení,
- hranice využití území.



Obr. 10: Mapa bariér vyskytujících se ve cvičné oblasti.

## 7.1 Barevné schéma

Aby bylo možné od sebe dané plochy rozlišit, byla každé ploše přiřazena barva korespondující s jejím typem. Např. vodní plochy mají modré odstíny, travnatý povrch je zelený. Avšak kategorií je tolik, že bylo těžké některé dostatečně odlišit. Například plochy různých typů budov mají šedivé odstíny. Tato legenda je používána po celou dobu práce s cvičnými daty (viz Obr. 11).

	AZAMEK	Areál zámku		ORNAPU	Orná půda
	BAZMOC	Bažiny, mokřady		OSTPRU	Ostatní, nerozlišený průmysl
	BUBLBU	Blok budov		AREZAS	Plocha u nádraží
	VANAZA	Budovy na nádraží		POTPRU	Potravinářský průmysl
	CAMPIN	Camping		POTELO	Povrchový důl
	CERPST	Čerpací stanice pohonných hmot		STAVHM	Průmysl skla, keramiky a staveb. hmot
	CISTVO	Čistírna odpadních vod		REKZAS	Rekreační zástavba
	ZDSOZA	Další zdravotní a sociální zařízení		ROZZRI	Rozvalina, zřícenina
	DREVPR	Dřevozpracující a papírenský průmysl		SADZAH	Sad, zahrada
	ELEKTR	Elektrárna		SKLHAN	Sklad, hangár
	GOLFAR	Golfový areál		SKLADK	Skládka
	HRBITO	Hřbitov		SKLENI	Skleníkové pěstování plodin
	HRISTE	Hřiště		SKUPGA	Skupinové garáže
	CHATKO	Chatová kolonie		SPORTA	Sportovní areál
	CHMELN	Chmelnice		STRPRU	Strojírenský průmysl
	CHOVZV	Chov hospodářských zvířat		STRELN	Střelnice
	KLASTE	Klášter		SKOLAA	Škola
	KOLEJI	Kolejiště		TECHSL	Technické služby
	KOUPAL	Koupaliště		TRTRPO	Travnatý povrch
	LETSCE	Letní scéna		UPRVOD	Úpravna vody
	LPKROV	Louka, pastvina s křovinami		USNAOD	Usazovací nádrž
	LPSTROM	Louky, pastvina se stromy		VEZENI	Věznice
	MUZEUM	Muzeum		VODPLO	Vodní plochy
	ROZTRA	Neidentifikovaný povrch		VODZEM	Vodojem zemní
	OSPLSI	Nějaké plochy		ZAHPAR	Zahrady, parky
	ODPOCI	Odpočívadlo		ZEMARE	Zemědělský areál ostatní

Obr. 11: Legenda typů ploch ve cvičných datech.

## 7.2 Příprava vstupních dat

V kapitole 6 byly popsány podmínky, kterými by se tvorba sektorů měla řídit. Také bylo uvedeno, že ve cvičných datech je 38 800 prvků, které se musí nějakým způsobem zpracovat na ploše o cca 62 000 ha. Kolik prvků může být na rozloze celé České republiky, která má cca 7,9 miliónů hektarů? Se zvyšujícím se počtem vstupních dat se zvyšuje i délka zpracování.

Dále uživatel musí zjistit, jestli jeho data vůbec obsahují rozdělení typů využití půd. Program pracuje tak, že mu stačí zadat název atributu, v němž se typy nacházejí a dále si to zpracuje sám.

Vyskytl se problém s poškozenými daty. Hlavním důvodem je vygenerování co nejmenších polygonů z liniové vsrtvy (použita metoda *Feature to Polygon* v *ArcGIS Desktop 10.6*). Mezi tyto problémy patří například výskyt děr mezi polygony, překrývání se a taky špatně vygenerované souřadnice, i když dva body ležely na sobě, tak nebyly brány jako stejné kvůli jinému číslu na cca šestém až devátém desetinném místě.

### 7.2.1 Eliminace nepotřebných dat

Jak bylo předtím řečeno, tak s rostoucím počtem dat roste i čas zpracování. Jednou možností, jak zrychlit výpočet, je eliminovat veškerá data, která nejsou pro výstup potřebná. Eliminací se rozumí sloučení polygonu nepotřebného typu s polygonem zájmového typu, v němž leží, popřípadě sloučení dvou polygonů stejného typu vnořených v sobě. Tento případ se hodně vyskytoval u školních pozemků, průmyslových a zemědělských areálů apod. (viz Obr. 12).



Obr. 12: Mapový výstřižek městské oblasti.



Jelikož se jedná o průzkum v rámci mimo městských oblastí, tak pro práci není potřeba mít budovy či jiné objekty, které zbytečně dělí prostor na menší nepoužitelné části. Mezi budovy patří vrstvy s budovami, budovami na nádraží, bloky budov, chladicí věže, elektrárna a skleníky. Nicméně elektrárny se nechají zachovány, protože by kolem nich měla být ochranná bariéra (viz Obr. 13).



Obr. 13: Příklad eliminace přebytečných polygonů ve zkoumané oblasti. Nahoře – původní data, dole – data po eliminaci.

Původní počet prvků, tj. 38 800, se snížil na 22 428 prvků. V rámci cvičné oblasti se počet snížil na 141 ze 199 prvků.

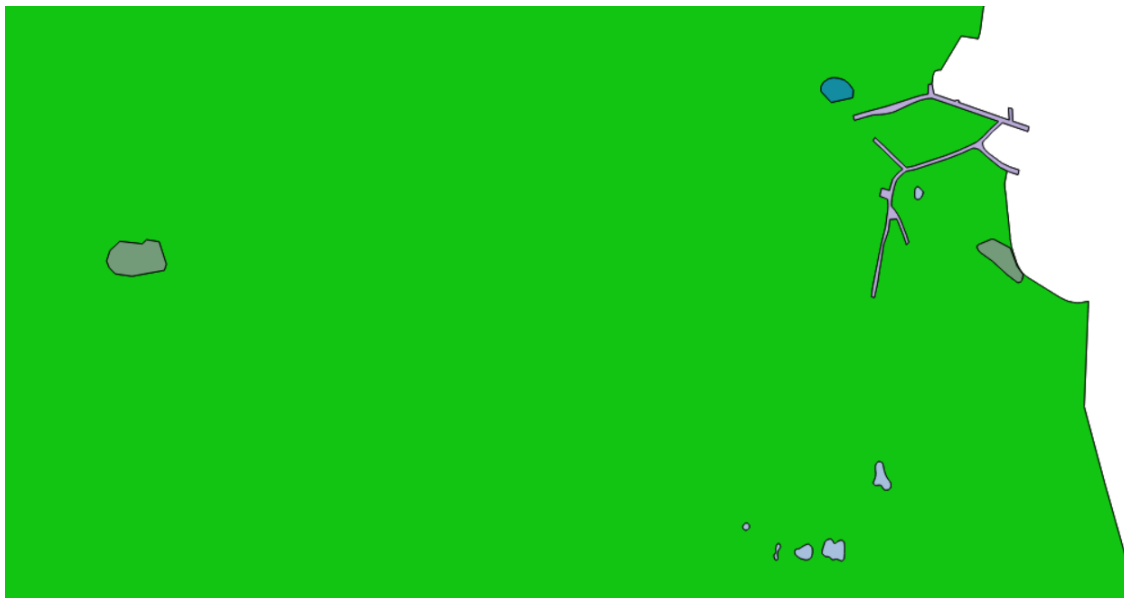
### 7. 2. 2 Generalizace souřadnic

Další problém, který je spojen s daty, je počet desetinných míst u souřadnic. V případě, že nad sebou leží dva body  $A$  a  $B$ . I po velkém přiblížení, kdy měřítko je např. 1:1, není stále patrné, že by body ležely vedle sebe. Tyto body mají souřadnice například:

- pro  $A$ : 123,123456789
- pro  $B$ : 123,123456799

Na první pohled lze říct, že jsou totožné, ale když se podíváme pozorněji, na osmém desetinném místě je přece jenom rozdíl. U bodu  $A$  tam je číslo 8, ale u  $B$  je číslo 9. Zde nastává problém.

Příklad je zobrazen na Obr. 14. Nad celou zkoumanou oblastí se spustila funkce pro sloučení všech polygonů do jednoho. Jelikož se souřadnice některých polygonů lišily na šestém až devátém desetinném místě, polygony nebyly sloučeny a místo jednoho výsledného polygonu jich zůstalo několik.



Obr. 14: Problém spojování oblastí.

Proto se nad vstupní vrstvou spustila funkce pro snížení přesnosti, která sníží počet desetinných míst na čtyři. Využila se funkce z balíčku *GDAL* a spouštěla se přes příkazový řádek. Stačí se pouze dostat do složky, kde se nachází požadovaná vrstva a následně v příkazovém řádku napsat:

```
ogr2ogr output.shp input.shp -simplify 0.0001
```

Soubor *output.shp* je název výsledku, který bude zpracován funkcí *ogr2ogr* a *input.shp* je vstupní soubor. Funkce zachovává nastavený souřadnicový systém a dále soubor typu *.dbf* a *.shx*.

### 7. 2. 3 Odstranění děr

Bylo vyřešeno několik problémů. Odstranily se přebytečné polygony, které jsou pro zpracování zbytečné, a tak se urychlil výpočet a dále se provedla generalizace souřadnic, aby bylo možné spojit zkoumanou oblast do jedné jediné. Pro tento test je využita funkce aplikace QGIS *Dissolve*.

Po sjednocení oblastí do jedné, zůstanou ve výsledném polygonu „rýhy“. Po nahlédnutí do atributové tabulky je vidět, že je zobrazen pouze jeden polygon, z čehož lze usoudit, že funkce zpracovala vstup dobře.

Pro snížení tohoto problému je využita funkce aplikace *GRASS*, která je součástí *QGIS*. Jedná se o funkci *v.clean*, která vyčistí topologii vstupní vektorové mapy. Tuto funkci lze najít: *Zpracování -> Sada nástrojů -> GRASS -> Vektor (v. \*) -> v.clean*.

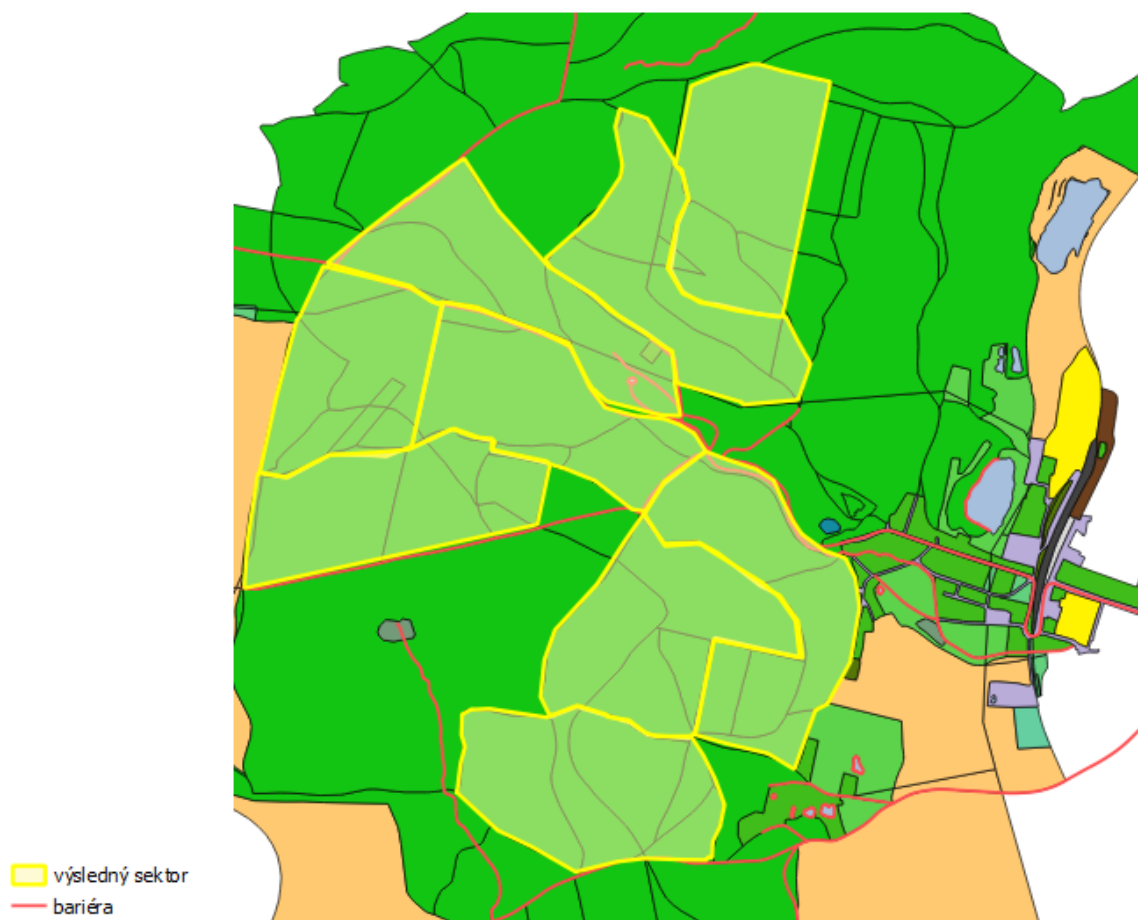


## 8 VYUŽITÍ KONVENČNÍCH PROSTŘEDKŮ

Možnost použít tradiční nástroje místo složitého programování je určitě na místě. Pokud by existovala metoda, která splňuje všechny podmínky zadavatele, nemuselo by se řešit žádné vlastní řešení a práce by dále pokračovala popisem, jak takovou metodu správně používat, testovala by se na různé množství dat a zjišťovaly by se její limity.

Na Obr. 15 je vidět výstup vytvořeného pomocí aplikace *OziExplorer* a zobrazeného pomocí *QGIS* 3.4.2, kde uživatel ručně tvořil jednotlivé sektory. Obrázek se nemusí přibližovat a jde dobře vidět, že hranice sektorů přesně neodpovídají hranicím polygonům vstupní vrstvy. Sektory se také překrývají a vyskytují se mezi nimi zvláštní tvary. To je důležité vědět, protože ne vždycky je v zájmu mít sektor v kompaktním tvaru. S příkladem výsledku manuálního zpracování je možné provést pozdější srovnání tvarů sektorů.

Pro možnost srovnání výstupů se bude pracovat s maximální plochou 50 ha a minimální plochou 20 ha. Dále bude hodnoceno, zda metoda dokáže do výpočtu zahrnout bariéru a typ využití půdy.



Obr. 15: Ukázka výstupu z manuální tvorby sektorů.

## 8.1 ArcGIS Desktop

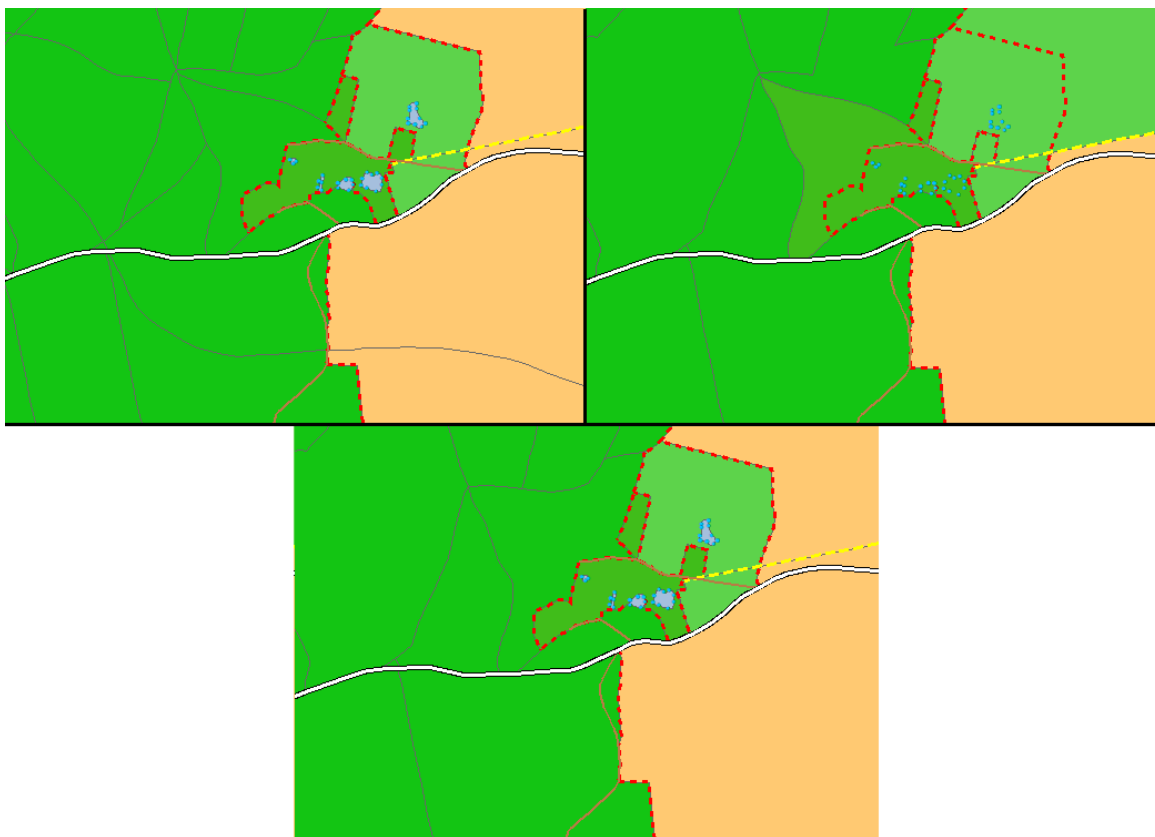
*ArcGIS Desktop* je geografický informační systém. Umožňuje práci s vektorovými, rastrovými či textovými formáty. Dále umožňuje připojení a načítání dat z geodatabáze, připojení mapových služeb nebo využívání dat publikovaných na internetu prostřednictvím služby *ArcGIS* serveru, který poskytuje možnosti pro zpracování a vizualizaci. [5]

V následujícím textu jsou popsány metody generalizace, konkrétně agregace, které vytvářejí nové polygony spojením více polygonů dohromady. Metody se nacházejí v *ArcToolbox* konkrétně v *Data Management Tools*, kde je složka *Generalization*. Názvy metod budou uváděny v anglickém jazyce, protože se pracuje s anglickou verzí softwaru.

