ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ

V PRAZE

FAKULTA ELEKTROTECHNICKÁ

KATEDRA KYBERNETIKY

BAKALÁŘSKÁ PRÁCE

Vizualizace grafových struktur pomocí genetických algoritmů

Vedoucí práce: Ing. Petr Buryan

Autor: Tomáš Meiser ml.

2009

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

BACHELOR PROJECT ASSIGNMENT

Student: Tomáš Meiser

Study programme: Software Engineering and Management

Specialisation: Intelligent Systems

Title of Bachelor Project: Visualisation of Graph Structures by Means Genetic Algorithms

Guidelines:

1. Learn the basic approaches to relational structures (graphs) visualization.
2. Find an approach from the field of genetic algorithms that would allow for fast and transparent layout of the structure.
3. Implement the approach in Java.
4. Verify the function of your implementation on graphs given by the supervisor.

Bibliography/Sources:

[1] Goldberg, David E. Genetic Algorithms in Search Optimization and Machine Learning, Addison-Wesley Publishing, Inc., Reading, Massachusetts, 1989.
[2] Koza, John R. Genetic Programming: On the Programming of Computers by Means of Natural Selection. Cambridge , MA : The MIT Press, 1992.

Bachelor Project Supervisor: Ing. Petr Buryan

Valid until: the end of the winter semester of academic year 2009/2010

L.S.

prof. Ing. Vladimír Mařík, CSc.
Head of Department

doc. Ing. Boris Šimák, CSc.
Head

Prague, Februar 23, 2009

Anotace

Předkládaná bakalářská práce je zaměřena na vývoj a implementaci algoritmů pro rozmisťování grafových struktur především ve třech dimenzích. Grafové struktury jsou rozmisťovány s ohledem na maximální čitelnost uživatelem. Kvalita vytvořeného řešení je popisována různě konstruovanými matematickými funkcemi, které jsou následně optimalizovány nedeterministickými optimalizačními algoritmy a to především genetickými. Výsledkem této práce je zejména implementovaná knihovna pro rozmisťování grafových struktur obsahující všechny využité přístupy ve formě maximálně modulárních a znovu použitelných částí programu. Druhou částí práce jsou testy a hodnocení implementovaných estetických kritérií pro kreslení různých jednoduchých grafových struktur.

Klíčová slova

Graf, rozmisťování, optimalizace, genetický algoritmus, 3D vizualizace, struktury.

Abstract

The main thesis of this project presents developing and implementing of graph drawing algorithms, especially in 3D. Graph structures are positioned with due regard for maximal operator understanding. The quality of solution is described by, more or less sophisticated mathematics functions and than they are optimized by nondeterministic algorithms, mostly by genetic algorithms. The outcome of this project is primarily implemented library for solving graph drawing problem. This library contains implementation of all introduced methods in modular and reusable form. The second part of this project consists of new methods evaluating and testing. This part compares the solution quality obtained by well known and newly implemented methods.

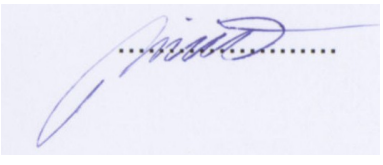
Key words

Graph structures, graph drawing, optimization, genetic algorithms, 3D visualization.

Prohlášení

Prohlašuji, že jsem svou bakalářskou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

V Praze dne 10.7.2009 Tomáš Meiser



Poděkování

Zde bych rád poděkoval Tomás García-Pozuelo Barrios [tomas.garcia.pozuelo@gmail.com] za souhlas s použitím jeho vizualizačního programu pro testování a demonstrování výsledků této práce.
V neposlední řadě bych tímto chtěl poděkovat také svému vedoucímu, panu Buryanovi, za trpělivost a jeho odbornou pomoc při psaní této práce.
Konečně bych rád poděkoval svému otci za pomoc s finální korekturou celé práce.

Obsah

Seznam použitých symbolů

$\mathbb{G} = \langle V, E, \rho \rangle$ ..... Obecný graf s množinou vrcholů V, množinou hran E a zobrazením incidence ρ

$V;\ |V| = n$ ..... Množina vrcholů grafu s mohutností n

$E; |E| = m$ ..... Množina hran grafu s mohutností m

$\rho: E \to (v_1, v_2);\ v_1, v_2 \in V$ .... Zobrazení incidence, které zobrazuje množinu hran grafu na (ne)uspořádanou množinu koncových vrcholů

$\mathbb{A}[n,m] = \begin{pmatrix} a_{1,1} & \cdots & a_{1,m} \\ \vdots & a_{i,j} & \vdots \\ a_{n,1} & \cdots & a_{n,m} \end{pmatrix}$ Matice $\mathbb{A}$ s n řádky, m sloupci a hodnotou $a_{i,j}$ v i-tém řádku a j-tém sloupci

d_{ij} ..... Délka nejkratší cesty v grafu mezi vrcholy i a j měřená buďto v počtu hran na cestě nebo jako součet vah hran na cestě

$\overrightarrow{p_i}$ ..... Vektor pozice vrcholu i v současném rozmístění grafu

D_{ij} ..... Vzdálenost v prostoru (nebo rovině) mezi body i a j v současném rozmístění grafu

1 Úvod

Tato práce je zaměřena na problém zobrazování grafů (v anglické literatuře jako Graph Layout Problem). Tato problematika patří mezi dlouho studovaná odvětví informatiky a existuje k ní rozsáhlá literatura, která je přehledně seřazena a anotována v (1). Vizualizace grafů (často se také používá kreslení či rozmisťování grafů, což budu v následujícím textu volně zaměňovat), je velice jednoduše řečeno, souhrn metod k automatickému nalezení „hezkého“ vyobrazení grafové struktury.

Grafy jsou velice často používány v různých odvětvích lidské činnosti, protože jsou ideálním prostředkem k zaznamenání vztahů, struktur a pracovních postupů. Z tohoto důvodu se také pro grafy za desítky let jejich studia vyvinuly mnohé techniky pro jejich efektivní zpracovávání. Grafy jsou považovány za základ moderní informatiky, a proto není divu, že většina problémů řešených počítačem se reprezentuje právě pomocí grafů. Algoritmy pro počítačové zpracování grafů, vycházející z teorie grafu, jsou jádrem většiny počítačových systémů. Je jasné, že grafové struktury jsou velice vhodné pro počítačové zpracování, avšak jejich nativní podoba použitá pro výpočty je následně velmi nečitelná pro uživatele. Z toho všeho plyne, že je zde velká poptávka po rychlých a robustních metodách, které získané výsledky ve formě grafu, vhodně zobrazí tak, aby jim byl uživatel schopen rychle a jednoduše porozumět.

Takto definovaný problém je však velmi vágní a nejasný. Hledáme takové „hezké“ zobrazení grafu, aby byl pro uživatele jasný a srozumitelný. Co však znamená slovo „hezké“? Tento pojem se liší pro každého člověka i pro každý obor. Hlouběji se ho pokusím rozebrat v kapitole věnované estetickým kritériím, protože právě estetika je věda, která dává tomuto nejasnému pojmu ostré obrysy.

V dalších kapitolách úvodu se budu podrobněji zabývat vysvětlením základů teorie grafu, které napomohou lepšímu pochopení celého následujícího textu. Dále rozeberu podrobněji podstatu

vizualizace grafů, ukážu, jak lze problém vizualizace převést na optimalizaci, a uvedu souhrn základních optimalizačních technik. Nakonec se budu zabývat optimalizačními kritérii používanými pro optimalizaci vizualizace.

1.1 Základy teorie grafů

Teorie grafu je, jak již bylo řečeno, velice dlouho studovaný obor, a proto k němu existuje velké množství kvalitní literatury. Já jsem čerpal především z (2) a (3). V této kapitole se zcela jistě nebudu snažit o vysvětlení teorie grafu v celé její šíři, ale zaměřím se na vysvětlení základů, které jak doufám napomohou k pochopení celého následujícího textu.

Graf je matematická struktura $\mathbb{G}$ složená z množiny vrcholů, zde značená jako V, množiny hran E a zobrazení incidence ρ. Hrana je definována jako relace mezi dvěma vrcholy, které jsou nazývány koncové. Tato relace je definována pomocí zobrazení incidence ρ, které zobrazuje každou hranu grafu na dvojici jejích krajních vrcholů. Tato dvojice může být buďto uspořádaná, v případě orientovaných grafů, nebo neuspořádaná, v případě grafů neorientovaných. Z toho plyne základní dělení grafů na skupinu orientovaných grafů, kde rozeznáváme počáteční a koncový vrchol každé hrany, dále pak grafy neorientované, kde jsou koncové vrcholy na hranách zaměnitelné a nakonec zřídka zmiňovanou skupinu grafů smíšených, které mohou obsahovat oba typy hran najednou. Pro naše potřeby se budeme zabývat pouze grafy neorientovanými.

Neorientované grafy můžeme dále rozdělit na grafy obecné, multigrafy a grafy obyčejné. Obecný graf je takový graf, ve kterém se vyskytují jak multihrany, tak smyčky. Multihranami nazýváme takovou skupinu dvou a více hran, která současně spojuje stejné koncové vrcholy. Smyčka je pak hrana spojující vrchol sám se sebou. Multigraf je tedy graf, ve kterém se mohou vyskytovat multihrany avšak už ne smyčky, a obyčejný graf je graf bez multihran a smyček. V následujícím textu se pak budu zabývat už jen grafy obyčejnými, aniž bych to zvlášť

zdůrazňoval. Důvod tohoto zjednodušení je dvojí. Zaprvé nepřítomnost multihran a smyček značně zjednoduší reprezentaci tohoto komplexního problému a zadruhé nedojde při nahrazení obecného grafu, či multigrafu obyčejným grafem k žádné výraznější újmě na obecnosti. Vysvětlení je následující. Pokud mám namalovat graf, který není obyčejný, pak mohu namalovat jeho podgraf očištěný o smyčky a multihrany, který obyčejný je, a do vzniklého obrázku pak domalovat smyčky a multihrany.

V oboru obyčejných neorientovaných grafů potřebujeme k popsání nějaké instance grafu znát pouze množinu jeho vrcholů a množinu neuspořádaných dvojic těchto vrcholů, která nám bude definovat hrany. Zobrazení incidence je tedy v takovémto případě nadbytečné, právě proto, že hrany jsou jednoznačně určené neuspořádanými dvojicemi svých koncových vrcholů. Je nutné poznamenat, že velice často nesou hrany grafu také informaci o takzvané váze. Váha hrany je hodnota, která může reprezentovat například cenu, případně vzdálenost. Pokud váha hrany reprezentuje požadovanou délku, případně poměr délek hran v hledaném zobrazení, musíme se touto informací také zabývat a vzít ji při výpočtech v úvahu.

Od výše zmíněné reprezentace pomocí množin, která je vhodná pro definování pojmů grafové teorie, se však potřebujeme dostat k reprezentacím, které jsou vhodnější pro zpracovávání grafů výpočetní technikou. Jednou z takových reprezentací je matice incidence. Matice incidence obyčejného neorientovaného grafu je maticí $\mathbb{B}[n,m]$ nad okruhem modulo 2, která má na pozici $b_{i,j}$ hodnotu 1 pokud hrana s indexem j inciduje s vrcholem s indexem i a hodnotu 0 jinak. Z této definice plyne, že matice incidence bude mít vždy součet sloupců roven 0 a že bude poměrně rozsáhlá pro husté grafy. (grafy s vysokým poměrem počtu hran k počtu vrcholů).

