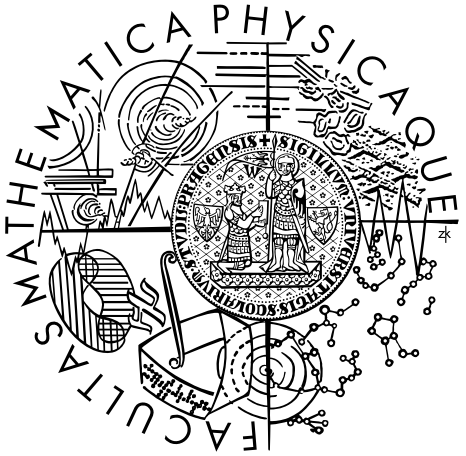
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Karolína Burešová

Umisťování mapových značek

Katedra aplikované matematiky

Vedoucí bakalářské práce: Mgr. Martin Mareš, Ph.D.
Studijní program: Informatika
Studijní obor: Správa počítačových systémů

Praha 2015

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ..... Podpis autora

Název práce: Umisťování mapových značek

Autor: Karolína Burešová

Katedra: Katedra aplikované matematiky

Vedoucí bakalářské práce: Mgr. Martin Mareš, Ph.D., Katedra aplikované matematiky

Abstrakt: Při tvorbě map představuje velký problém umisťování mapových značek tak, aby výsledná mapa vypadala dobře. V této práci se zabýváme automatizací tohoto procesu pomocí evolučního algoritmu. Vstupem algoritmu jsou požadavky na značky (rozcestníky, zříceniny, jména ulic, ...), které mají být rozmístěné na mapě, výstupem popis jejich umístění. Naším hlavním cílem je možnost vytvářet vizuálně hezké papírové mapy – zejména čitelné a přehledné. Na rozdíl od jiných prací se zabýváme popisky všech typů symbolů (nejen bodových) i umisťováním symbolů samotných. Dokázali jsme navrhnout evoluční algoritmus, který generuje přijatelné mapy a skýtá prostor pro další vylepšování získaných výsledků.

Klíčová slova: mapové značky, popisování map, kvalita mapy, evoluční algoritmy

Title: Placement of map symbols

Author: Karolína Burešová

Department: Department of Applied Mathematics

Supervisor: Mgr. Martin Mareš, Ph.D., Department of Applied Mathematics

Abstract: Placing map symbols in a way so that the resulting map looks well is a major problem in cartography. In this thesis, we deal with automatization of this process using an evolutionary algorithm. Input of this algorithm is a set of requests for symbols (signposts, ruins, street names etc) to be placed on the map, its output is a description of their placement. Unlike other studies, we deal with labels for all kind of features (not only point-features) as well as with placing the features themselves. We managed to design an evolutionary algorithm which produces acceptable maps and offers some possibilities to further enhance the quality of produced maps.

Keywords: map symbols, map labelling, map quality, evolutionary algorithms

Poděkování

V první řadě bych chtěla poděkovat své rodině. Svým rodičům za to, že si v mém dětství nacházeli čas na skládání z lega, obdivování špatně nacvičených kouzelnických vystoupení, poslouchání zážitků ze školy a vůbec všechny ty tehdy životně důležité činnosti, zatímco později mi věřili, i když jsem se vrhala do bláznivých akcí, stáli při mě, i když jsem se spálila, a smiřovali se s mým postupným roztahováním křídel, ač mě nikdy nepřestali vítat doma. Svému bráchovi děkuju za spoustu těch zážitků, které by se v jednom prostě nedaly nasbírat, a za to, že navzdory sourozeneckým válkám a hádkám byl mým parťákem vždycky, když jsem to potřebovala. Bez podpory své rodiny bych dnes těžko byla tam, kde jsem, a těžko bych byla tím, kým jsem.

Hned vzápětí bych chtěla poděkovat svému milému. Za ty roky, které už se mnou – většinou v dobrém, někdy v horším – prožil, za příležitosti poznávat jeho svět i za ochotu zkoumat ten můj, a zejména za to, že mě má rád a je mi podporou i ve chvíli, kdy se mnou nesouhlasí.

Trojici „hlavních“ poděkování uzavírá poděkování mému vedoucímu. Děkuji mu za všechny profesionální i lidské rady, a že jich za dobu mého studia bylo požehnaně. Děkuji mu za čas, kterého má sám málo, a přesto z něj pro mé projekty ukrajoval velký kus, za všechna povzbuzení a pomoc, když se práce zrovna nedařila, i za mírná vrčení, když jsem na něco šla hodně špatně.

Chtěla bych navíc přidat také poděkování všem svým kolegům a kamarádům z KSPčka, z Korespondenčního semináře z programování. Pojí nás spousta zážitků, pojí nás předávané vědomosti, pojí nás projekt, který snad má smysl, díky kterému tu po nás opravdu něco zůstává. A to je neuvěřitelně krásná a cenná zkušenost, za kterou všem, kteří se mnou drží (nebo někdy drželi) KSP v chodu, děkuji.

A přestože už teď možná poděkování působí, spíš jako bych psala životní dílo než „obyčejnou bakalářku“, přihodím ještě jedno. Děkuju Matě, která, ač to možná netuší a možná by mi to nevěřila, se mi během pouhých pár dní stačila stát velkou motivací a inspirací.

Obsah

Úvod

V této práci se budeme zabývat jednou z částí tvorby map, a to umisťováním mapových značek. Pod pojmem *mapová značka* si lze představit většinu z toho, co nás na mapě obvykle zajímá – ikony označující kostely, rozcestníky, zříceniny, ..., popisky udávající názvy ulic, pohoří, rozcestníků, ..., vrstevnice, hranice různých území, koryta řek atd.

Vhodné umístění značek, ač se to možná nezdá, představuje velmi složitý úkol. Problém spočívá zejména v kombinaci dvou věcí: značek, které musíme umístit, je mnoho; a prostor, do kterého tyto značky umisťujeme, je velmi omezený. I při takto omezeném prostoru je navíc žádoucí zamezit překryvům jednotlivých značek či jejich umístění příliš daleko od skutečné polohy objektu, který označují.

Malá změna v umístění značky často vynutí větší změny v jejím okolí, a proto obvykle trvá vytvoření výsledné mapy dlouhou dobu, případně se v zájmu ušetření času přijímají mapy navzdory reálnému předpokladu, že by ještě šly vylepšit. To vše dává dobrý důvod a zároveň potřebu celý tento proces automatizovat, umožnit strojové vytváření kvalitních map.

Přestože dnes již existují metody, které produkují mapy přijatelné, nebyl tento problém dosud uspokojivě vyřešen a po opravdu funkčním přístupu zatím počítačová kartografie stále pátrá.

Naše práce se snaží problém řešit zejména z hlediska přípravy kvalitních papírových map, které se nějakou (třeba i delší) dobu připravují, následně se vytisknou a pak teprve se v klasickém slova smyslu používají.

Orientace na papírové mapy pro nás znamená zejména tři následující vlastnosti: menší důraz na rychlé zpracování, neměnná zpracovávaná oblast (a zpracovávané značky), žádoucí podpora vysokého rozlišení.

Zatímco např. u online map by bylo naprosto nepřijatelné umisťovat značky několik vteřin, natož minut, při přípravě papírových map může být taková doba zpracování přiměřenou cenou za vyšší kvalitu výsledné mapy. To pro nás může být důležité právě proto, že některé postupy obecně při delším zpracování dosahují lepších výsledků. Jelikož můžeme dobu zpracování podřídit výsledné kvalitě, můžeme také využít časově náročné postupy.

Od začátku zpracování je k dispozici úplná informace o zpracovávané oblasti a v ní obsažených značkách, které se mají umístit. To umožňuje drobnými úpravami v jedné části mapy výrazně ovlivňovat umístění značek v jiné části mapy (s lehkou nadsázkou: tím, že o trochu posuneme popisek Brna, můžeme umožnit lepší umístění popisků pražských ulic). Zároveň to umožňuje nezavádět žádná pevná umisťovací pravidla, která by bránila kolizím při pozdějším přidání značek (např. nemusíme jméno města zásadně umisťovat vpravo – je-li vlevo od města jen prázdný prostor, nic nového se tam neobjeví, takže můžeme popisek dát tam).

Při uvažování kvalitních map je vhodné předpokládat jejich vyšší rozlišení a myslet na něj při návrhu algoritmu – algoritmus by ani v takovém případě neměl potřebovat příliš mnoho místa, stejně tak by se neměla dramaticky zhoršit jeho doba běhu.

Problém umisťování mapových značek pro takové mapy se v této práci pokusíme vyřešit novou metodou, resp. novým využitím existující metody, a to evolučním algoritmem.

Evoluční algoritmy jsou nedeklarativní metodou programování. Inspirovaly se v přírodě, ve vývoji života a přizpůsobování organismů zejména na principu „nejsilnější přežívá“. Dnes se s úspěchem využívají pro mnoho optimalizačních problémů, které neumíme jinak vyřešit.

V této práci si přiblížíme základní principy evolučních algoritmů a ukážeme, proč by takový přístup měl být pro náš problém funkční a vhodný. Následně se pokusíme navrhnout konkrétní evoluční algoritmus, který bude umisťování mapových značek dobře řešit. Zamyslíme se nad reprezentací problému, jeho řešení a nad jednotlivými evolučními operátory, a z těchto dílků následně seskládáme algoritmus.

Navržený algoritmus také naimplementujeme. Předvedenou implementaci následně otestujeme na reálných datech z projektu OpenStreetMap (OSM) a zhodnotíme dosažené výsledky.

Struktura práce

První kapitola poskytuje nutný kartografický základ. Připomíná podstatu map, jejich skladbu a podrobněji popisuje jednotlivé mapové značky. Dále diskutuje kritéria kvality mapy a také ukazuje některé způsoby, jak prohřešky vůči kvalitě mapy řešit.

Ve druhé kapitole zkoumáme některé existující nástroje a přístupy a následně zdůvodňujeme, proč budeme pro náš problém chtít použít evoluční algoritmus.

Třetí kapitola proto poskytuje nutný základ evolučních algoritmů. Vysvětluje princip evolučních algoritmů a podmínky, které by problém měl splňovat, aby byl evolučními algoritmy dobře řešitelný. Definuje pojmy používané při návrhu a popisu evolučního algoritmu.

Čtvrtá kapitola popisuje navržený algoritmus včetně jednotlivých rozhodnutí, která musíme při návrhu udělat, a argumentů pro tato rozhodnutí (resp. proti nim).

Pátá kapitola slouží jako vývojová dokumentace. Popisuje rozhodnutí učiněná při implementaci algoritmu, používané datové struktury a další důležité informace pro čtení či úpravu kódu. Její součástí je také krátká uživatelská dokumentace poskytující návod k instalaci a spuštění programu.

Šestá kapitola shrnuje provedené experimenty, uvádí číselné hodnoty, ukázky z výsledných map i slovní komentáře k získaným výsledkům.

V závěru shrnujeme dosažené výsledky a nabízíme také několik nápadů, jak algoritmus dále vylepšit.

1. Kartografický základ

V této kapitole uvedeme některé kartografické základy. Rozebereme jednotlivé druhy mapových značek a zamyslíme se nad kritérii kvality map. Některá z těchto kritérií pojmenujeme a uvedeme si příklady, jak skutečně kvalitu mapy ovlivňují.

Začněme tím, co je vůbec mapa. Spíše pro zajímavost uveďme dvě běžně používané formální definice mapy:

Mapa je zmenšený generalizovaný konvenční obraz Země, nebeských těles, kosmu či jejich částí, převedený do roviny pomocí matematicky definovaných vztahů (kartografickým zobrazením), ukazující podle zvolených hledisek polohu, stav a vztahy přírodních, socioekonomických a technických objektů a jevů. (ČSN 730402)

Mapa je zmenšené zevšeobecněné zobrazení povrchu Země, ostatních nebeských těles nebo nebeské sféry, sestrojené podle matematického zákona na rovině a vyjadřující pomocí smluvených znaků rozmístění a vlastnosti objektů vázaných na jmenované povrchy. (Mezinárodní kartografická asociace – ICA)

Pro potřeby této práce si definici mapy značně zjednodušíme. Předně, budeme uvažovat pouze mapu zemského povrchu, nikoliv mapu noční oblohy či vesmíru. Také budeme předpokládat klasickou geografickou či turistickou mapu, tedy mapu sloužící k získání představy o krajině a orientaci v ní. Nebudeme uvažovat např. mapy vyjadřující míru nezaměstnanosti či poměr hospodářského využití půdy. Je ovšem na místě podotknout, že tento předpoklad nám spíš usnadní uvažování o mapách, než že by výrazně ovlivnil výsledný algoritmus.

Jako mapu tedy budeme chápat podle matematických pravidel zjednodušený obraz zemského povrchu, který pomocí mapových značek vyjadřuje polohu (a případně vztahy) objektů na tomto povrchu. Zcela neformálně je mapa obraz, pomocí kterého se můžeme pohybovat někde v terénu a neztratit se.

Později podrobněji rozebereme kritéria kvality mapy, ale obecně budeme chtít, aby taková mapa byla užitečná – tedy zejména čitelná a pochopitelná.

Je jasné, že kvalitní mapa bude vyžadovat určité kompromisy. Čím více objektů, resp. jejich vlastností, budeme chtít na mapě zachytit, tím zahuštěnější bude muset mapa být, a tím méně bude čitelná. Zároveň čím méně objektů na mapě zachytíme, tím častěji v ní nenajdeme objekt ze skutečného světa a tím těžší bude se podle takové mapy orientovat.

1.1 Mapové značky

Mapové značky jsou symboly, které na mapě zastupují skutečné objekty, případně jejich vlastnosti. V kartografii tradiční, a zejména počítačové se běžně odlišují tři typy značek, resp. zastupovaných objektů, a to podle svého rozměru (dimenze):

- *body* (0-dimenzionální) – podle měřítka mapy zastupují např. památníky, kostely, parky, města ...
- *linie* (1-dimenzionální) – zastupují např. řeky, silnice, linky MHD, ...
- *oblasti* (2-dimenzionální) – zastupují např. budovy, lesy, státy, ...

My budeme kromě tohoto tradičního rozdělení uvažovat jako speciální typ značky také *popisek*, tedy textový popis typicky vázaný k jiné značce. Jako takovéto popisky se na mapách obvykle uvádí názvy, ať už ulic, restaurací, řek, pohoří, ... Na ostatní mapové značky, tedy na body, linie a oblasti, se občas budeme souhrnně odkazovat jako na *symboly*.

1.2 Kritéria kvality mapy

Kritérií kvality mapy obecně existuje mnoho a jednotlivá z nich spolu často splývají. Důležitost jednotlivých kritérií také do značné míry souvisí s účelem mapy, který v našem případě neznáme. Přesto stanovíme několik kritérií, podle kterých budeme kvalitu map hodnotit – ideálně je zohledníme při navrhování programu, případně se nad nimi alespoň zamyslíme při hodnocení výsledných map.

Velmi často se používají tři jednoduchá kritéria vycházející z prací Imhofa [10] a Yoeliho [19]. My těchto kritérií stanovíme více, zejména podle van Dijka a ostatních [7] a Čerby [5].

Po stanovení kritérií také navrhneme několik možností, jak případné prohřešky vůči stanoveným kritériím řešit. Z těchto možností opět můžeme vyjít při navrhování konkrétní podoby algoritmu.

- Srozumitelnost

  Mapa by měla být svému uživateli jednoduše srozumitelná. Uživatel by měl být schopný mapu pochopit bez dlouhého studování způsobu, jakým je tvořena či jakým uživateli předává informace.

  Základem srozumitelnosti je používání běžných značek. Kupříkladu symboly zastupující rozcestník, zříceninu nebo kapli mají běžně známý a očekávaný tvar. Tyto obvyklé tvary tedy můžeme použít a předpokládat, že uživatel bude mapě rozumět. Naopak bychom pro tyto objekty neměli zavádět nové symboly – jednak nemůžeme očekávat, že by je náhodný uživatel znal, jednak můžeme běžné uživatele mapy zmást. Při zkoumání mapy totiž nebudou očekávat, že by nové symboly označovaly něco, co má symbol léty zažitý, a skutečný význam se jim může o to složitěji hledat.

  Jakmile používáme značky, které nejsou obecně rozšířené ani intuitivně zcela jasné, musíme mapu doplnit o legendu, kde význam těchto značek vysvětlíme.

- Čitelnost

  Toto kritérium velmi úzce souvisí s celkovou srozumitelností mapy. Veškeré popisky, které se na mapě vyskytují, by měly být čitelné. Měl by být zvolený vhodný font, vhodná velikost písma, písmo by mělo být v porovnání s podkladem dostatečně kontrastní.

- Vyloučení překryvů

  Mapové značky by se neměly překrývat. Překryvy komplikují čtení mapy, v extrémním případě ho mohou zcela znemožnit. Při větších překryvech nemusí být některé značky vůbec vidět, částečné překryvy mohou také způsobit nečitelnost některých překrývajících se značek, zejména při překryvech

Obrázek 1.1: Různé překryvy symbolů

Obrázek 1.2: Překryvy popisků

popisků. Různou míru překryvu symbolů znázorňuje obrázek 1.1, problém překryvu popisků demonstruje obrázek 1.2.

K problémům s překryvy značek může jednoduše dojít v případě, kdy se na jednom fyzickém místě nachází více objektů. Na turisticky zajímavých místech tak např. může být zároveň zřícenina, restaurace, rozcestník, pěkný výhled a parkoviště. Zobrazit všechny značky na daném místě bez překryvu pak není možné.

Překryvy na mapě ale mohou nastat i v situaci, kdy jsou reprezentované objekty fyzicky dostatečně vzdálené, to kvůli zmenšení mapy a nepoměru mezi velikostí skutečného objektu a mapové značky, která ho reprezentuje.

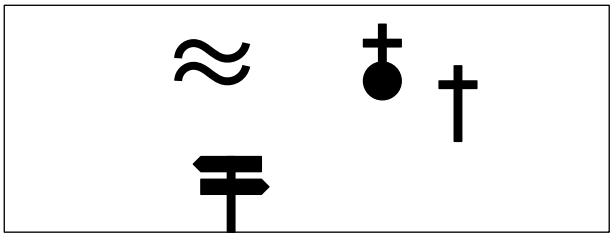
- Posunutí vůči realitě

  Mapa je kvůli svému zmenšení vždy trochu zkreslená, přesto by měla objekty zobrazovat tam, kde ve skutečnosti jsou. Posunout mapovou značku jinam, než kam z tohoto hlediska patří, může být žádoucí např. k zamezení vzniku zmíněných překryvů, přesto budeme takový posun považovat za prohřešek.

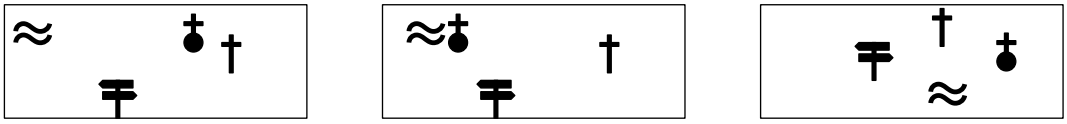
  Pokud už se značky na mapě vůči skutečné pozici posouvají, je žádoucí, aby zůstaly co nejlépe zachovány jejich vzájemné vazby. Představme si, že by věrná mapa vypadala jako na obrázku 1.3.

  Nyní předpokládejme, že věrnou mapu z nějakého důvodu sestrojit nešlo a bylo třeba značky posunout. Obrázek 1.4 ukazuje tři různě dobré, resp. různě špatné možné výsledky posunů.

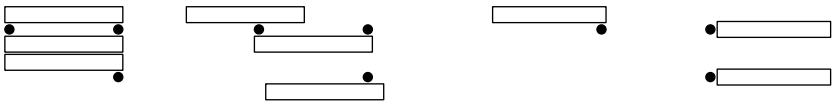
  Vlevo vidíme mapu, podle které bychom se nejspíš dokázali bezpečně zorientovat. Mapa uprostřed bude mnohem více matoucí, zatímco orientace podle mapy vpravo by nejspíš vůbec nebyla možná.



Obrázek 1.3: Referenční mapa pro demonstraci posunů



Obrázek 1.4: Různá posunutí značek na mapě



Obrázek 1.5: Různě jednoznačné umístění popisků

Podobně u různých značek může být posunutí různě závažným problémem. Pokud se, byť v centru města, posune značka kostela, pravděpodobně to bude menší problém, než když se v zástavbě posune značka restaurace či lékárny.

Složitě bychom pak mohli debatovat o posouvání a neposouvání značek v případě souběhu linií, např. pokud nějakým úsekem prochází vícero turistických tras.