V tomto případě nebude výstup nazýván jako sektor, ale pouze jako polygon.

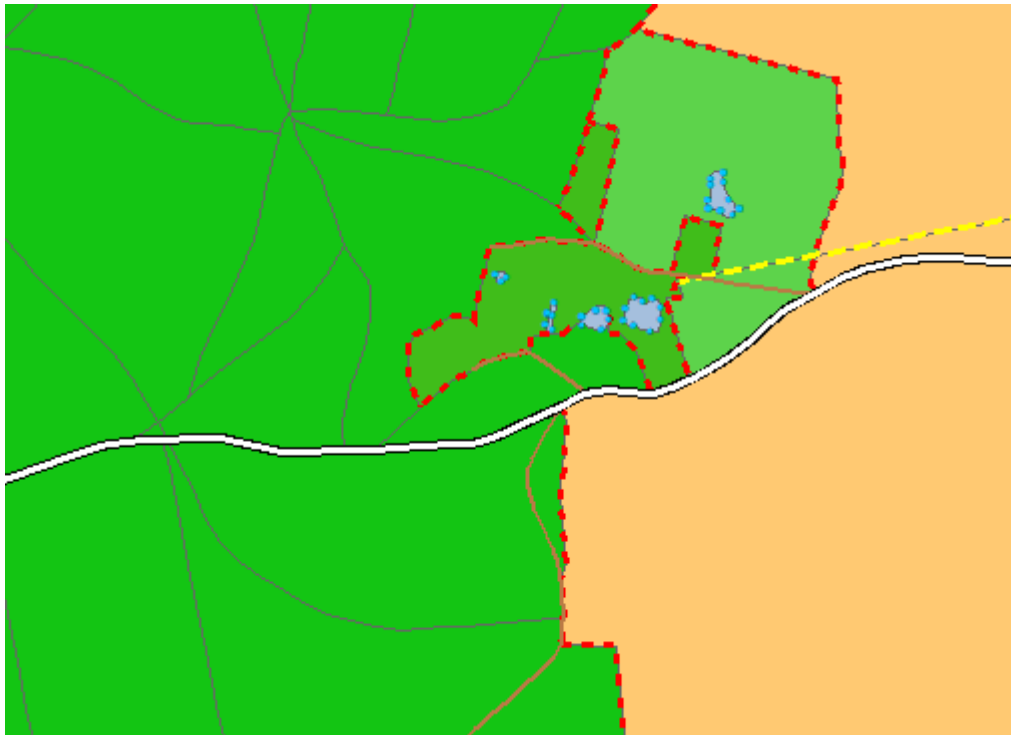
### 8.1.1 Eliminate

Aby bylo možné funkci *Eliminate* použít, musí se nejdříve vybrat oblast pro zpracování. Vstupem je polygonová vrstva cvičné oblasti. Funkce *Eliminate* také nabízí možnost připsání podmínky, přidání liniové či polygonové vrstvy, v tomto případě bariéry. Jak je vidět na Obr. 16, metoda nerozlišuje typy, a tudíž podmínky nespojování různých typů není splněna.



Obr. 16: Vlevo nahoře – původní polygonová vrstva, vpravo nahoře – upravená vrstva bez využití bariér, dole – upravená vrstva s využitím bariér.

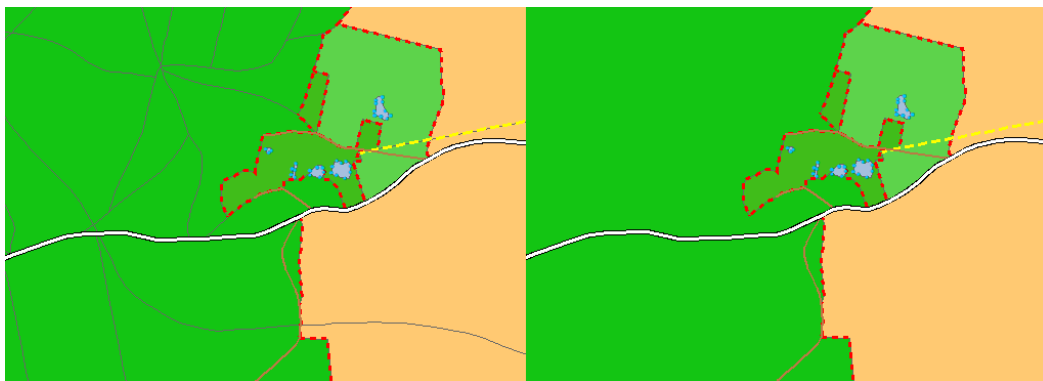
Při vyplnění pole *Exclusion Expression* výrazem „TYP“ = ‘LPSTROM‘ by se měl z výpočtu vyloučit tento typ plochy. Výstupem je spojení ostatních ploch a na obrázku dole se spojila pouze orná půda. Při použití výrazu „TYP“ = ‘ORNAPU‘ nedošlo ke spojení žádného polygonu.



Obr. 17: Vyloučení lesů z výpočtu.

### 8. 1. 2 Dissolve

Funkce *Dissolve* sjednocuje polygony na základě hodnot vybraného atributu. Při výběru atributu *TYP* se sloučí všechny polygony bez ohledu na podmínce o maximální velikosti plochy a omezení spojování přes bariéru. V tomto případě vzniklo celkem 10 polygonů, které jsou spojené napříč cvičnou oblastí. Při zrušení možnosti *Create multipart features* se vytvoří nové polygony tak, že pokud kolem sebe nemají žádného souseda stejného typu, je výstupem nový polygon (viz Obr. 18).

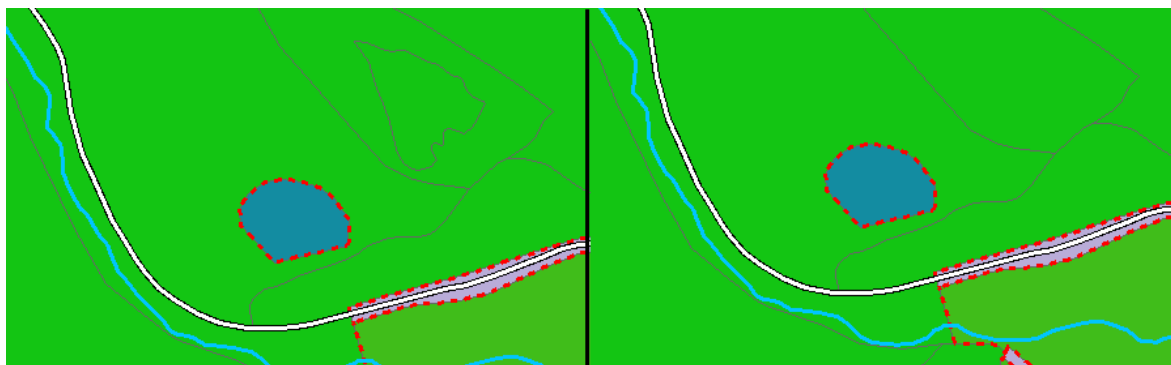


Obr. 18: Výstup funkce Dissolve. Vlevo původní cvičná oblast, vpravo výstup metody.

Tato metoda umožňuje spojování polygonů stejného typu, ale nesplňuje podmínku o maximální, popřípadě minimální, ploše a omezení spojování přes bariéry.

### 8.1.3 Eliminate Polygon Parts

Metoda *Eliminate Polygon Parts* umožňuje nastavení plochy polygonu, který bude spojen s jiným polygonem tak, aby jejich plocha byla menší než zadaná hodnota. Dále možnost *Eliminate contained parts only* říká, že se eliminují pouze polygony, které jsou uvnitř jiného a nedotýkají se hran. Při vyzkoušení metody bez zvolené možnosti eliminovat polygony uvnitř jiného polygonu, by byl výstup stále stejný. Sloučil pouze polygony uvnitř jiného, jak je na Obr. 19.



Obr. 19: Výstup metody Eliminate Polygon Parts. Vlevo původní cvičná oblast, vpravo výsledek.

## 8.2 QGIS

*QGIS* je uživatelsky přívětivý *open source* geografický informační systém licencovaný pod *GNU General Public Licence*. V základní verzi nabízí mnoho užitečných funkcí, a také nabízí připojení k databázi nebo načtení mapových služeb. Jeho výhodou pro uživatele je, že tento program se dá pořídit zdarma. V rámci *QGIS* je nabízeno mnoho modulů, které rozšiřují jeho funkcionalitu a stále vznikají nové nebo si uživatel může sám rozšířit aplikaci a modul pak poskytovat ostatním uživatelům. [18]

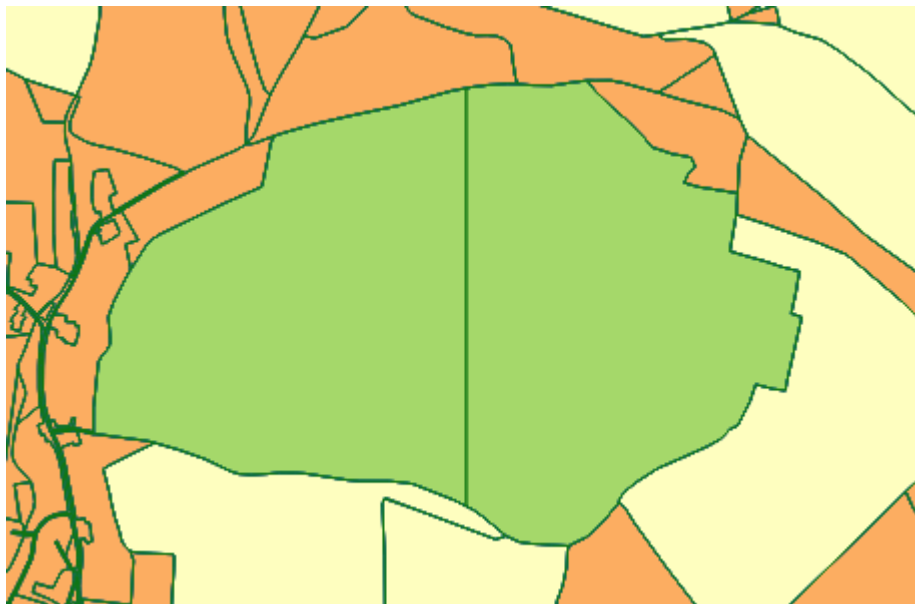
V této práci je použita verze *3.4.2-Madeira* v níž je implementováno mnoho funkcí a podobně jako u *ArcGIS* jsou rozděleny do skupin, např. vektorová analýza, vektorová geometrie apod.

### 8.2.1 Polygon Divider

Funkce *Polygon Divider* je užitečná, pokud je v dané oblasti mnoho sektorů, které přesahují maximální plochu. Může být zařazena do *post-processingu*, tj. do etapy, kdy se provádí dodatečné úpravy nad výsledky tvorby sektorů.

Uživatel zvolí vrstvu, která má být upravena. Následně zvolí plochu a určí, z jaké strany se bude provádět „stříhání“ polygonu, např. zprava doleva. Funkce efektivně rozdělí polygon a výstupem jsou dva polygony místo jednoho.

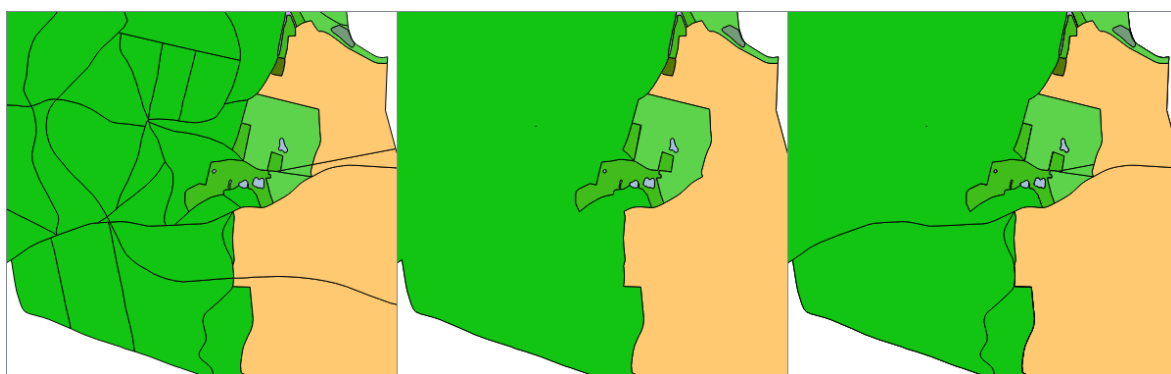
Z důvodu nekompatibility pluginu s novější verzí *QGIS* je zde uvedena ukázka ze starší verze aplikace. Na Obr. 20 barvy neodpovídají předdefinovaným barvám této práce, ale jde zde vidět, jak může být polygon rozdělen. Pozn. byla zkoušena i starší verze *QGIS*, ale při každém spuštění funkce celá aplikace padala.



Obr. 20: Rozdělení velkého sektoru.

### 8. 2. 2 Rozpustit (Dissolve)

Tato metoda má stejnou funkcionalitu jako u *ArcGIS* (viz kapitola 8. 1. 2) a tak byla rozšířena o rozdělení sjednocených polygonů pomocí linií. K tomu slouží funkce *Rozdělit liniemi*. Polygony byly rozděleny pomocí vrstvy bariér, ale ne vždy se polygon rozdělil. Důvodem je, že linie nevede přes celý polygon (viz Obr. 21).

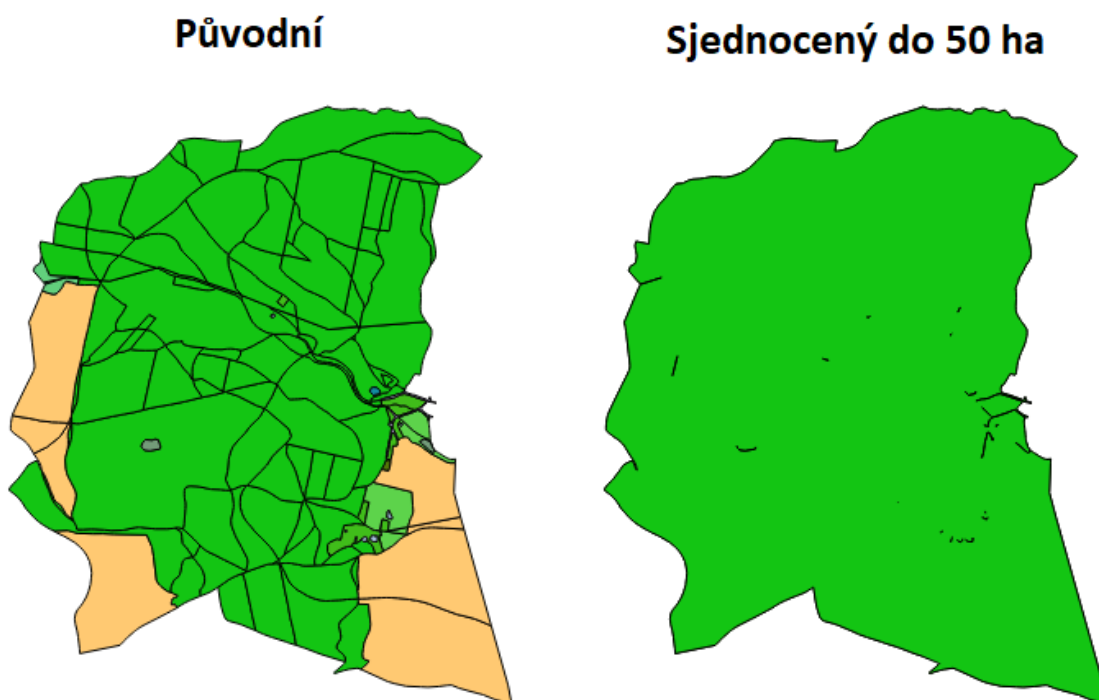


Obr. 21: Zleva: původní cvičná oblast, rozpustěná oblast, rozdělená oblast pomocí vrstvy s bariérami.

Výhodou této kombinace funkcí je, že zahrnuje podmínku stejných typů a bariér do výpočtu, i když některé bariéry vynechá, nedodržuje maximální plochu.