Další možnou reprezentací grafů je matice sousednosti. Sousední vrcholy grafu jsou takové vrcholy, mezi kterými existuje hrana. Sousedy daného vrcholu pak nazýváme množinu vrcholů, do kterých z daného vrcholu existuje hrana. Matice sousednosti neorientovaného obyčejného

grafu je potom taková čtvercová souměrná matice $\mathbb{A}[n,n]$ nad okruhem modulo 2, která má hodnotu $a_{i,j} = a_{j,i} = 1$ pokud existuje hrana mezi vrcholem s indexem i a vrcholem s indexem j a 0 v opačném případě. Je nutné zdůraznit, že v případě, kdy bereme v úvahu váhy jednotlivých hran, je možné definovat váhovou matici $\mathbb{W}$, která bude mít shodné vlastnosti jako matice sousednosti, pouze bude namísto hodnot 0,1 obsahovat přímo hodnotu váhy dané hrany. V tomto případě je pak nutné vhodně zvolit hodnotu, která bude reprezentovat neexistenci hrany. Často to bývá ∞, případně 0 pokud se nám nemůže vyskytnout jako váha. Reprezentace pomocí matice sousednosti, případně váhové matice, má velikou výhodu v rychlosti zpracování dotazu na existenci hrany mezi dvěma vrcholy případně na její váhu. Pro zmíněné vlastnosti byla tato forma reprezentace použita v další práci.

Jako poslední formu reprezentace grafu bych ještě v krátkosti zmínil takzvanou spojovou reprezentaci. Jedná se o pole spojových seznamů, které pro každý vrchol uchovává spojový seznam jeho sousedů, případně incidujících hran. Tato reprezentace se však ukázala velice neefektivní pro využití v této práci.

Konečně je potřeba definovat několik pojmů z oblasti cest v grafu. Cesta v obyčejném neorientovaném grafu mezi dvěma vrcholy, je taková posloupnost uzlů a hran, kde vždy hrana inciduje s vrcholy, které jsou v posloupnosti přímo před ní a přímo za ní. Navíc tato posloupnost nesmí obsahovat žádný uzel, a tudíž ani žádnou hranu, dvakrát. Nejkratší cesta v grafu je pak taková cesta, která je složena z nejmenšího počtu hran, případně má minimální součet vah, pokud se zabýváme grafem s váhami. Existují dobře známé algoritmy pro hledání takovýchto nejkratších cest v grafu, z nichž jeden bude popsán dále v implementační části. Na základě zjištěných nejkratších cest mezi všemi uzly jsme schopni zjistit některé obecné vlastnosti daného grafu. Jednou z nich je excentricita každého vrcholu. Excentricita je definována jako maximální nejkratší cesta mezi daným vrcholem a libovolným dalším vrcholem grafu. Maximální hodnota excentricity se pak nazývá průměr grafu a minimální poloměr grafu. Navíc vrcholy

s hodnotou excentricity rovnou poloměru se nazývají středy grafu. Tyto hodnoty pak vypovídají o grafu skutečnosti, které je možné použít při jeho kreslení.

1.2 Vizualizace grafu

V předchozích oddílech jsem se zabýval tím, co je to graf a k čemu slouží. V této kapitole budu mluvit o tom, proč je potřeba grafy zobrazovat, z jakých částí se zobrazení grafu skládá a o úskalích spojených s použitím různých médií k prezentaci výsledků vizualizace.

Jak již bylo řečeno výše, je velký rozdíl mezi podobou grafu, se kterou bude při svých výpočtech pracovat počítač, a podobou, kterou bude preferovat člověk pro zobrazení výsledků nějakých grafových operací. Stejně jako je nesmyslné předkládat grafovému algoritmu naskenovaný obrázek rukou nakreslené grafové struktury, je stejně bláhové chtít po člověku, aby dekódoval matici incidence s informacemi o struktuře vedení podniku. Hlavním problémem je fakt, že každý graf má nekonečně mnoho svých různých zobrazení. Z této nekonečné množiny variant pak máme za úkol vybrat jednu, která nejlépe splňuje jednoduchou podmínku, a to že je nejhezčí. Výborně, ale jak identifikujeme právě to jedno zobrazení? Při řešení toho úkolu se musíme zabývat pojmem estetika.

Estetika je naukou o kráse a vyvíjí se již od starověkého Řecka. Estetika jako součást filozofie byla pěstována mnoha filozofickými školami a snažila se vysvětlit pojem krásy pomocí exaktních pojmů. Například jedna z nejstarších škol, Pythagorejci, hledala krásu v dokonalém matematickém uspořádání světa. Jedním z fundamentálních estetických pozorování těchto myslitelů, byla definice „zlatého řezu“, který je i dnes často využíván umělci a designéry k dosažení líbivých proporcí. Podobným úkolem je také definování matematických a geometrických vztahů, které musí splňovat zobrazení grafu, aby bylo přehledné a čitelné. Estetika se v tomto smyslu využívá především k rozložení problému hezkého zobrazení na menší jednotky, které následně bude možné vyjádřit ve formě

matematických předpisů, jejichž splnění zapříčiní hezké vykreslení grafu. Podrobným výčtem a vlastnostmi takovýchto granulárních podproblémů se budu hlouběji zabývat v kapitole věnované estetickým kritériím.

Dříve než se budu zabývat požadavkem na „hezké“ zobrazení grafu, definuji ještě, co se myslí pod samotným pojmem zobrazení. V první řadě je potřeba definovat zobrazování jednotlivých komponent grafu. Odhlédneme-li od barvy a textury, které pro nás budou zcela irelevantní, vzhledem k tomu, že neděláme rozdíly mezi jednotlivými vrcholy a hranami, jsou pak vrcholy grafu tradičně kresleny jako koule, případně krychle. Hrany grafu se pak kreslí jako křivky spojující vrcholy. Ovšem máme několik možných typů křivek, které mohou hrany znázorňovat. Základním typem křivky je obecná lomená čára. Ta může být samozřejmě složená z různého množství segmentů, které mohou být napojené pod libovolným úhlem. Je důležité si uvědomit, že takto definovaná křivka nás nutí k určení bodů napojení všech segmentů, nebo ekvivalentně k určení délek všech segmentů a úhlů napojení. Drobným zjednodušením může být definování pevného úhlu mezi segmenty. Příkladem pak je ortogonální lomená čára, u které opět musíme určovat délku a směr napojení segmentů. Takovéto lomené čáry jdou následně vyhladit do líbivých křivek, avšak za cenu mohutného nárůstu hledaných neznámých. Proto se ve většině případů provádí zjednodušení problému, zavedením rovných nelomených čar, které reprezentují hrany. Takto definované zobrazování hran je pak plně určeno polohou vrcholů. I já se v této práci budu zabývat pouze kreslením grafů s rovnými čárami.

Podstatným úhlem pohledu, kterým se musíme na vizualizaci grafu dívat, je médium, pro které je hledaná vizualizace určená. Hlavním rozdělením v dnešní době může být dimenze prostoru, který je zařízení schopno zobrazit. Zde jsou v první skupině zařízení pro 2D vizualizaci, kam patří především tiskárny, případně obrazovky počítače bez podpory prostorového rozhraní. Ve 2D zobrazení na počítači máme ještě značnou volnost (oproti tiskárně), protože zde můžeme

s nalezeným zobrazením pracovat (například měnit rozlišení atd.). Na druhé straně výtisk z tiskárny determinován nalezeným zobrazením je neměnný a jednou pro vždy daný. Největší variabilitu a možnosti skýtá 3D zobrazení grafu a to především pro grafy, které ze své podstaty nejsou planární (planární graf je takový graf, který je možné nakreslit v rovině, aniž by se některé dvě hrany křížily). Pokud graf zobrazíme v třírozměrném prostoru, jsme schopni pomocí vhodných ovládacích prvků s vizualizací pohybovat. Můžeme ji otáčet podle dvou os a můžeme ji přibližovat a oddalovat. Tyto prostředky pak mohou napomoci k lepšímu porozumění grafu ze strany uživatele. Na druhé straně je nutné podotknout, že přidaný rozměr značně zvyšuje složitost všech geometrických operací a přidává množství hledaných neznámých.

Ze všeho co bylo řečeno v předchozích odstavcích, můžeme vizualizaci grafu shrnout následovně. Úkolem vizualizace grafu je vybrat právě jedno z nekonečného množství zobrazení daného grafu tak, aby splňovalo subjektivní estetická kritéria, kulturní a tradiční zvyklosti a navíc bylo vhodné pro zvolené zobrazovací médium.

1.3 Vizualizace grafu jako optimalizační proces

V této kapitole se pokusím ukázat, jakým způsobem se úloha kreslení grafu převádí na optimalizační úlohu. Nejdříve definuji optimalizační úlohu a optimalizační proces obecně a pak identifikuji jednotlivé prvky optimalizace na problému vizualizace grafu.

Pokud mluvíme obecně o optimalizaci, nejčastěji máme na mysli matematickou optimalizaci. Tu můžeme laicky ztotožnit s maximalizací, případně minimalizací nějaké matematické funkce, při dodržení definovaných omezení. Zde bych podotknul, že mezi maximalizací a minimalizací je poměrně snadné úlohy převádět. Nejčastěji se to provádí vynásobením dané funkce číslem -1 případně umocněním této funkce exponentem -1. Pokud se tedy nyní vrátíme k definici optimalizační úlohy jako maximalizace (minimalizace) nějaké matematické funkce, tak jako zadání takovéto úlohy můžeme chápat tuto funkci a nějaká omezení.

Definovanou optimalizační úlohu můžeme pak obecně řešit v optimalizačním procesu. Optimalizační proces je v zásadě pouze nalezení přiřazení hodnot pro proměnné optimalizované funkce, tak aby tato nabývala maxima (minima) a zároveň nebyla porušena omezení. Optimalizační proces provádějí optimalizační algoritmy, z nichž některé popíši podrobněji v následující kapitole.

Nyní když jsem definoval obecnou optimalizační úlohu, můžu se pokusit definovat optimalizační úlohu nalezení ideálního zobrazení grafu. Jak již bylo řečeno v kapitolách výše, původní komplexní úlohu jsme postupně zjednodušili na hledání zobrazení obyčejného neorientovaného grafu, kde se omezíme pouze na rovné čáry znázorňující hrany. Výsledek takto zjednodušené úlohy je pak dán pouze nalezením správných pozic pro jednotlivé vrcholy. Celá úloha se tedy bude zabývat rozmístěním vrcholů v prostoru daném vizualizačním zařízením, kde míra optimality je dána funkcí složenou z hodnocení jednotlivých estetických kritérií.

Pokud bych tedy tuto kapitolu shrnul, dospěl jsem k překladu původní vágní a komplexní úlohy, na optimalizační úlohu danou cílovou funkcí složenou z formulí, které hodnotí splnění estetických kritérií, v nichž v roli hledaných neznámých vystupují pozice všech vrcholů grafu a jako omezení slouží vlastnosti vizualizačního zařízení, jako například velikost papíru, či hloubka rozlišení obrazovky.

1.4 Optimalizační techniky

V této kapitole uvedu a srovnám základní přístupy k optimalizaci, a to jak ty, které jsou dále hlouběji popsány a implementovány, tak ty, které z nějakého důvodu dále použity nebyly. Provedu výčet jednotlivých skupin a předložím pro každou skupinu příklad konkrétního algoritmu s jeho základními rysy.

Pokud máme štěstí a cílová funkce je pěkná, neboli má nějaký specifický tvar či vlastnost, potom můžeme použít některou z analytických metod řešení takovéto optimalizační úlohy. Příkladem takové optimalizační techniky může být parciální derivace pro

diferencovatelné funkce. Je dobře známo, že pokud je funkce diferencovatelná, je možné najít extrém této funkce řešením normálních rovnic, které vzniknou položením všech parciálních derivací této funkce rovno nule. Podobné metody samozřejmě existují i pro mnoho dalších typů funkcí. Výhodou těchto metod je jejich vysoká efektivita, avšak nevýhodou je právě jejich úzké zaměření na konkrétní tvar cílové funkce. V tomto případě se nám může stát, že drobná změna zadání znemožní danou metodu použít. Shrneme-li tedy analytické metody řešení, tak jsou jistě vysoce efektivní, avšak málo robustní oproti ostatním metodám.