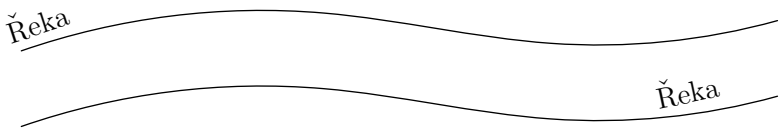
- Jednoznačnost popisků

  U popisků chceme, aby bylo jednoznačně určitelné, ke které značce patří. Různou míru jednoznačnosti reprezentuje obrázek 1.5:

  V levé variantě není možné jednoznačně určit, ke kterému bodu patří který popisek. V prostředním případě můžeme správné přiřazení předpokládat, přestože jiné umístění popisků by v tomto případě fungovalo lépe. V poslední, pravé, variantě jsou popisky umístěny jednoznačně.

  Jednoznačnost přiřazení popisků je velmi důležitá pro orientaci a případnou navigaci podle mapy – není-li podle mapy možné identifikovat daný objekt, není možné tento objekt zahrnout do své znalosti či plánování a mapa tak neslouží svému účelu. Práce s mapou, při které uživatel nemusí příliš přemýšlet o přiřazení popisků k ostatním značkám, je navíc pro uživatele mnohem příjemnější.

  Dalším rozměrem jednoznačnosti popisků je rychlost jejich nalezení na mapě. Při jednoznačném umístění popisků to není problém u bodových značek, ale může to být problém u oblastí, a zejména u linií. Ty se mohou, např. označují-li řeky či turistické trasy, táhnout velkou částí mapy, třeba se i různě klikatit. Najít k nim popisek pak znamená sledovat je na mapě až k umístění popisku a následně opět hledat původní místo. Situaci si můžeme lépe představit s využitím obrázku 1.6 ukazujícího různá umístění popisku k liniové značce.



Obrázek 1.6: Umístění popisku liniové značky

Obrázek 1.7: Překrývající se popisky řešitelné hladovou strategií

- Výběr značek

  Mapa nemůže obsahovat informace o všem, a proto je důležité správně vybrat, které značky by se na ní měly objevit a které ne. Tento výběr je ještě víc než jiná kritéria ovlivněný účelem mapy, přesto by měl být prováděn i s ohledem na ostatní kritéria.

  Značky, které se na mapě skutečně objeví, by měly odpovídat těm nejdůležitějším objektům, a zároveň by měly být co nejvíce zastoupeny objekty v nějakém smyslu unikátní či osamocené. Bez ohledu na přesný účel mapy může být např. žádoucí vynechat značku rozcestníku u zříceniny, ale už ne značku rozcestníku uprostřed lesa.

- Estetičnost

  Mapa by měla být estetická, pohled na ni by měl být příjemný.

Nyní navrhneme také některé způsoby řešení prohřešků a zlepšování kvality mapy.

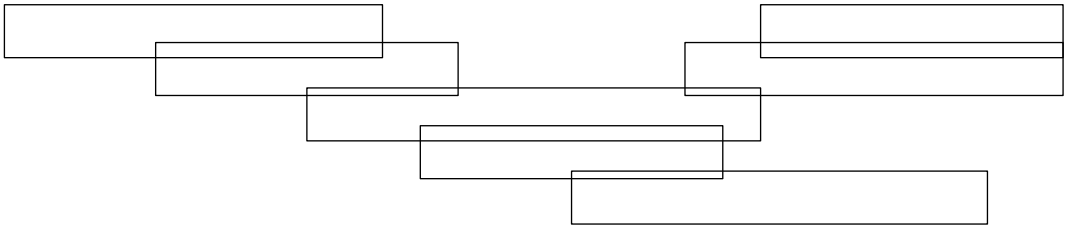
- Vynechání značek

  Nejjednoduším způsobem, jak řešit překryvy značek, je vynechávání některých z překrývajících se značek. Tímto způsobem je možné se překryvům zcela vyhnout.

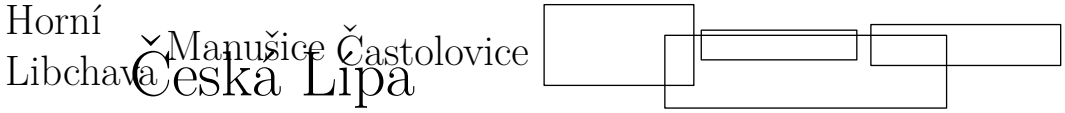
  Při vynechání značek ale pochopitelně dochází ke ztrátě informace. Je proto nutné vhodně volit vynechané značky. V některých případech může být dostatečnou strategií jednoduše odstraňovat ty značky, které mají nejvíc překryvů, dokud se všechny překryvy neodstraní (tedy použití hladové strategie). Takový postup by fungoval např. na obrázku 1.7, kde jsou zobrazené popisky (bez příslušných symbolů) čtvrti a několika obcí v okrese Česká Lípa, a pro lepší představu o problému také jejich ohraničující boxy.

  Často ale bude takováto hladová strategie suboptimální (alespoň z hlediska počtu vynechaných značek). Příkladem budiž obrázek 1.8 s „nějakými“ popisky (uveďme jen jejich ohraničující bosy). Hladově bychom mohli chtít odstranit prostřední popisek (má 3 překryvy), tím by ale zbyly tři překrývající se dvojice, tedy bychom museli odstranit ještě další 3 popisky. Pod naopak odstraníme všechny popisky, které se překrývají se středovým (každý z nich má 2 překryvy), odstraníme jen tyto tři popisky a všechny další můžeme ponechat.

  I v případě, že je vynechání značky s nejvíce překryvy optimální z hlediska počtu vynechaných značek, může být problematické z jiných hledisek. Například v situaci, kdy by samotné popisky vypadaly jako na obrázku 1.9, sice odstranění popisku České Lípy vyřeší překryvy, ale popisek okresního města je při obvyklém použití mapy důležitější než popisky, byť několika, vesnic v jeho okrese.



Obrázek 1.8: Protipříklad na hladovou strategii skrývání značek



Obrázek 1.9: Problematické překryvy popisků obcí

- Změna rozměrů značky

  Jedním ze způsobů, jak podpořit jednoznačnost popisků, může být změna rozměrů značky. Obrázek 1.10 ukazuje, jak je možné změnou velikosti symbolu a příslušného popisku zlepšit nejednoznačné přiřazení popisků k symbolům. Podobného efektu by pochopitelně šlo dosáhnout i zmenšením symbolu a jeho popisku.

  Zmenšení rozměrů značky může také vyřešit či zmírnit problém překryvu.

- Posunutí značky

  Ke způsobům, jak se vypořádat s překryvy či nejednoznačností popisků, patří posunutí značky. Jak jsme diskutovali už při stanovení kritérií, v takovém případě je potřeba myslet na to, že mapa má zobrazovat reálnou krajinu. Posouvaná značka by se neměla příliš vzdálit svému skutečnému umístění, ani by se neměla ocitnout ve výrazně nerealistické pozici vůči ostatním značkám ve svém okolí.

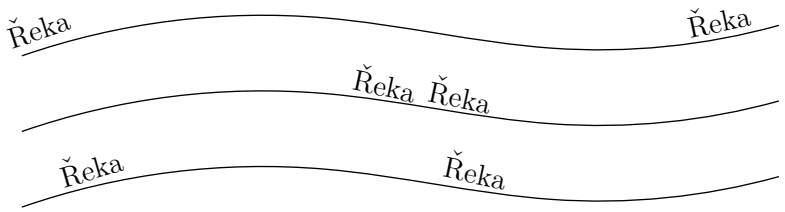
- Opakování popisků u liniových značek

  V tradiční kartografii je běžně využívaným způsobem, jak zlepšit dohledatelnost popisků u liniových značek, opakování popisků takových značek. Jedna linie je popsaná vícekrát, v různých místech. Takové opakování může být velmi efektivní, ovšem je nutné vhodně zvolit místa, kam se umístí jednotlivé popisky. Různou vhodnost volby demonstruje obrázek 1.11.

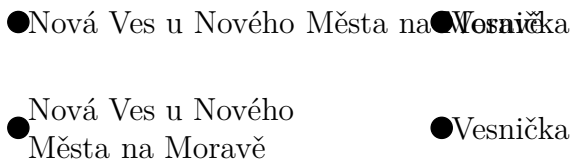
- Lámání textu v popiscích

  U popisků, a zvláště u těch dlouhých, může část problémů vyřešit rozlámání popisků na více řádků. Obrázek 1.12 ilustruje případ, kdy kvůli délce

Obrázek 1.10: Ukázka řešení nejednoznačnosti změnou velikosti



Obrázek 1.11: Možné volby opakování popisku u liniového symbolu



Obrázek 1.12: Ukázka řešení překryvu zalomením popisku

popisku dochází k nežádoucímu překryvu a přirozené rozlomení popisku na dva řádky problém zcela odstraní.

Samozřejmě je nutné takto lámat popisky pouze v místech, kde to „dává smysl“, tedy typicky mezi slovy. Jazykově a typograficky problematická je pak otázka, zda by mělo být povolené lámat popisky ve spojovníku.

V případě lámání popisků může být také žádoucí vyřešit, jak zalámaný text zarovnávat, zda striktně na levý praporek, striktně na střed, podle relativní pozice popisku vůči popisované značce, ...

- Používání zkratek v popiscích

  Použití zkratek představuje možnost, jak zmenšit rozměr popisku, a tedy omezit překryvy s jinými značkami. Při zkracování popisků či jejich částí je nutné myslet na srozumitelnost mapy.

  Extrémní možností zkracování popisků je jejich úplné nahrazení za kód, písmenný či číselný, a doplnění mapy o legendu vysvětlující tyto kódy. Představa o těchto kódech by ale neměla být zaměňována s prostým použitím všeobecně známých značek jako např. MÚ pro městský úřad.

2. Existující přístupy

V této kapitole se zaměříme na existující přístupy. Nejprve se podíváme na výstupy několika existujících nástrojů. Následně shrneme, jaké algoritmy již byly k řešení problému umisťování mapových značek zkoušeny. Nakonec se zamyslíme, jaký přístup by pro nás měl být nejlepší při návrhu vlastního algoritmu.

2.1 Některé současné mapové nástroje

Přestože se tato práce zaměřuje na papírové mapy, otestujeme několik webových nástrojů. To je zčásti dané horší dostupností programů pro tvorbu papírových map, zčásti jednoduchým faktem, že vykreslování online map můžeme považovat za v určitých ohledech těžší úkol. Kdyby se tedy webové nástroje osvědčily, mohli bychom se zkusit vydat cestou jejich mírné úpravy pro využití při tvorbě papírových map.

Webové mapy OpenStreetMap

Prvním testovaným nástrojem jsou webové mapy projektu OpenStreetMap.[1] Zde rozhodně musíme zmínit, že o vykreslování se stará nástroj Mapnik,[2] který je možné použít i offline právě k vykreslení papírové mapy.

Tento nástroj obsahuje různé prohřešky vůči kritériím kvality. Uvedeme několik příkladů takových prohřešků, všechny z mapy blízkého okolí Malostranského náměstí v Praze.

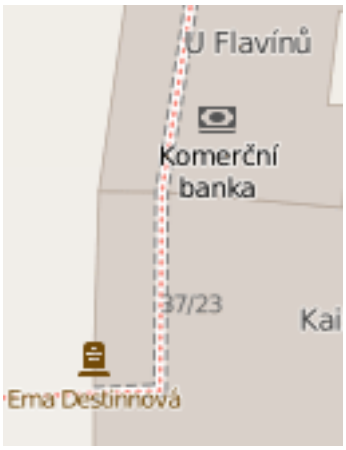
Obrázek 2.1 ukazuje několik překryvů popisku s liniovou značkou, zde s vyznačením podloubí. Potenciálně problematický je popisek pamětní busty Emy Destinové. Ten by se při posunu níže dostal relativně daleko od příslušného symbolu, a zejména by od něj byl oddělen právě liniovou značkou. U ostatních popisků by ale překryvu šlo zabránit jejich posunutím.

Na mapě se relativně často vyskytuje vytečení popisku oblasti z této oblasti (tedy to, že se část popisku nachází nikoliv uvnitř oblasti, ale za její hranicí). Obrázek 2.2 ukazuje situaci, kdy je takové vytečení nevyhnutelné, resp. kdy jeho jedinou alternativou je vynechání popisku, což v tomto případě můžeme považovat za větší problém. Obrázek 2.3 ovšem ukazuje situaci, kdy popisek vytekl z oblasti, přestože by se do ní dal umístit cele.

Přestože obvykle jsou značky i popisky dobře spojitelné se svým místem, resp. s objektem, který popisují, občas se objevují problematická umístění. Jedno z nich je zachyceno na obrázku 2.4. Číslo popisné, resp. orientační 200/11 není jednoduše přiřaditelné ke správnému domu. Teprve, když si všimneme, že dům vpravo je označený 200/5, dojdeme k tomu, že označení 200/11 patří k domu nalevo.

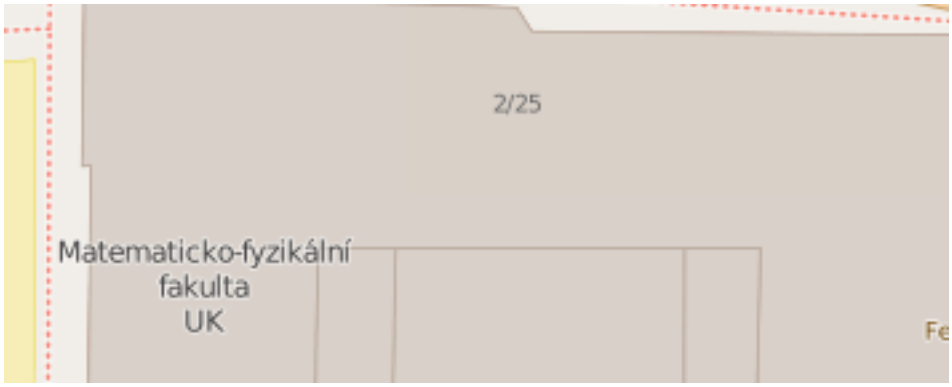
[1] `http://www.openstreetmap.org/`

[2] `http://mapnik.org/`



Obrázek 2.1: Mapa OSM: Překryv popisků s liniovou značkou

Obrázek 2.2: Mapa OSM: Nevyhnutelné vytečení popisku z oblasti



Obrázek 2.3: Mapa OSM: Vytečení popisku z oblasti

Obrázek 2.4: Mapa OSM: Nejednoznačné umístění popisku

Obrázek 2.5: Mapy.cz: Interaktivní zobrazení popisku

Obrázek 2.6: Mapy.cz: Kolize mapových podkladů s umístěnými značkami

Mapy.cz

Druhým testovaným nástrojem jsou internetové mapy Mapy.cz.[3]

Ani tento nástroj nám nevyhovuje. Na Mapách.cz je vidět, že jsou zamýšlené k online využití. Většina popisků není přímo umístěna na mapě, ale zobrazena interaktivně při najetí myší na příslušný symbol, jak naznačuje obrázek 2.5.

Velkým problémem výsledných map je to, že veškeré značky jsou zřejmě umisťovány na hotové podklady, které již samy o sobě značky, zejména popisky, obsahují. Tento problém dobře ilustruje obrázek 2.6 (i zde funguje zobrazení popisku při najetí myší na symbol).

Obrázek 2.7 zobrazuje hned několik problémů. V prvé řadě ukazuje velké množství bílého místa na mapě – to je pochopitelně věcí stylu, nikoliv chybou rozmisťovacího algoritmu jako takového, přesto to může být pro uživatele poněkud matoucí (uživatel si může klást otázku, zda je plocha bílá, protože se něco nenačetlo, nebo prostě proto, že být bílá má).

Pomineme-li zkratku UK v mapových podkladech, není žádným způsobem popsána Matematicko-fyzikální fakultra Univerzity Karlovy. Na mapě je sice umístěna interaktivní ikonka parkoviště na dolním náměstí (tedy na té části náměstí, která je napravo od budovy MFF UK), ale už ne na horním náměstí. Spekulovat bychom jistě mohli i o vhodnosti umístění ikonky zastupující restauraci Profesní dům, do které se ve skutečnosti vchází přibližně z místa, kde je popisek Malostranského náměstí.

Složitější otázkou pak může být, zda je vhodnější popsat Malostranské náměstí někde na ploše celého náměstí jako na této mapě, nebo opakovaným popisem ulic lemujících toto náměstí jako na jiných mapách.

[3] `http://www.mapy.cz/`

Obrázek 2.7: Mapy.cz: Zobrazení Malostranského náměstí

Obrázek 2.8: Mapy Google: Příliš časté opakování liniového popisku

Google Maps

Jako poslední otestujeme nástroj Google Maps.

I na mapách Google je znát zamýšlené online využití, přesto se popisky umisťují stejně jako ostatní symboly a konkrétní názvy jsou tak čitelné i bez nutnosti přejíždět myší na jednotlivé symboly. Přesto můžeme na mapě najít několik prohřešků.

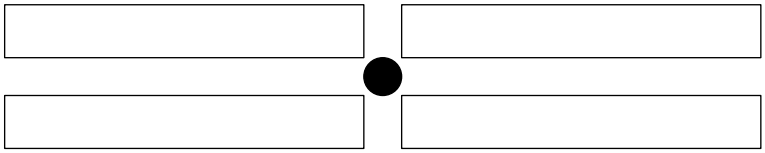
Často se zbytečně brzy opakuje popisek linie, jak můžeme vidět např. na obrázku 2.8.

Na obrázku 2.9 si také můžeme všimnout, že tramvajová zastávka je vykreslena pouze v jednom směru. To principiálně vůbec nemusí být špatně, zvlášť jsou-li si zastávky v obou směrech blízko. Zde to ovšem vypadá, jako by zastávka ve druhém směru vůbec neexistovala (a jelikož zastávky, které skutečně existují jen v jednom směru, v Praze občas najdeme, jedná se o dojem značně matoucí). Samozřejmě se nabízí otázka, zda jde o chybu umisťovacího algoritmu, nebo o vlastnost mapových dat.

S velkou pravděpodobností primárně chybou už v datech, podle kterých se značky k rozmístění generují, přesto chybou budící pozornost, je pak osamocený popisek sloupu Nejsvětější trojice v angličtině, jak ho zachycuje obrázek 2.10.

Obrázek 2.9: Mapy Google: Zobrazení tramvajové zastávky jen v jednom směru

Obrázek 2.10: Mapy Google: Osamocený popisek v odlišném jazyce



Obrázek 2.11: Běžně používané pozice pro umístění popisku

2.2 Předchozí práce a zvolená metoda

Možnosti automatizace tvorby map se řeší přibližně od sedmdesátých let minulého století, v posledních cca dvaceti letech výrazně intenzivněji. Za tu dobu bylo dosaženo mnoha zajímavých výsledků, přestože opravdu uspokojivé řešení problému zatím nikdo neodhalil. Pojďme nyní udělat malý přehled používaných metod.

Většina dosavadních přístupů řeší problém v nějak zjednodušené formě. Výzkum se orientuje zejména na umisťování popisků. Práce zabývající se umisťováním popisků se obvykle omezují na popisky bodových symbolů. Tyto popisky navíc neumisťují do spojitého prostoru celé mapy, resp. celého okolí popisovaného symbolu, nýbrž pro ně pouze vybírají jednu z předpočítaných variant, kam může být popisek umístěn. Tyto varianty jsou často čtyři a vypadají přibližně jako na obrázku 2.11, ale mohou být vůči bodové značce umístěny i jinak, případně jich může být větší množství.

Již o takto zjednodušené variantě problému, tedy umisťování popisků bodových symbolů do některé ze 4 předpočítaných pozic tak, aby se maximalizoval počet nepřekrývajících se značek, bylo prokázáno, že je NP-těžká [13][9], tedy ji pravděpodobně neumíme vyřešit v polynomiálním čase. To, že jsou NP-těžké, bylo dokázáno i o některých jiných formulacích problému, např. v práci Iturriagy a Lubiw[11].

K složitosti celého úkolu navíc přispívají různá kritéria na kvalitu mapy. Některá z nich jsme uvedli v kapitole 1 (a připomeňme, že jejich seznam zdaleka není vyčerpávající). Zde je ovšem na místě si uvědomit, že všechna tato kritéria jsou skutečně jen měřítkem kvality, nikoliv neporušitelnými podmínkami.

Za dobu, po kterou se odborná veřejnost umisťováním mapových popisků zabývá, již bylo vyzkoušeno několik různých přístupů. Původně se využívaly zejména systémy založené na pravidlech a jejich naplňování. Ty podle pravidel umisťovaly popisky, a pokud se dostaly do situace, kdy už popisek nemohly v souladu s pravidly umístit, vracely se (backtrackovaly) a zkoušely jiné možnosti. Umisťování značek v pravidlovém systému popisují např. Cook a Jones [3]. Uplatňovaly se také různé heuristiky.