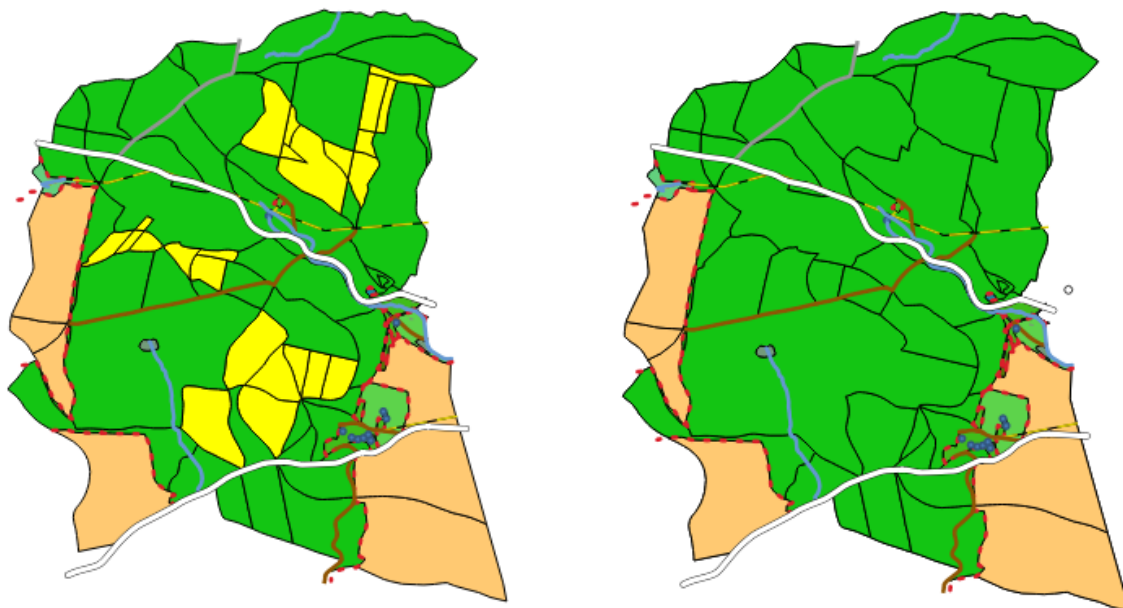
### 8. 2. 3 Eliminovat vybrané polygony

Funkce *Eliminovat vybrané polygony* se nachází v menu *Vektor -> Nástroje geoprocessingu*. Funguje následovně: vyberou se všechny polygony, které mají plochu do požadované velikosti a jsou spojeny se sousedním polygonem, který má buď největší či nejmenší plochu nebo mají společnou nejdelší hranici. Např. máme podmínku do 50 ha (viz Obr. 22). Jak už je vidět na obrázku, tato metoda spojí všechny vybrané polygony bez ohledu na to, že se překračuje maximální plocha. Také není možnost, jak do zpracování přidat omezení spojování přes bariéry a různých typů.



Obr. 22: Eliminace vybraných polygonů do 50 ha.

V tomto případě se jedná hlavně o manuální výběr polygonů. Možnost vybírání polygonů by mohl být proveden např. snížením velikosti plochy vybraných polygonů a to tak, že vybíraný polygon nemá souseda, kde přes společnou hranici vede bariéra a soused není jiného typu. Na Obr. 23 je tento postup dobře vidět. Je nutné dodat, že se nejedná o automatizovanou tvorbu sektorů, ale je dobré vědět, že taková metoda existuje.



Obr. 23: Příklad vybraní polygonů pro sjednocení tak, aby jejich soused nebyl jiného typu a nebyla mezi nimi bariéra.

## 9 VLASTNÍ ŘEŠENÍ

Během práce na vlastním řešení proběhlo několik změn požadavků, a ne vždy došlo ke zrychlení práce, spíše naopak. Nejvíce se měnil názor, jak velká musí být maximální plocha. V prvotní fázi byla velikost kolem 50 hektarů, která se nakonec snížila na asi 20 ha. Avšak pro názornost je lepší pracovat s maximální rozlohou 50 ha.

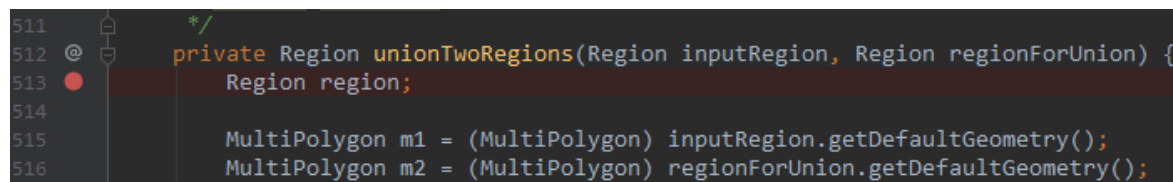
V této kapitole bude popsáno, jakým způsobem program pracuje. Nejedná se o aplikaci, jelikož program nemá žádné uživatelské prostředí. Budou zde popsány všechny prostředky využité pro tvorbu programu včetně programovacího jazyka, struktury projektu a nakonec budou popsány výstupy.

### 9.1 Použité technologie

Projekt je programován v objektovém programovacím jazyce **Java** s využitím knihovny **GeoTools**, což je *open source* knihovna Javy nabízející metody pro vizualizaci a zpracovávání prostorových dat a webových služeb využívající datové struktury postavených na *Open Geospatial Consortium* (OGC) specifikacích. Pomocí této knihovny se načítaly soubory ve formátu *Shapefile ESRI* a používaly především třídy *SimpleFeature* a *Geometry*.

*SimpleFeature* poskytuje metody umožňující čtení a nastavování atributů včetně geometrie. *Geometry* pak slouží převážně pro práci s geometrií. Nabízí topologické funkce pro nalezení například sousedů, kteří mají společnou hranici, vypočítání centroidu nebo vytvoření konvexní obálky kolem daného prvku. Všechny metody a specifikace jsou součástí dokumentace, která je dostupná online [7].

Vývoj programu probíhal v softwaru **IntelliJ IDEA**, který poskytuje například napovídání během psaní kódu, formátování či eliminování zbytečných řádků kódu, označování nesmyslně použitých vstupních parametrů apod. Nejsilnější funkcí je *Debug*. Tato funkce umožňuje spustit program a zastavit jej těsně před řádkem, kde docházelo k nějakému problému (viz. Obr. 24).



Obr. 24: Červený symbol kolečka označuje řádek, kde bude běh programu během debugování zastaven.

Uživatelské prostředí je možné si přizpůsobit, popřípadě přenášet mezi aplikacemi nastavení barev a rozložení horní lišty, kde se nacházejí tlačítka pro spuštění programu, debugování apod.



## 9.2 Algoritmus vlastního řešení

Nejjednodušším způsobem, jak popsat složitou logiku implementované kódem, je popsat ji pomocí algoritmu. Pod tímto pojmem si lze představit přesný návod nebo postup, řešící daný typ úlohy. Mezi základní vlastnosti algoritmu patří podle [1]:

- **Konečnost**, kdy končí v konečném počtu kroků.
- **Obecnost** řeší obecnou třídu problémů, a ne konkrétní problém.
- Každý krok algoritmu je přesně a jednoznačně definován. Tento princip se nazývá **determinovanost**.
- Má alespoň jeden **výstup**.
- Skládá se z konečného počtu jednoduchých kroků, tj. **elementárnost**.

Při řešení problematiky dělení sektorů byla v této práci použita metoda „*rozděl a panuj*“, tj. rozdělení složitého problému na jednoduché a jejich postupné spojování [1]. V případě vlastního řešení bylo prvním krokem načítání prvků z vrstev a jejich procházením. Tím bylo zjištěno, jaká data jsou poskytována (např. typy využití půdy, geometrii apod.). Následovalo vytvoření jednoduché metody, jak spojovat polygony dohromady a výstupem se stal jeden polygon. Další krokem bylo přidání podmínky maximální plochy a pro jednoduchost byly vypuštěny předem všechny polygony, které přesahovaly horní práh. Pak došlo k přidání podmínky stejného typu, omezení spojování přes bariéry, a nakonec definování intravilánu, i když tento krok provázelo několik změn, o kterých bylo stručně zmíněno v kapitole 6 v části intravilán.

Co se týče intravilánu, tak první myšlenkou bylo definovat ty typy, které se můžou spojovat i přes svou různost. Tak to původně definoval zadavatel. Postupem času bylo řečeno, že do výpočtu nemá vstupovat žádný intravilán a neměl by se nacházet ani ve výstupu. Avšak toto řešení by způsobovalo, že by se ve výsledné vrstvě nacházely díry. Proto byl všechen intravilán sloučen a označen jako typ *INTRAV*. Po vyloučení všech typů, které představují intravilán, zůstaly v rámci cvičné oblasti pouze typy *LPKROV* (louka, pastvina s křovinami), *LPSTROM* (louky, pastvina se stromy, les) a *ORNAPU* (orná půda), u kterých se sloučený vylučuje.

Intravilán je část území obce, která je z větší části zastavěna a zahrnuje další přilehlé plochy jako zahrady, pozemní komunikace, soukromou i veřejnou zeleň, vodní toky a plochy [13]. Sice intravilánem nejsou osamocené objekty jako vodní plochy, ale i přesto byly do intravilánu zahrnuty, protože mohou představovat nepřekročitelnou překážku.

### 9.2.1 Region

Pojem sektor byl popsán již v kapitole 3. 1 a v rámci vlastního řešení je v práci použit pojem region a je definován jako vstupní polygon s parametry *simpleFeature* (obsahuje

geometrii a atributy načtené ze souboru *Shapefile ESRI*), délka společné hranice, plocha, identifikátor regionu a poměr obvodů. Co přesně parametry znamenají, je popsáno v kapitole 9. 4. 1.

### 9. 2. 2 Tvorba sektorů

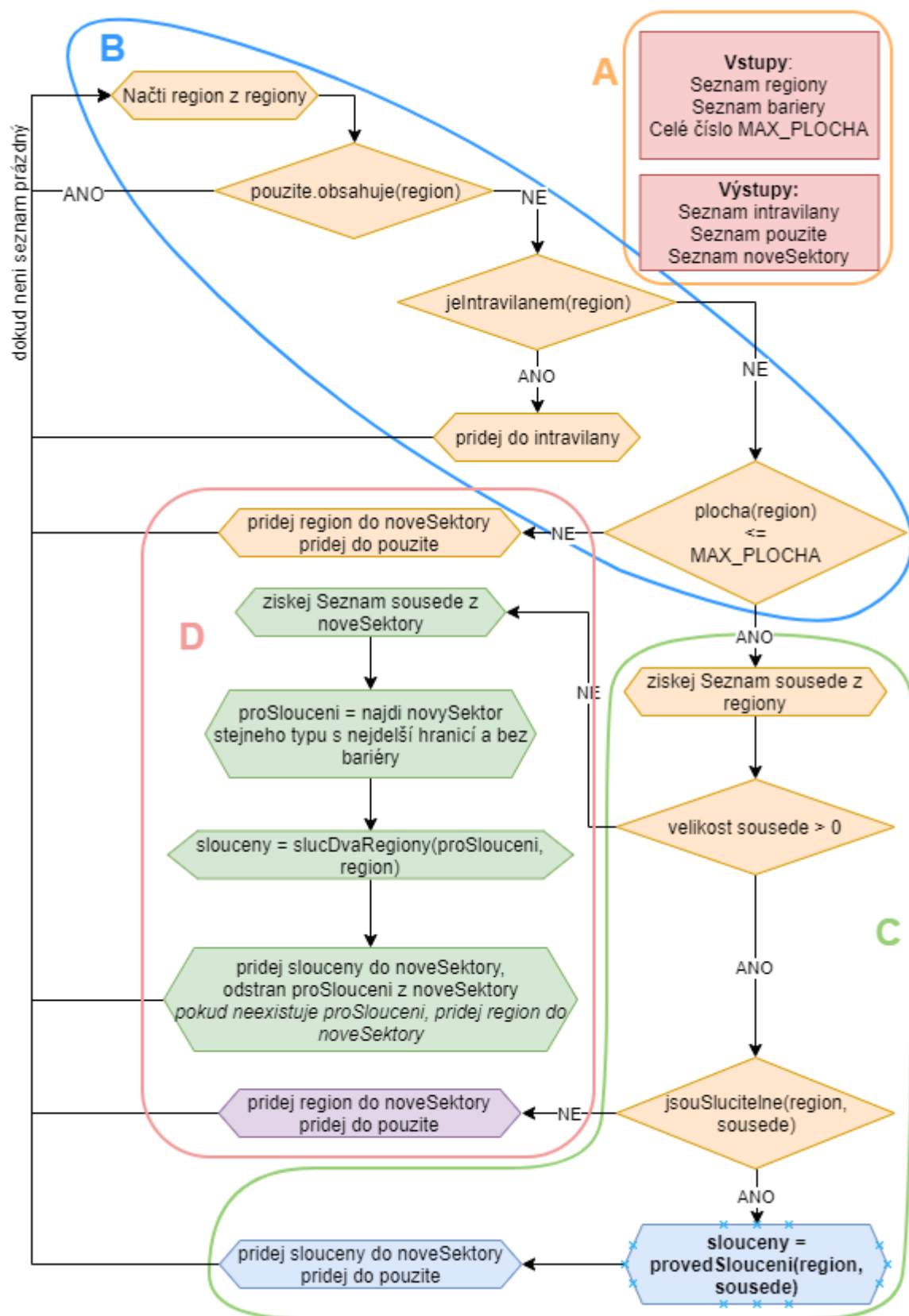
Prvním krokem tvorby sektorů je načtení vrstev do seznamů *regiony* a *bariery* a nastavení parametru *MAX\_AREA*, což určuje horní hranici velikosti výsledného sektoru. Taktéž jsou vytvořeny seznamy (viz Obr. 25 A):

- *intravilany*, do kterého se budou ukládat všechny regiony, jejichž typem je intravilán,
- *pouzite*, v němž budou uloženy ty regiony, které byly buď spojeny, uloženy do *intravilány* nebo neprošly podmínkou definující maximální plochu sektoru,
- *noveSektory* bude obsahovat výstupy vytvořených jako sjednocení regionů nebo regiony, které přesáhly maximální plochu.

Při spuštění algoritmu se postupně prochází celý seznam *regiony* a jako první se zjišťuje, zda je daný region v seznamu *pouzite*. Tímto krokem se dosáhne toho, že se nebudou znovu zpracovávat už zpracované regiony a zkrátí se tím časová náročnost algoritmu. Podmínka *jeIntravilanem()* hlídá, zda vstupní region je intravilánem a pokud ano, je uložen do seznamu *intravilány* a načte se další region v pořadí ze seznamu *regiony*. Poslední podmínka, která může přeskočit celý proces spojování zjišťuje, jestli region má větší plochu, než je maximální plocha. Pokud ano, je region označen jako sektor a přidán do seznamu *noveSektory* (viz Obr. 25 B).

Pokud region projde všemi třemi podmínkami, dostane se ke kroku, kde se hledají jeho sousední regiony, se kterými má společnou hranici. V podmínce *jsouSlucitelne()* se řeší, jestli alespoň jeden sousední region má stejný typ využití půdy jako současně zpracovávaný region. Pokud ano, je region sloučen se sousedními regiony stejného typu, označen jako sektor a vložen do seznamu *noveSektory*. Zároveň do seznamu *pouzite* vloží aktuálně zpracovávaný region a všechny regiony, které s ním byly sloučeny (viz Obr. 25 C).

Pokud ani jeden sousední region nebyl stejného typu, což mohlo nastat v případě, že je ohraničen sousedními regiony jiného typu, nebo se sousední regiony nacházejí v seznamu *pouzite* a tím pádem se s nimi dál nepracuje. Proto se znovu používá metoda pro získání sousedů, ale tentokrát ze seznamu *noveSektory*. Z nalezených sousedů se vybere ten, který je stejného typu, má nejdelší společnou hranici a neleží mezi nimi bariéra. Pokud takový soused existuje, je sloučen s regionem a je přidán do seznamu *noveSektory*, přičemž je z něj odstraněn ten sektor, který byl použit ke sloučení, aby nedocházelo k překrývání (viz Obr. 25 D).



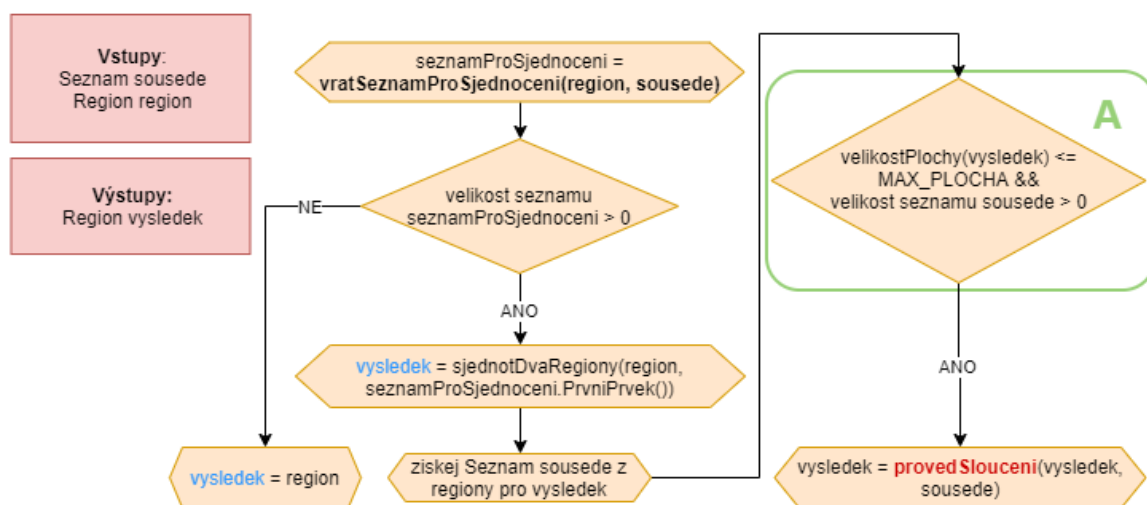
Obr. 25: Algoritmus popisující tvorbu regionů.