Další známou skupinou technik pro řešení optimalizačních úloh jsou lokální optimalizační techniky. Tyto algoritmy jsou založené na prohledávání okolí nějakého již známého řešení (to může být inicializováno třeba náhodně). Zde je potřeba říct, že okolím nějakého řešení se myslí množina řešení, které nejsou od tohoto řešení dál než předem definovanou vzdálenost. Pro určení vzdálenosti dvou řešení je potřeba použít vhodnou metriku tak, aby co nejvíce odpovídala druhu proměnných. Tím myslím, že pro binární proměnné bychom zvolili například Hammingovu metriku a pro reálná čísla například euklidovskou metriku.

Základní lokální optimalizátory se souhrnně nazývají hladové algoritmy (Greedy Algorithms) (4). Hladové algoritmy hledají ve svém okolí nějaké optimálnější řešení, do kterého se následně přesunou, a doufají, že tak dosáhnou optima funkce. Typický představitel této skupiny je metoda nejstrmějšího sestupu (4), kde se vybírá takové nové řešení, které je dáno gradientem funkce v okolí stávajícího řešení. Tato metoda však uvázne v prvním lokálním extrému, na který narazí. Tento fakt je hlavní nevýhodou hladových metod.

Některé další lokální optimalizátory se snaží kompenzovat nevýhodu hladových algoritmů. Hlavním představitelem této sorty optimalizačních technik je takzvané simulované žíhání (Simulated Annealing) (5). Tato metoda stejně jako jiné lokální optimalizační techniky prohledává okolí stávajícího řešení, avšak nepřechází pouze

do lepších řešení, jako hladové algoritmy, ale s určitou pravděpodobností může přejít i do řešení horšího. Metoda je založena na principu žíhání ocele, kde je materiál roztaven a poté postupně ochlazován. Správným postupem ochlazování dochází k lepšímu uspořádání struktury materiálu a tím i k lepším vlastnostem. Stejným způsobem je i simulované žíhání řízeno „teplotou“. Hodnota teploty je funkční parametr udávající pravděpodobnost přijetí horšího řešení. Nejpodstatnějším prvkem této techniky je pak takzvaný plán chlazení, který definuje jakým způsobem je teplota postupně snižována. Tento plán může být definován obyčejnou exponenciálou, nebo také složitějšími stochastickými metodami. V zásadě vede ochlazování v této metodě k přechodu od náhodného prohledávání, způsobeného vysokou teplotou, až k lokálnímu hladovému prohledávání, způsobeného teplotou nízkou. Statisticky je metoda simulovaného žíhání poměrně efektivní, avšak ani ona nemůže zcela vyloučit uváznutí v lokálním extrému. Výhodou je, že je možné použít tuto metodu i na problémy optimalizace černé skříňky (Black Box Optimization), kde nemáme explicitně danou cílovou funkci, máme pouze černou skříňku, která dokáže dvě řešení porovnat.

Stejně jako simulované žíhání, jsou také genetické algoritmy (6) stochastickou meta-heuristikou pro řešení optimalizačních úloh. Jak již napovídá téma této práce, budou genetické algoritmy hlavní sortou optimalizačních technik použitých v této práci. Z tohoto důvodu budu chování a vlastnosti těchto algoritmů popisovat v následující kapitole, zde je tedy uvádím jenom stručně v porovnání s ostatními technikami. Technika genetických algoritmů je založená na evoluční biologii. Vychází především z Darwinovy teorie přirozeného výběru a Mendelovy teorie dědičnosti. Základním rozdílem genetických algoritmů od předchozích je fakt, že genetické algoritmy nepracují pouze s jedním řešením, ale v každém okamžiku existuje celá populace (skupina) řešení. Celý algoritmus iteruje po jednotlivých generacích, tak že provádí vylepšování populace pomocí genetických operátorů. Jedna ze dvou základních implementací je takzvaný ustálený model (Steady-

State). Ten se velmi často používá pouze s omezenou škálou operátorů a často se pak jedná o paralelní běh několika lokálních optimalizátorů (lokální optimalizace každého jedince populace). Druhým přístupem je takzvaný generační model (Generational GA), kdy v průběhu iterací dochází k nahrazení jedné populace populací novou. Tato nová populace vzniká na základě rekombinace řešení z populace minulé. Mezi potomky se mohou vyskytovat i řešení z populace předchozí, to už však záleží na postupu nahrazování.

Zcela samostatnou skupinou optimalizačních technik, které se od předchozích fundamentálně liší, jsou techniky multikriteriální optimalizace (7). Multikriteriální optimalizační úloha se od standardní liší nahrazením jedné cílové funkce množinou cílových funkcí. V tomto případě nastává problém, pokud některé dvě funkce z této množiny mají rozdílná, případně protikladná optima. V takovém případě existuje množina optimálních řešení, u kterých nejsme schopni rozhodnout, které je lepší. Takovéto množině se říká nedominující optimální řešení a všechna takováto řešení tvoří takzvanou pareto optimální množinu. Právě tato množina všech nedominujících řešení je kýženým výsledkem multikriteriální optimalizace. Jednou z často používaných technik na poli multikriteriální optimalizace je právě speciální odnož genetických algoritmů. V tomto případě celá populace tvoří aproximaci pareto optimální množiny. Aplikací vhodných operátorů se pak snažíme nahrazovat jednotlivá nedominující řešení řešeními novými, která jim dominují, neboli jsou lepší aproximací pareto optimální množiny. Dílčím úkolem operátorů je pak co nejlépe pokrýt jednotlivými jedinci populace tuto pareto optimální množinu, která bývá často nekonečná.

1.5 Optimalizační kritéria

V této kapitole řeknu pár slov k podstatě optimalizačních kritérií a k postupu jejich korektního sestavení. V další kapitole věnované reprezentaci problému kreslení grafu pro potřeby optimalizace pak rozeberu jednotlivá optimalizační kritéria podrobně.

Jak již bylo řečeno v předchozích kapitolách, optimalizační úloha je dána především formulací své cílové funkce. Právě cílová funkce je pro nás kritérium, podle kterého optimalizaci provádíme. Cílová funkce nám vyjadřuje především kvalitu řešení, pokud bychom zašli až do extrému optimalizace černé skříňky. Proto právě cílová funkce musí obsahovat veškerou informaci o kýženém optimálním (ideálním) řešení. V předchozím textu již také bylo několikrát zmíněno, že právě vyjádření cíle problému kreslení grafu v dostatečné obecnosti pro použití na libovolný graf, a zároveň v explicitním tvaru matematické funkce, je největší problém celé úlohy.

Pokud začneme ze široka právě od zmíněné černé skříňky, budeme schopni poměrně snadno dosáhnout vysoké efektivity kreslení grafu, jen je zapotřebí najít dostatečně otrlého uživatele, který by byl ochoten odpovědět stotisíckrát na otázku, který graf se mu líbí víc, než by se dočkal ideálního řešení. Tento přístup je samozřejmě spíše žertem než návodem k řešení, proto se pokusíme problém rozebrat úžeji.

Ideálním postupem pro sestavení kriteriální funkce pro takto komplexní problém je jeho dekompozice na jednodušší granulární podproblémy. Následně pro každý tento podproblém najdeme matematické vyjádření jeho optimálního řešení. V zásadě hledáme zadání optimalizačních podúloh. Konečným krokem je pak vhodné sestavení těchto střípků až k zadání původní optimalizační úlohy. V tomto posledním kroku tedy nakonec řešíme převod přirozeně multikriteriální úlohy na úlohu jednokriteriální. Nejčastěji se toto provádí váženým součtem jednotlivých kriteriálních funkcí, kde váhy vyjadřují naše zaujetí pro to či ono kritérium. Ne jinak tomu bude i v mém případě, vzhledem k tomu, že jsem se nezabýval multikriteriální optimalizací.

Zcela jiným přístupem, který byl aplikován jako jedna z prvních metod, je zjednodušení problému a jeho převod na nějaký lépe řešitelný problém. Zde byl tím zjednodušeným problémem pružinový systém popsaný v následující kapitole. Tento postup vychází z pozorování

souvislostí mezi hezky vykresleným grafem a ustáleným pružinovým systémem. Tato metoda má však svá omezení a pro některé složitější grafy již zcela jistě nejde o ustáleném modelu, který je grafem indukován, říci, že by byl hezký.

2 Předchozí práce

V této kapitole se budu podrobně zabývat pracemi na poli kreslení grafů a optimalizačních metod, které jsem použil při implementaci této práce. Podrobně zde rozeberu metody a postupy, zmíněné v předchozí kapitole jen v obrysech. Samozřejmě zde budou uvedeny konkrétní rovnice a algoritmy s plnými podrobnostmi a odkazy na literaturu.

2.1 Formulace problému

V následujících oddílech představím formulaci problému kreslení grafů v podobě mechanického modelu a pomocí množiny estetických kritérií.

2.1.1 Mechanický model

Jak jsem napsal v předchozí kapitole, jeden z prvních přístupů k formulaci problému kreslení grafů bylo jeho zjednodušení na mechanický model, konkrétně pružinový model. Každý vrchol grafu byl nahrazen modelem železného kroužku. Hrany spojující vrcholy pak byly modelovány jako pružiny spojující kroužky. Hledané zobrazení grafu pak odpovídalo rozmístění modelu po tom, co zaujal polohu s nejmenší možnou potencionální energií. Podstatné je, že potencionální energie takového systému je dobře vyjádřitelná exaktní matematickou funkcí a proto je možné ji optimalizovat, některou ze zmíněných optimalizačních technik. Tato metoda byla poprvé na problém kreslení grafu použita v (8). V této práci, byl však model zjednodušen příliš, jelikož autor nedbal na Hookův zákon. Hookův zákon říká, že síla pružiny je úměrná změně její délky. Realističtější pružinový model s funkcí potenciální energie [2.1] byl pak představen v (9).

Tuto funkci lze použít jako cílovou funkci optimalizační úlohy, která je následně řešitelná pomocí Newton-Raphsonovy metody (též metoda tečen). Tato metoda patří mezi iterativní numerické metody optimalizace, jejíž výsledek je dobrou aproximací hledaného optima.

Funkce potenciální energie [2.1] je součtem potenciální energie mezi každým párem vrcholů. Potenciální energie mezi jedním párem je odvozena od kvadrátu rozdílu vzdálenosti těchto vrcholů v předloženém zobrazení D_{ij} a jejich ideální vzdálenosti l_{ij} vypočítané podle rovnice [2.2]. Ideální vzdálenost je zjištěna na základě vzdálenosti těchto vrcholů v grafu (délka nejkratší cesty mezi těmito vrcholy) a ideální délky hrany v hledaném zobrazení. Ideální délka hrany v zobrazení může být buďto dána explicitně nebo může být odvozena jako podíl nejkratší hrany kreslícího plátna a délky nejdelší cesty v grafu. Každá takto vypočtená odchylka ve vzdálenosti dvou vrcholů je následně násobena silou pružiny, vypočtené podle rovnice [2.3] jako podíl silové konstanty a kvadrátu grafové vzdálenosti těchto vrcholů.

$$\varepsilon = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{1}{2} k_{ij} \left(D_{ij} - l_{ij}\right)^2 \qquad [2.1]$$

$$l_{ij} = ideálníDélkaHrany \cdot d_{ij} \qquad [2.2]$$

$$k_{ij} = \frac{silováKonstanta}{d_{ij}^2} \qquad [2.3]$$

Současná nejnovější formulace mechanického modelu pro kreslení grafu, inspirovaná částicovou fyzikou, byla představena v (10). Toto vylepšení mechanického modelu však mělo za následek pouze zlepšení ve smyslu použité optimalizační metody. Vzhledem k tomu, že budu ve své práci řešit takto definovanou optimalizační úlohu pomocí genetických algoritmů, není relevantní zabývat se tímto posledním vylepšením. Proto zmiňuji toto vylepšení pouze ve zkratce.