Mezi ozkoušenými přístupy byl převod problému umisťování popisků na úlohu celočíselného lineárního programování (Cromley [4], Zoraster [20]) a její řešení, resp. řešení její relaxované verze. Při lineárním programování hledáme extrém nějaké lineární rovnice s mnoha neznámými, přičemž k této rovnici máme zároveň zadáno množství lineárních nerovnic, které omezují hodnoty neznámých. Celočíselné lineární programování pak vyžaduje, aby nalezené hodnoty neznámých byly celočíselné. Úlohy celočíselného lineárního programování jsou samy o sobě NP-těžké, ovšem na jejich řešení existují různé heuristiky, z nichž některé se v praxi dobře osvědčují.

Později se pozornost a pokusy přesunuly také k mnoha dalším metodám, mezi nimi zejména k simulovanému žíhání (simulated annealing) (Christensen [2]), tabu prohledávání (tabu search) (Yamamoto a ostatní [18]), převedení na úlohu nalezení nejmenší nezávislé množiny (Strijk a ostatní [15]) a jejímu heuristickému řešení, programování s omezujícími podmínkami (Wagner a Wolff [17]) a k evolučním algoritmům (např. Djouadi [8], Verner a ostatní [16]) včetně využití multikriteriálních evolučních algoritmů [1].

Při simulovaném žíhání uvažujeme o stavech odpovídajících řešení. Těmto stavům přidělujeme energii. Pracujeme jen s jedním stavem a při každé iteraci rozhodneme, zda přejdeme do některého ze sousedních stavů (které se určují na základě současného nějakým dobře definovaným způsobem); celý proces je řízený globálním parametrem – teplotou, která se v průběhu procesu snižuje. Rozhodnutí o přechodu do sousedního stavu je pravděpodobnostní, pravděpodobnost přechodu záleží na energiích obou stavů a na hodnotě teploty. Obecně platí, že pravděpodobnost přechodu je výrazně menší, pokud by se přechodem zvýšila energie; zároveň ale platí, že čím vyšší teplota, tím vyšší může být pravděpodobnost přechodu do horšího stavu (stavu s vyšší energií). Vyšší ochota přecházet do horších stavů v dřívějších fázích procesu snižuje riziko uvíznutí v lokálním optimu.

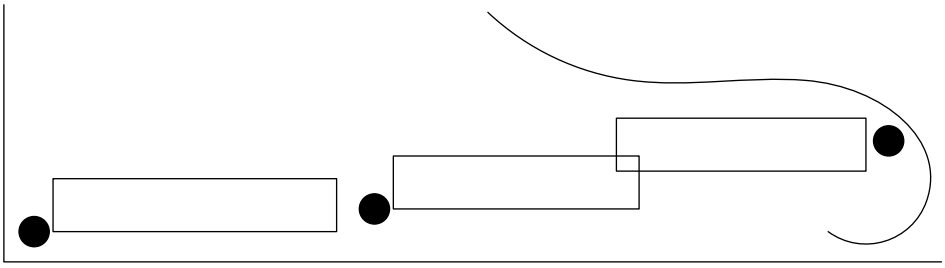
Tabu prohledávání prochází stavový prostor a vždy se přesouvá do nejlepšího možného sousedního řešení. K tomuto přesunu dochází jak v případě, kdy je nejlepší sousední řešení lepší než aktuální, tak v případě, kdy tomu tak není. To opět umožňuje návrat z lokálního optima (případně překonání prostoru, kde jsou všechna řešení stejně dobrá). Aby se ovšem hledání nezacyklilo, udržuje se seznam navštívených stavů, resp. nějakého množství posledních navštívených stavů. Do těchto stavů se prohledávání nesmí znovu vrátit (alespoň dokud nejsou odstraněny z tohoto seznamu).

Programování s omezujícími podmínkami spočívá v určení proměnných, jejich domén (možných nabývaných hodnot) a podmínkách na přiřazení konkrétních hodnot proměnným. Řešení problémů s omezujícími podmínkami je obecně těžké, ale existují pro něj heuristické algoritmy. Cílem typicky bývá nalézt libovolné řešení splňující zadané podmínky.

Evoluční algoritmy pracují s mnoha možnými řešeními najednou. Pomocí upravování a kombinování dosud nejlepších řešení vytvářejí nová, která mohou nahradit ta původní. Aby byl problém dobře řešitelný evolučním algoritmem, musí být kvalita jeho řešení v určitém smyslu spojitá (optimální řešení musí být postupně odvoditelné z méně dobrých drobnými úpravami) a v prostoru jeho řešení musí být dostatečně hustě zastoupena přípustná řešení.

V dosavadních pracích se obvykle jako hlavní kritérium vzal počet nepřekrývajících se popisků a jakákoliv jiná kritéria byla buď zcela ignorována, nebo řešena způsobem podobným pravidlovému a podřízena minimalizaci překryvů.

Takové přeformulování problému má své výhody: umožňuje nám velmi jednoduše hodnotit kvalitu řešení (prostý počet překrývajících se popisků), navíc je to elegantní a objektivní hodnocení kombinatorické stránky celého problému. Tento přístup ovšem vylučuje řešení, která mohou být lepší v širším úhlu pohledu, při zohlednění všech stránek problému. Pokud uvážíme například umístění popisků jako na obrázku 2.12, může být drobný překryv menším problémem než vynechání libovolného z popisků.



Obrázek 2.12: Možná situace, kdy je překryv popisků přijatelný

Při tomto přístupu je navíc obecně těžké algoritmus dále rozšířit, aby začal zohledňovat nějaké další kritérium, zvlášť pokud bychom ho chtěli považovat za rovnocenné kritériu počtu překryvů.

My se proto pokusíme neomezit na počet překryvů, a zejména budeme navrhovat náš algoritmus tak, aby mohla být další kritéria v budoucnosti snadno přidaná.

Přestože se neomezíme na počet překryvů, hlavní odlišnosti naší práce od předchozích budou spočívat ve dvou jiných ohledech.

Nebudeme uvažovat pouze popisky bodových symbolů, nýbrž i popisky liniových symbolů a oblastí. Kromě nich budeme uvažovat také bodové symboly samotné, tedy umisťujeme-li např. popisek se jménem kostela, dovolíme pohybovat nejen s tímto popiskem, ale také s ikonkou příslušného kostela. To nám umožní tvořit rovnou celou mapu.

Van Dijk ve své tezi [6] na rozdíl od ostatních umisťuje nejen popisky bodových symbolů, ale také liniových. Ve své práci dospívá k závěru, že současné umisťování liniových popisků zlepšilo výslednou kvalitu (tedy počet překrývajících se značek) jen o málo a že popisků oblastí bude na mapě nejspíš dostatečně málo na to, aby se na výsledku příliš neprojevily. Přesto bychom si měli všimnout, že k určitému zlepšení došlo, a zejména bychom neměli zapomenout, že van Dijk hodnotí výsledek pouze počtem překryvů (případná další kritéria jsou zohledňována pomocí lokálních optimizátorů stylem best-effort) – budeme-li při hodnocení mapy zohledňovat i jiná kritéria, může být současné umisťování všech popisků velmi příhodné. Počet popisovaných oblastí navíc velmi závisí na typu mapy. Budeme-li tvořit např. plánek města, může se stát, že většinu popisků budou tvořit právě popisky oblastí.

Druhý podstatný rozdíl, který učiníme vůči dosavadním pracem, bude uvažování celého spojitého prostoru místo diskrétních možností pro umístění popisku. Omezení prostoru na předpočítané možnosti má pochopitelně výhodu v omezení prostoru řešení, a tedy výpočetní složitosti. My ale víme, že stejně nemůžeme prozkoumat všechna možná řešení, a omezením jejich prostoru můžeme vyloučit i velmi kvalitní řešení. Některé práce zmiňují i možnost spojitého prostoru řešení, např. de Nascimento a Eades [14], ovšem při jejím popisu předpokládají, že výsledné umístění popisku se bude dotýkat popisovaného symbolu. My nebudeme prostor řešení omezovat ani tímto způsobem, případné nedotýkání se popisovaného symbolu postihneme raději v rámci hodnocení řešení.

I když uvažujeme tyto odlišnosti od dosavadních prací, zůstává umisťování mapových značek pochopitelně problémem, který nejspíš neumíme efektivně řešit přesně. Co (pravděpodobně) umíme, je nějak rozumně zhodnotit výslednou mapu.

Jelikož na mapu nemáme žádná striktní omezení, nedává příliš smysl formulovat problém jako problém s omezujícími podmínkami. Pokud bychom se snažili problém formulovat jako úlohu hledání maximálního párování, hledání maximální nezávislé množiny nebo podobnou grafovou úlohu, nejspíš narazíme na spojitost prostoru řešení. Problém bychom museli nejprve diskretizovat, a tomu se chceme vyhnout.

Kdybychom se rozhodli vydat cestou celočíselného lineárního programování, budeme výrazně omezení v podmínkách i formulaci kriteriální (hodnoticí) funkce. To vše by muselo být lineární. Speciálně bychom tedy nemohli navrhnout hodnocení, resp. žádnou jeho složku tak, aby se měnila např. kvadraticky, což může být žádoucí (např. při hodnocení vzdálenosti umístění značky od její skutečné polohy).

Abychom ale vybrali metodu, kterou použít chceme, zamyslíme se ještě nad spojitostí kvality řešení. Můžeme předpokládat, že existuje mnoho srovnatelně dobrých map, z nichž navíc vedou srovnatelně dobré cesty (v podobě postupných úprav) do dalších, ale zároveň mohou prvotní mírná zhoršení vést k výrazně lepší mapě. Zdá se tedy oprávněné chtít pracovat s mnoha mapami naráz. Mírnými úpravami mapy určitě můžeme zlepšit její kvalitu, platí také, že dostatečně dobrou mapu můžeme získat postupným upravováním méně dobrých map. Jelikož na mapu nemáme striktní podmínky, nemůžeme vyrobit nepřípustnou mapu (ač samozřejmě můžeme vyrobit mapu, která bude tak špatná, že bude efektivně nepoužitelná). Koncentrace přípustných řešení tak určitě bude vysoká.

Vše tedy nasvědčuje tomu, že náš problém by mohl být velmi dobře řešitelný pomocí evolučního algoritmu. Proto se budeme dále zabývat evolucí a nějaký evoluční algoritmus pro umisťování mapových značek navrhneme.

3. Evoluční algoritmy

V této kapitole představíme jednoduchý úvod do evolučních algoritmů. Náš text nemůže v žádném případě nahradit v celém rozsahu semestrální výuku probírající takové algoritmy do hloubky. Měl by ale posloužit jako připomenutí těm, kteří takovou výukou prošli, a jako dostatečný základ pro porozumění zbytku práce těm, kteří takovou výuku neměli.

Evoluční algoritmy jsou algoritmy inspirované přírodou a vývojem života (resp. současnými teoriemi o něm). Jedná se o algoritmy stochastické, tedy využívající náhodu, ale nikoliv o náhodné prohledávání.

Princip evolučních algoritmů spočívá ve vylepšování existujících řešení. Nejprve se „nějak“, často náhodně, vygeneruje počáteční sada řešení. Následně se pomocí upravování a kombinování existujících řešení odvozují řešení nová. Do dalšího průběhu se ponechávají pouze některá z dosud získaných řešení. Jednotlivé algoritmy se pak liší zejména tím, jak přesně vygenerují tuto počáteční sadu řešení, jak řešení upravují a kombinují, jak vybírají, která řešení budou upravena či zkombinována, jak určují, která řešení jsou dobrá, a také tím, jak určují, která řešení se ponechají a která se zahodí.

Z přírody si evoluční algoritmy vzaly zejména dva principy. Tím prvním je dědičnost, tedy to, že některé informace jsou dědičné, udržované v genetické informaci a při křížení předávány od rodičů jejich potomkům.

Druhým je pak přírodní výběr a související snaha o přežití. Řešení, která jsou lepší (ať už to znamená cokoliv), mají větší šanci přežít a podílet se na odvozování dalších řešení. Jejich genetická informace se tak víc a víc rozšiřuje a stává se běžnou součástí řešení, a to až do doby, kdy je vytlačena informací ještě lepších řešení.

Kvůli nutnosti vygenerovat mnoho různých řešení bývají evoluční algoritmy pomalé, navíc jsou velmi náchylné na složitost častých operací – tedy zejména ohodnocení existujícího řešení a vygenerování nového. Pokud některá z těchto operací trvá dlouho, bude mít velmi špatnou složitost i celý algoritmus. Naopak, zrychlení takové operace může výrazným způsobem urychlit celý algoritmus.

Jako stochastické nedávají evoluční algoritmy žádnou záruku nalezení optimálního řešení. Kvůli postupnému vylepšování řešení také obecně platí, že pro nalezení lepšího řešení potřebuje algoritmus delší dobu. Při vhodně realizovaném odvozování a hodnocení řešení ale má smysl předpokládat, že poskytneme-li algoritmu dostatečně dlouhý čas, bude výsledkem řešení dostatečně blízké optimálnímu. Podotkněme, že délka běhu nutná pro tento předpoklad se často určuje empiricky.

I přes relativní pomalost a nejistou kvalitu výsledku se evoluční algoritmy dobře uplatňují v praxi. Zvláště pro problémy, ke kterým neznáme přesný polynomiální algoritmus, mohou být velmi dobrou aproximací.

Aby byl problém řešitelný evolučními algoritmy, musí splňovat dvě, resp. tři podmínky.

Kvalita řešení musí být v určitém smyslu spojitá. Potřebujeme, aby se při úpravách řešení postupně měnila kvalita a aby optimální (resp. dostatečně dobré) řešení bylo podobné mnoha jiným řešením s kvalitou jen o málo menším, tedy aby se z těchto řešení dalo odvodit.

Evoluční algoritmy mohou ve skutečnosti uspět i v případě, že jsou dostatečně dobrá řešení izolovaná, resp. obklopená řešení výrazně horšími, a stejně tak v případě, kdy se kvalita řešení mění velmi skokově. V obou případech ale potřebují k odvození vhodného řešení mnohem více pokusů, a tedy mnohem více času; při podobném času dají v průměrném případě výrazně horší výsledek.

Druhou podmínkou je, aby nově odvozené řešení bylo s dostatečnou pravděpodobností přípustné, tedy aby splňovalo všechny podmínky problému, který řešíme. V opačném případě by opět výrazně narůstala doba potřebná na nalezení vhodného řešení.

V neposlední řadě je pochopitelně potřeba, aby bylo vůbec možné problém, resp. jeho řešení zapsat jako řešení evolučního algoritmus a navrhnout jednotlivé operace.

Závěrem pro popisování evolučních algoritmů zaveďme několik formálních pojmů. Pojmy budeme používat stejné jako Kubalík [12]. Jako *chromozom* označíme zápis řešení, s tímto zápisem bude pak celý algoritmus pracovat. Chromozom je tvořen jednotlivými *geny*.

Pro řešení budeme používat pojem *jedinec*. Tím budeme myslet nejen příslušný chromozom, ale také *hodnocení* tohoto chromozomu, tedy hodnotu vyjadřující, jak dobře či špatně řeší zadaný problém. Sadě řešení, resp. sadě jedinců, budeme říkat *populace*.

Odvození nového řešení tím, že upravíme existující, označíme za *mutaci*, zatímco odvození nového řešení zkombinováním existujících označíme jako *křížení*. Navíc budeme předpokládat, že křížíme vždy dvě řešení, přestože technicky je možné zkřížit jich dohromady i více.

Chromozomy mohou mít i velmi složitou podobu, může jít např. o obecné grafy, ale pro naše potřeby bude postačující představovat si chromozom jednoduše jako posloupnost genů.

Zavedené pojmy včetně těch, které přidáme v jednotlivých sekcích této kapitoly, na konci této kapitoly pro přehlednost shrneme do tabulky.

Pro úplnost ještě dodejme, že v existující literatuře se občas používá také pojem genetické algoritmy. V některých případech jsou jimi označovány evoluční algoritmy tak, jak je popisujeme v této práci. Někdy jsou dokonce pojmy evoluční a genetické algoritmy v literatuře volně zaměňovány. Jindy se tyto pojmy odlišují podle podoby chromozonu. V takovém případě se jako genetické označují ty algoritmy, které mají chromozomy tvořené výhradně binárními hodnotami, zatímco jako evoluční se označují ty s obecnější podobou chromozomu.

3.1 Hodnocení jedinců

Abychom mohli vybírat, které jedince zachováme a umožníme jim tak stát se součástí budoucích populací, musíme být schopní jedince porovnávat, a tedy i nějak hodnotit. K tomu nám bude vždy sloužit *hodnoticí funkce*. Ta na vstupu vždy dostane jedince, resp. jeho chromozom, a vrátí reálné číslo – *hodnocení*, resp. *penalizaci* tohoto jedince. Mezi pojmy hodnocení a penalizace nebudeme důsledně rozlišovat; obvykle použijeme pojem penalizace, budeme-li chtít zdůraznit, že naším cílem je hodnocení minimalizovat.

Jak už bylo naznačeno, hodnoticí funkce musí mít malou složitost, neboť ji voláme často. Proti požadavku na rychlost hodnocení stojí požadavek na jeho

přesnost. Potřebujeme, aby spočítaná hodnocení opravdu vypovídala o kvalitě hodnocených jedinců, jinak nebudeme schopní vhodně vybírat, které jedince ponechávat a případně upřednostňovat při mutaci a křížení.

U některých problémů existuje jednoduchá objektivní a přímočará hodnoticí funkce (tou může být např. očekávaný zisk vyvinutého plánu), u jiných problémů taková funkce není a úspěšnost evolučního algoritmu záleží také na vhodnosti návrhu této funkce. Zdůrazněme také, že hodnoticí funkce může k určení hodnocení využít různé prostředky – analytický výpočet, simulaci, ...

Zejména u problémů, pro které neexistuje přímočará hodnoticí funkce, nám ovšem její nalezení samo o sobě nezaručuje jednoduché porovnání jedinců. Mělo by platit, že vyšší hodnocení (resp. nižší penalizace) znamená lepší řešení. Nemusí už ale být pravda, že dvojnásobné hodnocení znamená dvakrát lepší řešení. Obecně funkční způsob, jak se s tímto faktem vyrovnat, přitom není.

Proč je vlastně taková porovnatelnost důležitá? Kvůli inspiraci přirozeným výběrem bychom chtěli, aby lepší řešení měla větší šanci přežít, a také větší šanci předávat své geny (tedy účastnit se mutace nebo křížení). Ideálně bychom si přáli, aby dvakrát lepší řešení mělo dvojnásobnou šanci – na něco takového bychom ale museli umět říct, že řešení A je dvakrát lepší než řešení B, což právě ne vždy umíme.

3.2 Evoluční operátory

Pro odvozování nových řešení při evoluci používáme dvě základní operace: mutaci, tedy úpravu existujícího řešení, a křížení, tedy kombinaci dvou existujících řešení. Ještě než si přiblížíme smysl, potenciální problémy a běžné realizace každé z nich, zamyslíme se nad výběrem jedinců pro tyto operace.

Výběr jedinců

Jak jsme již několikrát zmínili, chceme, aby jedinci s lepším hodnocením měli vyšší šance účastnit se evolučních operací a šířit své geny. Už jsme naznačili, že toto přání občas naráží na nemožnost říct, o kolik lepší je které řešení, a tedy o kolik větší by jeho šance na šíření svých genů měly být.

S preferováním výrazně lepších jedinců jsou ale svázané dva možné problémy evoluce, resp. dvě možné vady jejího průběhu, které se mohou objevit v různých fázích evoluce.

V začátku, při tvorbě prvních populací, může dojít k *předčasné konvergenci*. Ta nastává, pokud je nějaký jedinec J výrazně lepší a my následně většinu nových jedinců získáme operacemi s tímto jedincem J. Ostatní geny se z populace ztratí, populace se omezí na jedince J a jeho drobné mutace. To jednak zpomaluje další vývoj, jednak zvyšuje riziko uvíznutí v lokálním optimu.

Naopak v pozdějších fázích hrozí *stagnace*. Pokud si jsou všechna řešení podobná a my je pro evoluční operace vybíráme stejnoměrně, nemohou se výrazněji rozšířit geny lepších řešení. Populace se pak téměř nevyvíjí, přestože v ní jsou zastoupeni silnější jedinci.

Předčasné konvergenci i stagnaci je při určitých typech výběru jedinců možné předcházet *škálováním* spočítaných hodnocení. Při něm v každé iteraci upravíme hodnocení všech jedinců tak, aby hodnocení nejlepšího jedince bylo nově

konstanta-násobkem (za tuto konstantu se obvykle volí hodnota 1,5-2) průměrného hodnocení a hodnocení průměrných jedinců aby zůstala průměrná.

Nejjednodušším způsobem výběru jedinců je *náhodný výběr*, případně náhodný výběr z určitého množství nejlepších jedinců. Ten do značné míry porušuje stále připomínaný princip, že lepší jedinec má mít vyšší šanci předat své geny, přesto může fungovat dobře.