### 9. 2. 3 Slučování regionů

Předchozí algoritmus popisoval obecné fungování tvorby sektorů a jeho výstupem byl seznam nových sektorů. V této kapitole bude popsán konkrétní algoritmus, kde výstupem je pouze jeden sektor vzniklý sloučením regionů. Na Obr. 25 C je jeden krok označen tučným písmem a to *slouceny* = *provedSlouceni(region, sousede)*. Právě zde se řeší, které regiony budou sloučeny a jakým způsobem.

Algoritmus na Obr. 26 je rekurzivní, což znamená, že volá sám sebe. Opět se v jeho těle nachází další algoritmus se svou vlastní vnitřní logikou (tj. *vratSeznamProSjednoceni()*). Na vstupu je tentokrát region (vstupní) a jeho sousede. Zatím se neví, které sousední regiony jsou vhodné pro sloučení, a proto se použije algoritmus *vratSeznamProSjednoceni()* a výstup se uloží do proměnné *seznamProSjednoceni*. Pokud tento seznam má více než 0 regionů, pak se vezme první sousední region v *seznamProSjednoceni* a provede se jeho sjednocení se vstupním regionem. Výsledek sjednocení se uloží do *vysledek*. Nyní se pro *vysledek* hledají sousední regiony a pokud jsou splněny podmínky (viz Obr. 26 A), pak algoritmus volá sám sebe a výsledek se stává vstupním regionem.

Mohou nastat dva jediné případy, kdy nedojde k rekurzi. Buď *seznamProSjednoceni* je prázdný a přeskočí sjednocování nebo nebyla splněna alespoň jedna podmínka v A. V obou případech je výstupem *vysledek* (označen modře).



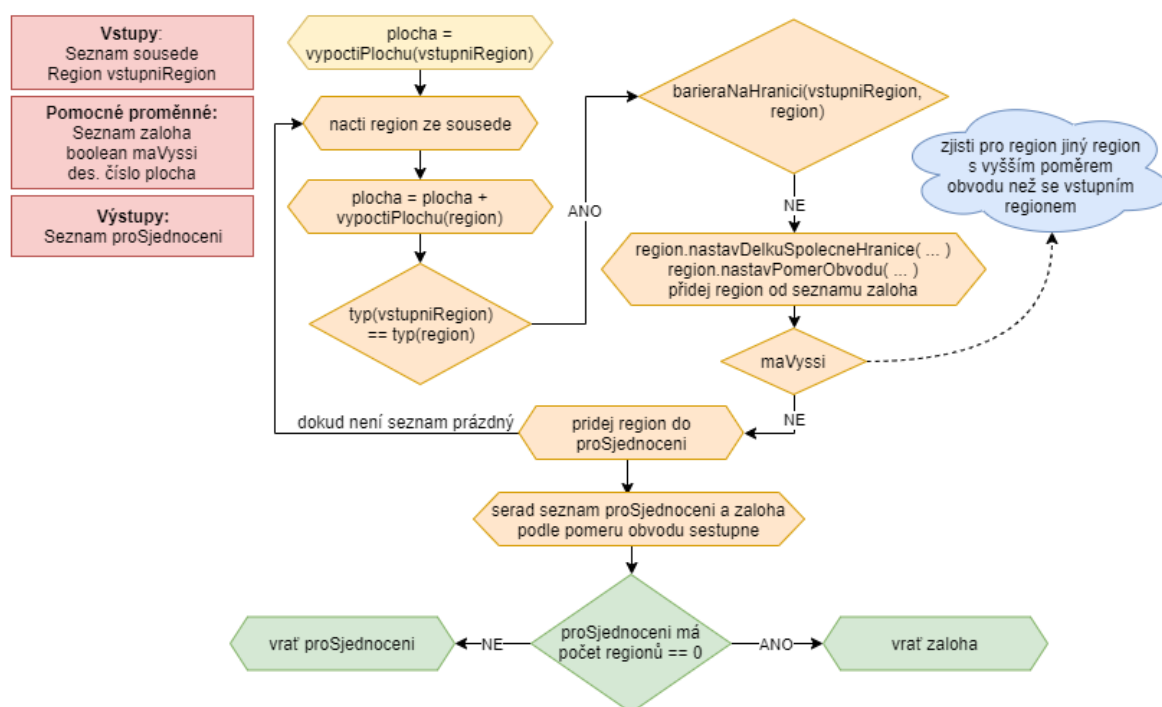
Obr. 26: Rekurzivní algoritmus *provedSlouceni()* pro vytvoření jednoho sektoru z právě zpracovávaného regionu a seznamu sousedních regionů.

### 9. 2. 4 Seznam pro sloučení

Algoritmus *vratSeznamProSjednoceni()* plní funkci filtru, jehož výsledkem jsou všechny sousední regiony, které splňují podmínky (viz kapitola 6). Do zpracování není zahrnuta podmínka minimální plochy, protože se s ní pracuje až v *post-processingu*.

Jako první se do proměnné *plocha* ukládá velikost *vstupního regionu* a potom se prochází *region* po regionu v seznamu *sousede*. K ploše se přičte plocha *region*

pro zjištění aktuální velikosti a na řadě je podmínka, která zjišťuje, jestli je *region* stejného typu jako *vstupniRegion*. Pokud ano, zjišťuje se, zda mezi nimi existuje bariéra. Podmínka je splněna při neexistenci bariéry a v této větvi se pro *region* nastavuje délka společné hranice a poměr obvodu, což se vypočte jako  $\frac{\text{délka společné hranice}}{\text{délka hranice připojovaného regionu}}$ . Do seznamu *zaloha* se přidá *region* z důvodu následující podmínky, která pro daný *region* navíc hledá jiné sousedy, kteří mají vyšší poměr obvodu, než jaký má se vstupním regionem. V této fázi může nastat, že do seznamu *proSjednoceni* nebude přidán ani jeden *region* a tímto způsobem se alespoň dostane na výstup *region* s nejvyšším poměrem obvodu. Je důležité oba dva seznamy seřadit podle velikosti poměru obvodů, tzn. od největšího po nejmenší. Poslední podmínka pouze kontroluje, zda v *proSjednoceni* je alespoň jeden *region* a pokud ne, vrátí seznam *zaloha*. Jinak vrátí *proSjednoceni* (viz Obr. 27).



Obr. 27: Algoritmus *vratSeznamProSjednoceni()* s výstupem seznam s regiony určených pro sjednocení.

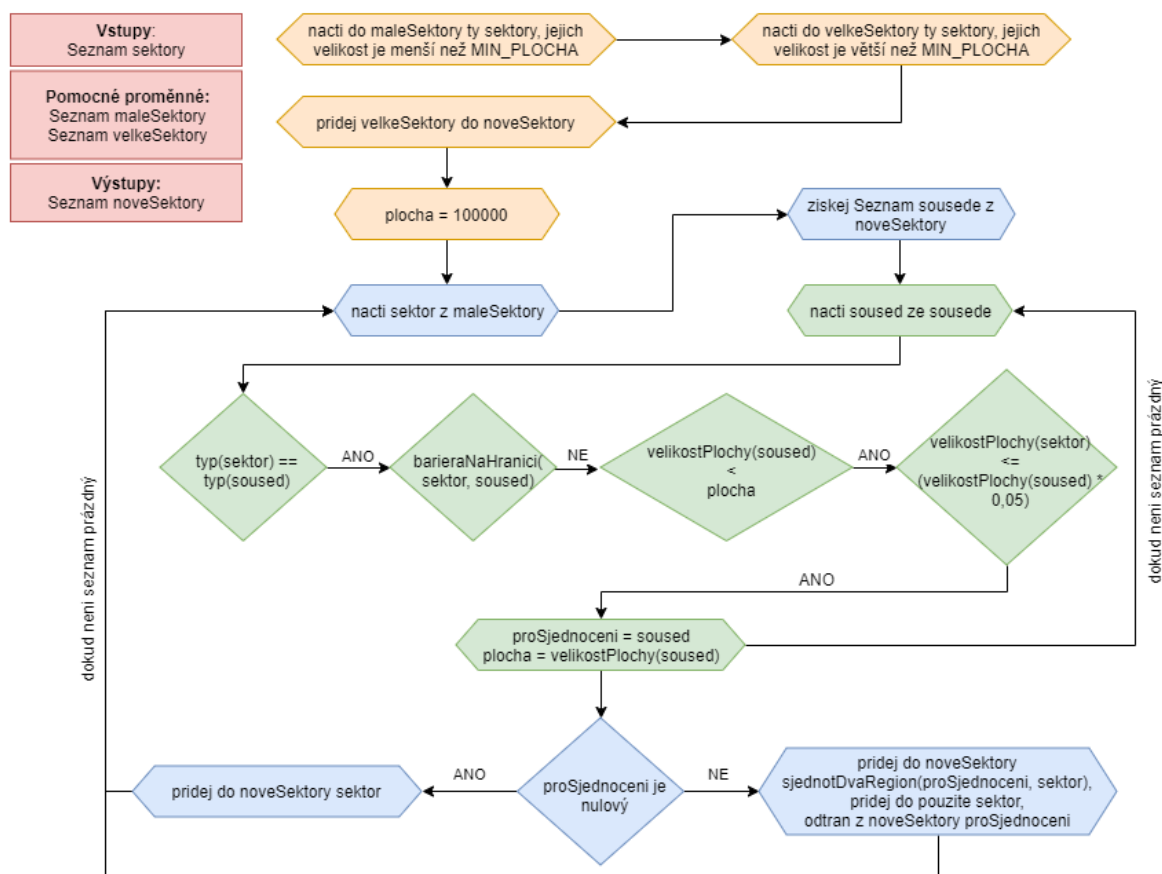
### 9.2.5 Post-processing

Poslední algoritmus, který zde bude uveden, se týká úprav výstupu. Podle výše definovaných algoritmů může nastat případ, kdy se sektorem stane sjednocený region nižší velikosti, než je minimální velikost. Díky menšímu počtu vstupních dat a absenci rekurzivní metody probíhá zpracování rychleji a je zde použita jednodušší logika (viz Obr. 28).

Tentokrát zde nebude použit pojem *region*, nýbrž pojem *sektor*, protože se pracuje nad vrstvou s nově vytvořenými sektory. Do seznamů *maleSektory* a *velkeSektory* se přidávají sektory podle toho, zda mají menší nebo větší plochu než je minimální plocha. Seznam *velkeSektory* jsou rovnou přidány do seznamu *noveSektory* a *maleSektory* se postupně

prochází. U každého sektoru se naleznou sousedé a vybere se ten soused, jehož typ je stejný jako typ sektoru, přes jejichž společnou hranici nevede bariéra, jeho plocha je nejmenší ze všech sousedů a zároveň je jeho plocha menší než 5 % z plochy sektoru. Důvodem je, že hlavním cílem je odstranit takové sektory, jejichž velikost byla tak malá, že se s nimi nedalo dál pracovat. Tuto hodnotu může uživatel nastavit podle sebe.


Podmínka na úrovni seznamu *maleSektory* kontroluje, zda je v *proSjednoceni* uložen nějaký soused a pokud ne, znamená to, že je ohraničen bariérami nebo jeho velikost je větší než 5 % plochy sektoru, ke kterému by měl být připojen.



Obr. 28: Algoritmus pro *post-processing* nad vrstvou s vytvořenými sektory.

### 9.3 Implementace

Než se začne psát kód, je důležité si nejdříve založit nový projekt a načíst všechny potřebné knihovny. V tomto případě se jedná o knihovnu *GeoTools*. Knihovna se načítá pouze v rámci projektu, takže pokud se založí nový, musí se znovu nastavit závislosti a počkat, až je vše staženo. Knihovna *GeoTools* je velmi rozsáhlá a zabere i hodinu, než se kompletně načte, ale při dalším spuštění už není třeba nic stahovat. Informace o tom, jak nainstalovat knihovnu *GeoTools* v prostředí *IntelliJ IDEA*, se nacházejí v dokumentaci [7].

Každý projekt má svou hlavní třídu (= *Main.java*), která slouží ke spuštění celého programu. Většinou se vytváří při založení projektu, ale pokud se tak nestane, je důležité tuto třídu vytvořit a v konfiguraci projektu ji nastavit (*Run -> Edit Configuration*). Hlavní třída je označena symbolem: . Není potřeba v ní mít celý program, ale stačí nějaká iniciální metoda, která spustí celý běh aplikace, např. *groupSectors()*.

V rámci programovacího jazyka *Java* se užívá název metoda a ta představuje nějakou funkcionalitu. Metody se píšou v rámci tříd, které mohou být buď objekty, tj. představují nějaký prvek reálného světa v abstraktní formě, nebo obsahují pouze statické metody, a tudíž se nemusí volat konstruktor, který inicializuje konkrétní objekt. Objekt je uložen na disku a proměnná obsahuje pouze jeho referenci.

Co se týče programování, tak názvy metod, proměnných, parametrů, tříd apod. jsou v anglickém jazyce a jejich názvosloví bude dodržováno i v této práci. Pouze text v komentářích je psán v českém jazyce.

## 9.4 Struktura projektu

Třídy jsou rozděleny do balíčků (angl. package) a každý balíček představuje nějakou funkcionalitu (viz Obr. 29). Hlavní balíček se nazývá *Main* a obsahuje třídu *Main*, kde si uživatel může vybrat část programu, která bude spuštěna. O to se stará enumerátor, což je speciální třída sloužící pouze pro definování konstant a neobsahuje žádné metody. V projektu je v balíčku *Enum* vytvořený enumerátor *MethodChoser* a obsahuje názvy části aplikace neboli inicializační metody: *GROUP\_SECTORS*, *LOG\_2\_SVG*, *LAND\_USE*, *GET\_BARRIERS*, *Z\_DIMENSION*, *POST\_PROCESSING*, *EVALUATION*.

Ve třídě *Main* je definován *switch*, který podle zvolené hodnoty enumerátoru vybere danou třídu a spustí se program.

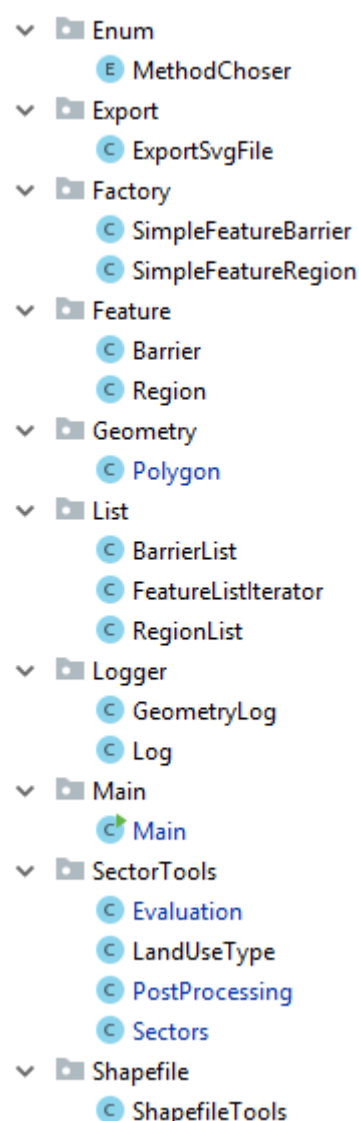
Příklad zdrojového kódu v *Main* třídě:

```
public class Main {
```

Každá třída obsahuje *TAG* nesoucí název třídy, tj. cesta k balíčku včetně názvu. Příklad výpisu: Log.d: cz.vsb.gis.sla0193.patrac.SectorTools.Sectors

```
private static final String TAG =
    Main.class.getName();
```

Dřív, než se začne vybírat metoda, je důležité nastavit vstupní parametry. Uživatel si může nastavit cestu ke vstupním datům, kam se bude ukládat výsledek, nastavit minimální a maximální plochu,



Obr. 29: Struktura projektu.



nastavit cestu k bariérám apod. Pokud se uvede pouze název, pak musí být data umístěna v kořenové složce projektu. Doporučuje se v ní alespoň vytvořit složku *data*.

```
private static final String TAG = Main.class.getName();
private static final double MIN_AREA = 20;
private static final double MAX_AREA = 50;
private static final double SCALE = 0.75;
private static final String REGION_INPUT = "data/nazev_vstupu.shp";
private static final String SECTOR_OUTPUT = "data/nazev_vystupu.shp";
private static final String BARRIER_INPUT = "data/bariery_vstup.shp";
private static final String BARRIER_OUTPUT = "data/bariety_vystup.shp";
private static final String BUILT_UP_AREA = "data/intravilan.txt";
private static final String POST_PROCESSING_OUTPUT = "data/
postProcessing.shp";
```

V tuto chvíli je vybrána část programu, které provádí tvorbu sektorů skrze metodu *groupSectors()*.

```
MethodChoser chosenMethod = MethodChoser.GROUP_SECTORS;
...
switch (chosenMethod) {
    case GROUP_SECTORS:
        //Hlavní metoda pro spojování do sektorů
        new Sectors(REGION_INPUT, BARRIER_INPUT, MIN_AREA * SCALE, MAX_AREA *
SCALE, SECTOR_OUTPUT, BUILT_UP_AREA).groupSectors();
        break;
    ...
}
```

#### 9.4.1 Balíček Feature

Tento balíček obsahuje dvě třídy představující objekty *Region* a *Barrier*. *Region* nese informace v proměnných:

```
private SimpleFeature simpleFeature;
private float lenghtOfCommonBorder;
private double area;
private int regionId;
private float circuitRatio;
```

*SimpleFeature* je třída, které rozšiřuje třídu *Feature* z *org.opengis.feature* a poskytuje metody umožňující získání hodnot atributů, kde na 0. pozici je uložena geometrie, popřípadě nabízí metodu pro získání základní geometrie a taktéž metody pro nastavení hodnot atributů a geometrie.