Zde je důležité podotknout, že mechanický pružinový model byl původně navržen a v (8) a (9) použit pouze pro kreslení grafů v rovině. Ovšem podle závěrů v (11) není problém formulaci upravit, tak, aby se dala použít i v prostorovém případě.

Myšlenku mechanického pružinového medulu lze však využít i jiným způsobem. V této práci byl použit pružinový model pro optimalizační techniku založenou na zjednodušené diskrétní simulaci tohoto mechanického modelu. Pokud je model systému stabilní, pak v diskrétní simulaci konverguje do stavu s minimální potenciální

energií. Podrobnosti o fungování této metody popíši hlouběji v kapitole věnované jednotlivým optimalizačním metodám.

2.1.2 Estetická kritéria

Jak již bylo řečeno, mechanický pružinový model, byl jedním z prvních pokusů o zjednodušení a exaktní vyjádření problému kreslení hezkých grafů. Ukázalo se však, že rozmístění pružinového modelu s minimální potenciální energií ne u všech grafů zaručuje jejich hezké vykreslení. Tento fakt plyne ze způsobu, jakým tato reprezentace problému vznikla. Celá myšlenka pružinového modelu je postavena na pozorování, že se pružinový systém v rovině většinou ustálí v nějaké hezké konfiguraci. Pokud ho tedy necháme reprezentovat graf, bude i tento graf přehledný a hezký. Jak jsem již předeslal v úvodu, existuje také jiný postup. Můžeme se na problém kreslení hezkých grafů dívat rovnou ze strany samotných grafů. Podstatné je umět vysvětlit a popsat co to znamená, že je graf hezký, případně přehledný. Tímto oborem se v obecné rovině zabývá nauka zvaná estetika.

Estetika nám umožňuje pojem krásný graf rozdrobit na menší pojmy, které ho tvoří. Těmto jednoduchým prvkům pak můžeme říkat estetická kritéria krásy. Podstatné na tomto dekompozičním postupu je, že tato základní kritéria je možné poměrně snadno, vzhledem k jejich jednoduchosti, popsat také jednoduchými matematickými vztahy. Faktem ovšem je, že jedno estetické kritérium lze vyjádřit pomocí různých matematických vztahů. Příkladem může být kritérium, které se pokouší udržet graf v nějaké rozumné oblasti, případně uvnitř kreslící plochy. Tento cíl nám může plnit hned několik matematických vztahů. Triviálním přístupem ke splnění tohoto kritéria je penalizovat všechna řešení, která kritérium poruší dostatečně vysokou penaltou, a tím zabránit jejich přijetí. Tento postup je však neefektivní v předcházení takovýmto řešením. Problém je v tom, že kritérium má pak pouze dva stavy. Lepším řešením tedy bude kritérium s více polohami. Takovým může být například součet vzdáleností vrcholu od všech okrajů kreslící plochy, nebo jeho vzdálenost od středu kreslící plochy. Tato kritéria

sice nezaručí splnění stoprocentně, ale na druhou stranu předcházejí jeho porušení. Obě výše zmíněné formulace kritéria sledují stejný cíl, avšak každé jiným způsobem a s jinými vedlejšími efekty.

Poprvé byl tento přístup, založený na parciálních estetických kritériích, využitý k definování problému kreslení hezkých grafů představen v práci (12). Autor zde představil sadu estetických kritérií, která, jak již název článku napovídá, měla zaručit kreslení hezčích grafů. Jako metodu pro optimalizaci autor použil simulované žíhání, zmíněné v úvodu, a jednotlivá kritéria spojoval do jedné cílové funkce pomocí váženého součtu. Podstatný na této práci není jen nový přístup autora, ale také diskuze nad jednotlivými představenými kritérii, ve které se autor zabývá různými možnostmi matematického vyjádření, tak jak to bylo popsáno v předchozím odstavci.

Dalším významným použitím a rozborem estetických kritérií pro kreslení grafů byla aplikace AGLO popsaná v článku (13). Autoři se zde kromě samotných kritérií zabývají také jejich rozdělením do skupin podle významu a způsobu použití. Rozeznávají kritéria definovaná jako omezující, statická a dynamická. Za omezující estetická kritéria považují taková, která jsou vyjádřena imperativně a dané zobrazení je může pouze splňovat nebo nesplňovat. Statická kritéria jsou pak taková, která definují požadovaný stav zobrazení, oproti tomu dynamická kritéria deklarují způsob jakým je možné jedno zobrazení změnit na jiné (definují způsob vývoje).

V práci (13) se autoři, kromě výčtu jednotlivých estetických kritérií sebraných z různé literatury a jejich kritického zhodnocení, zabývají myšlenkou použití technik multikriteriální optimalizace tak, jak byla popsána v úvodu. Je zcela zřejmé, že jednotlivá kritéria by bylo vhodné optimalizovat separátně vzhledem k jejich složité normalizaci. Vážený součet kritérií, který je často používán jako trik pro předefinování multikriteriální optimalizační úlohy na úlohu jednokriteriální, může být dosti ošidný. Je podstatné si uvědomit fakt, že jednotlivá kritéria mohou, podle způsobu jejich vzniku, nabývat diametrálně různých hodnot. Například kritérium, které měří počet křížení hran v grafu, bude

nabývat pouze celých čísel a jeho maximální mez bude silně záviset na hustotě grafu. Na druhou stranu součet odchylek v délkách hran bude nabývat hodnot reálných a bude silně závislý na měřítku a velikosti kreslící plochy. Pokud dáme tato dvě kritéria dohromady pomocí váženého součtu, pak nám váhy musí v první řadě zaručit normalizaci obou složek. Je však poměrně složité určit vztah, který by tato kritéria normalizoval automaticky. Multikriteriální optimalizace pak tento problém do jisté míry řeší. Umožní úlohu korektně optimalizovat bez ohledu na rozdíly ve škálách jednotlivých kritérií a umožní pak také uživateli vybrat právě to řešení, které mu vyhovuje.

Tabulka 2.1: Výčet sebraných estetických kritérií

Vrcholy nesmí být příliš blízko u sebe	$\mathcal{O}_{n^2}$
Vrcholy od sebe nesmí být příliš vzdálené	$\mathcal{O}_{n^2}$
Hrany mají být krátké	$\mathcal{O}_m$
Hrany nesmějí být příliš krátké	$\mathcal{O}_m$
Hrany by měli být stejně dlouhé (nebo v daném poměru)	$\mathcal{O}_m$
Hrany by se neměli křížit (jen 2D)	$\mathcal{O}_{m^2}$
Hrany by neměli být příliš blízko u sebe (3D)	$\mathcal{O}_{m^2}$
Úhel, který sousední hrany svírají, by neměl být moc malý	$\mathcal{O}_{m^2}$
Vrcholy by neměli být příliš blízko hranám	$\mathcal{O}_{n \cdot m}$
Vrcholy by měli být uvnitř kreslící plochy	$\mathcal{O}_n$
Vrcholy by měli být blízko středu kreslící plochy	$\mathcal{O}_n$
Vrcholy by měli být rozptýleny rovnoměrně	$\mathcal{O}_n$
Plocha zabraného prostoru by měla být malá (konvexní obálka vrcholů, případně obdélník)	$\mathcal{O}_{n \cdot log\ n}$

V [Tabulka 2.1] je uveden výčet jednotlivých estetických kritérií, tak, jak byly sebrány z literatury. U každého kritéria pak navíc uvádím složitost jejich nejoptimálnějších výpočetních metod z literatury. Je potřeba zdůraznit, že tyto údaje jsou velice důležité pro praktickou aplikaci. V zásadě je největší problém estetických kritérií jejich časově náročné vyhodnocování. Zároveň upozorňuji, že většina uvedených

kritérií (bez poznámky 3D) je použitá pro případ rovinného kreslení grafů. Ve chvíli kdy se přeneseme do oblasti kreslení grafů v prostoru, některá kritéria, jako například počet křížení hran, ztrácejí svůj význam a musí být nahrazena jinými, případně se značně zkomplikuje jejich výpočet. V tomto případě autoři často provádějí některé přípravné operace, které pak pomohou při opakovaném výpočtu jeho náročnost zmírnit.

2.2 Optimalizační algoritmy

Následující dva oddíly budou pojednávat o použitých optimalizačních algoritmech. První bude metoda simulovaného pohybu, po níž bude následovat optimalizace pomocí genetických algoritmů.

2.2.1 Simulovaný pohyb

Tato optimalizační metoda, která využívá formulaci problému pomocí pružinového modelu, je založena na velmi zjednodušené diskrétní simulaci mechanického systému. Nejedná se o diskrétní simulaci v pravém smyslu tohoto pojmu, ale spíše o její nápodobu. Celá metoda je založen na principu samovolné konvergence všech stabilních mechanických systémů, do stavu s nejnižší potenciální energií. Kupodivu v této metodě nepočítáme v žádném místě hodnotu samotné potenciální energie systému.

Model, který budeme simulovat je zjednodušen až na rovnoměrný přímočarý pohyb. Veškeré výpočty se opírají o dobře známou rovnici pro výpočet změny polohy při rovnoměrném přímočarém pohybu z Newtonovské mechaniky, která je shrnuta v rovnici [2.4]. Metoda začíná v náhodně rozmístěném grafu uvnitř kreslící plochy. Následně iteruje, dokud se model v nějaké poloze neustálí, případně není dosaženo limitního počtu iterací. V každé iteraci je pro všechny vrcholy spočten aktuální kumulativní vektor síly. Tento vektor je součtem příspěvků silového působení od všech okolních vrcholů. Výpočet jednotlivých příspěvků se liší, podle toho, jestli jsou vrcholy v grafu spojeny hranou nebo ne. V zásadě, mezi sousedními vrcholy je síla

úměrná odchylce vzdálenosti těchto vrcholů od ideální délky hrany, jak je popsáno v rovnici [2.5]. Naopak mezi vrcholy, které spolu nesousedí, působí vždy odpudivá síla nepřímo úměrná kvadrátu jejich vzdálenosti, což odpovídá rovnici [2.6].

$$\vec{s} = \frac{1}{2} \cdot \frac{\sum \vec{F}}{m} \cdot t^2 \qquad [2.4]$$

$$\vec{F}_{ij} = \frac{1}{2}\left(\widehat{\overrightarrow{p_i p_j}}\right) \frac{D_{ij}}{(D_{ij} - L)} \qquad [2.5]$$

$$\vec{F}_{ij} = \frac{1}{2}\left(\widehat{\overrightarrow{p_i p_j}}\right) \frac{-1}{D_{ij}^2} \qquad [2.6]$$

V rovnicích [2.5] a [2.6] je symbolem $\widehat{\overrightarrow{p_i p_j}}$ myšlen normalizovaný jednotkový vektor určující směr působení síli a symbolem D_{ij} vzdálenost vrcholů v aktuálním rozmístění grafu. Navíc v rovnici [2.5] je symbolem L myšlena ideální délka hrany. Potom, co jsou spočítány všechny kumulativní síly, dojde k přepočítání poloh všech vrcholů v rozmístění podle rovnice [2.4] v diskrétním časovém intervalu daném konstantou t.

Nastavitelným parametrem této metody je hmotnostní konstanta m v rovnici [2.4]. Tato konstanta ovlivňuje rychlost konvergence, avšak jak již bylo řečeno v předchozích kapitolách, nemá tento parametr zásadní vliv na výsledné rozmístění grafu. Je nutné zdůraznit, že hmotnostní konstanta m má zcela zásadní význam pro stabilitu systému. Nevhodně zvolená konstanta m má za následek rozkmitání systému a jeho divergenci.