Běžným způsobem výběru je *ruletový výběr*, kdy jsou jedinci virtuálně rozmístěni po obvodu rulety. Z této rulety jim je vyhrazena poměrná část, která odpovídá jejich hodnocení, resp. poměru jejich hodnocení vůči hodnocení celé populace. Virtuální ruletou se zatočí a vybere se ten jedinec, jemuž odpovídá část u ukazatele.

Posledním způsobem výběru, který zmíníme, je výběr *turnajem*. Při něm se nejprve náhodně vybere určité malé množství jedinců a z těchto jedinců se zvolí ten s nejlepším ohodnocením. (Striktně vzato je možné obyčejný náhodný výběr považovat za speciální případ výběru turnajem, kdy se jako malé množství volí pouze jeden jedinec.)

Mutace

Mutace představuje malou změnu na chromozomu existujícího jedince, a jako taková pomáhá zachovat, případně obnovit rozmanitost populace. Pomocí mutace se mimo jiné mohou obnovit dříve ztracené hodnoty genů. To, že jde o malou změnu, je důležité – kdyby byla změna příliš velká, bude se mutace chovat podobně jako náhodné vygenerování nového jedince.

Hlavním rizikem mutace je pochopitelně to, že změna chromozomu se ukáže být změnou k horšímu. To se ale odrazí na horším hodnocení zmutovaného jedince, a v ideálním případě tedy i jeho postupném vymizení z populace.

U některých problémů, případně některých realizacích mutace, může také mutováním chromozomu vzniknout chromozom reprezentující řešení, které není přípustné (jinými slovy neodpovídá podmínkám, které musí výsledné řešení splňovat). V takovém případě je pochopitelně nutné přípustnost vzniklého řešení kontrolovat a mít stanovený postup i pro situaci, kdy vznikne řešení nepřípustné (ať už by tím postupem měla být nová mutace původního chromozomu, použití původního chromozomu tak, jak je, nebo postup úplně jiný).

Nejčastějším způsobem, jak mutaci realizovat, je *změna hodnoty*. U tradičních binárních řetězců jde vždy o převrácení binární hodnoty, u složitějších genů může jít o nahrazení libovolnou hodnotou přípustnou na daném genu nebo o mírnou úpravu současné hodnoty (u reálných čísel např. přičtení náhodného čísla v rozsahu 0-1).

Křížení

Křížení vytváří nové jedince, *potomky*, nakombinováním chromozomů existujících jedinců, *rodičů*. Tradičně křížením dvou rodičů vznikají dva potomci; každý z potomků přitom dostává část genů od jednoho rodiče, část od druhého. Na rozdíl od mutace nemůže křížení vnést do populace nové hodnoty genů, pouze jejich nové kombinace. Zato umožňuje mnohem rychlejší a dynamičtější vývoj populace, jelikož nově vzniklí jedinci jsou od původních odlišnější než při mutaci.

Podotkněme, že někdy se právě pro podporu obměny genů nově vzniklí potomci ještě s určitou pravděpodobností mutují, takže se z nich nestává přímo kombinace jejich rodičů, ale její mírně upravená verze.

Podobně jako mohou nepřípustná řešení vznikat při mutaci, mohou u některých problémů či při některých realizacích vznikat jako výsledek křížení. Je tedy třeba případně přípustnost kontrolovat a nepřípustná řešení ošetřovat.

Podle charakteru problému, resp. podoby chromozomu může také křížení negativně ovlivnit kvalitu získaných jedinců. Pokud se jednotlivé geny ovlivňují, může se stát, že u křížených jedinců se podporují, případně si alespoň neškodí, zatímco nakombinované geny obou rodičů si budou škodit výrazně. Opět je nutné na toto riziko myslet při návrhu algoritmu a realizaci křížení zvolit s ohledem na vazby mezi jednotlivými geny.

Jedním z běžných způsobů realizace křížení je *náhodné rozhodnutí o každém genu*. V takovém případě se u každého genu náhodně určí, který z potomků jej zdědí po prvním rodiči a který po druhém.

Jinými běžnými realizacemi jsou *jednobodové* a *dvoubodové křížení*. Při nich se na chromozomu určí jeden, resp. dva rozdělující body mezi geny. Tyto body chromozom rozdělí na dvě, resp. tři kratší sekvence genů. Následně se celé tyto sekvence kopírují jako jeden celek. První potomek zdědí první sekvenci od prvního rodiče, druhou od druhého (případnou třetí opět od prvního). Druhý potomek naopak zdění první sekvenci od druhého rodiče atd.

3.3 Průběh evoluce

Jak už jsme předesílali, evoluční algoritmus principiálně funguje tak, že nejprve vygeneruje prvotní populaci, následně populaci pomocí mutací a křížení obměňuje a uchovává nejlepší jedince, dokud nenalezne nějakého dostatečně dobrého.

Na celém procesu je ale pochopitelně k vyřešení mnohem víc věcí, než jen jak přesně realizujeme hodnoticí funkci, výběr jedinců a jednotlivé evoluční operátory.

My se nyní blíže podíváme na vlastnosti populace a také na možnost evoluci úspěšně ukončit.

Populace

K vlastnostem populace, a tedy i celého evolučního algoritmu, patří přesný způsob *obměny populace*. Populaci můžeme obměňovat nárazově, tedy tak, že vždy nahradíme celou starou populaci populací novou, nebo postupně, kdy vždy nahrazujeme jen několik jedinců. Tradičně se předpokládá pouze to, že populace, se kterou algoritmus pracuje, je po celou dobu jeho běhu stále stejně velká. Pokud nahrazujeme celou populaci, mluvíme obvykle o *generacích*.

Při prostém nahrazování populace novou generací může dojít ke zhoršení kvality nejlepšího jedince v populaci. To nastane, pokud se dosud nejlepší jedinci vlivem náhody kříží a mutují pouze způsobem, který vyprodukuje horší jedince. V takovém případě se navíc geny původně kvalitních jedinců zcela ztratí.

Takové ztrátě můžeme předejít aplikováním tzv. *elitismu*, při kterém do nové generace zahrneme i určité množství nejlepších jedinců z předchozí generace.

Faktorem, který dále musíme zvážit, je *velikost populace*. Velikost populace ovlivňuje rozličnost jedinců a schopnost populace konvergovat k dobrým řešením.

Obvykle je nutné určit vhodnou velikost populace experimentálně, standardně se ale pohybuje v řádu vyšších desítek či nižších stovek jedinců.

Ukončení evoluce

Ukončit celý vývojový cyklus můžeme různými způsoby. Tím pravděpodobně nejčastějším je *dosažení stanoveného počtu iterací*, tedy vytvoření určitého množství populačních generací, nebo nahrazení určitého množství jedinců ve stálé populaci (pokud nenahrazujeme populaci celou, ale pouze několik jedinců v ní).

Bohužel i vhodný počet iterací k ukončení evoluce je obvykle třeba určit experimentálně. Ideální počet navíc souvisí s velikostí populace, do omezené míry platí, že čím větší populace, tím méně iterací stačí k nalezení dobrého řešení.

Výhodou takového přístupu je, že mají-li evoluční operátory a hodnoticí funkce omezenou dobu běhu, je omezená i doba běhu celého algoritmu.

Za ukončovací podmínku můžeme zvolit také *stagnaci* populace, tedy stav, kdy se ohodnocení nejlepších jedinců již dál nezlepšuje. V takovém případě sice nemáme zaručenou žádnou maximální dobu běhu, ale můžeme předpokládat, že nalezeného jedince již nešlo evolucí dál vylepšit.

Zde se sluší podotknout, že vlivem náhody může zejména v pozdějších fázích evoluce zůstávat nejlepší ohodnocení jedince několik iterací stejné (neboť jedinec je sice křížen a mutován, ale změny se s velkou pravděpodobností dějí k horšímu), načež se nejlepší ohodnocení zlepší (nastane méně pravděpodobná varianta a efekt mutace je kladný). Může tedy být vhodné neimplementovat ukončení při stagnaci jako ukončení při prvním zopakování nejlepšího hodnocení, ale jako ukončení při opakování nejlepšího hodnocení po několik iterací.

Potenciálně můžeme chtít i ukončení při *dosažení určitého hodnocení*. Při volbě takové ukončovací podmínky ale musíme mít na paměti, že řešení může předčasně zkonvergovat, případně naše žádoucí hodnocení vůbec nemusí být dosažitelné, a pak se program nikdy nezastaví.

Tabulka pojmů

Pojem	Význam pojmu
chromozom	zápis řešení
gen	dílčí část chromozomu
jedinec	konkrétní řešení s ohodnocením
populace	sada jedinců
mutace	úprava existujícího jedince
křížení	kombinace existujících jedinců
rodič	výchozí jedinec při křížení
potomek	jedinec vzniklý křížením
hodnoticí funkce	funkce přiřazující jedincům hodnocení
hodnocení, penalizace	míra (ne)kvality jedince
generace	populace v konkrétní iteraci

Tabulka 3.1: Pojmy v souvislosti s evolučními algoritmy

4. Navržený algoritmus

V této kapitole se budeme zabývat konkrétním řešením problému umisťování mapových značek. V sekci 2.2 jsme uvedli důvody, proč by mělo být právě použití evolučních algoritmů vhodným způsobem k řešení tohoto problému. Proto se pokusíme vhodný evoluční algoritmus navrhnout.

Nejprve se přesněji zamyslíme nad tím, co by mělo být jeho vstupem a výstupem, pak navrhneme vhodné genetické kódování jednotlivých řešení (vhodnou podobu chromozomů).

Následně navrhneme vhodnou realizaci křížení a mutace s ohledem na řešený problém a dále navrhneme hodnoticí funkci. Nakonec se zamyslíme nad celkovým průběhem evoluce.

Sluší se říct, že návrh celého algoritmu samozřejmě není takto jednoduchý, resp. až takto postupný. Volba genetického kódu může ovlivnit vhodnost určité realizace křížení, zvolený způsob mutování může zvyšovat potřebu elitismu, hodnoticí funkce může pro rychlý běh vyžadovat specifické genetické kódování atd. My zde ale vše popíšeme postupně.

4.1 Vstup a výstup algoritmu

Teoreticky je vstup a výstup algoritmu jasný – vstupem algoritmu jsou požadavky na umístění mapových značek, výstupem pozice těchto značek. Co to ale znamená pořádněji?

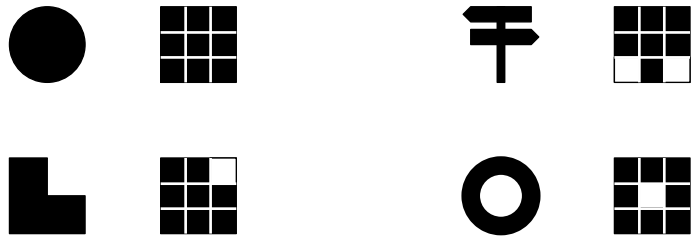
Předně se zamysleme nad tím, co je vlastně požadavek na umístění mapové značky. Takový požadavek je informace

- že se vůbec má něco umístit,
- co se umisťuje (bodový symbol, popisek atp.),
- jak to vypadá (modrá čára, černý puntík, text daného znění, ikonka, ...)
- a kam se to má umístit (na místo odpovídající souřadnicím, někam podél konkrétní čáry, ...).

Navrhujeme algoritmus pro umisťování mapových značek, nikoliv pro celou tvorbu mapy, dovolme si tedy zanedbat otázky ohledně toho, jak přesně značku vykreslit. Předpokládejme ale, že máme informaci o její velikosti a tvaru.

U bodových symbolů bude tedy požadavek parafrázovatelný jako „umísti něco zadané velikosti, ideálně na zadané souřadnice (tedy na přesně určený bod)“. Pochopitelně se nabízí otázka, jak se u plošného objektu určuje, zda byl umístěný přesně na konkrétní bod.

Dohodněme se nyní, že naše informace o velikosti a tvaru značky bude mít podobu čtvercové sítě sloužící jako bitová maska. Tato maska bude mít minimální rozměry takové, aby obsáhla celou značku (tedy bude mít rozměry jejího ohraničujícího (bounding) boxu). Jedničky v této masce pak znamenají, že v dané části ohraničujícího boxu se vyskytuje grafika značky. Obrázek 4.1 uvádí několik příkladů masek pro různé značky (černá políčka představují jedničky, bílá nuly).



Obrázek 4.1: Hrubé bitové masky pro vybrané značky

Nyní můžeme říct, že požadavek na umístění bodového symbolu na konkrétní místo je požadavkem na umístění levého horního rohu masky příslušné značky na toto místo. Kontrola toho, zda byl na konkrétní bod umístěný jiný bod, už je dobře definovaná.

Tím jsme ale buď vynutili úpravu vstupních dat, aby takový smysl umisťování respektovala, nebo snížili přesnost mapy. Umisťované značky se nám tak totiž přirozeně ocitají vpravo dole od skutečné pozice, nikoliv na ní. To ovšem můžeme jednoduše napravit – umožníme rozšířit požadavek o specifikaci, jak by měl být levý horní roh masky posunutý vůči zadaným souřadnicím. Tím dosáhneme podobného efektu, jako kdybychom umožnili specifikovat, který bod masky se má na souřadnice umístit, a navíc jsme povolili i body mimo masku (žádoucí posunutí může být technicky vzato třeba dvojnásobek šířky masky doleva).

Informaci o žádoucím posunutí můžeme získávat přímo ze vstupních dat, z jiných částí mapového programu, případně ji můžeme počítat sami.

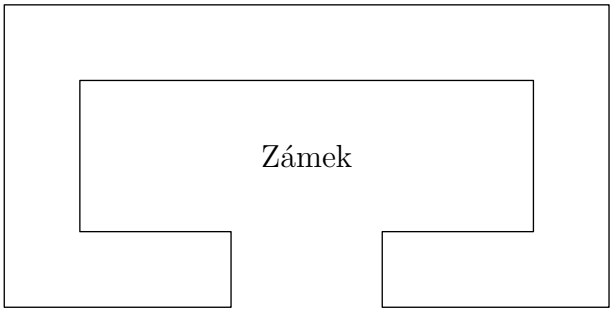
Umožněním specifikace takového posunutí jsme zároveň vyřešili ještě jednu komplikaci při umisťování symbolů, a to že různé symboly by se vůči své poloze měly umisťovat různě. Třeba značka rozcestníku by se měla objevit přímo nad svou skutečnou polohou (tedy skutečné poloze bude odpovídat střed dolní hrany této značky), zatímco značky studánek či restaurací se spíše umisťují tak, aby na skutečnou polohu připadl střed značky.

Shrňme si to: požadavek na umístění bodového symbolu je požadavkem, aby se značka reprezentovaná maskou objevila v určité pozici vůči zadanému bodu. Výstupem našeho algoritmu je pak přesné umístění levého horního rohu této masky.

Požadavky na umístění liniových symbolů a oblastí jsou požadavkem, aby se tyto symboly v přesné podobě objevily na přesných místech. Přestože jsme v kapitole 1 naznačili, že zejména s liniemi se pojí mnoho problémů, které je třeba řešit (např. při souběhu různých turistických stezek), nebudeme se komplikacemi spojenými s těmito požadavky zatím zabývat a symboly vykreslíme přesně tak, jak je v požadavku popsáno. Při implementaci je tedy nedotčené předáme dalším částem mapového programu, pomyslný výstup algoritmu je pro tyto požadavky prázdný.

I pokud nebudeme linie ani oblasti nijak složitě umisťovat, může pro nás být užitečné vědět, že na ně tyto požadavky přišly a kde se tyto symboly nachází. Mimo jiné můžeme takovou informaci využít při pozdějších kontrolách překryvů.

K popiskům bodových symbolů se můžeme chovat velmi podobně jako k bodovým symbolům samotným. Je to požadavek na umístění značky (popisku), resp. jejího levého horního rohu na konkrétní místo dané posunutím vůči zadaným souřadnicím.



Obrázek 4.2: Příklad situace, kdy střed oblasti není její součástí

Podívejme se nyní na popisky oblastí, které se vlastně chovají dost podobně. I tyto požadavky jsou požadavkem na umístění značky, ale tentokrát ne na konkrétní místo, nýbrž do nějaké oblasti.

My si pro jednoduchost konkrétní místo vyrobíme, a to střed popisované oblasti. V takové chvíli se můžeme i k popiskům oblastí chovat podobně jako k popiskům bodových symbolů (a tedy i jako k bodovým symbolům samotným). Přesto bychom informaci o oblasti neměli zcela zahazovat – můžeme ji využívat pro lepší hodnocení umístění popisku oblasti (např. kontrolovat, zda je popisek umístěn *uvnitř* této oblasti; jednoduchý příklad, kdy se tak při umístění na střed nestane, je naznačený na obrázku 4.2).

I pro popisky oblastí je přímočarým výstupem přesné umístění rohu masky.

Nakonec se podívejme na popisky liniových symbolů. Jak jsme naťukli již v kapitole 1, takových popisků může být žádoucí umístit více, aby se na mapě neobjevil příliš dlouhý nepopsaný kus.

Takové popisky by si neměly být příliš blízko, ani by od sebe neměly být příliš vzdálené. Zároveň by měly kopírovat tvar linie, kterou popisují. Jenomže součástí vstupního požadavku je jen informace, k jaké linii popisek patří, a žádné nápovědy k tomu, kde ji popsat.

Mohli bychom vždy nějak (třeba na základě délky) určit, kolika popisky bude linie popsaná, a takový počet popisků náhodně rozmístit po její délce. To by ale znamenalo, že bychom buď museli mít přesnou informaci o podobě popisku (jeho textu, řezu a velikosti písma, ...), abychom jeho tvar mohli přizpůsobit tvaru linie v daném místě, nebo bychom se museli smířit s tím, že popisky nebudou tvar linie kopírovat.

Ani jedna z popsaných možností není dobrá. Navíc nám takové náhodné umisťování nijak nezaručuje, že jednotlivé popisky od sebe budou rozumně daleko. Pojďme na to tedy úplně jinak.

Trochu si upravíme vstupní požadavek. Stanovíme si maximální délku, jakou může nepopsaný úsek mít (resp. polovinu této délky), a rozdělíme zadanou linii na úseky této délky. Těmto úsekům budeme říkat *sekce*. Nyní řekneme, že každá sekce musí být právě jednou popsaná.

Tím jsme vyloučili možnost, že se objeví příliš dlouhý nepopsaný kus linie (ale můžeme si všimnout, že jsme nevyloučili přílišnou blízkost popisků).

Chtěli bychom, aby popisky respektovaly tvar linie, a proto jednotlivé sekce dále rozdělíme, a to na jednotlivé rovné úseky. Takové úseky mohou být někdy velice krátké, třeba v místech, kde linie nějak zatáčí. Obvykle ale budou relativně dlouhé.

Podobu popisku na rovném úseku určit umíme – základní popisek, resp. jeho masku, otočíme o vhodný úhel. Vybereme tedy ty úseky, které jsou dostatečně dlouhé, aby se do nich popisek vešel, a prohlásíme je za dílčí prvky celé sekce. Budeme jim říkat *segmenty*.

Nyní se domluvíme, že každá sekce bude popsaná právě jednou, a to v některém ze svých segmentů.

Pochopitelně by se nám mohlo stát, že v celé sekci není ani jeden dostatečně dlouhý úsek, a tedy nemáme žádný segment, v kterém bychom linii mohli popsat. V takovém případě bychom danou sekci buď museli nechat nepopsanou, nebo využít informací o přesné podobě popisku a ohout text přímo podle tvaru linie. Tím můžeme vytvořit speciální segment s příslušným popiskem a získat sekci tvořenou alespoň jedním segmentem.

Popisky segmentů se již chovají velmi podobně popiskům oblastí či bodových symbolů. Mají svou masku, jejíž roh se má umístit. Nemají přímo referenční souřadnice jako popisky bodových symbolů, ale počáteční souřadnice jim můžeme spočítat podobně jako popiskům oblastí, tentokrát třeba jako střed segmentu, ke kterému patří. Finálním výstupem pro ně bude jako pro ostatní značky přesné umístění rohu masky.

Naopak u sekcí algoritmus vybírá pouze to, v kterém segmentu budou popsané, výstupem u nich tedy není umístění, ale jakási meta-informace. Výstup pro původní požadavky na popisky liniových symbolů není žádný, resp. je prázdný a nahrazen výstupy pro požadavky na sekce a příslušné segmenty.

Vraťme se ještě k některým z možných řešení kvality mapy. Možným řešením problémů může být použití zvětšení či zmenšení značky, u popisků jiné zalámání textu či použití zkratek. Tuto myšlenku můžeme zobecnit – některé prohřešky vůči kvalitě je možné řešit použitím jiné varianty stejné značky.