Třída *Region* využívá metody ze *SimpleFeature* a dále obsahuje metody vracející hodnoty globálních proměnných, protože k nim má přístup jenom tato třída z důvodu nastavení proměnné jako *private*. Metoda *getType()* vrací hodnotu atributu s názvem *TYP* přímo ze *SimpleFeature*. Pro definici, o který atribut se jedná, je použit *String* (tj. textový řetězec), protože je jednodušší kontrolovat název atributu než jeho pozici.

```
public String getType() {
    return simpleFeature.getAttribute("TYP").toString();
}
```



Dále třída *Region* poskytuje metody pro vypočítání délky společné hranice nebo poměru obvodu. Aby bylo možné zjistit délku, popřípadě obsah, je nutné převést *SimpleFeature* na geometrii, čemuž odpovídá třída *Geometry* z `com.vividsolutions.jts.geom`. V příkladě je použita konkrétní geometrie a to *MultiPolygon*, protože vstupní regiony jsou polygony.

```
public double countLengthOfCommonBorder(Region region) {
    MultiPolygon thisPolygon = (MultiPolygon) this.getDefaultGeometry();
    MultiPolygon regionPol = (MultiPolygon) region.getDefaultGeometry();
```

*Geometry*, respektive *MultiPolygon* či *MultiLineString* pro linie, nabízí také metody s topologickou funkcionalitou, např. *intersection()*, která vrací *Geometry* a tudíž na ni lze použít metodu *getLength()*. *Java* pracuje s tečkovou notací, a *IntelliJ IDEA* nabízí funkci, že když se za objekt, popř. statickou třídu, přidá tečka, software nabídne všechny metody, které objekt, popř. statická třída, nabízí a pokud je dobrá znalost anglického jazyka, je velmi jednoduché najít takovou metodu, která vrací konkrétní výsledek.

```
    return thisPolygon.intersection(regionPol).getLength();
}
```

Pokud jsme v rámci dané třídy s objektem *Region*, pak *this* ukazuje přímo na objekt a není třeba jej získávat přes *get* metody.

```
public float countCircuitRatio(Region region) {
    return ((float) (this.countLengthOfCommonBorder(region) /
region.getLengthOfBorder())) * 100;
}
```

Třída *Barrier* slouží pouze pro uložení dat, respektive *SimpleFeature*, typu bariéry a indexu. Index se generuje sám a číslo představuje pořadí, ve kterém daná linie vstoupila do čtení. Ty si získává z atributu liniové bariéry.

#### 9.4.2 Balíček List

Důležitým balíčkem je balíček *List*. Třídy *RegionList* a *BarrierList* mají za úkol načíst prvky ze souboru ve formátu *Shapefile ESRI* a uložit je do seznamu daného typu:

```
List<Region> regions = new ArrayList<>();
List<Barrier> barriers = new ArrayList<>();
```

Tato inicializace znamená, že *ArrayList* bude obsahovat objekty typu *Region* nebo *Barrier* a *List* je pouze seznam o jednom řádku. Jelikož se vytváří nový objekt, tak se musí použít *new*.

*RegionList* nabízí navíc metody pro vypočítání plochy: *getArea(Region region)* a třídění podle plochy vzestupně a sestupně: *sortByAreaAsc(List<Region> list)*, *sortByAreaDesc(List<Region> list)*, podle délky společné hranice: *sortByLength(List<Region> list)* a podle poměru obvodu: *sortByCircuitRatio(List<Region> list)*.

Třída *FeatureListIterator* pouze definuje iterátor pro `ArrayList<T>`, kde *T* znamená nějaký objekt. Může to být například *Region*. Iterátor má za úkol prvek po prvku procházet seznamem a *hasNext()* pouze říká, jestli má seznam další prvek.

```
Iterator<Region> regionIterator = listOfSectors.iterator();
...
while (regionIterator.hasNext()) {
...

```

Místo iterátoru se může použít například *for* ve tvaru:

```
for(Region region : regions) {...}
```

### 9. 4. 3 Balíček Factory

Tento balíček nabízí dvě třídy *SimpleFeatureBarrier* a *SimpleFeatureRegion*, které převádějí *Barrier* nebo *Region* na *SimpleFeature*. V obou případech se definuje typ, což je vytvoření struktury prvku. U *Region* se na první pozici ukládá geometrie, následuje *ID*, *TYP* a *AREA*.

```
public static SimpleFeatureType createSimpleFeatureType() throws Exception {
    return DataUtilities.createType("Sectors",
        "the_geom:MultiPolygon," +
        "ID:Integer," +
        "TYP:String," +
        "AREA:double");
}
```

Metoda *buildSimpleFeature(Region region)* následně „sestaví“ a vrátí *SimpleFeature*. Důvodem je vstup u metody vytvářející nový soubor *Shapefile ESRI* ze *SimpleFeature*.

### 9. 4. 4 Balíček SectorTools

Zde se nachází čtyři třídy a to *Sectors*, *LandUseType*, *PostProcessing* a *Evaluation*. Pokud se bude postupovat chronologicky, začne se s *LandUseType*, protože patří do přípravné fáze než se data začnou zpracovávat. Metoda *getTypes()* postupně čte zadaný atribut a jeho hodnotu ukládá do seznamu *listOfTypes* za podmínky, že se daný typ v seznamu už nenachází.

```
public static List<String> getTypes(String nameOfAtr, List<Region>
simpleFeaturesList) {
    List<String> listOfTypes = new ArrayList<>();
    for (Region simpleRegion : simpleFeaturesList) {
        String type = simpleRegion.getAttribute(nameOfAtr).toString();
        if (!listOfTypes.contains(type)) {
            listOfTypes.add(type);
        }
    }
    return listOfTypes;
}
```

Pak metoda *writeAtrToLog()* vezme seznam a vloží jej do textového souboru *\*.txt* a ke každému kódu typu přiřadí jeho význam, např:

```
LPSTROM;Louky, pastvina se stromy  
ORNAPU;Orná půda  
TRTRPO;Travnatý povrch  
SADZAH;Sad, zahrada  
LPKROV;Louka, pastvina s křovinami
```

S tímto seznamem pak může uživatel dál pracovat a definovat, které typy jsou intravilány a použít soubor jako vstup. Díky tomu uživateli odpadá nutnost ručně procházet typy v atributové tabulce. Názvy jsou definovány pro konkrétní kódy typů využití půdy, a pokud by tam byl jiný, pak tato metoda nebude fungovat správně.

Ve třídě *Sectors* se pracuje s regiony a vytvářejí se nové sektory. Způsob, jakým se to dělá, je popsán v kapitole 9. 2. Jako první se musí zavolat konstruktor, který inicializuje a definuje objekt. Vstupem je cesta k vrstvě s regiony a s bariérami, minimální a maximální plocha, cesta pro výstup a cesta k textovému souboru s intravilány.

```
public Sectors(String REGION_INPUT, String BARRIERS, double MIN_AREA, double  
MAX_AREA, String SECTORS_OUTPUT, String builtUpArea) {...}
```

Po načtení prvků z vrstev do *listOfRegions* se seznam regionů seřadí od největšího po nejmenší region podle plochy.

```
RegionList.sortByAreaDesc(listOfRegions);
```

S využitím iterátoru se prochází region po regionu a postupuje se podle algoritmu z kapitoly 9. 2. 2 (zdrojový kód na další straně).

```

while (regionIterator.hasNext()) {
    ...
    if (!removed.contains(currentRegion)) {
        //region je intravilán... nemá cenu cokoli řešit
        if (isBuiltUpArea(currentRegion)) {
            ...
            builtUpAreas.add(currentRegion);
            removed.add(currentRegion);
        } else {
            //region má plochu menší než MAX_AREA
            if (RegionList.getArea(currentRegion) <= MAX_AREA) {
                listOfTouches = getTouches(currentRegion, listOfRegions);
                groupableByType = areGroupableByType(currentRegion,
                                                        listOfTouches);
                //Pokud region nemá žádné sousedy, nemá to cenu dál řešit
                if (listOfTouches.size() > 0) {
                    //Pokud se v seznamu sousedů nachází alespoň jednoho typu,
                    jinak nemá smysl řešit
                    if (groupableByType) {
                        ...
                        Region groupedRegion = group(listOfTouches,
                                                        currentRegion);
                        addNewRegion(groupedRegion, currentRegion);
                    } else {
                        ...
                        addNewRegion(currentRegion);
                    }
                } else {
                    //znovu najde touches, ale z nově vytvořených sektorů a
                    spojí je dohromady
                    ...
                    List<Region> touches = getTouches(currentRegion,
                                                        newSectors);

                    Region forUnion = new Region();
                    double lengthOfCommonBorder = 0;
                    for (Region touch : touches) {
                        if (currentRegion.getType().equals(touch.getType())) {
                            if (
                                getIntersectionLength(currentRegion, touch) > lengthOfCommonBorder &&
                                !intersectionHasBarrier(currentRegion, touch)) {
                                    forUnion = touch;
                                }
                            }
                        }
                    }
                    RegionList.println(touches, TAG);
                    //tato podmínka kontroluje, jestli existuje nějaký sektor
                    pro sloučení a pokud ne, přidá původní sektor do newSectors
                    if (forUnion.getSimpleFeature() != null) {
                        addNewRegion(unionTwoRegions(forUnion, currentRegion));
                        newSectors.remove(forUnion);
                    } else {
                        addNewRegion(currentRegion);
                    }
                }
            } else {
                addNewRegion(currentRegion);
            }
        }
    }
}
}

```

Další důležitou metodou je `group()`. Jak už bylo řečeno předtím, jedná se o rekurzi, kdy metoda volá sama sebe. Volání přestane ve chvíli, kdy je překročena `MAX_AREA` nebo dojdou prvky v `listOfTouches`.

```
private Region group(List<Region> listOfTouches, Region region) throws
Exception {
    ...
    Region result;
    //vrátí seznam sousedů, které jsou vhodné pro sloučení
    List<Region> listForUnion = getListForUnion(region, listOfTouches);
    //vezme první region a spojí dohromady
    ...
    if (listForUnion.size() > 0) {
        result = unionTwoRegions(region, listForUnion.get(0));
        listOfTouches = getTouches(result, listOfRegions);
        ...
        //Dokud je plocha spojeného regionu menší než MAX_AREA, bude metoda
        //volat sama sebe
        if (result.getArea() <= MAX_AREA && listOfTouches.size() > 0) {
            result = group(listOfTouches, result);
        }
    } else {
        result = region;
    }
    return result;
}
```

Rekurze může být velmi nebezpečná. V případě, že je rekurze špatně ošetřena, může se stát, že nikdy neskončí. Tuto metodu doprovázelo nejvíce potíží. Často se stávalo, že docházelo k neustálému volání metody a slučované regiony nikdy nedosáhly přesahu maximální plochy. Proto je nutné, aby k přesahu maximální plochy došlo a důsledkem toho se pak plocha u většiny sektorů pohybovala nad danou hranicí.

```
private static final double MAX_AREA = 50;
private static final double COEF = 0.75;
...
new Sectors(..., MIN_AREA * COEF, MAX_AREA * COEF, ...)
```

Číslo 0,75 bylo zvoleno po pokusech, kdy se sledovalo, jak musí být vysoké, aby se dosáhlo co nejbližších výsledků k maximální ploše. Samozřejmě to záleží na vstupních datech a může být kdykoliv upraveno. Další možnost ošetření nekonečné rekurze je přičtení k regionu plochu sousedního regionu v rámci metody `group()`.

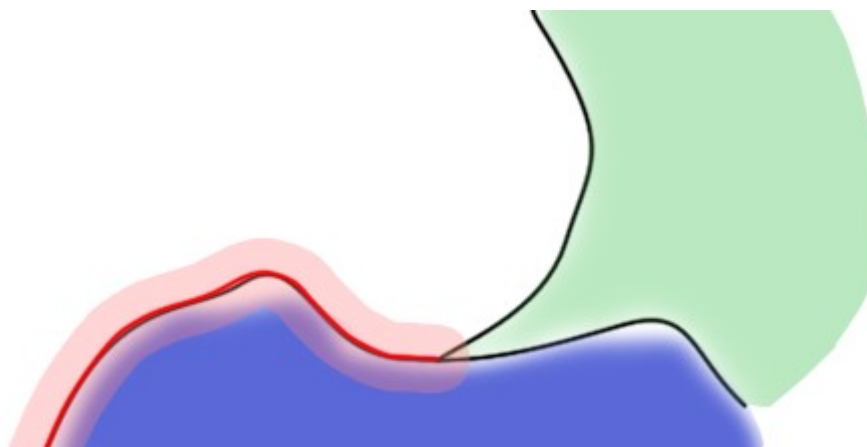
Metoda `getListForUnion()` vrací seznam regionů, které jsou vhodné pro sloučení. Pracuje se zde s podmínkami ohledně stejného typu, omezení spojení přes bariéru a jestli se pro souseda vyskytuje jiný region, se kterým má větší poměr obvodu než se vstupním. Často se stávalo, že pro parametr `hasHigher` často vycházel `true` úplně pro všechny sousedy a padalo to na výjimce `NullPointerException`. Proto se zde zavedl záložní seznam. Pokud se stane, že `listForUnion` bude prázdný, pak se vrátí `backupForUnion` a v metodě `group()` se v obou případech vybere první region.

```

private List<Region> getListForUnion(Region inputRegion, List<Region>
listOfTouches) {
    ...
    //zjišťuje, zda oba regiony mají stejný typ
    if (inputRegion.getType().equals(currentRegion.getType())) {
        //zjišťuje, zda na společné hranici leží bariéra
        if (!intersectionHasBarrier(inputRegion, currentRegion)) {
            //Vypočítá se délka společné hranice a poměru obvodu a
            //vloží se do parametru regionu.
            currentRegion.setLenghtOfCommonBorder((float)
inputRegion.countLengthOfCommonBorder(currentRegion));
            currentRegion.setCircuitRatio(inputRegion.countCircuitRatio(currentRegion));
            backupForUnion.add(currentRegion);
            //Zjišťuje se, zda daný region má jiný regon s delší
            //hranicí či větším poměrem obvodů.
            hasHigher = hasHigherCircuitRatio(currentRegion);
            //pro možnost spojování v závislosti na nejdelší hranici
            if (!hasHigher) {
                listForUnion.add(currentRegion);
            }
        }
    }
    ...
    RegionList.sortByCircuitRatio(listForUnion);
    RegionList.sortByCircuitRatio(backupForUnion);
    ...
    if (listForUnion.size() == 0) {
        return backupForUnion;
    } else {
        return listForUnion;
    }
}

```

Při zjišťování, zda na společné hranici leží bariéra, se vyskytl problém různé přesnosti souřadnic a ani po zjednodušení neboli snížení desetinných míst na čtyři nepomohlo. Proto se musely všechny linie bariér obalit bufferem do jednoho metru (tj. obalová zóna do zadané vzdálenosti). I když nyní fungovalo zjišťování, že se nad společnou hranicí nachází bariéra, nastal problém u společných hranic, kde bariéra přes ně nevedla, ale i tak na ně dosáhl buffer (viz Obr. 30).



Obr. 30: Bariéra nevede přes společnou hranici modrého a zeleného regionu, ale buffer se jí dotýká.

Proto se v kódu ošetřuje délka části hranice, která je obsažena v bufferu. Několika pokusy se došlo až k délce 4 m. Čím je úhel menší směrem k bariéře, tím je délka hranice vnořená do bufferu delší. Je stále možné, že ve výstupu nedojde ke sjednocení některých regionů, i když mezi nimi neexistuje žádná bariéra. Tento problém je spíše pro další studie, které nejsou součástí této práce.

```
if (inputIntersection.intersects(bufferedBarrier) &&
inputIntersection.intersection(bufferedBarrier).getLength() >= 4) {...}
```

Balíček *SectorTools* obsahuje navíc třídu *Evaluation*, která pro výsledky vypočítá parametry jako je kulatost, poměr mezi obsahem a obsahem konvexní obálky a kompaktnost.