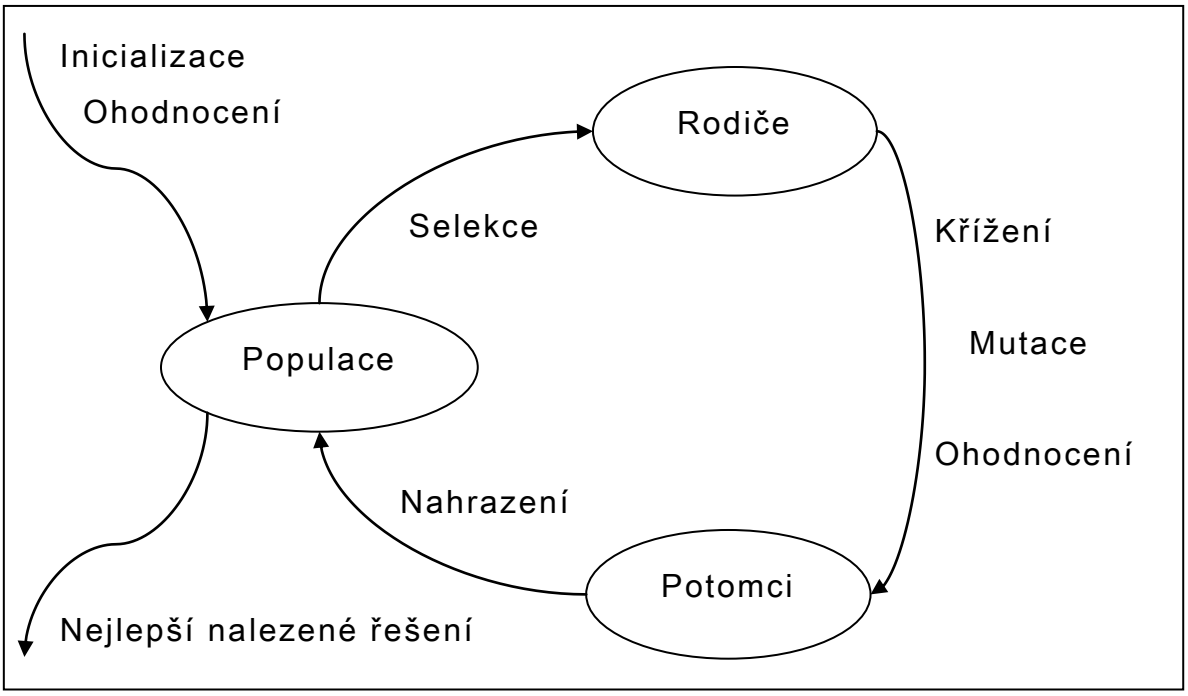
2.2.2 Generační genetický algoritmus

Genetické (evoluční) algoritmy jsou založeny na evolučních procesech probíhajících v přírodě k automatickému přizpůsobování životních forem na jejich prostředí. Princip přírodní evoluce vychází z prací Darwina o přirozeném výběru a Mendela o dědičnosti. Evoluční princip byl následně v padesátých a šedesátých letech dvacátého století zpracováván a nezávisle publikován několika vědci z oboru informatiky, z nichž každý využíval principu evoluce v jiné metodě. Genetické algoritmy byly studovány a publikovány především

J. H. Hollandem v (14), dále pak jako evoluční strategie publikované I. Rechenbergem v (15) a konečně jako simulovaná evoluce v (16).

Dnes jsou již tyto různé přístupy sjednoceny v optimalizační metodě obecně nazývané evoluční algoritmus. Evoluční algoritmy jsou chápány jako meta-heuristika pro optimalizaci a řešení problémů. Pro jejich značnou nezávislost na konkrétním problému se staly velice oblíbenou stochastickou metodou. V zásadě je princip evolučních algoritmů založen na inteligentním prohledávání prostoru všech řešení, tak, že pomocí již nalezených kvalitních řešení odhadujeme, kde by se mohlo vyskytovat řešení lepší. Je podstatné poznamenat, že evoluční algoritmy neřeší daný problém optimálně, ale o to víc nás hlavně u těžkých problémů často překvapí řešení, která naleznou.

Generační genetický algoritmus prohledává prostor řešení pomocí populace řešení (skupina řešení), kterou vhodným způsobem pomocí genetických operátorů zpracovává do generace následující. Běh algoritmu naznačený v [Obr. 2.1] je založen na cyklickém nahrazování



Obr. 2.1: Průběh generativního genetického algoritmu

staré populace populací novou. Iterace probíhá, dokud není nalezeno řešení s požadovanou kvalitou, případně dokud není dosaženo limitu počtu iterací. Vytvoření nové populace probíhá vybráním vhodných

jedinců z populace staré, aplikováním operátoru selekce, a následným sekvenčním aplikováním operátorů křížení a mutace s danou pravděpodobností. Nastavení pravděpodobnosti pro použití jednotlivých operátorů je jedním ze způsobů, jak řídit prohledávání stavového prostoru. Můžeme tak dosáhnout jak rychlé konvergence do lokálního optima, tak maximálního prohledání prostoru řešení. V některých technikách je také využíváno změny těchto pravděpodobností v průběhu iterací, pak algoritmus svými vlastnostmi připomíná simulované žíhání, popsané v předchozím bloku.

Jedním z problémově závislých prvků genetického algoritmu je definice chromosomu. Chromosom je zakódované řešení optimalizační úlohy, které je v zásadě bodem ve stavovém prostoru prohledávaném genetickým algoritmem. Zde je nutné zdůraznit, že právě vhodná reprezentace chromosomu, spolu s použitými operátory křížení a mutace, mají fundamentální vliv na schopnost dobrého prohledávání genetického algoritmu a tím i na jeho výkonnost. Chromosom může být vyjádřen ve dvou tvarech. Prvním z nich je takzvaný genotyp, který je vhodnou reprezentací pro aplikaci genetických operátorů. Tento tvar dobře vyjadřuje strukturu řešení, a proto lze jeho části dobře kombinovat, či měnit. Druhým tvarem chromosomu je takzvaný fenotyp, který vyjadřuje konkrétní vlastnosti řešení z pohledu optimalizační úlohy, a proto je vhodný pro výpočet kvality řešení, neboli jeho fitness. Typicky je chromosom vyjádřen jako řetězec znaků a to nejčastěji binárních, celočíselných nebo reálných.

Kvalita řešení, v předchozím odstavci nazývaná jako fitness, je v zásadě ekvivalentem klasické cílové funkce optimalizační úlohy. Tato funkce má za úkol určit kvalitu daného chromosomu podle toho, jak dobré řešení optimalizační úlohy představuje. Nejčastěji funkce fitness ohodnocuje řešení reálným číslem. Historicky se fitness vyjadřuje ve tvaru maximalizace (na který jde, jak bylo popsáno výše, každá minimalizační úloha převést), tedy čím vyšší fitness, tím kvalitnější řešení. Nakonec je potřeba zmínit, že pro její konstrukci se často, kromě samotné cílové funkce, používají také další prvky, které mají za

úkol například zvětšovat rozpětí hodnot výsledné fitness pro lepší funkčnost operátorů selekce.

Operátor selekce je využíván k výběru jedinců, kteří se budou svým genetickým materiálem podílet na vzniku nové generace. Pravděpodobnost výběru daného jedince by měla být odvozena od jeho kvality (fitness), přičemž kvalitní jedinci mohou být vybráni vícekrát, avšak existuje také malá pravděpodobnost výběru špatného jedince. Poměru mezi pravděpodobností výběr nejlepšího a průměrnou pravděpodobností výběru ostatních řešení se říká selekční tlak. Hodnota selekčního tlaku ovlivňuje s dalšími operátory rychlost konvergence. Pokud budou neúměrně často vybírána nejkvalitnější řešení na úkor horších, dojde k rychlé konvergenci do lokálního optima nejlepšího řešení, které následně obsadí celou populaci. Na druhou stranu, pokud bude selekční tlak malý, nebude genetický algoritmus schopen konvergovat do žádného optima a dostaneme tak náhodné prohledávání. V následujícím odstavci představím některé dobře známé metody selekce, které jsem použil při implementaci.

Nejjednodušším použitým operátorem selekce je Turnajová metoda (6), která náhodně vybere k jedinců z populace, z nichž je následně vybrán ten s nejvyšší hodnotou fitness. Číslo k je v tomto případě možné považovat za hodnotu selekční tlaku a pohybuje se v intervalu $1 \leq k \leq n$, kde pro $k = 1$ metoda degeneruje na náhodný výběr a pro $k = n$ je vždy vybrán nejlepší jedinec. Další metody pak většinou existují ve dvou formulacích a to zaprvé metody selekce proporcionální k fitness a zadruhé proporcionální k pořadí (rank). Metody proporcionální k fitness mají nejčastěji statickou hodnotu selekčního tlaku, která je přímo dána poměrem mezi fitness nejlepšího jedince a průměrnou hodnotou ve zbytku populace. Mezi zástupce této skupiny patří především Ruletový výběr (Roulette Sampling) (6) a univerzální stochastický výběr (Universal Stochastic Sampling) (6). Algoritmus výběru obou metod bude podrobněji popsán v implementační části. Oproti tomuto druhá skupina, s výběrem pomocí pořadí, má nastavitelnou hodnotu selekčního tlaku a díky tomu nabízí větší

variabilitu. Tato skupina je zde zastoupena metodou Linear-Rank Roulette Sampling (6). Metoda využívá pro výběr ruletovou metodu s tím, že namísto použití hodnot fitness využívá hodnoty vypočtené pomocí vztahu [2.7] na základě pořadí i daného jedince v populaci seřazené podle fitness v neklesající posloupnosti a hodnoty selekčního tlaku v intervalu $0 \leq k \leq 1$.

$$\mathcal{L}_{i=1..n} = 1 - k + 2 \cdot k \cdot \frac{i-1}{n-1} \qquad [2.7]$$

Operátory křížení a mutace jsou hlavními prvky genetického algoritmu, zodpovědnými za samotné prohledávání stavového prostoru. Pomocí operátorů křížení se genetický algoritmus pokouší vygenerovat jednoho či více nových potomků za pomoci informací ze dvou či více rodičů. Předpokládáme, že vhodně zvolený operátor křížení je schopný nacházet nová řešení úlohy tak, že uchová z rodičů všechny dobré vlastnosti. Na druhou stranu operátor mutace je většinou aplikován pouze na jednoho rodiče, kterého vhodným způsobem pozmění a najde tak řešení nové. Mutace se především stará o udržování diversity (různorodosti) populace a zároveň provádí lokální prohledávání ve stavovém prostoru. Je vhodné poznamenat, že často samotné použití operátoru mutace spolu s vhodnou selekční metodou simuluje paralelní běh více lokálních optimalizátoru a může dosahovat velmi dobrých výsledků.

Obecně se dá o operátorech mutace a křížení říct, že jsou problémově závislé, avšak existuje poměrně široká skála dobře známých meta-operátorů, které jsou vhodné pro různá použití. Tyto meta-operátory popisované v literatuře, například v (6), jsou určeny vždy pro daný typ genotypu. Jedná se například o jednobodové, případně uniformní křížení nebo o perturbaci či prohození náhodných genů v případě mutace. Samozřejmě musíme zaručit především to, aby operátory křížení a mutace co nejvíce respektovaly přirozenou strukturu genotypu, protože pokud neexistuje logická spojitost mezi chromosomem a použitým operátorem, pak genetický algoritmus nemůže generovat dobrá řešení. Operátory mutace a křížení použité

v této práci v principu využívají výše zmíněných meta-operátorů, avšak vzhledem k reprezentaci chromosomu se od nich v parcialitách liší. Podrobně budou tyto operátory popsány v implementační části.