Dovolme tedy, aby požadavky na umístění značky obsahovaly ještě informaci, „jak jinak to taky může vypadat“. Ve skutečnosti tato informace nemusí být nutně součástí požadavku, můžeme ji odvozovat i my v našem algoritmu.

Požadavek tedy nemusí být požadavkem na umístění jedné konkrétní značky reprezentované jednou konkrétní maskou, ale požadavkem na umístění právě jedné z několika konkrétních značek reprezentovaných svými maskami. Jednotlivým značkám říkejme *varianty*.

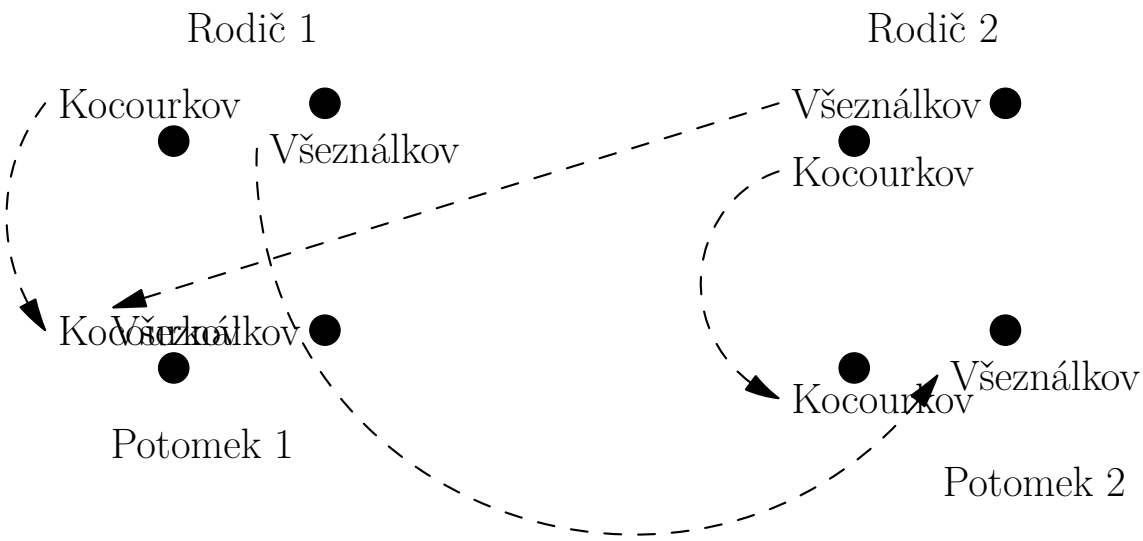
Součástí výstupu pro jednotlivé požadavky (pro ty, které měly dosud výstup neprázdný) nyní bude i informace o tom, která varianta se má při umisťování použít.

4.2 Genetická reprezentace

Nejprve si uvědomme, že požadavky na umístění značek jsou celou dobu stejné, součástí řešení jsou skutečně jen výstupy, tedy informace, která značka bude použita (resp. u sekcí v kterém segmentu bude sekce popsána) a kam se má roh této značky umístit. Souhrnně budeme těmto informacím říkat *umístění* požadavku.

U zadání tedy budeme uvažovat jako o posloupnosti požadavků a jednotlivce reprezentujeme jako posloupnosti umístění těchto požadavků.

Hodnotou i-tého genu bude obecně umístění i-tého požadavku. Jelikož umístění požadavku je souborem informací (souřadnice, použitá varianta), budou hodnoty genů při implementaci zřejmě realizovány jako struktury či objekty. Přímo



Obrázek 4.3: Ukázka vzniku překryvů křížením

jako věc implementace pak ponecháme hodnoty genů odpovídajících požadavkům, které se v nějakém smyslu neumisťují.

4.3 Křížení

K rizikům křížení patří, že namíchané geny rodičů si budou navzájem škodlivé. Přesně takové riziko skutečně nastává při křížení jednotlivých map. Pokud bychom přímočaře vzali část umístění z jednoho rodiče a část ze druhého, snadno se může stát, že se v potomcích budou umístění překrývat, přestože v rodičích byly již překryvy eliminovány. Jednoduchý případ takové situace demonstruje obrázek 4.3.

To, které geny se ovlivňují, je navíc dáno jejich hodnotami. Není možné předem určit vhodné pořadí požadavků tak, aby se ovlivňovaly vždy pouze sousední geny – i kdybychom zkusili požadavky např. lexikograficky uspořádat, mohou se negativně ovlivnit takové, které jsou v chromozomu relativně daleko od sebe. Použití několikabodového křížení pro nás tedy není rozumným řešením.

Potřebovali bychom vzít ze stejného rodiče taková umístění, abychom nevytvořili kombinací umístění z různých rodičů nové překryvy. Od této myšlenky se odrazíme a odvodíme *symbolový uzávěr*.

Pomocí tohoto uzávěru určíme právě ty geny, jejichž hodnoty musíme kopírovat společně ze shodného rodiče. Pokud by se značka z při umístění do opačného rodiče překrývala s jinými značkami, určitě chceme vzít ze shodného rodiče jak umístění těchto jiných značek, tak značky z. Podobně kdyby u kterékoliv z těchto jiných značek hrozil překryv s umístěním dalších značek ve druhém rodiči, chceme i tyto další značky vzít ze stejného rodiče.

Takto budeme množinu značek, jejichž umístění máme vzít ze shodného rodiče, rozšiřovat, dokud budou nějaké překryvy hrozit. Následně umístění všech těchto značek překopírujeme do potomků vždy ze shodného rodiče a poznačíme si, že tyto značky jsme již zpracovali.

Dokud existuje nezpracovaná značka, vždy si nějakou vybereme a opět pro ni nalezneme symbolový uzávěr.

Tvorbu symbolového uzávěru pro konkrétní umístění u z jedince I s ohledem na jedince J tedy můžeme popsat takto:

- 1) Vlož do uzávěru u
- 2) Vezmi dosud nezpracované umístění v z uzávěru; neexistuje-li, skonči
- 3) Nalezni všechna umístění, s kterými by se v překrývalo, kdyby bylo součástí řešení J
- 4) Vezmi umístění stejných značek v I a přidej je do uzávěru (pokud v něm ještě nejsou)
- 5) Jdi na krok 2)

Mohlo by se zdát, že tímto způsobem do uzávěru přidáme velkou většinu značek a křížení tak bude spíš mutací než opravdu křížením. V praxi ale značky na mapě velmi často tvoří velké množství menších shluků, a uzávěr bude rozumně malý.

4.4 Mutace

Mutace navrhneme jen velmi jednoduché. Můžeme očekávat, že i takto jednoduché operátory jako prostředek vylepšování získaných jedinců poslouží dobře.

Posun

Libovolné umístění značky dovolíme posunout. Umístění bodových symbolů, jejich popisků a popisků oblastí umožníme posunout libovolným směrem o malý kousek. Popisky segmentů ovšem dovolíme posunout pouze po směrnici segmentu, který popisují.

Přegenerování umístění

S malou pravděpodobností umožníme u každého umístění značky úplné přegenerování umístění, tedy náhodné vygenerování místa, kam značku umístíme.

Změna varianty

U všech umístění značek, a také u genů odpovídajících požadavkům na jednotlivé sekce umožníme změnu varianty. Typicky tedy půjde o změnu značky, kterou při vykreslení skutečně použijeme; v případě sekcí půjde o informaci, v kterém segmentu je popsaná.

Podotkněme pro úplnost, že použitou variantu volíme náhodně, bez jakékoliv apriorní preference větších variant, zalámaných popisků atp.

4.5 Hodnoticí funkce

Hodnoticí funkci navrhneme tak, aby zohledňovala některá z kritérií. Nebudeme zatím navrhovat složitou funkci zohledňující vše, co se dá hodnotit. Raději navrhneme jednoduchou funkci hodnotící nejdůležitější aspekty. Pokud se umisťování mapových značek pomocí evolučního algoritmu ukáže jako funkční přístup, můžeme hodnoticí funkci dále vylepšovat a rozšiřovat, aby postihla i další kritéria.

Rovnou také řekněme, že nebudeme počítat kladné ohodnocení, nýbrž penalizaci. Výsledek hodnoticí funkce tak bude vyjadřovat, jak moc se výsledná mapa prohřešuje proti kritériím kvality.

Zohledňování více různých kritérií vyřešíme tím, že budeme výslednou penalizaci počítat jakou součet dílčích penalizací. Takový přístup pochopitelně není bezproblémový, jednak mohou být některé dílčí penalizace důležitější než jiné, jednak v určitém smyslu sčítáme jablka a hrušky. Speciálně není jasné, zda stejná hodnota různých dílčích penalizací znamená stejně závažné prohřešení vůči danému kritériu.

Nebudeme proto určovat prostý součet, ale vážený součet dílčích penalizací. To s sebou nese další komplikace. V první řadě to pro nás znamená nutnost stanovit váhy dílčích penalizací, což budeme muset provést experimentálně. Ve druhé řadě nám tím vzniklo riziko, že při budoucím rozšíření hodnoticí funkce bude nutné všechny váhy určit znovu, neboť se tím změní i vzájemná důležitost dosavadních kritérií.

Jinou možností by mohlo být omezení dílčích penalizací na reálné číslo z rozsahu 0-1 a výpočet výsledné penalizace jako součin penalizací dílčích. Dílčí penalizace by představovaly jakýsi výsledek srovnání s nejhorším případem. To by ale nutně vyžadovalo umět právě nejhorší možný případ určit. Uvažujeme-li čistě počet překrývajících se značek, můžeme nejhorší případ určit – překrývají se všechny značky.

Komplikovanější situace by nastala např. při uvažování vzdálenosti skutečného umístění značky od žádaného. Asi bychom zvládli definovat nejhorší případ (třeba jako situaci, kdy mají všechny značky od skutečného umístění vzdálenost odpovídající delšímu rozměru mapy), ale problém by nastal se srovnáním případů. Je horší, když je jedna značka vzdálená 10 cm, když jsou dvě značky vzdálené 5 cm, nebo je to obojí stejně špatné?

Přestože bychom i takové otázky zvládli vyřešit, zůstaňme raději u varianty sčítání penalizací, která je po stanovení vah velmi přímočará.

Nyní se už zaměřme na to, jak budeme určovat jednotlivé dílčí penalizace.

První dílčí penalizací bude *penalizace za překryvy*. Mohli bychom jednoduše spočítat značky, které se překrývají s alespoň jednou další, případně spočítat dvojice značek, které se překrývají. Takový údaj ale říká velmi málo. Obrázek 4.4 zachycuje dvě situace, kdy se překrývají popisky. V prvním případě dochází k překryvu mezi dvěma dvojicemi značek, ve druhém k překryvu mezi jednou dvojicí. První případ bychom tedy při takto jednoduchém určování penalizace označili za výrazně horší.

Ve skutečnosti je ale první případ výrazně čitelnější, a pro mapu pravděpodobně vhodnější. To proto, že oba překryvy jsou jen velmi malé. Naopak ve druhém případě je sice překryv jediný, ale velký. My tedy penalizaci spočítáme jako celkovou plochu překryvů. Určovat ji budeme jako součet překrývajících se ploch

Obrázek 4.4: Rozdílné míry překryvu popisků

každé značky, tedy např. při překryvu tří značek započítáme plochu překryvu třikrát.

Podotkněme, že zajímavým přístupem k určení penalizace za překryvy by mohlo být také určování překrývající se části značky. Při vhodném způsobu kombinování by mohlo podobně jako náš přístup preferovat více malých překryvů před malým množstvím velkých překryvů.

Druhou dílčí penalizací bude *penalizace za neumístění*. Tu vyřešíme velmi přímočaře – za každou značku, která nebude umístěná, započítáme fixní penalizaci. Pro dosažení lepších výsledků by samozřejmě bylo možné neurčovat penalizaci fixně, ale počítat ji např. na základě typu značky či důležitosti značky (kdyby byla specifikovaná na vstupu).

Třetí a poslední dílčí penalizací bude *penalizace za vzdálenost*. Za každou umístěnou značku započítáme penalizaci za vzdálenost skutečného umístění této značky od žádaného. Zdůrazněme, že za neumístěnou značku penalizaci za vzdálenost nezapočítáváme.

Penalizace za vzdálenost by měla reflektovat vzdálenost v tom směru, že při vyšší vzdálenosti bude i tato penalizace výrazně vyšší. Zejména u popisků si snadno můžeme představit, že přiřadit popisek ke značce je při vzdálenosti 4 cm více než dvojnásobný problém než při vzdálenosti 2 cm. Podobně se horší orientace v situaci, kdy bodové symboly neodpovídají realitě. Penalizaci za vzdálenost budeme proto určovat jako druhou mocninu vzdálenosti značky od žádaného umístění.

4.6 Průběh evoluce

Náš algoritmus pojmeme jako generační, v každém kroku tedy nahradíme vždy celou populaci. Abychom nepřišli o nejlepší řešení, uplatníme elitismus a součástí nové generace se vždy stane několik nejlepších jedinců z generace předchozí.

Většinu parametrů evoluce budeme považovat za nastavitelnou, přestože samozřejmě poskytneme výchozí hodnoty. Uživateli tím umožníme experimentovat s výsledky algoritmu a možná dosáhnout lepších výsledků pro specifická data.

Za výchozí stav budeme ovšem považovat rovnoměrné vytváření nové generace křížením a mutací existujících jedinců, s náhodným výběrem ze všech kromě několika nejhorších jedinců. Umožníme mutaci potomků. Evoluce se zastaví po určitém počtu iterací.

5. Implementace

V této kapitole popíšeme implementaci navrženého algoritmu včetně problémů, které se během ní vyskytly, a rozhodnutí, která bylo třeba udělat.

5.1 Vývojová dokumentace

Navržení vhodného algoritmu pro umisťování mapových značek je samo o sobě těžkým problémem, ovšem jeho otestování na reálných datech představuje další velký problém. Je potřeba buď začlenit implementaci algoritmu do existujícího programu, nebo kromě samotného algoritmu implementovat i mnoho dalších vrstev – zpracování reálných dat, skutečné vykreslení značek atp. My jsme se rozhodli vydat první cestou a implementovat navržený algoritmus v rámci existujícího mapového programu.

Kromě samotného zakomponování algoritmu do existujícího programu jsme museli při implementaci řešit další problémy, které popíšeme později v této sekci.

Napojení na program Hic est leo

Jak už bylo zmíněno, rozhodli jsme se implementovat navržený algoritmus v rámci existujícího programu. Programem, který jsme pro to zvolili, je Hic est leo.[1]

Tento program vznikl v červnu 2014 a v současné době je stále ve vývoji (i my jsme během své práce narazili na některé chyby nebo omezení). Přesto se jeví jako velmi šikovný nástroj. Jelikož je malý a jednoduchý, jsou jeho úpravy (zejména typu doplnění nové vrstvy) relativně snadné. Mohli jsme tedy většinu času věnovat skutečně samotnému algoritmu na umisťování značek.

Vstupem tohoto programu jsou jednak mapová data žádoucí oblasti z projektu OpenStreetMap,[2] a to ve formátu XML dumpu, jednak popis stylů, které se mají použít při vykreslování mapy. Syntaxe těchto stylů vychází z MapCSS[3]. Běh programu je řízen pomocí konfiguračního souboru.

Hic est leo je psaný v jazyce C s využitím standardních knihoven a nestandardní, volně dostupné knihovny ucwlib.[4] I naše implementace algoritmu proto využívá jazyk C s touto knihovnou.

Program má několik virtuálních vrstev (modulů), která si postupně předávají a dále upravují zpracovávaná data. Nejprve jsou zpracována mapová data ze souboru XML, tato data se převedou na interní reprezentaci. Dále jsou načteny styly a podle nich jsou vygenerovány značky (resp. opět jejich interní reprezentace), které se mají umístit na mapu. Následně jsou vykresleny.

Přímo o práci se značkami se starají tzv. symbolizery. Symbolizery mají na starost zejména vygenerování značek (rozhodnutí, zda se pro daný mapový objekt bude nějaká značka umisťovat, nastavení přesných hodnot v interní reprezentaci atp.) a jejich skutečné vykreslení do výsledné mapy. Symbolizerů je více druhů, každý z nich určený pro jiný typ značky. Jejich dělení je jemnější, než jaké jsme

[1] Program zatím nemá webovou stránku, je dostupný z repozitáře git://git.ucw.cz/leo.git

[2] http://www.openstreetmap.org/

[3] http://wiki.openstreetmap.org/wiki/MapCSS

[4] http://www.ucw.cz/libucw/

zatím používali my: body, ikony, linie, oblasti, texty, liniové obrázky. Při generování značek je pro každý mapový objekt zavolaný každý symbolizer, který pro něj může a nemusí vygenerovat značky k umístění.

Náš algoritmus je volaný mezi vygenerováním značek a jejich skutečným vykreslením. Kromě jednoduchého zásahu do hlavní části programu bylo potřeba upravit některé symbolizery, aby s existencí umisťovacího algoritmu počítaly, a také je naučit některé nové operace se značkami (visuálně porovnat, vytvořit kopii).

Spojování a segmentace linií

Již několikrát jsme v textu zmínili, že bychom si přáli, aby byly dlouhé liniové symboly popisované vícekrát. Bohužel v mapových datech může být to, co je ve skutečnosti dlouhou linií, reprezentováno jako mnoho kratších.

Specificky v projektu OpenStreetMap má každý objekt přiřazenou nějakou množinu příznaků (tagů). Pokud by se však příznaky linie (v terminologii OSM cesty – posloupnosti bodů, jejichž spojnicemi je tato linie tvořená) v její délce lišily, musí být rozložena na několik menších, kterým bude možné takovou množinu přiřadit. Tedy např. ulice, která je částečně jednosměrná, bude rozdělena minimálně na část, v které jednosměrná je, a část, v které jednosměrná není.

Takové rozdělení se zpropaguje i při generování popisků – vznikne požadavek na popsání jednosměrné i nejednosměrné části.

Prvním úkolem tedy je takto rozdělené linie zrekonstruovat. V dalším popisu si přizpůsobíme terminologii a budeme *linie* říkat rekonstruovaným liniím, *cesty* liniím přímo odpovídajícím vstupním datům a *úseky* jednotlivým rovným spojnicím bodů tvořícím cesty.

Nejprve jsme sestavili graf tvořený všemi úseky ze vstupu. V tomto grafu jsme následně zpracovali každou hranu. Pokud zpracovávaná hrana dosud nebyla součástí žádné linie, zavedli jsme novou linii a tuto hranu označili za její součást. Následně jsme pro tuto linii z tého hrany spustili prohledávání do šířky (po hranách). Pokud prohledávání narazilo na hranu, která dosud nebyla součástí žádné linie a její popisek byl stejný jako popisek aktuální linie, označilo i tuto hranu za součást aktuální linie; v opačném případě dál nepokračovalo.

Výsledné linie jsme následně, v souladu s navrženým algoritmem, rozdělovali na sekce (o nastavitelné maximální délce) a segmenty. Toto zpracování již bylo relativně přímočaré.

Procházeli jsme postupně každou linii od jednoho konce ke druhému, a to po úsecích. Pro každý úsek jsme určili jeho délku a ověřili, zda po jeho přidání nebude současná sekce příliš dlouhá. Pokud ano, hranu odpovídající tomuto úseku jsme rozdělili (tak, aby její první část doplnila sekci právě na maximální délku), a to faktickým rozdělením úseku. Asi se sluší podotknout, že toto místo (tedy zásah přímo do mapových dat) je vhodným kandidátem na budoucí vylepšení.

Dále jsme určili směrnici úseku a zjistili, zda by se do něj vešel otočený popisek. Pokud ano, označili jsme tento úsek za nový segment. V každém případě jsme přičetli délku úseku k současné délce sekce. Tím byl úsek zpracovaný a bylo možné přejít na další.

Situace, kdy nějaká sekce nakonec zůstala bez jediného segmentu (protože všechny úseky byly příliš krátké na to, aby se do nich vešel popisek), jsme zatím řešili jejich odstraněním. Linie tedy v takovém místě vůbec není popsána.

Hledání překrývajících se značek

Mapových značek může být mnoho a často se pro značku hledají její překryvy (vždy při vytváření symbolového uzávěru během křížení a při každém hodnocení jedince). To dává požadavek na rychlé nalezení překryvů.

Samotná detekce překryvů (zejména pouhý fakt, že se značka překrývá s nějakou jinou) by se dala řešit dostatečně jemnou maticí o velikosti mapy, kde by byla poznamenaná informace, kolik značek se v dané části mapy nachází. Taková mapa by ale byla velká a musela by se udržovat pro každého jedince zvlášť.