#### 9.4.5 Balíček ShapefileTools

Po dokončení metody pro tvorbu sektorů *groupSectors()* se musí nové sektory (*newSecotrs*) uložit do nového souboru. *ShapefileTools* obsahuje pouze jednu metodu a to *newShp()*, kde vstupy jsou:

```
public static void newShp(List<?> inputList, String newPath) throws Exception {
```

*List<?>* znamená, že se zatím neví, jaký typ objektů uchovává. V tomto případě se bude jednat o objekty typu *Region* nebo *Barrier*. Textový řetězec *newPath* definuje cestu, kam se uloží nově vytvořený soubor. Cesta byla definována v konstantě *SECTOR\_OUTPUT* nebo *BARRIER\_OUTPUT* (viz kapitola 9. 4)


Volání metody se provádí následovně:

```
ShapefileTools.newShp(newSectors, SECTOR_OUTPUT);
```


Když přijde na vstup *List<?>*, je důležité rozeznat, jaký typ objektu v sobě uchovává. K tomu slouží operátor *instanceof*, který testuje, jaký typ je v instanci a s použitím podmínek se zvolí řešení pro konkrétní objekt. Aby bylo možné uložit nové sektory do nového souboru, musí se nejdříve převést na *SimpleFeature*. Pokud ani jeden ze seznamů neobsahuje objekt typu *Region* nebo *Barrier*, je zde navíc přidána *else* větev, a ta upozorní uživatele, že jeho vstup neodpovídá předem definovaným typům.

```
if (inputList instanceof List) {
    if (inputList.size() > 0 && inputList.get(0) instanceof Region) {
        List<Region> regions = (List<Region>) inputList;
        TYPE = SimpleFeatureRegion.createSimpleFeatureType();
        for (Region region : regions) {
            list.add(SimpleFeatureRegion.buildSimpleFeature(region));
        }
    } else if (inputList.size() > 0 && inputList.get(0) instanceof Barrier) {
        List<Barrier> barriers = (List<Barrier>) inputList;
        TYPE = SimpleFeatureBarrier.createSimpleFeatureType();
        for (Barrier barrier : barriers) {
            list.add(SimpleFeatureBarrier.buildSimpleFeature(barrier));
        }
    } else {
        Log.d(TAG, "Na vstupu je jiný typ než Region nebo Barrier.");
        return;
    }
}
```

## 9.5 Výstupy

Z důvodu ušetření času byla aplikace testována nad menším vzorkem, tj. nad cvičnou oblastí. Pro představu, jak dlouho může trvat výpočet, byl program spuštěn nad celým Královéhradeckým krajem a trvalo to cca 30 hodin. Cvičná oblast po přípravě obsahuje 141 prvků (viz Obr. 9) a u každého příkladu bude uveden čas zpracování, výsledný počet prvků, počet sektorů u *post-processingu* a hodnocení tvaru u *post-processingu*, protože důležitý je výsledek. Pro porovnání bude přidána vrstva s manuálně vytvořenými sektory. Symbol této vrstvy v legendě je .

Pro **hodnocení tvaru** budou vypočítány hodnoty pouze pro kompaktnost, kulatost a poměr mezi obsahem a obsahem konvexní obálky, protože jejich implementace v rámci programovacího jazyka byla nejjednodušší a nejrychlejší. Použije se průměrná hodnota bez extrémů, čímž je intravilán, protože se spojuje do jednoho sektoru napříč celou oblastí.

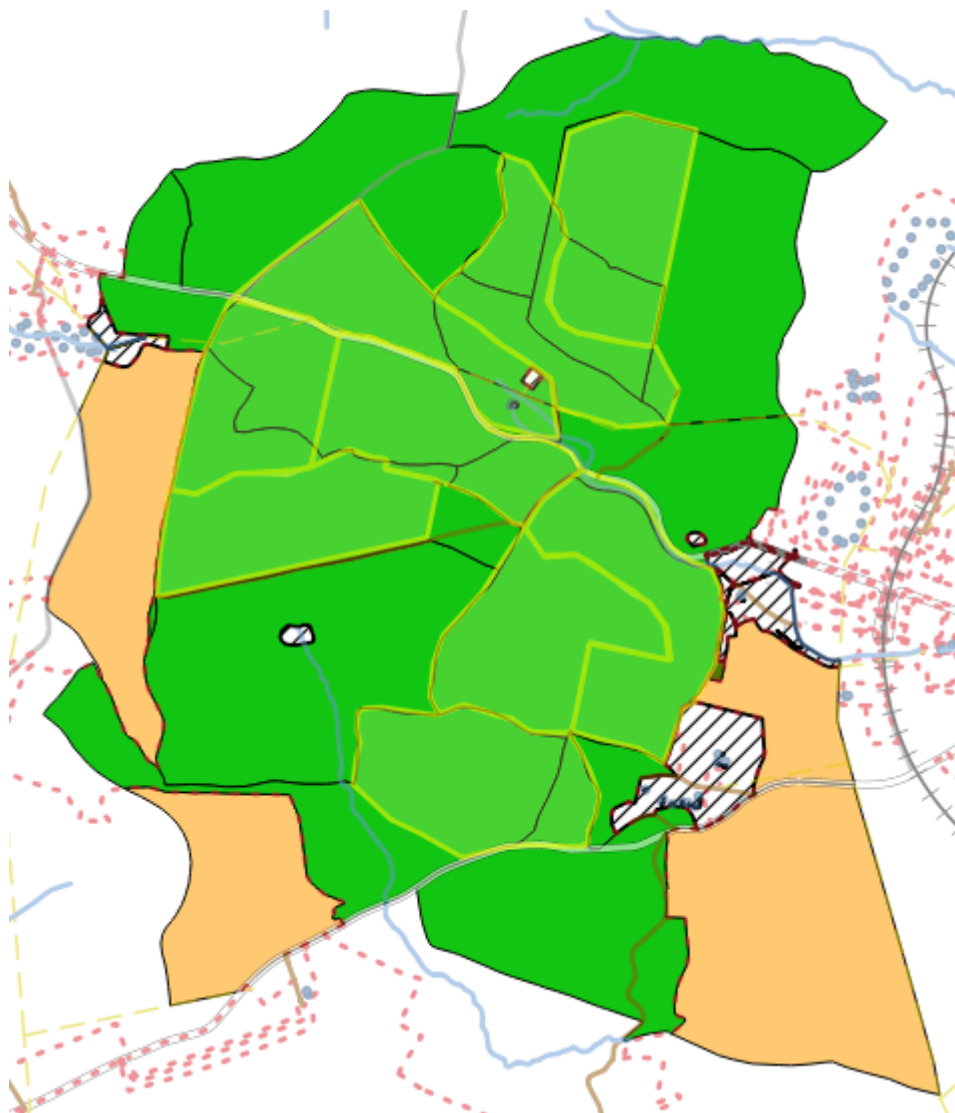
Typy využití půd definované jako intravilány jsou popsány v příloze 1 a v legendě je označen .

Zpracování probíhalo na počítači se čtyřmi procesory o frekvenci 3,2 GHz až 3,6 GHz, 16 GB RAM s daty uloženými na disku se 7 200 otáčkami za minutu.



### 9. 5. 1 Cvičná oblast bez bariér

Na Obr. 31 je vidět výstup ze zpracování pro vytvoření nových sektorů, kde nebyly použity bariéry. Bylo vytvořeno celkem 20 sektorů a jelikož po *post-processingu* nedošlo k žádné změně, nebude zde uveden jeho příklad. Sektory se spojují i přes bariéry. V ukázce jsou vloženy i manuálně vytvořené sektory. V Tab. 1 jsou vypočtené hodnoty. Kompaktnost, kulatost a poměr mezi obsahem a obsahem konvexní obálky jsou popsány v kapitole 5.



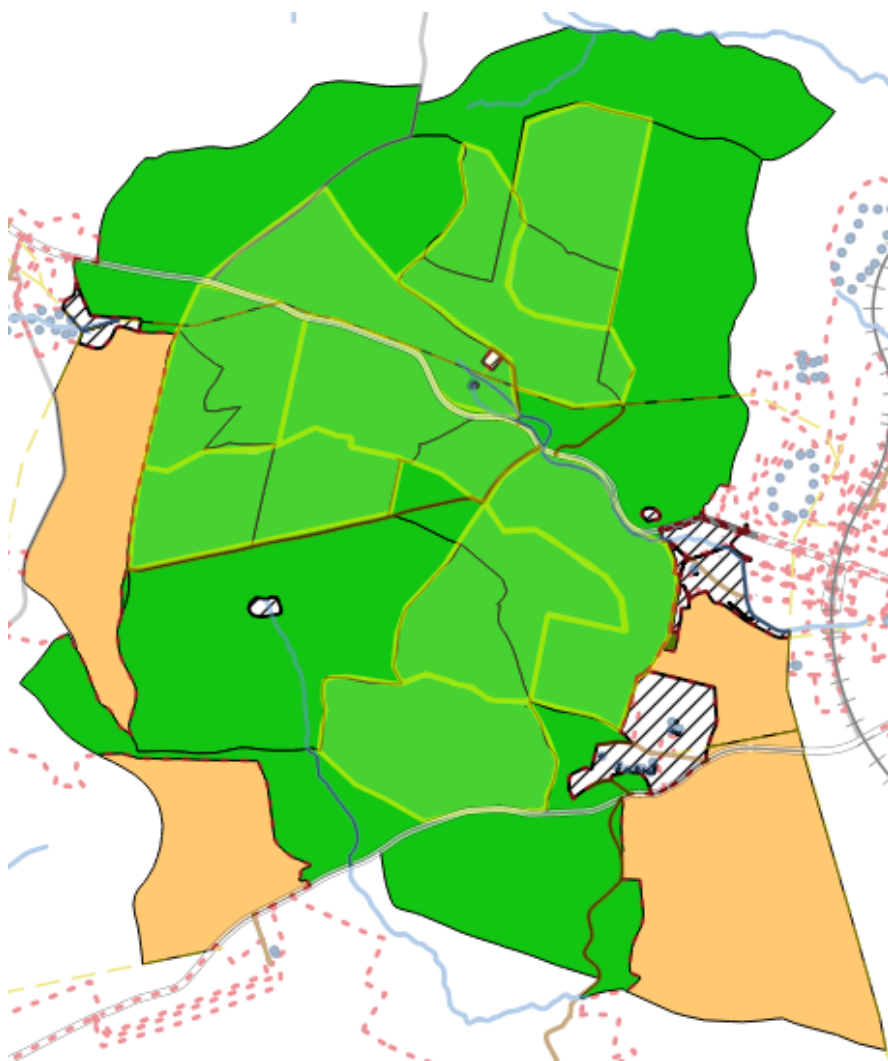
Obr. 31: Výstup tvorby sektorů bez použití bariér.

Tab. 1: Zjištěné hodnoty pro výstup zpracování z cvičné oblasti bez použitých bariér.

<b>Počet sektorů:</b>	20	<b>Kompaktnost:</b>	4,26
<b>Čas zpracování (s):</b>	5,67	<b>Kulatost:</b>	25,63
<b>Počet sektorů po post-processingu:</b>	20	<b>Poměr obsahu a obsahu konvexní obálky:</b>	0,74

### 9. 5. 2 Cvičná oblast s bariérami

Na rozdíl o tvorby sektorů bez bariér je cvičná oblast více rozdělena a má vyšší počet menších sektorů, než je minimální hodnota, protože jejich spojení nedovolí bariéra. Rozdíl mezi zpracováním, kde se použily bariéry a zpracováním bez bariér, není moc vidět. Také lze porovnat rozdíl mezi manuální tvorbou a automatizovanou tvorbou sektoru. Program neví, jak co nejlépe spojit polygony a v tuto chvíli se řídí spojováním podle největšího poměru mezi délkou společné hranice a délkou hranice připojovaného regionu. Využití jiné logiky přinese jiné výsledky, a kdyby se nepoužila žádná, regiony budou spojené způsobem *first-in first-out* neboli *FIFO* (viz Obr. 32).



Obr. 32: Výstup tvorby sektorů s použitím bariér.

V tabulce jsou uvedeny vypočtené hodnoty parametrů (viz Tab. 2).

Tab. 2: Zjištěné hodnoty pro výstup zpracování z cvičné oblasti s použitím bariér

<b>Počet sektorů:</b>	33	<b>Kompaktnost:</b>	4,76
<b>Čas zpracování (s):</b>	9,52	<b>Kulatost:</b>	20,93
<b>Počet sektorů po post-processingu:</b>	27	<b>Poměr obsahu a obsahu konvexní obálky:</b>	0,76

Tentokrát *post-processing* spojil v rámci možností sektory, kdy jejich velikost byla menší než minimální velikost. Dodrželo se nespojování přes bariéry. Na Obr. 33 jsou žlutě označena místa, kde ke spojení došlo. Počet sektorů se snížil z 33 na 27 a označená místa jsou viditelná na malém rozlišení ukázky.



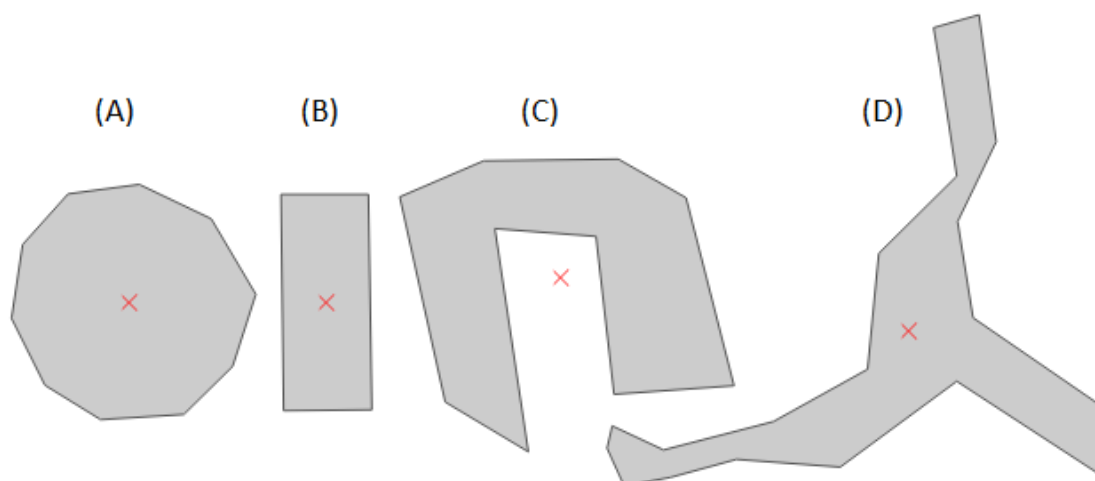
Obr. 33: Ukázka, kde došlo ke sloučení menších sektorů.

## 10 HODNOCENÍ

Hodnocení se dá rozdělit na dva pohledy. Z prvního pohledu bude zhodnocení výstupů podle vypočtených hodnot přinášejících informaci o výstupu. Toto hodnocení bude více objektivní, protože se budou mezi sebou porovnávat čísla. Hodnocení z druhého pohledu bude spíše subjektivní, protože se bude hodnotit vizuální stránka výstupu.

### 10.1 Hodnocení polygonů

Dříve, než se bude hodnotit tvar sektorů, provedly se výpočty kompaktnosti, kulatosti a poměru mezi obsahem a obsahem konvexní obálky na čtyřech ručně vytvořených polygonech (viz Obr. 34). Účelem je porovnat různé tvary a vysvětlit, co výsledné hodnoty pro hodnocení tvarů znamenají.



Obr. 34: Manuálně vytvořené polygony různých tvarů. Červený křížek označuje centroid (tj. těžiště) polygonu.

Kompaktnost testuje, zda je tvar polygonu kulatý. V tomto případě je hodnota rovna 1. Nejvíce kompaktním tvarem je polygon *A* s hodnotou 1,04. Naopak nejméně kompaktní tvar má polygon *D* s hodnotou 5,77. Na obrázku nahoře se tvar polygonů stává postupně více složitějším a v Tab. 3 se hodnoty kompaktnosti postupně zvyšují.

Kulatost také hodnotí kulatý tvar polygonu a úplně kulatému tvaru odpovídá hodnota 0. Jak lze vidět na obrázku Obr. 34, za nejvíce kulatý tvar byl označen polygon *B* s hodnotou 0,01. Důvodem je, že do výpočtu vstupují pouze lomové body polygonu, a ne body všech úhlů od 0° do 360°. Polygon *A* je také blízký kulatému tvaru s hodnotou 0,21 a nejméně kulatým polygonem je *D* (viz Tab. 3). I v tomto případě hodnota kulatosti stoupá, pokud polygon má menší kulatost.

Jako poslední se hodnotí poměr mezi obsahem polygonu a obsahem konvexní obálky polygonu. Pokud polygon neobsahuje žádné díry a všechny jeho lomové body jsou součástí konvexní obálky, tzn. leží na hranici konvexní obálky, je výslednou hodnotou 1. Tento

výsledek vyšel u polygonů *A* a *B*. Čím je hodnota nižší, tím je obsah původního polygonu menší a tvar není kompaktní ani kulatý (viz Tab. 3 a Obr. 34). Nejhuře na tom je polygon *D* s hodnotou 0,35.

Tab. 3: Výsledky hodnotící tvar polygonů A, B, C a D.