2.3 Závěr

Mechanický model jako reprezentace problému kreslení grafů je metodou vhodnou jen pro určité typy grafů. Výhodou je, že pro každou zde představenou variantu byl zároveň navrhnut robustní algoritmus pro hledání optima. Značnou nevýhodou je pak fakt, že pokud se nám nalezené zobrazení grafu nelíbí, neposkytuje tato reprezentace téměř žádné prostředky jak výsledek ovlivnit. Jedinou nastavitelnou hodnotou je zde silová konstanta, které však spíše než výsledek ovlivňuje rychlost konvergence, případně stabilitu systému.

Kreslení grafu pomocí množiny estetických kritérií problém vystihuje mnohem přesněji než předchozí model. Nižší míra zjednodušení je však zaplacena v podobě vyšší výpočetní složitosti jednotlivých kritérií a jejich víceznačnosti z pohledu exaktní matematické formulace. Estetická kritéria s sebou navíc přinášejí problém v jejich multikriteriální podstatě, který můžeme řešit buďto použitím multikriteriální optimalizační metody, nebo normalizováním a vážením jednotlivých kritérií v jedné cílové funkci.

Metoda simulovaného pohybu je velice zjednodušeným ekvivalentem diskrétní simulace mechanického systému. V tomto zjednodušení byla zcela zanedbána spojitá změna vzájemného silového působení mezi vrcholy, stejně jako zrychlení všech prvků. Navržený model uvažuje odpudivou sílu mezi všemi nesousedními vrcholy a sílu závislou na odchylce od ideální délky hrany mezi uzly sousedními. Jak bude následně ukázáno v kapitole demonstrující výsledky, je tento algoritmus velice efektivní metodou pro rozmisťování i poměrně složitých grafů jak v rovině, tak v prostoru.

Genetické algoritmy jsou přírodou inspirovanou meta-heuristikou pro řešení optimalizačních úloh, která je dnes pro svou značnou problémovou nezávislost a dobré výsledky velice oblíbená. V kapitole

byly popsány jednotlivé prvky běhu genetického algoritmu, jejich použití a příklady s tím, že některé části budou teprve předvedeny v implementační části, která následuje.

3 Implementace

Jako hlavní součást této práce byla implementována knihovna pro rozmisťování vrcholů grafů v jazyce Java. V této kapitole se budu zabývat konkrétní podobou představované knihovny. V první části popíši použité algoritmy a postupy pro rozmisťování vrcholů grafů, které byly v knihovně implementovány. V další části zaměřené na strukturu programu budu hovořit o některých důležitých třídách a rozhraních.

3.1 Algoritmy

Zde popíši konkrétní postup implementace metod zobrazování s ohledem na maximální výkon. Podrobněji se budu zabývat výpočetně náročnějšími částmi kódu a postupy jakými jsem je řešil. Zvláště bych pak vytknul geometrické operace ve třech dimenzích, které se ukázaly být největším kamenem úrazu.

3.1.1 Zobrazovaní pomocí simulovaného pohybu

Tato velice jednoduchá metoda, která byla popsána v předchozí kapitole, se ukázala být velice efektivním využitím pružinového modelu. Při její implementaci jsem se snažil zachovat v maximální míře rychlost výpočtu. Celý proces je iterativní a hlavní části jsou naznačené v [Algoritmus 3.1]. Funkce accForce, která počítá vzájemné silové působení vrcholů i a j pracuje v souladu s rovnicemi [2.5] a [2.6]. Pro omezení aritmetických operací byly všechny konstantní prvky výpočtů provedeny předem a uloženy do konstanty Q.

Algoritmus navíc využívá faktu, že síla působící mezi dvěma vrcholy grafu je vždy symetrická. Takto lze omezit počet kombinací vrcholů z n^2 na $\frac{n\cdot(n-1)}{2}$.

```
Input:                          Graf   G
                Hmotnostní konstanta   mass
                 Počet kroků iterace   steps
             Délka diskrétního úseku   timeStep
-------------------------------------------------------------
Output:              Pozice vrcholů   points[]
-------------------------------------------------------------
1    points[] = random positions
2    Q = (mass*timeStep^2)/2
3    for steps
4    | for every vertex j and k<j from G
5    | | force = computeForce(j, k);
6    | | accForce[j] += force;
7    | | accForce[k] -= force;
8    | end
9    |
10   | for each point p from points[]
11   | | p += accForce[p] * Q;
12   | end
13   end
```

Algoritmus 3.1: Metoda zobrazování grafů simulovaným pohybem

Parametry vstupující do algoritmu jsou tedy objekt grafu, počet kroků iterace a konstanty času a hmotnosti. Vhodné nastavení konstant času a hmotnosti, může být provedeno uživatelem, který tak může dosáhnout rychlejší konvergence modelu. Ovšem jak již bylo uvedeno, nastavení těchto konstant zásadně ovlivňuje stabilitu systém. Konečně nízký počet kroků iterace může způsobit špatné rozmístění, pokud model do svého optima nestihne konvergovat.

Závěrem tedy mohu říct, že algoritmus pro zobrazování grafu simulovaným pohybem byl zaimplementován s důrazem na maximální efektivitu výpočtů, přičemž asymptotická složitost algoritmu je $\mathcal{O}(s \cdot n^2)$, kde s je počet kroků iterace.

3.1.2 Zobrazování generačním genetickým algoritmem

V této kapitole přehledně popíši implementaci jednotlivých prvků genetického algoritmu. Bude se jednat především o širokou škálu operátorů selekce, křížení a mutace. Dále popíši mechanismy pro efektivní ohodnocování populace. Na konci kapitoly se pak budu zabývat algoritmy pro výpočet fitness. Především se zaměřím na

výpočet fitness pro jednotlivá estetická kritéria. Postup těchto výpočtů ukážu především z pohledu složitějších geometrických operací ve třech dimenzích.

Reprezentace řešení

Zde je nutné zmínit důležité zásady pro zachování konzistence v datech za běhu algoritmu. Je zřejmé, že v průběhu genetického algoritmu dochází ke vzniku velkého množství nových řešení, která jsou však velmi podobná svým rodičům. Proto byl navrhnut postup, který v maximální míře využije existujících objektů a sníží tak režii spojenou s tvorbou nových a destrukcí nepoužitých.

V první řadě bylo potřeba zaručit správnou funkci metody clone pro objekty reprezentující řešení. Tato metoda má za úkol vrátit novou instanci řešení, které má stejnou hodnotu fitness jako původní instance a navíc její chromosom je deep kopií (nekopíruje se pouze reference na pole, ale je vytvořeno pole nové s kopií hodnot) originálního chromosomu. Dále je potřeba, aby jakákoliv implementace selekčního operátoru vracela ne vybrané řešení, ale právě jeho klon. V tuto chvíli je jasné, že mezi sebou jednotlivá řešení sdílí reference na vektory vrcholů grafů, a proto musí také ostatní operátory, které nějakým způsobem manipulují s vrcholy grafu, zaručit neměnnost původních hodnot. Jinými slovy, pokud operátor mění souřadnice nějakého vrcholu, musí si udělat nejdříve kopii tohoto vrcholu, kterou následně edituje.

Nakonec je také nutné, aby operátory měnící chromosom tuto změnu explicitně deklarovaly nastavením hodnoty fitness řešení na NaN (Not a Number). Jak bude dále připomenuto, obecně existuje pravděpodobnost, že řešení projde z jedné generace do druhé v nezměněné podobě, a proto ho ohodnocovací mechanismus nebude znovu ohodnocovat. Právě nastavení hodnoty fitness na NaN ohodnocení řešení vyvolá.

Operátory křížení a mutace

V knihovně byly implementovány a použity, jak standardní operátory křížení (konkrétně jednobodové, dvoubodové a uniformní) z (6), jejichž implementací se zde nebudu zabývat, tak dva speciální operátory křížení, převzaté z (17). První z nich, v literatuře nazývaný jako absolutní křížení, spočívá ve výměně umístění jednoho náhodně vybraného vrcholu mezi dvěma rodiči. Druhý operátor, nazývaný v literatuře jako relativní křížení, vybere náhodně dva vrcholy i a j, pro které následně zkombinuje jejich umístění z obou rodičů podle [3.1]. Ostatní vrcholy, pak tyto operátory nechají vždy nepozměněné.

$$\Delta_{ji} = \frac{\left(p_j{}^{par\,1} - p_i{}^{par\,1}\right) + \left(p_j{}^{par\,2} - p_i{}^{par\,2}\right)}{2}$$
$$p_i{}^{off\,1} = p_i{}^{par\,1} + \Delta_{ji} \qquad [3.1]$$
$$p_j{}^{off\,2} = p_j{}^{par\,2} - \Delta_{ji}$$

Byly implementovány a využity také dva operátory mutace, převzaté z (17). První z nich provádí mutaci jediného vrcholu tak, že k náhodně vybranému vrcholu přičte náhodný vektor, který však není větší, než velikost ideální délky hrany v hledaném zobrazení. Druhý operátor mutace pak provádí výměnu pozic dvou náhodně vybraných vrcholů. Jako poslední operátor mutace byla implementována varianta perturbačního operátoru mutace, ovšem s tím rozdílem, že je aplikován vždy a pravděpodobnost mutace se používá jako procentuální vyjádření počtu vrcholů, na které bude perturbační operátor mutace aplikován.

Operátory selekce

Jednotlivé dobře známé operátory selekce byly popsány již v kapitole věnované předchozí práci a to do značné teoretické hloubky. Především bylo vysvětleno fungování turnajové metody, která byla zaimplementována v této knihovně přesně podle tohoto předpisu. Dále byl také vysvětlen výpočet hodnoty Linear-Rank, která je následně použita spolu s metodou ruletového výběru. Tato selekční metoda byla také implementována podle rozboru z předchozí kapitoly a je nutné poznamenat, že byla nejčastěji požívána při testování. V této kapitole

tedy zbývá popsat implementaci metody univerzálního stochastického výběru a metodu ruletového výběru.

Metoda univerzálního stochastického výběru, která provádí selekci nejčastěji proporcionálně k hodnotě fitness, je odlišná od předchozích metod především tím, že je nutné předem definovat počet požadovaných jedinců, kteří jsou vybráni v okamžiku vytvoření. Selekce začne vytvořením pole neklesající posloupnosti kumulovaných hodnot fitness všech jedinců. Následně je určen krok jako podíl sumy fitness v celé populaci a počtu požadovaných jedinců. Při samotném výběru je pak pole procházeno pomocí ukazatele, který leží v intervalu od nuly do velikosti kroku. K tomuto ukazateli je v každé iteraci přičtena hodnota kroku. Při tomto průchodu jsou jednotliví jedinci přidáváni do skupiny vybraných jedinců do té doby, dokud ukazatel nepřeroste jim odpovídající hodnotu kumulativní fitness. V poslední fázi je skupina vybraných jedinců zamíchána. Celková složitost takovéto implementace odpovídá pro populaci o velikosti p a výběru s jedinců asymptotické složitosti $\mathcal{O}(p + s + p \cdot \log p)$.

Metoda ruletového výběru provádí při svém vzniku, podobně jako metoda předchozí, vytvoření pole neklesající posloupnosti kumulovaných hodnot fitness (případně hodnot Linear-Rank). Selekce jednoho jedince se následně provádí vygenerováním náhodného čísla v rozsahu od nuly do sumy fitness (Linear-Rank) v celé populaci. Vybrán je pak ten jedinec, jemuž odpovídající kumulativní hodnota tvoří horní mez intervalu, ze kterého je náhodné číslo. Nalezení odpovídajícího intervalu se provádí metodou binárního vyhledávání (4). Celá metoda má tedy pro výběr s jedinců z populace o velikosti p časovou složitost $\mathcal{O}(p + s \cdot \log p)$.