Předvedeme si proto úspornější řešení, které má při alespoň trochu rovnoměrném rozložení značek také dobrou složitost. Mapu rozdělíme na malé části (o rozměrech v řádu jednotek milimetrů). Každá část bude obsahovat odkazy na všechny značky, které do ní potenciálně zasahují (ty, jejichž maska zasahuje do dané části). Zároveň bude mít každé umístění značky odkazy na všechny části mapy, do kterých (potenciálně) zasahuje.

Při hledání překryvů pak vezmeme všechny části, do kterých značka při současném umístění může zasahovat. Z těchto částí získáme seznam všech značek, jejichž umístění může do některé z těchto částí zasahovat také. Ze seznamu vytřídíme duplicity a následně můžeme ověřit, zda a nakolik se původní značka skutečně překrývá s těmito získanými.

Neumístění značky

Neumístění značky je reprezentováno jako použití speciální varianty (v kódu jako použití varianty s indexem -1). Skrytí ani odkrytí značky není realizováno jako speciální typ mutace, nýbrž jako speciální případ změny použité varianty.

Při změně segmentu, v kterém je popsaná sekce, se všem segmentům této sekce přepisuje použitá varianta. Přepisování explicitní informace o tom, zda je popisek umístěný, nebo ne, zejména umožňuje snazší implementaci kontroly překryvů či výpočtu penalizace za vzdálenost. Není totiž potřeba přizpůsobovat kontrolu toho, zda je značka umístěná, typu značky, a dohledávat tuto informaci prostřednictvím značky jiné.

Do budoucna ovšem může stát za zvážení např. reprezentovat neumístění obecně zápornou hodnotou. Pravděpodobně by pak bylo nutné implementovat explicitní skrytí a odkrytí značky, ale bylo by možné mimo jiné uchovat informaci o vybrané variantě pro segment, v kterém aktuálně sekce není popsaná.

Rotace popisků segmentů

Standardně je rotace popisku segmentu odvozena od směrnice tohoto segmentu. U svislých úseků bychom ale mohli narazit na to, že pro ně není směrnice definovaná, a problémy bychom mohli potkat také u úseků, které jsou téměř svislé a hodnoty jejich směrnice se blíží nekonečnu.

U „příliš svislých“ segmentů je proto rotace rovnou nastavena na úplné otočení o 90 stupňů a směrnice se vůbec nepoužívá.

5.2 Popisy využívaných struktur

Nyní předvedeme některé zavedené datové struktury a přiblížíme smysl jejich nejdůležitějších polí.

request

K základním strukturám patří `request`, tedy struktura reprezentující požadavek na umístění nějaké značky.

V poli `type` je uložený typ požadavku, tedy buď bod, oblast, linie, sekce, nebo segment. Podotkněme, že popisek ve výčtu chybí záměrně – odlišení symbolů a popisků probíhá na základě typu značky, resp. typu symbolizeru, který ji vygeneroval.

Pole `ind` slouží k uložení indexu požadavku. Ten je požadavkům generovaný postupně tak, jak přicházejí. Určuje pozici genu v chromozomu (a jako takový se občas uplatňuje při udržování informace, které požadavky již byly zpracovány a která ne).

Poslední položka, `variants`, představuje pole všech možných varint umisťované značky.

Podle typu požadavku jsou bližší informace o požadavku uloženy v příslušné struktuře `request_*`. Ty obvykle obsahují i položky `sym` či `label`, což je odkaz na strukturu popisující umisťovanou značku, a položku `zindex` potřebnou pro pozdější vykreslení symbolu, resp. potřebnou při předávání symbolu k vykreslení. V budoucnu může stát za zvážení přesun této informace do struktury reprezentující značku.

request_point

Reprezentace bodových značek obsahují kromě obvyklých částí položky `x` a `y` sloužící jako referenční souřadnice, tedy souřadnice, na která se má značka v ideálním případě umístit.

request_segment

Požadavky na popisek segmentu obsahují jednak souřadnice krajních bodů (`x1`, `y1`, `x2`, `y2`), jednak hodnotu směrnice v poli `slope`. U segmentů, které jsou při segmentaci linií vyhodnoceny jako příliš svislé, je tato hodnota nastavena na magickou konstantu 142.

request_area

Požadavky na popisek oblasti obsahují v polích `cx`, `cy` souřadnice středu oblasti, ty slouží jako referenční. Navíc mají tyto požadavky v poli `o` odkaz na reprezentaci mapového objektu odpovídajícího dané oblasti. Můžeme totiž předpokládat, že v budoucnosti se umístění popisku bude hodnotit podrobněji, mimo jiné s ohledem na polohu uvnitř/vně oblasti či protínání hranic.

variant

Jak jsme zmínili, součástí požadavku je pole možných variant značky. Tyto varianty obsahují informaci o šířce (`width`) a výšce (`height`) příslušné masky, dále tuto masku (`bitmap`) a také informaci o žádoucím horizontálním (`x_offset`) a vertikálním (`y_offset`) posunutí levého horního rohu masky vůči referenčním souřadnicím.

placement

Hodnotami jednotlivých genů jsou struktury `placement` obsahující podrobné informace o umístění příslušných značek. Nejdůležitějšími prvky těchto struktur jsou jednak pole `x` a `y` udávající souřadnice skutečného umístění značky, jednak pole `variant_used` udržující informaci o vybrané variantě. Jak už bylo zmíněno dříve, hodnota -1 v tomto poli znamená, že značka momentálně není vůbec umístěna.

map_part

Struktury pro části mapy obsahují v první řadě seznam značek, které do těchto částí mohou zasahovat (`placements`). Krom něj mají uložený (`placements`) také svůj index (`ind`) v rámci celé mapy, resp. v rámci sítě těchto jejích částí. Uložení indexu umožňuje jednoduché dohledání stejných či sousedních částí ve stejném či jiném jedinci.

individual

Reprezentace jedince je velmi přímočará – obsahuje odkaz na jeho chromozom (`placements`), dále pole částí mapy (`map`) a současnou penalizaci jedince (`penalty`).

graph_edge

Spolu s `graph_node` patří `graph_edge` mezi interní struktury části implementace zabývající se předzpracováním linií (tedy jejich spojováním a segmentací), obě slouží k reprezentaci grafu, na kterém se rekonstruují a segmentují linie.

Důležitými prvky hranové struktury jsou pochopitelně její krajní body (`n1`, `n2`). Během zpracování grafu se hojně využívá pole `visited` udržující informaci, v které iteraci (při rekonstrukci které linie) byla tato hrana naposledy navštívena. Uložení této informace přímo v podobě čísla iterace, nikoliv jako příznak navštívení v současné iteraci, umožňuje nenulovat tuto hodnotu u každé hrany po skončení prohledávání. Kdybychom nulovat museli, nemohli bychom se vyhnout kvadratické složitosti.

Číslo rekonstruované linie, do které byla hrana zařazena, se udržuje v poli `longline` (zejména v komentářích můžeme občas potkat označení této rekonstruované linie jako *logical line*). Při zařazení do linie se hraně nastavuje také `dir`, tedy její směr v rámci linie. Jinými slovy toto pole říká, zda při zpracování celé linie máme za první bod tohoto úseku považovat `n1`, nebo `n2`.

V průběhu zpracování jsou také nastavené hodnoty `prev` a `next`, tedy předchozí a následující hrana ve stejné linii.

6. Používání programu

K vyzkoušení algoritmu si musíme opatřit program Hic est leo rozšířený o implementaci umisťovacího algoritmu. To v současné době znamená naklonovat si repozitář a program zkompilovat ze zdrojových kódů ve větvi umisťování.

Shrňme napřed požadavky: k používání Hic est leo je zapotřebí podpora jazyka C, dále knihovny LibUCW, LibUCW-XML, LibUCW-charset, PangoFT2 a FreeType. Pro snadnou konfiguraci je nutná také podpora jazyka Perl.

S ohledem na nutnost naklonovat si repozitář vzniká také požadavek na přítomnost gitového klienta.

Získání programu a jeho příprava k používání probíhá v následujících krocích:

- 1) Naklonování repozitáře

  Repozitář programu je dostupný na adrese: *git://git.ucw.cz/leo.git*

  Při použití gitu z terminálu je možné pořídit si do pracovního adresáře kopii repozitáře příkazem:

  ```
  git clone git://git.ucw.cz/leo.git
  ```

- 2) Přepnutí na větev umisťování značek

  Jelikož změny nejsou sloučené do hlavní větve, je potřeba se přepnout do větve *labelling*, v které je umisťování značek vyvíjeno. Z terminálu tedy:

  ```
  git checkout labelling
  ```

- 3) Konfigurace instalace

  Před samotnou instalací je potřeba spustit konfigurační skript. Ten také otestuje, že jsou k dispozici všechny potřebné knihovny. Z terminálu ho spustíme následovně:

  ```
  ./configure
  ```

- 4) Překlad programu

  Nyní již stačí přeložit (a slinkovat) program. K dispozici je Makefile, který celý proces řídí, stačí tedy využít nástroj make. V terminálu píšeme:

  ```
  make
  ```

Hotovo, nyní máme program připravený k použití. Součástí repozitáře je také vzorový konfigurační soubor, `map.cf`, a využitelný soubor stylů `poskole.css`.

Pro skutečné použití si ale potřebujeme pořídit data, na kterých program poběží. Asi nejjednodušším způsobem pořízení dat je navštívení internetových map OSM na adrese `http://www.openstreetmap.org/`. Na mapě si najdeme tu část, která by nás zajímala. Následně můžeme v horní liště vybrat záložku Export. V levé části stránky se objeví informace o vybrané části – minimální a maximální zeměpisná šířka a délka exportovaného obdélníku. Pokud chceme jinou oblast, případně si podle souřadnic neumíme oblast představit, můžeme využít odkazu hned pod touto informací *Manually select a different area*. V takovém případě se na mapě objeví vyznačení exportovaného obdélníku s možností přetahování

rohů, a tedy změně vybrané oblasti. Nakonec stačí kliknout na tlačítko Export a stáhnout vygenerovaný soubor.

Podotkněme, že známe-li hraniční souřadnice, můžeme výstup stáhnout přímo, např. pomocí nástroje wget, z adresy následujícího typu:
http://api.openstreetmap.org/api/0.6/map?bbox=14.532,50.668,14.540,50.673.

Budeme-li chtít stáhnout data o větší oblasti, může nám náš pokus selhat s vcelku nicneříkající chybou Bad request (špatný požadavek). V tom případě je možné využít např. OverpassAPI na adrese *http://overpass-api.de/*.

V konfiguračním souboru musíme nastavit promítnuté souřadnice oblasti, resp. okrajů oblasti, pro kterou chceme mapu vytvořit. Tato oblast nemusí být shodná s oblastí, pro kterou jsme stáhli mapová data, je technicky možné např. stáhnout data pro celou Prahu a pak si postupně nechat generovat plánky malých částí města.

K možnostem, jak takové promítnuté souřadnice získat, patří použití nástroje `proj`. Přímo v konfiguračním souboru je uvedeno, jaké parametry mu předat, tedy se může volat následovně:

```
proj +proj=utm +zone=33 +ellps=WGS84
```

Ač by to mělo být z podstaty věci jasné, raději zdůrazněme, že se zadávají souřadnice bodu, tedy se musí zadat zároveň zeměpisná šířka i délka. Upozorněme také, že `proj` očekává jako první zeměpisnou délku. Pokud bychom chtěli zadávat první zeměpisnou šířku, musíme `proj` volat s přepínačem `-r`.

Již volitelně můžeme v konfiguračním souboru nastavit jméno souboru s mapovými daty a také jméno souboru definujícího styly.

Dále můžeme nastavit různé parametry výsledné mapy a rozmisťovacího algoritmu, tyto parametry jsou dokumentovány komentáři přímo ve vzorové konfiguraci.

Máme-li nakonfigurováno, můžeme pustit program. Jako argument mu musíme předat jméno konfiguračního souboru, tedy pokud jsme pouze upravovali výchozí, voláme:

```
./run/bin/leo map.cf
```

Program během svého běhu vypisuje informační hlášení a upozornění na chyby. Hlášky typu „Multipolygon:(číslo): Boundary not closed“ znamenají, že při zpracování oblasti program narazil na neukončenou hranici – velmi neformálně řečeno, „někde mu chybí kus čáry“. To se snadno může stát, pokud oblast leží částečně na mapě, částečně mimo ni.

Výsledná mapa se zapíše jako soubor SVG pod názvem specifikovaným v konfiguračním souboru, při výchozím nastavení tedy jako `output.svg`. Je možné si ji prohlédnout v nějakém prohlížeči SVG, případně exportovat do jiného formátu, např. PDF.

7. Experimenty

V této kapitole popíšeme několik experimentů, které jsme provedli po implementaci našeho algoritmu. Ozkoušíme různé hodnoty parametrů a budeme sledovat, jak se mění jednak penalizace v populaci, jednak výsledné mapy. Výsledné mapy následně zhodnotíme vzhledem ke kritériím stanoveným v sekci 1.2 a nakonec neformálně porovnáme s mapami generovanými jinými nástroji.

Data, se kterými jsme při testování pracovali, jsou jako příslušné OSM dumpy dostupné na přiloženém CD. Stejně tak na CD poskytujeme vždy i vzorový konfigurační soubor pro dané testování a k němu soubor `.values` popisující hodnoty měněné mezi jednotlivými měřeními. Každé měření bylo opakováno dvacetkrát.

7.1 Váhy dílčích penalizací

Jedním z problémů, které představuje skládání celkové penalizace z více dílčích, je nutnost vyvážit tyto dílčí penalizace. My nyní vyzkoušíme, jaký vliv mají váhy dílčích penalizací na výsledné mapy.

Necháme vygenerovat mapy se čtyřmi různými nastaveními vah dílčích penalizací. První tři budou jednu z penalizací započítávat výrazně větší vahou než ostatní. Poslední bude započítávat všechny dílčí penalizace stejnou vahou. U vygenerovaných map vizuálně zhodnotíme, jak moc se toto nastavení vah projevuje a co způsobuje.

Při měření jsme používali OSM dump `vysehrad.osm` a konfiguraci `vahy.cf`.

Nejvíce se odlišují mapy s vysokým postihem za neumístění značky, na těch je typicky umístěno výrazně více značek než na ostatních mapách. Jednou z oblastí, kde je tento jev dobře vidět, je oblast samotného Vyšehradu. Obrázky 7.1, 7.2, 7.3 a 7.4 zachycují tuto oblast na reprezentativních mapách při každém ze zkoušených nastavení vah dílčích penalizací.

Jiným místem, kde se zejména postih za neumístění značky projevuje výrazně, jsou podolské koleje. Ty jsou zobrazené na výsecích z map na obrázcích 7.5, 7.6, 7.7 a 7.8. Můžeme si také všimnout, že přikládání větší váhy penalizaci za vzdálenost mělo při našem měření jen malý význam.

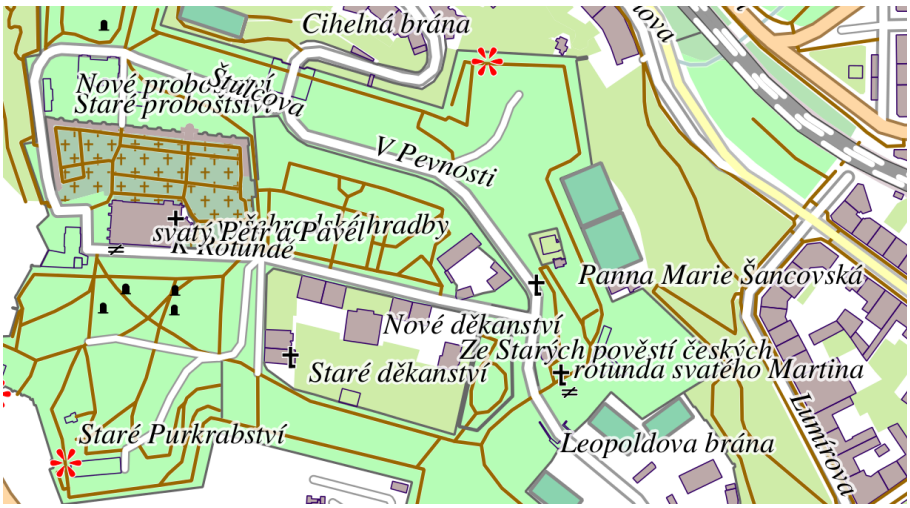
Výřez na obrázku 7.6 je záměrně větší, můžeme si na něm všimnout absurdního posunutí popisku ulice Pod Děkankou.

Zdá se tedy, že při vyvažování parametrů se projevují již menší změny váhy penalizace za neumístění značky, ale až mnohem větší změny váhy penalizace za vzdálenosti.

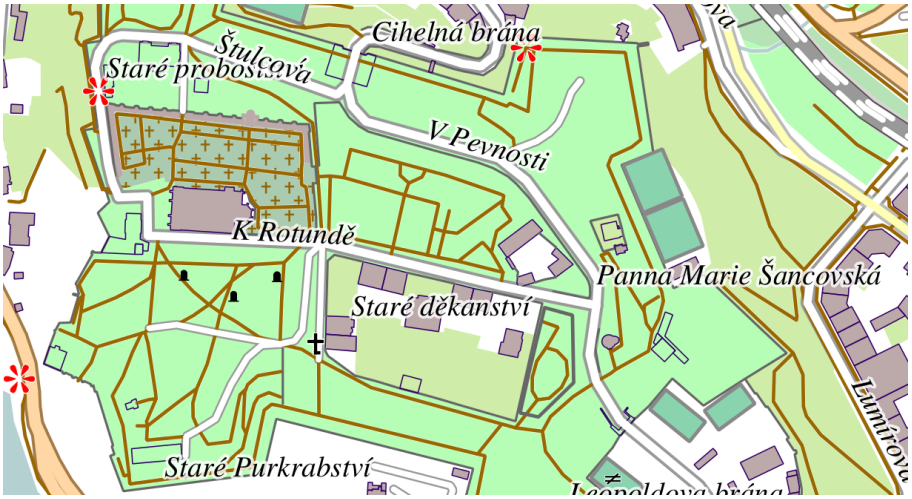
7.2 Poměr křížení a mutace

V sekci 4.6 jsme se rozhodli, že novou generaci budeme vytvářet rovnoměrně mutací a křížením a že uplatníme elitismus, abychom nepřicházeli o dobrá řešení.

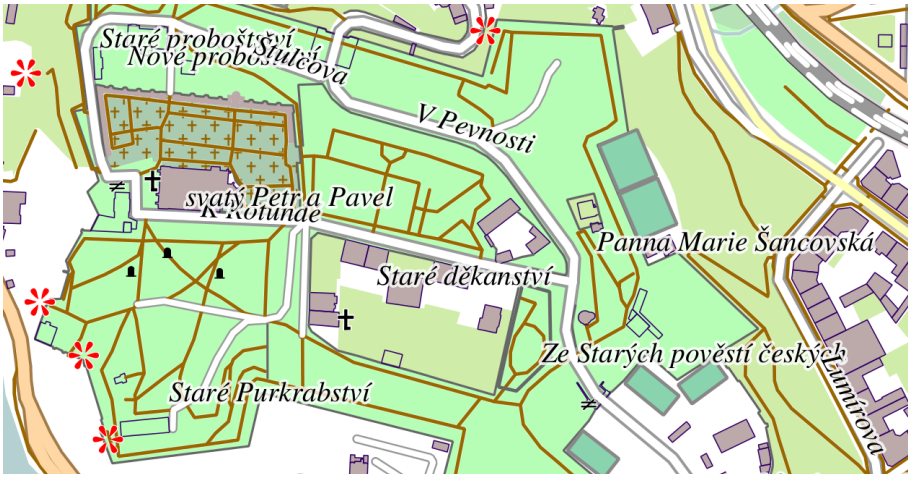
Nyní ovšem otestujeme, jak se vyvíjí penalizace v populaci při různě nastavených poměrech jedinců vzniklých mutací a křížením (vždy však s 10% elitismem, kdy je 10 % nejlepších jedinců původní generace překopírováno do nové). Pomocí mutace a křížení tak vytváříme 90 % nové populace.