	<b>A</b>	<b>B</b>	<b>C</b>	<b>D</b>
<b>Kompaktnost</b>	1,04	1,55	3,10	5,77
<b>Kulatost</b>	0,21	0,01	2,41	7,48
<b>Poměr obsahu a obsahu konvexní obálky</b>	1,00	1,00	0,70	0,35

## 10.2 Hodnocení výstupů

Počet sektorů se bude měnit v závislosti na tom, kolik bariér se v oblasti bude nacházet. Použitím bariér byl zvýšen čas průběhu zpracování o cca 4 s. *Post-processing* provedl změnu pouze u výsledku z tvorby bariér. Kompaktnost se liší v desetinách a dopadla lépe při použití bariér. Důvodem může být kontrola tvaru hranice díky bariéře, protože cesty se příliš neklikatí, pokud to dovoluje terén. Poměr mezi obsahem a obsahem konvexní obálky vyšel téměř stejně. Liší se pouze o cca 0,02 (viz Tab. 4).

Tab. 4: Srovnání vypočtených hodnot pro tvorbu sektorů bez a s bariérami.

	<b>bez bariér</b>	<b>s bariérami</b>
<b>Původní počet regionů</b>	141	141
<b>Počet sektorů</b>	20	33
<b>Čas zpracování (s)</b>	5,67	9,52
<b>Počet sektorů po <i>post-processingu</i></b>	20	27
<b>Kompaktnost</b>	4,26	4,76
<b>Kulatost</b>	25,63	20,93
<b>Poměr obsahu a obsahu konvexní obálky</b>	0,74	0,76

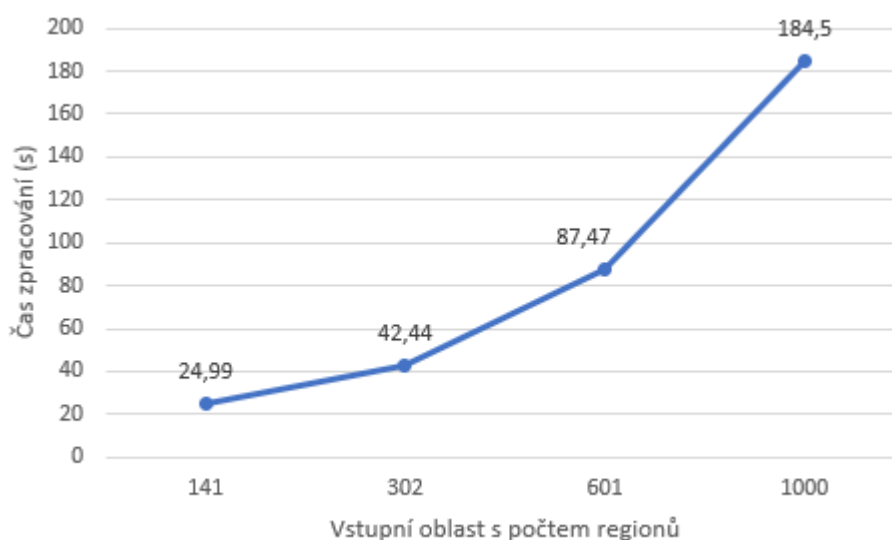
Pro srovnání, jak na tom budou vstupní data s jiným počtem prvků, byly vybrány cvičné oblasti s cca 300, 600 a 1000 prvky. Jak už bylo řečeno dříve, s vyšším počtem se zvyšuje časová náročnost, a proto by se měl čas zpracování u každé cvičné oblasti zvyšovat. Výstupy se budou srovnávat s cvičnou oblastí se 141 prvky. Ve všech případech se použije jenom tvorba sektorů s bariérami, protože zadavatel hlavně chce, aby se výstup tvořil s použitím této liniové vrstvy. Pro všechny výstupy se použije stejná vrstva s bariérami o 588 prvcích, což může měnit výsledky u polygonové vrstvy s 141 prvky.

Hodnoty poměru mezi obsahem a obsahem konvexní obálky jsou téměř stejné a nejvyšší rozdíl je cca 0,06. Čas zpracování se podle předpokladu zvyšoval a u nejvyššího původního počtu regionů dosáhl délky 3 min a 4 s. Kompaktnost byla rozdělena na hodnotu kolem 4,8 pro 141 a 302 regionů a kolem 6,3 pro 601 a 1000 regionů. Během *post-processingu* bylo odstraněno nejvíce malých sektorů u 601 a to kolem 20 % (viz Tab. 5).

Tab. 5: Srovnání hodnot pro 141, 302, 601 a 1000 regionů na vstupu.

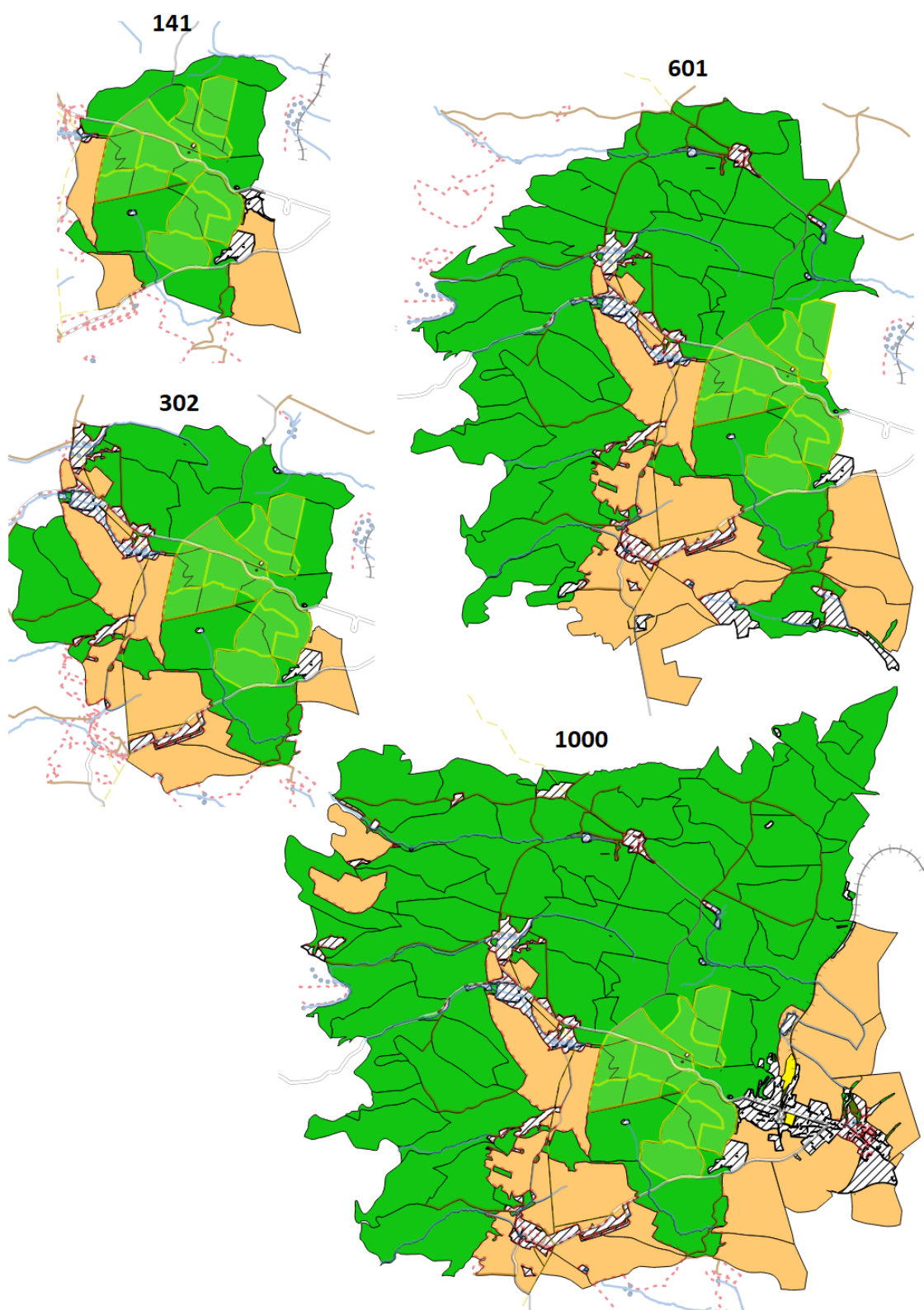
Původní počet regionů	141	302	601	1000
Počet sektorů	25	72	153	213
Čas zpracování (s)	24,99	42,44	87,47	184,50
Počet sektorů po <i>post-processingu</i>	23	63	123	177
Odstraněno sektorů při <i>post-processingu</i> (%)	8	13	20	17
Kompaktnost	4,81	4,84	6,05	6,60
Kulatost	25,54	22,42	30,37	30,35
Poměr obsahu a obsahu konvexní obálky	0,74	0,80	0,76	0,76

Čas má téměř exponenciální růst a jeho křivka by se prudčeji zvedala, kdyby pro každý vstup o jiném počtu regionů byla vytvořena vrstva bariér s různým počtem prvků (viz Graf 1).



Graf 1: Porovnání času zpracování pro všechny vstupy o daném počtu regionů.

Pro všechny vstupy o různých počtech regionů, byla použita oblast v okolí manuálně vytvořených sektorů. Ve všech případech se vyskytují intravilány, které nebyly zahrnuty do výpočtu a největší plochu zabírá les, i když v názvosloví typů je označen jako pastviny se stromy. Tvary sektorů jsou velmi různorodé a určitě by některé šly sloučit s jinými. Důvodem, proč nebyly sloučeny během *post-processingu* je, že je tam nastavena podmínka, že připojovaný sektor musí mít plochu menší než 5 % plochy sektoru, k němuž se bude připojovat. Zvýšením tohoto čísla by se zredukovalo více malých sektorů, ale narostlo by riziko, že se rapidně zvýší počet sektorů, jejichž plocha bude vyšší než maximální plocha v rámci celého programu. Neovlivní se nesloučení těch regionů, které mají kolem sebe regiony jiného typu (viz Obr. 35).



Obr. 35: Výstupy tvorby sektory pro o 141, 302, 601 a 1000 regionech jako vstup.

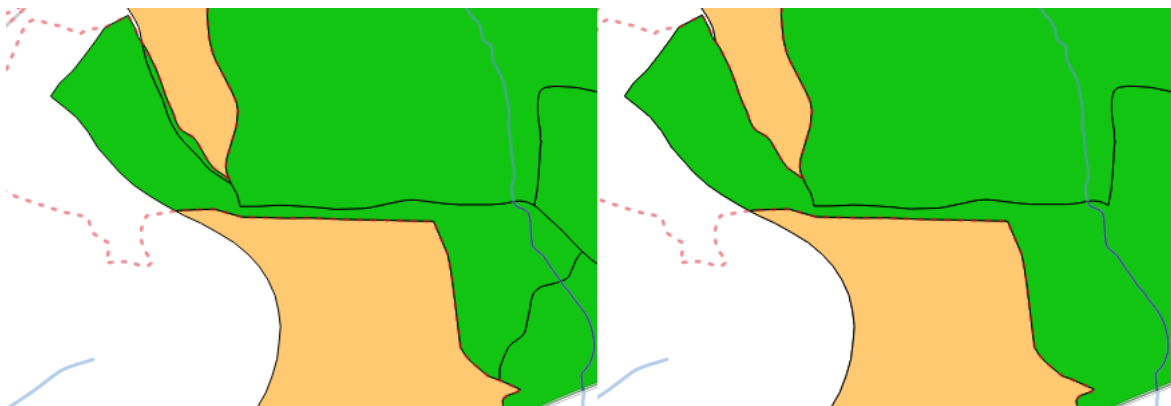


### 10.3 Omezení programu

Mezi první omezení patří **nerozdělování regionů**. Jejich velikost je větší než maximální zadaná plocha. Existují konvenční prostředky, jako například *Polygon Divider*, který rozdělí plochu na dva stejné sektory. Kde vzniká problém, že například region o velikosti 51 ha rozdělí na 25,5 ha a 25,5 ha. Jeho velikost bude cca 2krát menší, než je maximální plocha, se kterou se v rámci testování pracuje. Pak nastává otázka, zda region rozdělit na 50 ha a 1 ha a ten menší připojit k jinému. Tím by opět narůstal počet procházení seznamy uchováující regiony, popř. sektory a tím by opět narůstala časová náročnost. Další možnost je to zakomponovat do *post-processingu*.

Nevýhodou práce se seznamy je, že pokud je procházen, nelze do něj přidávat nebo měnit objekty, aby to mělo efekt na další zpracování. Prvotním cílem bylo procházet seznam *listOfRegions* a ihned v něm provádět změny. Místo toho se musely definovat další dva seznamy *removed* a *newSectors*, aby se více kontrolovalo procházení prvním seznamem a zbytečně se nezpracovávaly již použité regiony. Z tohoto důvodu nastala chyba, kdy se tvořilo spoustu malých sektorů o velikosti pod 1 ha, protože se kolem nich nacházely už použité regiony (tj. regiony nacházející se v *removed*). Pokud došlo k jejich spojení, přidaly se do *newSectors*. Jednodušší je to v případě regionů, kolem kterých se nacházejí použité regiony, a tak se hledají sousedé z *newSectors*. S tímto seznamem se nepracuje v cyklu. Je možné do něj ukládat, měnit a odstraňovat sektory i při běhu programu.

Některé regiony měly na vstupu zvláštní tvary a navíc byly obklopeny buď regiony jiného typu, regionem jehož velikost je vyšší než maximální plocha nebo na jeho hranici ležela bariéra. Příklad je na Obr. 36, kde se uprostřed nachází region, který má velmi dlouhou a úzkou střední část. Jednou z možností, jak se takového regionu zbavit, je rozdělit jej na tři části při přípravě dat, ale neeliminuje se jeho opětovné sloučení. Druhá možnost je opět *post-processing* a navrhnutí takové metody, která by sama rozpoznala, že se jedná o „špatný“ sektor a provedlo by se rozdělení a sloučení.

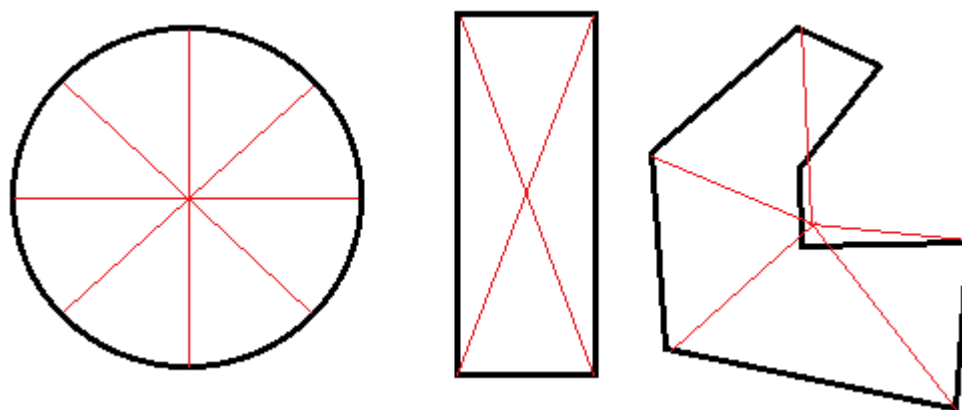


Obr. 36: Příklad zvláštního tvaru vstupního regionu. Vlevo vstupní region, vpravo vytvořený sektor.

Výpočet kulatosti by se mohl stát cenným pomocníkem, jak získat bližší informaci o charakteristice tvaru regionu. V aplikaci se hodnota vypočítá jako suma poměrů rozdílu



vzdálenosti mezi centroidem a lomovým bodem a průměrnou vzdáleností mezi centroidem a lomovými body. Hodnota blíží se k nule udává, že se tvar podobá buď kruhu, čtverci, obdélníku a může odhalit regiony ve tvaru písmene U (viz Obr. 37) Pozn.: neplést si kulatost s plošnou zakulaceností.



Obr. 37: Příklad tvarů pro výpočet kulatost. První a druhý s hodnotou 0, třetí s hodnotou větší než 0.

V terénu se může nacházet místo, kde dochází ke skokové změně výšky, například strž či prudký krátký svah. Pro vygenerování takové bariéry by stačilo mít k dispozici digitální model terénu a v místech prudších skoků výšek by se generovala linie a přidala by se do vrstvy bariér. Záleží na tom, jaká data má uživatel k dispozici a s čím vším je mu umožněno pracovat.

Největší omezení programu je časová náročnost. Samozřejmě platí, že čím výkonnější počítač, tím rychleji bude probíhat výpočet od načtení vrstev po vytvoření nové. Jedním řešením je rozdělit oblast na několik podoblastí, kde vzniká riziko hraničního efektu, protože se v rámci jiné podoblasti může vyskytovat vhodnější region pro sjednocení. Popřípadě by se každá podoblast mohla spustit na jenom procesoru například ze čtyř a tím by šlo o paralelní zpracování.

Problém hledání sousedních regionů, kdy se musí neustále dokola procházet seznamy, by mohla vyřešit topologie, kde by se předem u každého regionu určilo, se kterými regiony sousedí a vytvořila by se *hashmap*. Java takovou třídu obsahuje. Značně by se tím urychlilo vyhledávání sousedů.

Snížení časové náročnosti by se mohlo vyřešit také spouštěním aplikace jen jako aktualizace. První zpracování by se pustilo nad celou Českou republikou, a další pouze nad oblastí, kde došlo k nějakým změnám. Nepředpokládá se, že les za rok změní svou hranici nebo řeka své koryto.