Ohodnocování populace

Ohodnocování populace je jistě časově nejnáročnější operací, kterou genetický algoritmus při svém výpočtu provádí. Tento fakt, je z velké části dán obecnou výpočetní složitostí jednotlivých fitness

funkcí, avšak také způsobem, jakým je samotné ohodnocování prováděno.

Pro maximální využití potenciálního výpočetního výkonu dnešních víceprocesorových počítačů byl navrhnut a implementován paralelní ohodnocovač populace. Vstupem tohoto objektu je populace určená k ohodnocení a objekt fitness funkce. Paralelní ohodnocování pak provádí Java ThreadPoolExecutor do nějž je vždy předána klonovaná funkce fitness s nastaveným dosud neohodnoceným jedincem populace. Po tom, co ThreadPoolExecutor dokončí výpočet ve všech svých vláknech, je ohodnocování populace dokončeno.

Kromě paralelního ohodnocovače byl navrhnut také sekvenční ohodnocovač, který je mnohem výkonnější, pokud počítač neumožňuje výpočet více vláken opravdu paralelně. V takovém případě je totiž režie spojená se spuštěním, naplněním a správou ThreadPoolExecutoru mnohem vyšší než výkon ušetřený paralelním výpočtem. Sekvenční ohodnocovač provádí postupné předání všech neohodnocených jedinců v populaci do funkce fitness a následné spuštění její metody run.

Funkce fitness

Zcela jistě nejpodstatnější a nejkomplikovanější částí implementace byly objekty pro výpočet hodnot fitness jednotlivých jedinců populace. Celkově byly navrhnuty dva takové objekty, z nichž první počítá pouze hodnotu potenciální energie pružinového modelu a druhý sadu estetických kritérií pro zobrazování ve 3D.

Funkce fitness implementuje interface Cloneable, aby bylo možné vytvářet kopie objektů fitness bez nutnosti znovu počítat některá statická data, a interface Runable, skrze nějž je prováděno ohodnocování. Pro zvýšení efektivity výpočtu fitness byly v maximální míře rozdělené výpočty závislé pouze na struktuře grafu, které jsou spočteny při vytvoření objektu, a výpočty závislé na rozmístění vrcholů, které se počítají při každém ohodnocování.

Nejjednodušší hodnotící funkcí je výpočet potenciální energie pružinového modelu podle rovnice [2.1]. Vstupem pro vytvoření tohoto

objektu je hmotností konstanta a instance grafu. Objekt provede při svém vzniku výpočet matic k_{ij} a l_{ij}. Výpočet ohodnocení pak probíhá ve dvou vnořených cyklech formou každý vrchol se všemi ostatními. Z toho vyplívá, že prvotní vytvoření a inicializace objektu se provede v čase $\mathcal{O}\left(\frac{n\cdot(n-1)}{2}\right)$, stejně jako každé následné ohodnocení jednoho jedince.

Tabulka 3.1
Přehled estetických kritérií

1. Součet délek všech hran
2. Součet absolutních odchylek délek hran oproti ideální teoretické délce hrany
3. Multiplikativní inverze součtu vzdáleností mezi všemi vrcholy
4. Součet absolutních odchylek vzdáleností vrcholů od středu oproti ideální vzdálenosti od středu
5. Součet vzdáleností všech bodů od středu
6. Multiplikativní inverze součtu vzdáleností všech vrcholů od hran
7. Multiplikativní inverze součtu vzdáleností mezi všemi hranami

Druhý objekt pro ohodnocování, který byl navrhnut a použit pro testování, počítá fitness jako vážený součet estetických kritérií uvedených v [Tabulka 3.1] a hodnoty potenciální energie spočtené stejným postupem jako v její samostatné implementaci z předchozího odstavce. Jednotlivá kritéria jsou spočtena v sadě vnořených cyklů tak, jak je naznačeno v [Algoritmus 3.2]. Toto uspořádání nemá vliv na asymptotickou časovou složitost celého výpočtu, avšak napomáhá k minimalizaci režijních nákladů na provádění cyklů a znovu využívá jednou spočtených hodnot. V následujících dvou odstavcích bude podrobněji popsán způsob, jakým je implementován výpočet vzdálenosti vrcholů od hran a vzdálenosti hran od ostatních hran. Výpočty ostatních kritérií jsou triviální a nebudu je zde tudíž popisovat.

Výpočet vzdálenosti daného vrcholu od hrany vychází z výpočtu kolmé vzdálenosti bodu od přímky prostředky analytické geometrie. Pokud máme přímku p danou dvěma body A a B, pak jsme schopni

```
Input:     Centroid
           Ideal centripetal distance
           Ideal edge length
           Matrices k_ij and l_ij
-------------------------------------------------------------
Output:    Criteria values
-------------------------------------------------------------
1     edges = empty list;

2     for each vertex v
3     | compute centripetalDistance(v);
4     | compute centripetalDeviation(v);
      |
5     | for every vertex u<v
6     | | compute distance(u,v);
7     | | compute potentialEnergy(u,v);
      | |
8     | | if (u,v) is edge
9     | | | edgeLength = distance(u,v);
10    | | | compute edgeDeviation;
      | | |
11    | | | for each edge in edges list
12    | | | | compute edgeToEdgeDistance(edge, (u,v));
13    | | | end
      | | | add (u,v) to edges list;
14    | | |
15    | | | for each vertex w
16    | | | | compute vertexToEdgeDistance(w, (u,v));
17    | | | end
18    | | end
19    | end
20    end
```

Algoritmus 3.2

Výpočet estetických kritérií

vyjádřit jí v parametrickém tvaru [3.2], kde směrový vektor $\vec{u} = B - A$. Následně potřebujeme zjistit průsečík této přímky s rovinou ρ, která je na tuto přímku kolmá (směrový vektor přímky je zároveň normálový vektor roviny) a leží v ní bod K jehož vzdálenost od přímky hledáme. Tato rovina je tedy jednoznačně dána rovnicí [3.3]. Nyní můžeme vyjádřit průsečík roviny s přímkou spojením těchto dvou rovnic. Po úpravách pak získáme rovnici [3.4], která byla použita k výpočtu parametru t. Pokud bychom parametr t nyní dosadili do rovnice [3.2] získáme souřadnice průsečíku I přímky p a roviny ρ, jehož vzdálenost od bodu K je právě hledanou kolmou vzdáleností bodu K od přímky p.

$$p: X = A + \vec{u} \cdot t \; ; \; t \in \mathbb{R} \qquad [3.2]$$

$$\rho: Y \cdot \vec{u} = K \cdot \vec{u} \qquad [3.3]$$

$$t = \frac{A \cdot \vec{u} - K \cdot \vec{u}}{\vec{u} \cdot \vec{u}} \qquad [3.4]$$

Zde je však nutné si uvědomit, že původním úkolem nebylo hledat vzdálenost vrcholu od přímky, nýbrž od hrany, čili úsečky na přímce. Všechny body na hraně jsou vyjádřitelné pomocí rovnice přímky tak, že omezíme parametr t na interval $\langle 0,1 \rangle$. Z toho plyne, že je potřeba po výpočtu upravit parametr t takto: pokud $t < 0$, pak $t = 0$, nebo pokud $t > 1$, pak $t = 1$. Pokud nyní vypočteme vzdálenost bodu K od bodu daného parametrem t, dostaneme buďto kolmou vzdálenost od hrany nebo vzdálenost od nejbližšího krajního bodu hrany, což jsme chtěli.

Druhým kritériem, jehož výpočet je poměrně náročný, je vzdálenost dvou hran. Toto kritérium je použito jako náhrada za kritérium počtu křížících se hran v rovinném případě zobrazování grafu. Výpočet se bude opírat opět o nalezení kolmé vzdálenosti prostředky analytické geometrie. Tentokrát však mezi dvěma přímkami p a q ve směrnicovém tvaru dle rovnice [3.2]. Vzdálenost mezi dvěma přímkami v prostoru je dána vzdáleností mezi průsečíky těchto přímek a jejich společné příčky d. Příčka je přímka definovaná směrovým vektorem $\vec{w}$, jež je kolmí na směrové vektory obou přímek $\vec{u} = B - A$ a $\vec{v} = D - C$. Tento vektor získáme snadno operací cross product, čili $\vec{w} = \vec{u} \times \vec{v}$. Pokud tedy příčku vyjádříme obecně pomocí rovnic pro obě přímky, získáme rovnici [3.5]. Tato rovnice je ve třech dimenzích řešitelná, protože se jedná o lineární soustavu tří rovnic o třech neznámých $r, s, t \in \mathbb{R}$.

$$p: X_p = A + \vec{u} \cdot r; r \in \mathbb{R}$$

$$q: X_q = C + \vec{v} \cdot s; s \in \mathbb{R}$$

$$X_q = X_p + \vec{w} \cdot t; t \in \mathbb{R} \qquad [3.5]$$

Pro řešení těchto rovnic jsem zaimplementoval jednoduchý řešitel, založený na Gaussově eliminační metodě. Tento řešitel je schopný detekovat pravděpodobný výskyt singulární matice, což značí fakt, že přímky jsou rovnoběžky. V případě, že je nalezeno řešení, pak

nalezené parametry r a s, spolu s příslušnými rovnicemi přímek, jednoznačně určují body, jejichž vzájemná vzdálenost je námi hledaná vzdálenost přímek. Opět jako v předchozím odstavci však musíme provést úpravu parametrů do intervalu $\langle 0,1 \rangle$ a potom teprve zjišťovat vzdálenost bodů. Ve výše zmíněném případě, kdy se zřejmě jedná o rovnoběžky, musíme zjistit vzdálenosti mezi všemi kombinacemi krajních bodů hran a kolmou vzdálenost přímek. Výsledná vzdálenost je pak rovna minimu z těchto všech hodnot.

Výpočet jednotlivých kritérií je především v prostoru značně náročný a skýtá mnohá úskalí. Obecně je celková výpočetní složitost pro všechna kritéria rovna $\mathcal{O}(n^2 + m^2 + m \cdot n)$. Výsledná hodnota fitness, která je po výpočtu vrácena, je, jak již bylo řečeno, rovna váženému součtu jednotlivých kritérií, jejichž váhy jsou vstupním parametrem při konstrukci objektu této fitness. Zde bych uvedl, že pro správné vážení jednotlivých kritérií je nutné tyto vhodně normovat. Jak je vidět, tak výsledek výpočtu všech kritérií je v jednotkách kreslící plochy, avšak jednotlivá kritéria jsou závislá na počtu různých prvků grafu (například deviace hran na počtu hran). Tohoto poznatku bylo využito pro aproximaci normování jednotlivých kritérií v této práci tak, že je každé kritérium děleno číslem odpovídajícím počtu iterací při jeho výpočtu.

3.2 Třídy a rozhraní

Knihovna byla navržena tak, aby svou strukturou umožňovala maximálně jednoduché testování nových metod rozmisťování i jednotlivých dílčích prvků použitých v již existujících metodách. Z tohoto důvodu bylo podstatné navrhnout širokou škálu rozhraní, která umožňují pracovat s konkrétními implementacemi ve smyslu polymorfizmu. Kromě těchto rozhraní byly také navrhnuty některé abstraktní třídy implementující základní postupy. Kompletní přehled všech tříd a rozhraní spolu se stručným popisem je obsahem přílohy. Některé vybrané třídy budou popsány podrobně v následujících kapitolách vzhledem k jejich fundamentálnímu charakteru.