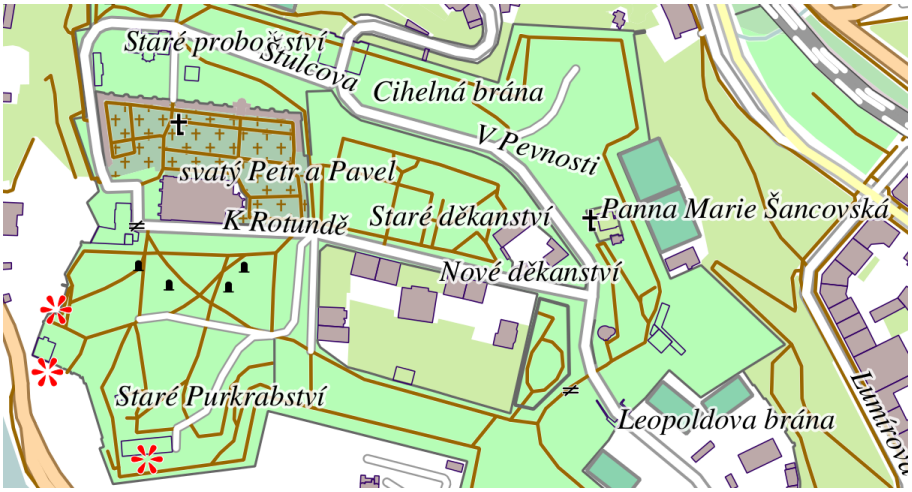
Obrázek 7.1: Oblast Vyšehradu při výraznějším postihu neumístění značky



Obrázek 7.2: Oblast Vyšehradu při výraznějším postihu překryvů

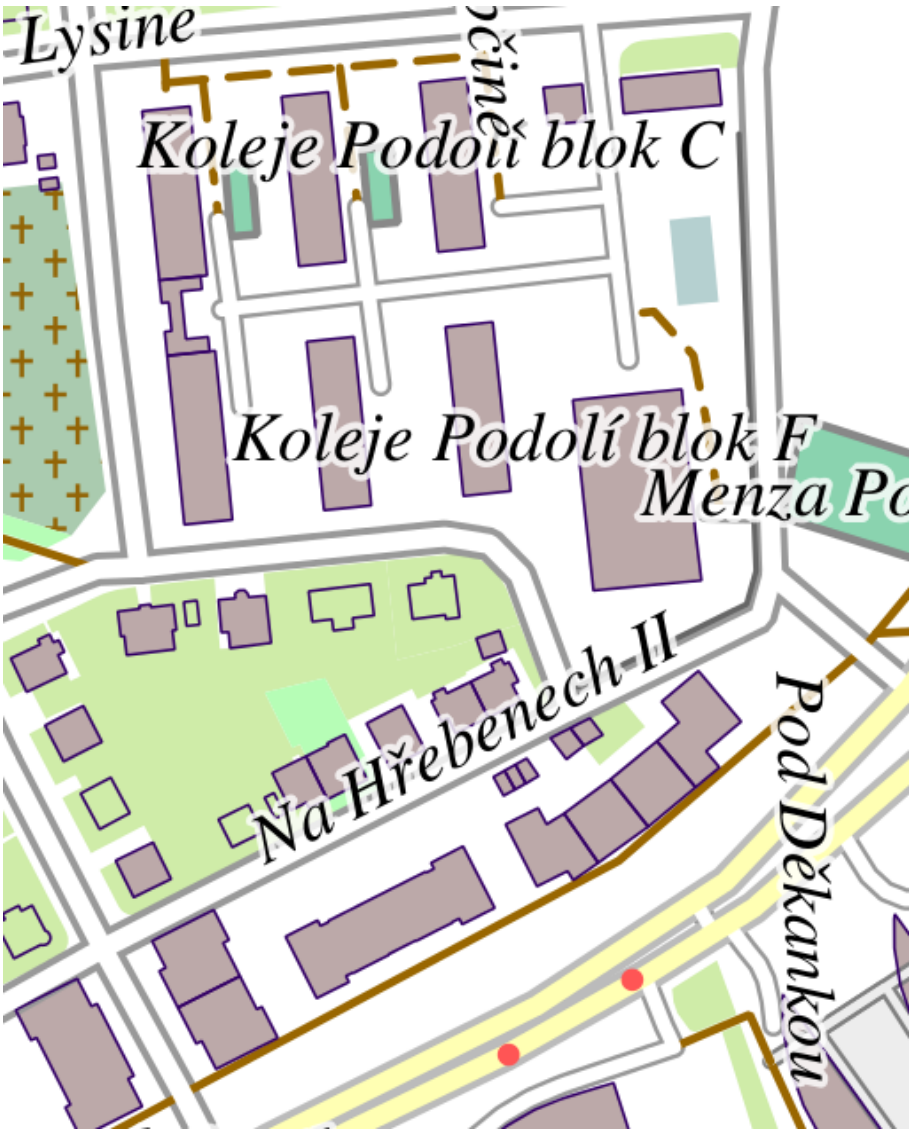


Obrázek 7.3: Oblast Vyšehradu při výraznějším postihu vzdáleností



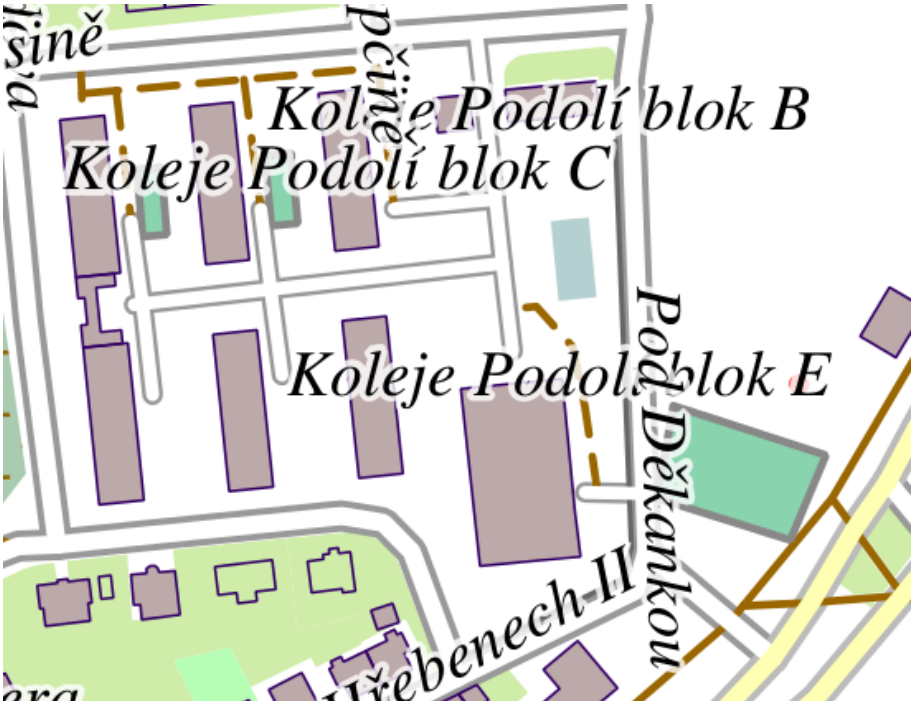
Obrázek 7.4: Oblast Vyšehradu při vyváženém postihu

Obrázek 7.5: Oblast podolských kolejí při výraznějším postihu neumístění značky



Obrázek 7.6: Oblast podolských kolejí při výraznějším postihu překryvů

Obrázek 7.7: Oblast podolských kolejí při výraznějším postihu vzdáleností



Obrázek 7.8: Oblast podolských kolejí při vyváženém postihu

Při měření jsme používali OSM dump `vysehrad.osm` a konfiguraci `pomery.cf`.

V grafu 7.9 je zachycený průběh minimální, průměrné a maximální penalizace při námi zvoleném poměru (mutace 45 %, křížení 45 %) během prvních 200 iterací. Grafy 7.10 a 7.11 tento průběh zachycují v situaci, kdy novou generaci vytváříme pouze pomocí mutace, resp. pouze pomocí křížení.

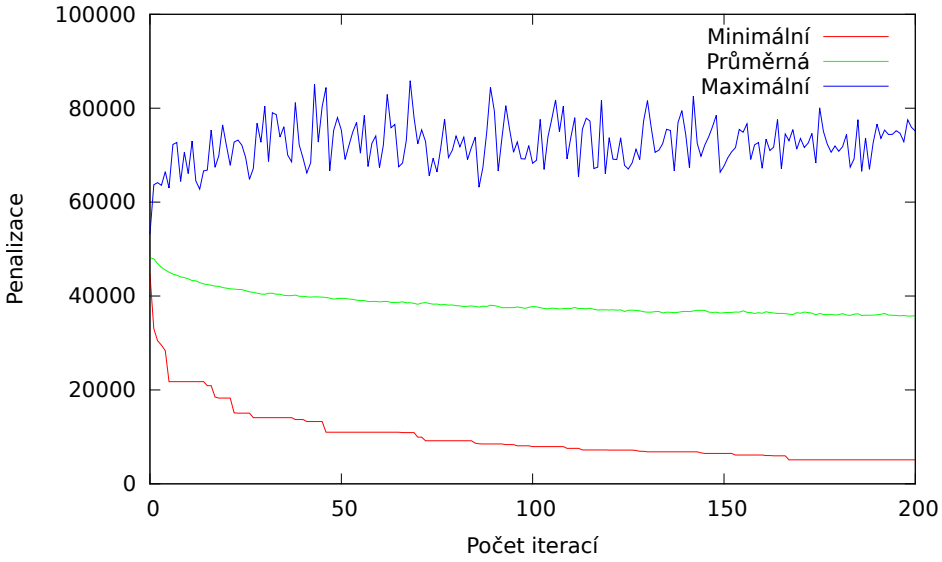
Můžeme si všimnout, že při omezení na mutace se penalizace v populaci snižuje velmi pomalu. Při rovnoměrném vytváření jedinců mutací i křížením se rychle sníží minimální penalizace, zatímco průměrná klesá pomalu a objevují se velké výkyvy u maximální. Naopak při vytváření jedinců výhradně křížením rychle klesá jak minimální, tak i průměrná a maximální penalizace. Celá populace tedy konverguje k nějakému řešení. Naopak v případě, kdy jedinci vznikají oběma způsoby, je zachována větší rozmanitost populace, a tedy i šance vylepšit geny, ve kterých se nejlepší jedinci shodují.

Pro zajímavost je uvedený také graf 7.12, který ukazuje výsledný stav minimální a průměrné penalizace po 200 iteracích při různých poměrech křížení a mutace, resp. v závislosti na míře zastoupení křížení v tvorbě nové generace.

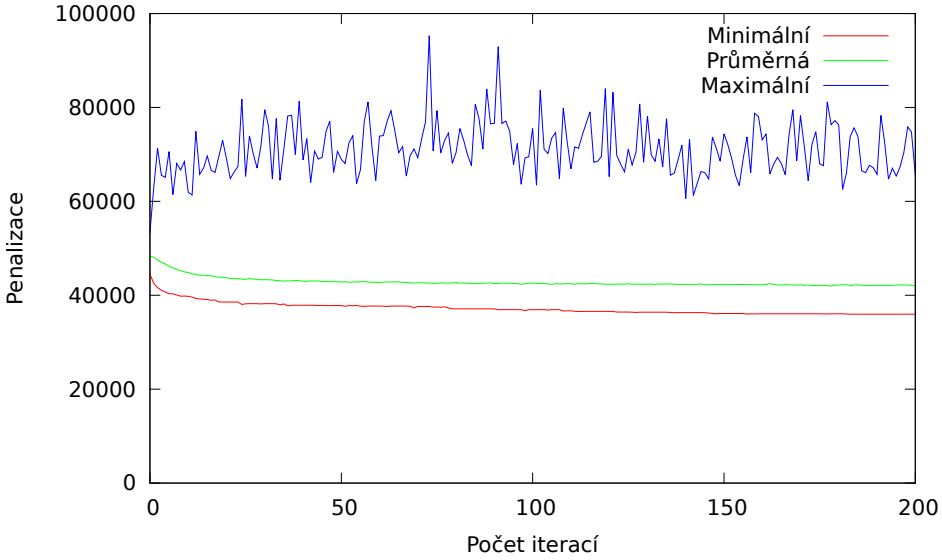
7.3 Velikost populace

Velikost populace má podstatný vliv na výkon evoluce. Příliš malá populace by neumožnila konvergenci k vhodnému řešení, protože by nebylo možné zachovat dostatečnou rozmanitost. Velikost populace navíc ovlivňuje rychlost konvergence. Prozkoumáme proto, jak si náš algoritmus při jinak fixních parametrech vede s různě velkými populacemi.

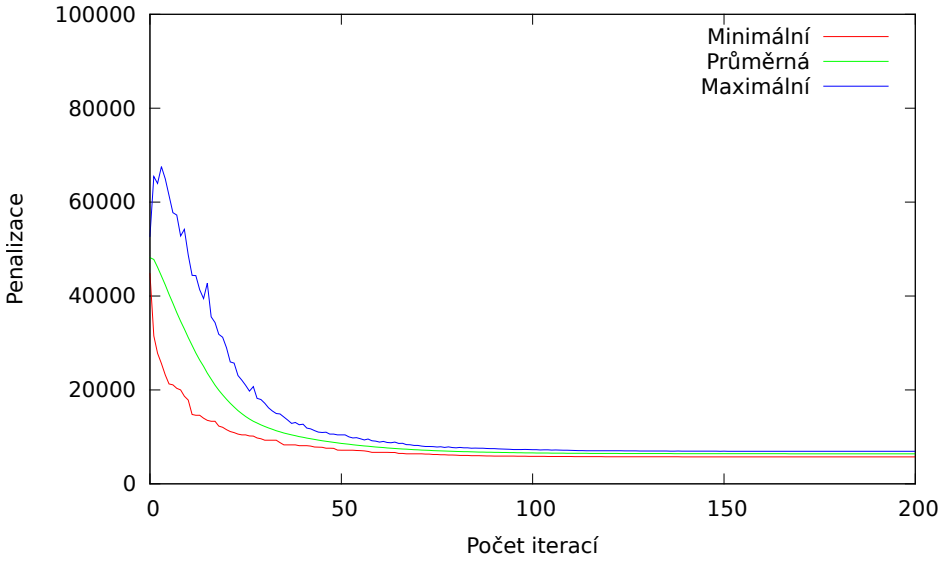
Při měření penalizací jsme používali OSM dump `vysehrad.osm` a konfiguraci `populace.cf`.



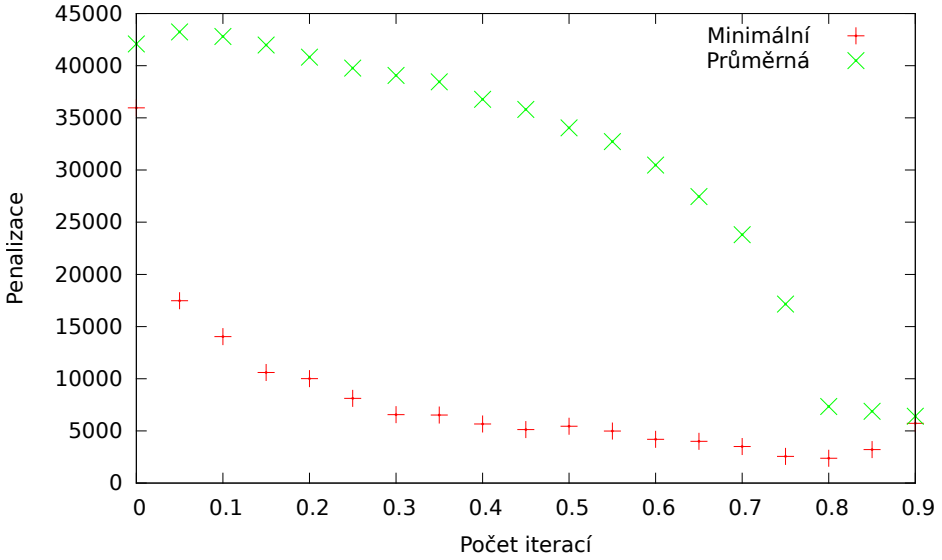
Obrázek 7.9: Průběh penalizace při rovnoměrném zastoupení křížení a mutace



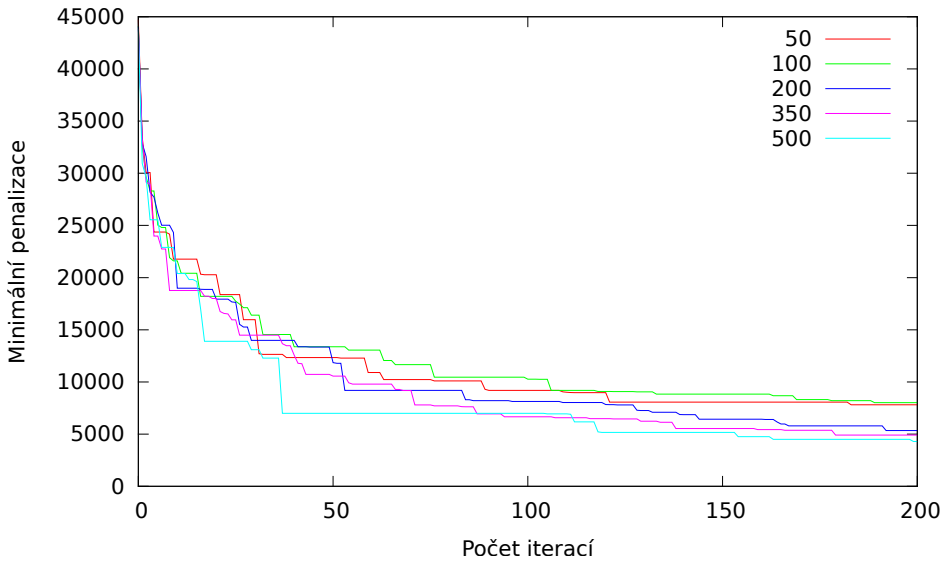
Obrázek 7.10: Průběh penalizace při výhradním zastoupení mutace



Obrázek 7.11: Průběh penalizace při výhradním zastoupení křížení



Obrázek 7.12: Penalizace po 200 iteracích při různém zastoupení křížení



Obrázek 7.13: Minimální penalizace v závislosti na iteraci pro několik velikostí populace

V grafech 7.13 a 7.14 jsou zachycené průběhy minimální a průměrné penalizace v prvních 200 iteracích pro zvolené velikosti populace.

7.4 Vizuální hodnocení map

V této části práce vygenerujeme mapy pro několik míst a zhodnotíme je s ohledem na kritéria ze sekce 1.2. Tyto mapy generujeme za použití konfigurace `mapy.cf`.

Vyšehrad

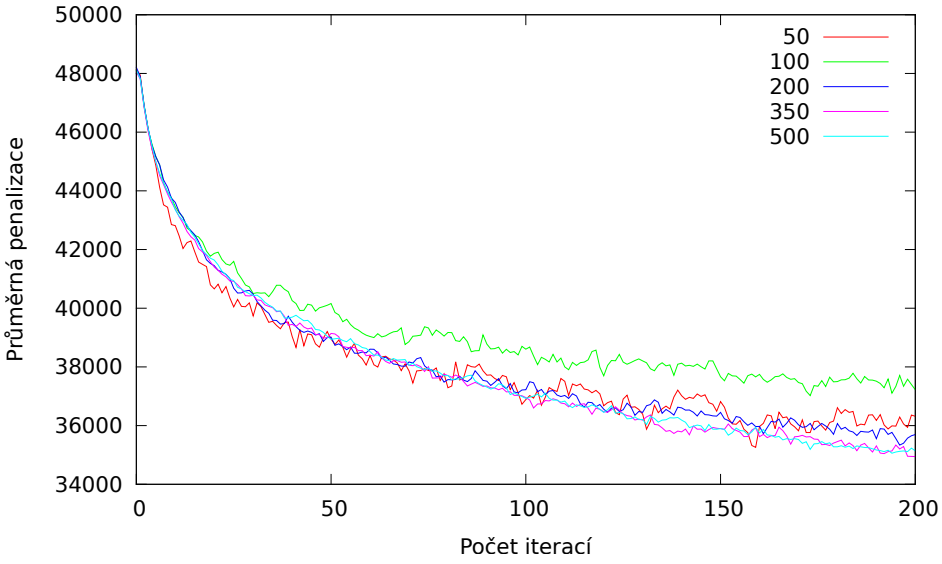
První mapou, kterou budeme hodnotit, je mapa Vyšehradu a jeho okolí. Pro vytvoření mapy jsme použili OSM dump `vysehrad.osm`. Celá výsledná mapa je uložená na přiloženém CD, a to pod názvem `vysehrad.pdf`.

Mapa vypadá na první pohled přijatelně. Při bližším zkoumání na ní objevíme větší množství prohřešků, z nichž některé okomentujeme podrobněji, přesto je jistě možné mapu použít k jejímu účelu.

Na obrázku 7.15 vidíme osamocený symbol. Mohli bychom předpokládat, že v průběhu evoluce došlo ke skrytí jeho popisku – faktem ale je, že mapová data žádný popisek pro tento kostel neobsahují. Naopak na obrázku 7.16 vidíme popisek, který jako by nepatřil k žádnému symbolu. V tomto případě skutečně došlo k tomu, že příslušný symbol nebyl umístěn.

Několikrát si můžeme všimnout stejných popisků blízko sebe, příklady jsou zachyceny na obrázcích 7.17 a 7.18. Na vině jsou částečně mapová data (např. Nuselský most je reprezentovaný jako dvě linie, a jelikož na sebe tyto dvě linie nejsou nikdy napojeny, naše předzpracování je nespojí), částečně absence nějakého sjednocování popisků v našem algoritmu.