## 11 ZÁVĚR

Přijít na vlastní řešení nebylo jednoduché a podmínky se během této práce několikrát změnilly. V první fázi logika fungovala tak, že se k regionu připojovali všichni sousedé ze seznamu najednou. Znamenalo to rychlejší zpracování, ale nedovolovalo to najít nejvhodnějšího souseda pro souseda právě zpracovávaného regionu. Proto se muselo slučování změnit na spojování pouze dvou regionů. Často se stávalo, že se pro region nenalezl ani jeden vhodný soused, protože se pro sousedy našel jiný vhodnější sousední region ke sloučení. Tento problém způsoboval nekonečnost rekurzivní metody a kompromisem bylo, že se stane vhodným sousedem alespoň jeden region.

Tvorba řešení postupovala od nejjednoduššího po nejsložitější. Během každé fáze se narazilo na nějaký problém, který šlo nebo nešlo vyřešit. Po domluvě s vedoucím se buď řešení našlo nebo se problém nechal být s tím, že se dodělá buď v rámci této práce nebo v rámci jiné. Rozsah práce je velký a vývoj aplikace se neustále potýkal se změnami nejenom v logice zpracování, ale také v podmínkách. Mezi změny patří například přepsání zdrojového kódu do přehlednější formy.

Zpočátku byl zdrojový kód velmi nepřehledný, a dokonce se nepracovalo s objekty jako *Region* a *Barrier*. Po získání nemalých zkušeností ať už v rámci diplomové práce nebo práce mimo vysokou školu, bylo rozhodnuto udělat zásadní změnu. Vytvořením objektů se zjednodušilo uchovávání informací, protože se uložily v konkrétním objektu a bylo možné jej kdykoliv volat. Dále se metody roztrídily podle funkčnosti. Nesložitější byla úprava z *List<SimpleFeature>* na *List<Region>* a *List<Barrier>*, protože většina metod pracovala se *SimpleFeature*. Tato úprava trvala cca týden týden.

Tvorbu sektorů je třeba dále zdokonalovat a nalézat další řešení, jak zrychlit běh programu a optimalizovat jej. Existuje velmi mnoho oblastí, které by se daly vylepšit, přidat nebo vymazat. Chtělo by to více testování na různém typu vstupních dat a vyřešit, jak si poradit s typem, pokud nebude podle zadané předlohy. Vstupní data nejsou nikdy stejná, ale čtení z konkrétního atributu je nastaveno tak, že stačí typy využití půd mít v atributu s názvem *TYP* a identifikátor v *ID*. Vytvoření nového sloupce například v QGIS není problém.

Na závěr této práce bych chtěla dodat, že mi tato práce dala minimálně to, jak se dívat na programování objektově, a ne jako na kus zdrojového kódu, který něco dělá. Vyzkoušela jsem si svoje schopnosti nejenom v psaní kódu, ale i v uvažování nad logikou řešení a představování, jak tvorba sektorů může probíhat. V duchu si představit, jak se dílky pomalu skládají a podle toho vytvořit algoritmus. Dopomohly mi k tomu i existující podobná řešení.

## SEZNAM LITERATURY

- [1] Algoritmus. *Wikipedie: Otevřená encyklopedie* [online]. Wikimedia Foundation, 2001, 20. 2. 2019 [cit. 2019-04-07]. Dostupné z: <https://cs.wikipedia.org/wiki/Algoritmus>
- [2] Convex Hull | Set 1 (Jarvis's Algorithm or Wrapping). *GeeksforGeeks* [online]. Uttar Pradesh [cit. 2019-03-31]. Dostupné z: <https://www.geeksforgeeks.org/convex-hull-set-1-jarviss-algorithm-or-wrapping/>
- [3] Convex Hull | Set 2 (Graham Scan). *GeeksforGeeks* [online]. Uttar Pradesh [cit. 2019-03-31]. Dostupné z: <https://www.geeksforgeeks.org/convex-hull-set-2-graham-scan/>
- [4] ČERBA, Otakar. *Generalizace*. Plzeň, 2011. Prezentace. Západočeská univerzita v Plzni.
- [5] Desktopový GIS: Využijte svá data. *ARCDATA PRAHA* [online]. Praha: ARCDATA PRAHA, 2019 [cit. 2019-04-06]. Dostupné z: <https://www.arcdata.cz/produkty/arcgis/desktopovy-gis>
- [6] Geodata. *Wikipedie: Otevřená encyklopedie* [online]. Wikimedia Foundation, 2001, 2018 [cit. 2019-03-30]. Dostupné z: <https://cs.wikipedia.org/wiki/Geodata>
- [7] *GeoTools: GeoTools The Open Source Java GIS Toolkit* [online]. Sphinx, 2011 [cit. 2019-04-10]. Dostupné z: <https://geotools.org>
- [8] HÁTLE, J., J. JAROŠ a J. LYSÁK. Vybrané aspekty znázorňování hranic na mapách. *Geografické rozhledy*. 2015, **24**(3), 12-13.
- [9] Heuristika. *Úvod do studia dějepisu*. Brno: Masarykova univerzita, 2014, s. 92. ISBN 978-80-210-7012-7.
- [10] Heuristika. *Wikipedie: Otevřená encyklopedie* [online]. Wikimedia Foundation, 2001, 2018 [cit. 2019-03-30]. Dostupné z: <https://cs.wikipedia.org/wiki/Heuristika>
- [11] Hranice. *Wikipedie: Otevřená encyklopedie* [online]. Wikimedia Foundation, 2001, 2018 [cit. 2019-03-30]. Dostupné z: <https://cs.wikipedia.org/wiki/Hranice>
- [12] CHALOUPKOVÁ, Helena, Ivona SVOBODOVÁ a Vladimír MAKEŠ. Využití vyspělých technologií a čichových schopností psů pro zvýšení efektivity vyhledávání pohřešovaných osob v terénu (PÁTRAC). Ministerstvo vnitra České republiky, 2017.
- [13] Intravilán. *Wikipedie: Otevřená encyklopedie* [online]. Wikimedia Foundation, 2001, 26. 12. 2015 [cit. 2019-04-07]. Dostupné z: <https://cs.wikipedia.org/wiki/Intravilán>
- [14] KYLIÁN, Jakub. *Tvarová analýza obrazových dat* [online]. Brno, 2017 [cit. 2019-03-31]. Dostupné z: <http://hdl.handle.net/11012/69393>. Bakalářská práce. Vysoké učení

technické v Brně. Fakulta elektrotechniky a komunikačních technologií. Ústav biomedicínského inženýrství. Vedoucí práce Jiří Chmelík.

- [15] PAVLIŠTA, Rostislav. *Řízení pátracích akcí po pohřešovaných osobách*. Ostrava, 2009. Dostupné také z: <https://dspace.vsb.cz/handle/10084/73741>. Diplomová práce. Vysoká škola báňská - Technická univerzita Ostrava. Vedoucí práce Kpt. Ing. Vladimír Makeš.
- [16] PEARL, Judea. *Heuristics: intelligent search strategies for computer problem solving*. 1984. Reading, Mass.: Addison-Wesley Pub. Co., c1984. ISBN 02-010-5594-5.
- [17] PETROVÁ, Kristýna. *Pátrání po osobách a věcech*. Brno, 2013. Bakalářská práce. Masarykova univerzita.
- [18] QGIS: The Leading Open Source Desktop GIS. *QGIS* [online]. Creative Common, 2019 [cit. 2019-04-06]. Dostupné z: <https://qgis.org/en/site/about/index.html>
- [19] Quickhull Algorithm for Convex Hull. *GeeksforGeeks* [online]. Uttar Pradesh [cit. 2019-03-31]. Dostupné z: <https://www.geeksforgeeks.org/quickhull-algorithm-convex-hull/>
- [20] RAPANT, Petr. *Základy geoinformatiky* [online]. Ostrava, 2014 [cit. 2019-03-30]. Dostupné z: [https://www.researchgate.net/profile/Petr\\_Rapant/publication/309618977\\_Zaklady\\_geoinformatiky/links/581a31d308ae30a2c01c9d04/Zaklady-geoinformatiky.pdf](https://www.researchgate.net/profile/Petr_Rapant/publication/309618977_Zaklady_geoinformatiky/links/581a31d308ae30a2c01c9d04/Zaklady-geoinformatiky.pdf). Skripta. Vysoká škola báňská - Technická univerzita Ostrava.
- [21] SERGEY, Ilya. Experience report: Growing and shrinking polygons for random testing of computational geometry algorithms. *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming - ICFP 2016*. New York, New York, USA: ACM Press, 2016, 2016, , 193-199. DOI: 10.1145/2951913.2951927. ISBN 9781450342193. Dostupné také z: <http://dl.acm.org/citation.cfm?doid=2951913.2951927>
- [22] SVOBODOVÁ, Jana. *Algoritmus pro automatizovanou kartografickou generalizaci shluků budov metodou agregace*. Praha, 2012. Diplomová práce. Univerzita Karlova v Praze. Vedoucí práce Ing. Tomáš Bayer, Ph.D.
- [23] ŠUBA, Radan. *On-the-fly generalizace nad daty katastru nemovitostí*. Plzeň, 2012. Diplomová práce. Západočeská univerzita v Plzni. Vedoucí práce Ing. Karel Janečka, Ph.D.
- [24] Záchrana pohřešovaných osob - pátrací akce v terénu. In: *Katalog typových činností integrovaného záchranného systému*. Česká republika, 2010, STČ 07/IZS. Dostupné z: <https://www.hzscr.cz/soubor/stc-zpat07-final-pdf.aspx>

- [25] ZEMAN, Jiří. *Návrh informačního portálu a informačního systému pro SAR (Search and Rescue) složky integrovaného záchranného systému*. Brno, 2009. Dostupné z: <https://is.muni.cz/th/unm11/>. Bakalářská práce. Masarykova univerzita, Filozofická fakulta. Vedoucí práce Ppor. Mgr. Petr Vodička.

## **SEZNAM ZKRATEK**

ESRI	Společnost zabývající se vývojem softwaru pro práci s GIS.
GIS	Geografický informační systém.
IZS	Integrovaný záchranný systém.
Java	Objektově orientovaný programovací jazyk.
KPT	kynologický pátrací tým
PÁTRACĚ	Využití vyspělých technologií a čichových schopností psů pro zvýšení efektivity vyhledávání pohřešovaných osob v terénu.

## SEZNAM OBRÁZKŮ

Obr. 1: Základní geometrické prvky. ....	9
Obr. 2: Metoda výběru prvků. ....	10
Obr. 3: Zjednodušení linií a polygonů. ....	11
Obr. 4: Prostorová redukce vodního toku. ....	11
Obr. 5: Ukázka Gift Wrapping algoritmu. [2] ....	12
Obr. 6: Vlevo: původní polygon (černý), připojovaný polygon (červený), segment (zelený); vpravo: výsledný polygon po spojení (modrý). ....	13
Obr. 7: Příklad výstupu typů využití půdy vyskytujících se v oblasti zájmu. ....	17
Obr. 8: Příklad nového sektoru ohraničeného bariérami ze všech stran. (Žlutá barva označuje vybraný prvek pro lepší přehlednost.) ....	18
Obr. 9: Mapa typů využití půd vyskytujících se ve cvičné oblasti určené pro zpracování. ....	19
Obr. 10: Mapa bariér vyskytujících se ve cvičné oblasti. ....	20
Obr. 11: Legenda typů ploch ve cvičných datech. ....	21
Obr. 12: Mapový výstřižek městské oblasti. ....	22
Obr. 13: Příklad eliminace přebytečných polygonů ve zkoumané oblasti. Nahoře – původní data, dole – data po eliminaci. ....	23
Obr. 14: Problém spojování oblastí. ....	24
Obr. 15: Ukázka výstupu z manuální tvorby sektorů. ....	25
Obr. 16: Vlevo nahoře – původní polygonová vrstva, vpravo nahoře – upravená vrstva bez využití bariér, dole – upravená vrstva s využitím bariér. ....	26
Obr. 17: Vyloučení lesů z výpočtu. ....	27
Obr. 18: Výstup funkce Dissolve. Vlevo původní cvičná oblast, vpravo výstup metody. ....	27
Obr. 19: Výstup metody Eliminate Polygon Parts. Vlevo původní cvičná oblast, vpravo výsledek. ....	28
Obr. 20: Rozdělení velkého sektoru. ....	29
Obr. 21: Zleva: původní cvičná oblast, rozpuštěná oblast, rozdělená oblast pomocí vrstvy s bariérami. ....	29
Obr. 22: Eliminace vybraných polygonů do 50 ha. ....	30
Obr. 23: Příklad vybraní polygonů pro sjednocení tak, aby jejich soused nebyl jiného typu a nebyla mezi nimi bariéra. ....	31

Obr. 24: Červený symbol kolečka označuje řádek, kde bude běh programu během debugování zastaven. ....	32
Obr. 25: Algoritmus popisující tvorbu regionů. ....	35
Obr. 26: Rekurzivní algoritmus <i>provedSloucení()</i> pro vytvoření jednoho sektoru z právě zpracovávaného regionu a seznamu sousedních regionů. ....	36
Obr. 27: Algoritmus <i>vratSeznamProSjednocení()</i> s výstupem seznam s regiony určených pro sjednocení. ....	37
Obr. 28: Algoritmus pro post-processing nad vrstvou s vytvořenými sektory. ....	38
Obr. 29: Struktura projektu. ....	39
Obr. 30: Bariéra nevede přes společnou hranici modrého a zeleného regionu, ale buffer se jí dotýká. ....	46
Obr. 31: Výstup tvorby sektorů bez použití bariér. ....	49
Obr. 32: Výstup tvorby sektorů s použitím bariér. ....	50
Obr. 33: Ukázka, kde došlo ke sloučení menších sektorů. ....	51
Obr. 34: Manuálně vytvořené polygony různých tvarů. Červený křížek označuje centroid (tj. těžiště) polygonu. ....	52
Obr. 35: Výstupy tvorby sektory pro o 141, 302, 601 a 1000 regionech jako vstup. ....	55
Obr. 36: Příklad zvláštního tvaru vstupního regionu. Vlevo vstupní region, vpravo vytvořený sektor. ....	56
Obr. 37: Příklad tvarů pro výpočet kulatost. První a druhý s hodnotou 0, třetí s hodnotou větší než 0. ....	57



## SEZNAM TABULEK

Tab. 1: Zjištěné hodnoty pro výstup zpracování z cvičné oblasti bez použitých bariér. ....	49
Tab. 2: Zjištěné hodnoty pro výstup zpracování z cvičné oblasti s použitím bariér.....	51
Tab. 3: Výsledky hodnotící tvar polygonů A, B, C a D. ....	53
Tab. 4: Srovnání vypočtených hodnot pro tvorbu sektorů bez a s bariérami. ....	53
Tab. 5: Srovnání hodnot pro 141, 302, 601 a 1000 regionů na vstupu.....	54

## **SEZNAM GRAFŮ**

Graf 1: Porovnání času zpracování pro všechny vstupy o daném počtu regionů. ....	54
--	----

## **SEZNAM PŘÍLOH NA CD**

Příloha č. 1 – vypracovaná diplomová práce: dp\_sla0193\_2019.pdf

Příloha č. 2 – IntelliJ IDEA projekt v programovacím jazyce Java: program.zip

## PŘÍLOHA 1: INTRAVILÁN

AZAMEK;Areál zámku  
BAZMOC;Bažiny, mokřady  
BUBLBU;Blok budov  
VANAZA;Budovy na nádraží  
CAMPIN;Camping  
CERPST;Čerpací stanice pohonných hmot  
CISTVO;Čistírna odpadních vod  
ZDSOZA;Další zdravotní a sociální zařízení  
DREVPR;Dřevozpracující a papírenský průmysl  
ELEKTR;Elektrárna  
GOLFAR;Golfový areál  
HRBITO;Hřbitov  
HRISTE;Hřiště  
CHATKO;Chatová kolonie  
CHMELN;Chmelnice  
CHOVZV;Chov hospodářských zvířat  
KLASTE;Klášter  
KOLEJI;Kolejiště  
KOUPAL;Koupaliště  
LETSCE;Letní scéna  
MUZEUM;Muzeum  
ROZTRA;Neidentifikovaný povrch  
OSPLSI;Neznámá plocha  
ODPOCI;Odpočívadlo  
OSTPRU;Ostatní, nerozlišený průmysl  
AREZAS;Plocha u nádraží  
POTPRU;Potravinářský průmysl  
POTELO;Povrchový důl  
STAVHM;Průmysl skla, keramiky a staveb. hmot  
REKZAS;Rekreační zástavba  
ROZZRI;Rozvalina, zřícenina  
SADZAH;Sad, zahrada  
SKLHAN;Sklad, hangár  
SKLADK;Skládka  
SKLENI;Skleníkové pěstování plodin  
SKUPGA;Skupinové garáže  
SPORTA;Sportovní areál  
STRPRU;Strojírenský průmysl  
STRELN;Střelnice  
SKOLAA;Škola  
TECHSL;Technické služby  
TRTRPO;Travnatý povrch  
UPRVOD;Úpravna vody  
USNAOD;Usazovací nádrž  
VEZENI;Věznice  
VODPLO;Vodní plochy  
VODZEM;Vodojem zemní  
ZAHPAR;Zahrady, parky  
ZEMARE;Zemědělský areál ostatní