Navzdory penalizacím se na mapě několikrát vyskytují překryvy značek, příklad můžeme nalézt na obrázku 7.19. Celkově ale překryvů vidíme na mapě málo,



Obrázek 7.14: Průměrná penalizace v závislosti na iteraci pro několik velikostí populace

což musíme hodnotit kladně. Méně kladně už budeme hodnotit relativně časté vytečení popisku segmentu z tohoto segmentu, jak můžeme vidět na obrázku 7.20.

Zarazit nás může také chybějící popisek největší budovy v areálu České televize, zvlášť když ostatní budovy jsou popsané. Situaci si můžeme prohlédnout na obrázku 7.21.

Naopak zcela profesionálně je označena ulice 5. května. Ukázka by se ale do této práce buď vůbec nevešla, nebo by se kvůli zmenšení stala nejasnou a nečitelnou. Zde tedy pouze odkazujeme na vlastní soubor mapy na přiloženém CD.

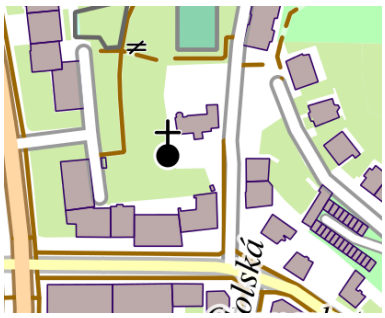
Mapa používá běžné značky i běžné barvy např. pro oblasti a jistě ji můžeme považovat za srozumitelnou. Na mapě se vyskytuje několik překryvů, které ale nebrání čitelnosti mapy. Písmo některých konkrétních popisků částečně splývá s okolím, přesto lze mapu označit za dobře čitelnou.

Jak už jsme párkrát zmínili, překryvy nejsou úplně vyloučené, ale na mapě jich je pouze málo. Některé značky jsou posunuté vůči realitě, ale případné posuny se vyskytují u objektů, které by i tak měly být v realitě jednoznačně identifikovatelné a jejichž nalezení by nemělo představovat problém. Přiřazení popisků je obvykle jednoznačné, v několika případech téměř jednoznačné. Opakování popisku skutečně usnadňuje identifikaci ulic.

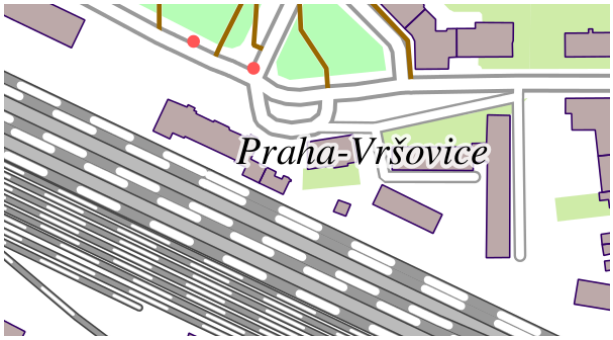
Celkovou estetičnost mapy lze hodnotit kladně.

Když se podíváme na webové mapy OSM, zjistíme, že např. nepopsaným symbolem trpí také. Nepopsaný symbol je zachycený na obrázku 7.22. Mnohé oblasti, u kterých na naší mapě dochází k malým překryvům popisků, na webových mapách popsané nejsou, toho si můžeme všimnout na obrázku 7.23. Na obrázku 7.24 můžeme pozorovat, že k mnoha výskytům stejných popisků při srovnatelné podrobnosti nedochází. Jak je ale znázorněno na obrázku 7.25, při přiblížení oblasti jsou i zde důsledně popsány všechny ulice.

Již v sekci 2.1 jsme zkonstatovali, že na Mapách.cz je výrazně vidět jiné zamýšlené využití. Přesto ve srovnání s naší mapou upozorněme na jednu odlišnost. Na Mapách.cz je popisek Nuselského mostu uveden jen jednou společně pro oba



Obrázek 7.15: Nepopsaný symbol



Obrázek 7.16: Popisek bez příslušného symbolu

Obrázek 7.17: Zdvojený popisek

Obrázek 7.18: Mnoho popisků se stejným smyslem

Obrázek 7.19: Nežádoucí překryvy

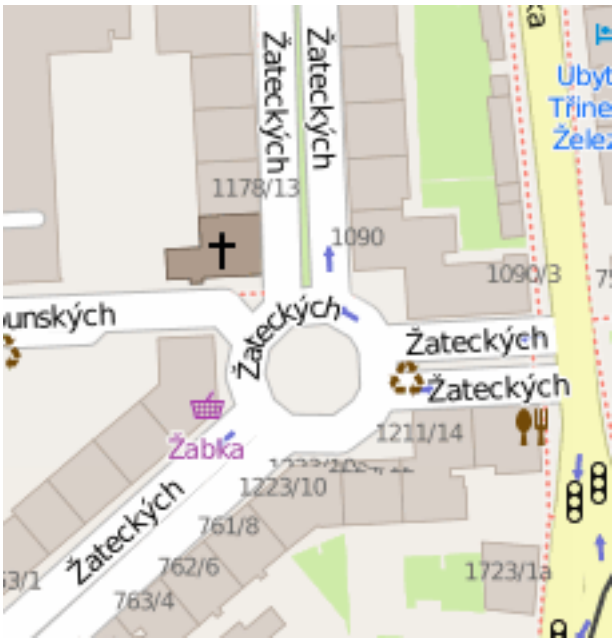
Obrázek 7.20: Vytečení popisku segmentu

Obrázek 7.21: Areál české televize s nepopsanou největší budovou

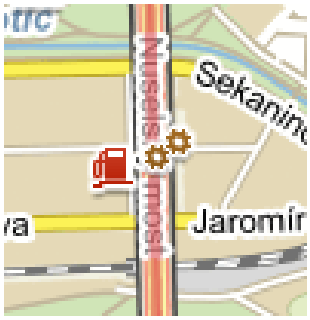
Obrázek 7.22: Mapy OSM: Nepopsaný symbol

Obrázek 7.23: Mapy OSM: Nepopsané objekty

Obrázek 7.24: Mapy OSM: Zobrazení jen některých popisků ulic se shodným jménem



Obrázek 7.25: Mapy OSM: Zobrazení všech popisků ulic se shodným jménem



Obrázek 7.26: Mapy.cz: Sjednocený a částečně překrytý popisek Nuselského mostu

Obrázek 7.27: Mapy Google: Absence znázornění budov

jízdní pruhy, jak se můžeme podívat na obrázku 7.26. Jeho čitelnost je ovšem snížena vyznačením trasy metra, popisek je navíc částečně překryt dalšími symboly.

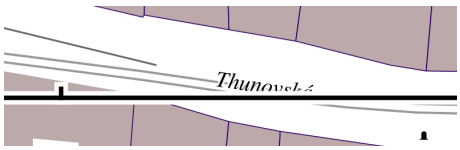
Srovnávat naši mapu s mapou z map Google příliš nelze, neboť na srovnatelné úrovni zobrazení ještě nejsou znázorněny jednotlivé budovy, jak zachycuje obrázek 7.27.

Malostranské náměstí

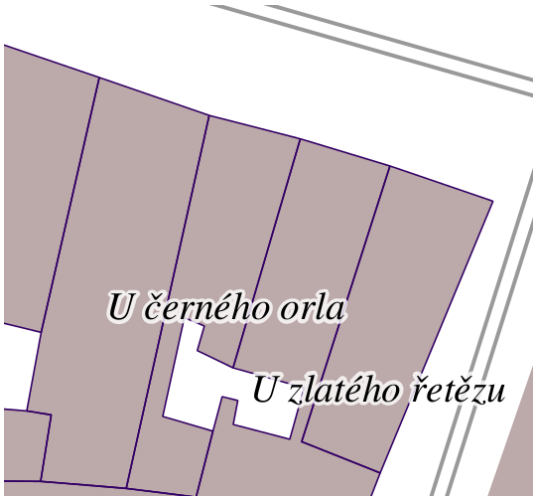
Druhou mapou, kterou zhodnotíme, je mapa nejbližšího okolí malostranské budovy MFF UK. Použili jsme dump `mff-ms.osm`, výsledná mapa je přiložená na CD apojmenovaná `mff-ms.pdf`.

Program mapu vytvořil velmi jednoduchou. Díky malému množství použitých symbolů je spolehlivě srozumitelná. Jediný překryv, ke kterému dochází, je překryv popisku ulice s měřítkem mapy, jak je naznaceno na obrázku 7.28. Podotkněme, že umístění měřítka není věcí naší implementace, ale jiné části programu. Při pominutí tohoto překryvu je mapa velmi dobře čitelná.

Popisky bodových symbolů jsou velmi dobře spojitelné s příslušnými symboly. Bohužel některé popisky se správně přiřadit nedají, příklad uvádíme na obrázku 7.29. To je dané zejména tvarem oblastí – většina z nich je příliš úzká na to, aby se do nich dal popisek umístit. Jelikož v současném návrhu navíc nepostihujeme vytečení z oblasti, není z hlediska programu takové umístění špatné. Všechny nástroje, s kterými náš algoritmus srovnáváme, tedy webové mapy OSM, Mapy.cz a mapy Google uplatňují lámání popisků, a proto jsou výsledné mapy jednoznačnější. Samotné umístění popisků je však i na naší mapě dobré.



Obrázek 7.28: Kolize s měřítkem mapy



Obrázek 7.29: Problematické popisky oblastí

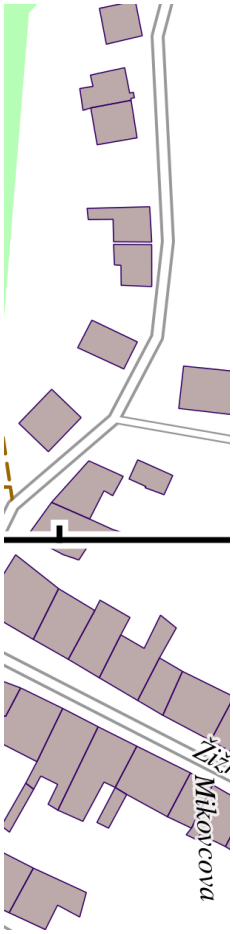
Centrum České Lípy

Třetí mapou, kterou zhodnotíme, je mapa centra České Lípy. Použili jsme dump `cl.osm`, výsledná mapa je přiložená na CD a pojmenovaná `cl.pdf`.

Pro tuto mapu platí velmi podobné hodnocení jako pro mapu předchozí. Díky tomu, že je na mapě celkově méně malých oblastí, tu navíc nedochází k velkému množství vytékání popisků z oblasti, a tedy je celá mapa jednoznačnější. Přesto uvádíme příklad nejednoznačného umístění na obrázku 7.30. Na obrázku 7.31 si můžeme všimnout absurdního vytečení popisku ulice z příslušného segmentu.

Mapu můžeme označit za přijatelnou. Přesto doporučíme další experimenty s penalizováním vzdáleností, aby se zabránilo právě případům z obrázku 7.31.

Obrázek 7.30: Problematický popisek oblasti



Obrázek 7.31: Extrémně posunutý popisek ulice

Závěr

Navrhli jsme evoluční algoritmus pro umisťování mapových značek – náročný, a přesto nevynechatelný proces tvorby map. Na rozdíl od dosavadních prací na toto téma jsme se neomezili na popisování bodových symbolů, ale umisťovali jsme popisky všech typů symbolů, a navíc také přímo bodové symboly.

Navržený algoritmus jsme implementovali jako novou součást existujícího programu pro kreslení map a následně otestovali na reálných datech.

Zjistili jsme, že algoritmus produkuje přijatelné a dobré mapy, srozumitelné a čitelné. Všimli jsme si, že v některých případech se na mapě objeví popisek velmi daleko od svého místa, ale jinak jsou popisky v rámci výsledných map umístěné dobře.

Z hodnocení vytvořených map a srovnání s jinými nástroji vyplynulo, že největší slabinou našeho algoritmu jsou popisky malých oblastí. Náš algoritmus neumí zalamovat popisky. Pokud ovšem umístí delší popisek do úzké oblasti a tato oblast sousedí s dalšími, stane se přiřazení popisku nejednoznačným.

Algoritmus již má podporu pro více variant značky, nabízí se tedy např. možnost doplnit předzpracování, kdy se k popisku vygenerují také různě zalámané varianty.

Máme tedy důvod se domnívat, že jsme vytvořili dobrý základ k případnému dalšímu rozšiřování. V této chvíli náš algoritmus produkuje přijatelné až dobré mapy, po zohlednění více kritérií v hodnoticí funkce a lepším využití potenciálu algoritmu (např. využití podpory více variant) může produkovat mapy dobré až skvělé.

Možná budoucí rozšíření

V úplném závěru této práce ještě nabídneme několik nápadů na možná rozšíření a obecně další práci s navrženým algoritmem. Vhodná realizace těchto nápadů může velmi pravděpodobně zvýšit kvalitu výsledků navrženého algoritmu, resp. jeho implementace.

Další experimenty s hodnotami parametrů

První z možností je komplexnější analýza parametrů a obsáhlejší experimenty týkající se jejich nastavování a jejich vztahu k dosaženým výsledkům. My jsme provedli několik základních pokusů a jejich výsledky zdokumentovali v kapitole 7. Tyto výsledky potvrzují očekávání, že výkon algoritmu (zejména ve smyslu výsledné kvality) na nastavení parametrů závisí, a dávají návod k nastavení těch základních. Parametrů algoritmu je ale mnoho a zřejmě každý z nich může kvalitu výsledné mapy ovlivnit.

Kontrola vytečení z oblasti

Náš algoritmus ani program v současné době nekontroluje, zda se popisek oblasti vejde do oblasti. Abychom byli přesní, nekontroluje ani to, zda se popisek

alespoň částečně nachází *uvnitř* popisované oblasti. Zavedení kontroly na pozici popisku vůči příslušné oblasti a na jejich vzájemné překryvy by tedy mohlo přispět ke kvalitě výsledných map.

Slučování značek

Estetický dojem z některých map vygenerovaných naším programem snižují místa, kde se blízko sebe vyskytuje několik stejných popisků. To nastává např. při popisu zastávek MHD v různých směrech, při popisu dvou stejně pojmenovaných jednosměrných ulic nebo při popisu několika budov patřících stejnému subjektu.

Možným řešením je již v prvních fázích algoritmu vyhledat popisky, které stejným textem popisují typově stejné objekty, a všechny tyto popisky nahradit jedním společným.

Podobným způsobem by mohlo kvalitu výsledných map zvýšit obecné slučování značek, např. bodových symbolů pro zastávky MHD. Slučování značek může pomoci zejména v oblastech, kde je vysoká hustota značek, a tedy buď velké množství překryvů, nebo nutnost mnoho značek vynechávat.

Lámání popisků

Implementace našeho algoritmu v současné době vůbec neumožňuje lámání popisků, resp. jiné zalámání, než jaké vygeneruje symbolizer. Naimplementovali jsme ale podporu pro více variant značky a tu bychom mohli pro zalámané popisky velmi dobře využít. Dokonce tak ani nemusíme předem říkat, jaké zalámání bude optimální. Nabízí se tedy v budoucnosti upravit ať už implementaci algoritmu, nebo symbolizer tak, aby popisky dostávaly více variant značky, a to s různým zalámáním.

Propojení symbolu s popiskem

Algoritmus, který jsme navrhli, nezohledňuje vztahy mezi značkami, zejm. nevyužívá vazbu mezi popiskem a jím popisovaným symbolem. I popisky totiž interně patří k reálnému objektu. Kdybychom upravili algoritmus, aby dovolil přiřadit popisek k mapovému symbolu, mohli bychom např. počítat penalizaci za vzdálenost od tohoto symbolu, nikoliv od skutečného objektu. To by pravděpodobně pomohlo kvalitě výsledku.

Důležitost značek

V současné implementaci není možné vyjádřit důležitost značek. Ta by přitom mohla být využívána např. k určení penalizace za neumístění značky. I tato úprava by vygenerovaným mapám zřejmě prospěla.

Seznam použité literatury

[1] Bradstreet, Lucas, Luigi Barone a Lyndon While. *Map-labelling with a multi-objective evolutionary algorithm.* Proceedings of the 2005 conference on Genetic and evolutionary computation - GECCO '05. New York, New York, USA: ACM Press, 2005. DOI: 10.1145/1068009.1068335. ISBN 1595930108

[2] Christensen, J., J. Marks a S. Shieber. *Algorithms for cartographic label placement.* Proceedings of the American Congress on Surveying and Mapping I, 1993. strany 75-89

[3] Cook, Anthony C. a Christopher B. Jones. *Rule-based name placement with Prolog.* Proceedings of the Ninth Auto-Carto Conference - ACSM/ASPRS. 1989. strany 231-240

[4] Cromley, R. G. *An LP relaxation procedure for annotating point features using interactive graphics.* Proceedings of the Seventh Auto-Carto Conference - ACSM/ASPRS. 1985. strany 127-132

[5] Čerba, Otakar. *Přednáškové slidy.* Předmět Tematická kartografie. Fakulta aplikovaných věc Západočeské univerzity v Plzni. Dostupné online z http://gis.zcu.cz/studium/tka/Slides/. [cit. 25. 3. 2014]

[6] van Dijk, Steven. *Genetic Algorithms for Map Labeling.* Doktorská teze. Department of Computer Science, Utrecht University, listopad 2001.

[7] van Dijk, Steven, Marc van Kreveld, Tycho Strijk a Alexander Wolff. *Towards an evaluation of quality for names placement methods.* Technická zpráva UU-CS-2001-43, Department of Computer Science, Utrecht University, 2001.

[8] Djouadi, Y. *Cartage: A cartographic layout system based on genetic algorithms.* Proceedings of the Fifth European Conference and Exhibition on Geographical Information Systems. 1994. strany 48-56

[9] Formann, Michael a Frank Wagner. *A packing problem with applications to lettering of maps.* Proceedings of the seventh annual symposium on Computational geometry - SCG '91. New York, New York, USA: ACM Press, 1991, strany 281-288. DOI: 10.1145/109648.109680. ISBN 0897914260.

[10] Imhof, E. *Positioning names on maps.* The American Cartographer. 1975. 2(2):128-144.

[11] Iturriaga, Claudia a Anna Lubiw. *NP-hardness of some map labeling problems.* Technická zpráva CS-97-17, University of Waterloo, Kanada, 1997.

[12] Kubalík, Jiří. *Přednáškové slidy.* Předmět Biologicky inspirované algoritmy. Fakulta elektrotechnická Českého vysokého učení technického. Dostupné online z https://cw.fel.cvut.cz/wiki/courses/a4m33bia/lectures. [cit. 31. 5. 2015]

[13] Marks, Joe a Stuart Shieber. *The Computational Complexity of Cartographic Label Placement*. Technická zpráva TR-05-91, Harvard CS, 1991.

[14] do Nascimento, Hugo A. D. a Peter Eades. *User Hints for map labeling*. Journal of Visual Languages. 2008. 19(1):39-74. DOI: 10.1016/j.jvlc.2006.03.004. ISSN 1045926x.

[15] Strijk, T., B. Verweij a K. Aardal. Algorithms for maximum independent set applied to map labelling. Technická zpráva UU-CS-2000-22, Department of Computer Science, Uttrecht University, 2000.

[16] Verner, O., R. Wainwright and D. Schoenefeld. *Placing text labels on maps and diagrams using genetic algorithms with masking*. INFORMS Journal on Computing. 1997. 9(3):266-275.

[17] Wagner, F. a A. Wolff. *A combinatorial framework for map labeling*. S. H. Whitesides, editor, Lecture Notes in Computer Science, Volume 1547: Proceedings of the Symposium on Graph Drawing. Springer-Verlag, 1998. strany 316-331

[18] Yamamoto, M., L. A. N. Lorena a G. Camaara. *Tabu search heuristic for point-feature cartographic label placement problems*. Proceedings of the Third Metaheuristics International Conference. 1999.

[19] Yoeli, P. *The logic of automated map lettering*. The Cartographic Journal. 1972. 9:99-108.

[20] Zoraster, S. *Integer programming applied to the map label placement problem*. Cartographica. 1886. 23(3):16-27

Příloha A

V této příloze popisujeme soubory na přiloženém CD.

- adresář `leo` — úplný obsah repozitáře, resp. jeho umisťovací větve, kde byl algoritmus implementován, k 28. 7. 2015
- soubor `patch` — patch oproti větvi master k 28. 7. 2015, tedy veškeré změny, které jsme udělali v této práci
- soubor `styly.css` — styly používané při experimentech v této práci
- adresář `dumpy` — OSM dumpy dat použitých při vykreslování hodnocených map
- adresář `konfigurace` — konfigurační soubory a seznamy měněných hodnot použitých při získávání podkladů pro kapitolu 7
- adresář `mapy` — hodnocené mapy ve formátu PDF