3.2.1 Instance grafu

Objekt grafu je vstupním prvkem všech navržených metod rozvrhování grafů, proto byla implementaci této třídy věnovaná maximální péče. Pro zaručení co největší variability obsahuje tato třída velkou škálu konstruktorů pro zpracování různých reprezentací vstupního grafu. Mezi základní patří konstruktor zpracovávající data v podobě celočíselné (případně reálné) matice incidence. Další varianty pak slouží ke zpracovávání různě zaznamenaných matic incidence v textové podobě. Oba přístupy zpracovávají pouze dolní polovinu matice, protože předpokládají vstup v podobě neorientovaného grafu, čili symetrické matice incidence. Toto omezení bylo nezbytné, vzhledem k tomu, že by bylo nutné při překladu grafu orientovaného na neorientovaný řešit případné nekonzistence ve vahách antisymetrických hran. Zde je nutné říct, že v knihovně existuje také třída pro čtení textových souborů a jejich následné převedení do objektu grafu, avšak nebyla použita při testování.

Objekt grafu udržuje reprezentaci grafu ve dvojí podobě. Jednou je binární matice incidence, která slouží k zpracovávání grafu bez ohledu na váhy hran. Druhou je reálná matice incidence, kde jsou udržovány informace o váhách. Kromě těchto datových struktur, které vznikají přímo uvnitř příslušného konstruktoru, udržuje objekt grafu ještě další data, jež byla při vytvoření spočtena, určená ke zrychlení návazných výpočtů.

Pomocí Floyd-Warshallova algoritmu (4) jsou na konci běhu každého konstruktoru vypočteny nejkratší cesty mezi všemi uzly a to jak v počtech hran, tak jako sumy vah těchto hran. Matice s těmito údaji jsou v objektu grafu uschovány a dá se na ně dotazovat. Stejně tak je nalezena a zaznamenána délka maximální cesty pro obě matice nejkratších cest. Poslední fáze pak vytvoří spojový seznam všech hran grafu, aby bylo možné snížit časovou náročnost některých výpočtů, především pak znatelnou u řídkých grafů.

Shrneme-li představené vlastnosti objektu reprezentujícího zpracovávanou grafovou strukturu, je možné říct, že objekt grafu má velké množství způsobů, jakými může být vytvořen. Kromě matic incidence, sloužících k reprezentaci struktury grafu a vah jeho hran, jsou vytvořeny i další užitečné údaje a struktury. Takovýto postup vytváření má pak asymptotickou časovou složitost $\mathcal{O}(n^3)$.

3.2.2 Rozhraní vizualizátoru

Rozhraní vizualizátoru (GraphDrawer) a abstraktní třída, která ho implementuje, jsou základní stavební kameny používané pro tvorbu konkrétních implementací pro rozvrhování grafů. Samotné rozhraní umožňuje zpracovávání navržených metod zobrazování grafů ve smyslu polymorfizmu. Oproti tomu abstraktní třída slouží pro implementaci základních metod práce s nalezeným zobrazením, které jsou nezávislé na konkrétní implementaci. Mezi zajímavější z těchto metod patří centerGraph a scaleGraph, které slouží na doladění naleznutého rozvržení. Jediná metoda, jež zůstává neimplementována, je metoda placeGraph, která provádí samotné rozmístění vrcholů grafů.

3.2.3 Modulární genetický algoritmus

Navrhnutá knihovna pro rozmisťování vrcholů grafů využívá ve velké míře generační genetický algoritmus jako obecnou metodu optimalizace. S ohledem na tuto skutečnost byla navrhnuta také implementace genetického algoritmu, jako maximálně modulárního rámce s širokou škálou rozhraní a abstraktních tříd.

Třída GraphDrawerGAAbstractInstance obsahuje kompletní implementaci metody placeGraph tak, jak je to popsáno v [Algoritmus 3.3], spolu se všemi potřebnými proměnnými, pro řízení běhu tohoto genetického algoritmu. Třídy, které jsou jejími potomky, pak implementují metody pro křížení a mutaci. Testování pravděpodobnosti aplikace operátorů mutace a křížení je potřeba provádět v následnických třídách a to kvůli možnosti sekvenční aplikace několika operátorů s nezávislým testováním pravděpodobnosti.

Nakonec je následnická třída povinná implementovat ještě inicializační metodu, která provádí zaprvé inicializaci pomocných objektů, jako je

```
Input:                          Graf    G
                  Velikost populace    PopulationSize
                     Počet generací    GenerationLimit
-------------------------------------------------------------
Output: Nejlepší nalezené řešení    BSF
-------------------------------------------------------------
1    initialize random population;
2    evaluate population;

3    for GenerationLimit do
4    | BSF = get best from population;
5    | create empty NewPopulation;
6    | add BSF into NewPopulation;
     |
7    | while(NewPopulation is not full)
8    | | parents = selectParents();
9    | | offspring = crossover(parents);
10   | | mutate(offspring);
11   | | add offspring into NewPopulation;
12   | end
13   |
14   | evaluate(NewPopulation);
15   | replace population;
16   end
```

Algoritmus 3.3 - Generační genetický algoritmus

například ohodnocovací mechanismus, případně operátor selekce, a zadruhé inicializaci počáteční populace.

Jedinec (reprezentace řešení) je v knihovně reprezentován objektem, který v sobě uchovává chromosom, hodnotu své fitness a informaci o tom, jestli byl od poslední změny chromosomu ohodnocen. Chromosom řešení je v mém případě reprezentován polem referencí na třírozměrné vektory bodů. Z pohledu teorie genetických algoritmů je pak překlad na fenotyp a genotyp totožný s chromosomem.

Všechny implementované operátory křížení a mutace jsou koncipovány jako statické metody umístěné v jedné třídě operátorů. Obecně byl dodržen model, kdy jsou rodiče předány k reprodukci v poli spolu s požadovanými parametry. Na druhou stranu operátory mutace přijímají vždy jednoho jedince a parametry. Operátory nevracejí žádnou

hodnotu, protože změny jsou prováděny přímo v dodaném jedinci či poli jedinců.

Nakonec operátory selekce, fitness funkce a ohodnocovače jsou vždy implementací příslušného rozhraní a je možné je v odvozených třídách metod zobrazování pomocí genetických algoritmů jednoduše nahradit přepsáním inicializační funkce. Tato koncepce umožňuje velice snadno provádět testování například nové fitness funkce odvozením třídy od již fungujícího vizualizátoru a přepsáním metody inicializace v rozsahu několika řádků.

4 Experimenty

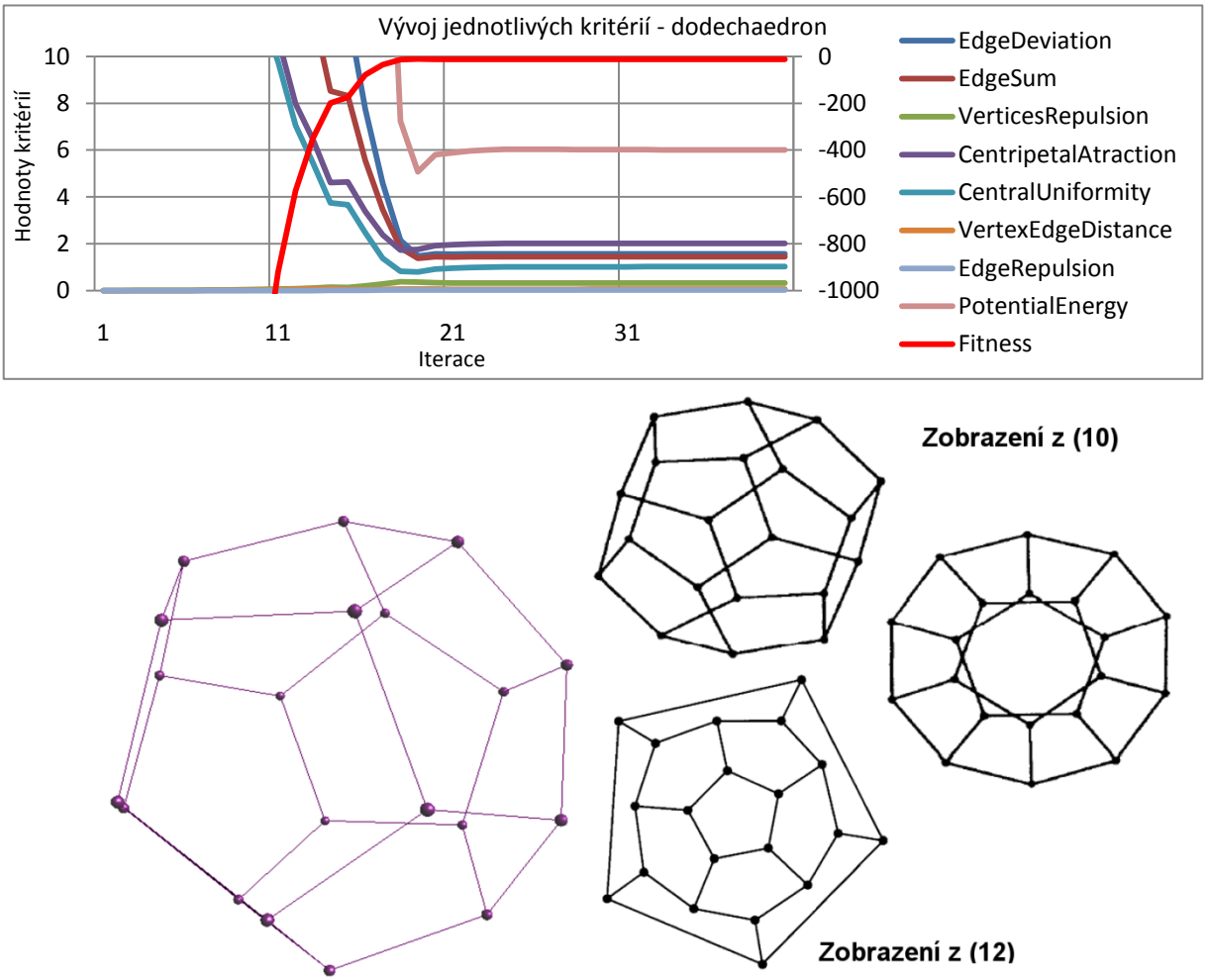
V následující kapitole představím výsledky experimentů rozmisťování některých grafů převzatých z literatury. Experimentovat budu s algoritmy implementovanými v předkládané knihovně, přičemž výsledné vizualizace jsou prováděny pomocí vizualizačního nástroje, který vytvořil Tomás García-Pozuelo Barrios (viz. Poděkování). Zde bych rád předem upozornil na poněkud problematickou vizualizaci výsledků rozmisťování grafových struktur v prostoru pomocí obrázků. Je zcela pochopitelné, že při absenci interaktivního prostředí nemusí být takovýto výsledek čitelný. Přesto jsem se pokusil používat takové úhly pohledů, aby byly výsledky čitelné.

Postupně představím výsledky získané zobrazováním simulovaným pohybem, pro které vždy ukážu ustálené hodnoty jednotlivých estetických kritérií. V další části pak nejprve korektně naladím genetický algoritmus a takto připravený ho použiji pro demonstraci účinků různých estetických kritérií použitých samostatně. Nakonec předvedu účinnost některých vhodně zvolených skupin estetických kritérií, přičemž výsledky budu prezentovat získanými vizualizacemi a záznamy vývoje hodnot jednotlivých kritérií.

4.1 Simulovaný pohyb

V této kapitole budu pro rozmisťování některých předem vybraných grafů z literatury používat rozmisťování simulovaným pohybem. Parametry pro testování byly nastaveny takto; $timeStep = 1$, $mass = 1$, přičemž iterace probíhá do ustálení celého modelu. Jednotlivé vizualizace grafů budou předkládány s odkazy na literaturu, kde bylo jejich zobrazování také testováno. Problematické je srovnání mích výsledků s literaturou, vzhledem k tomu, že v literatuře se jedná o rozmisťování ve 2D. Z toho důvodu budu srovnání provádět pouze popisně.

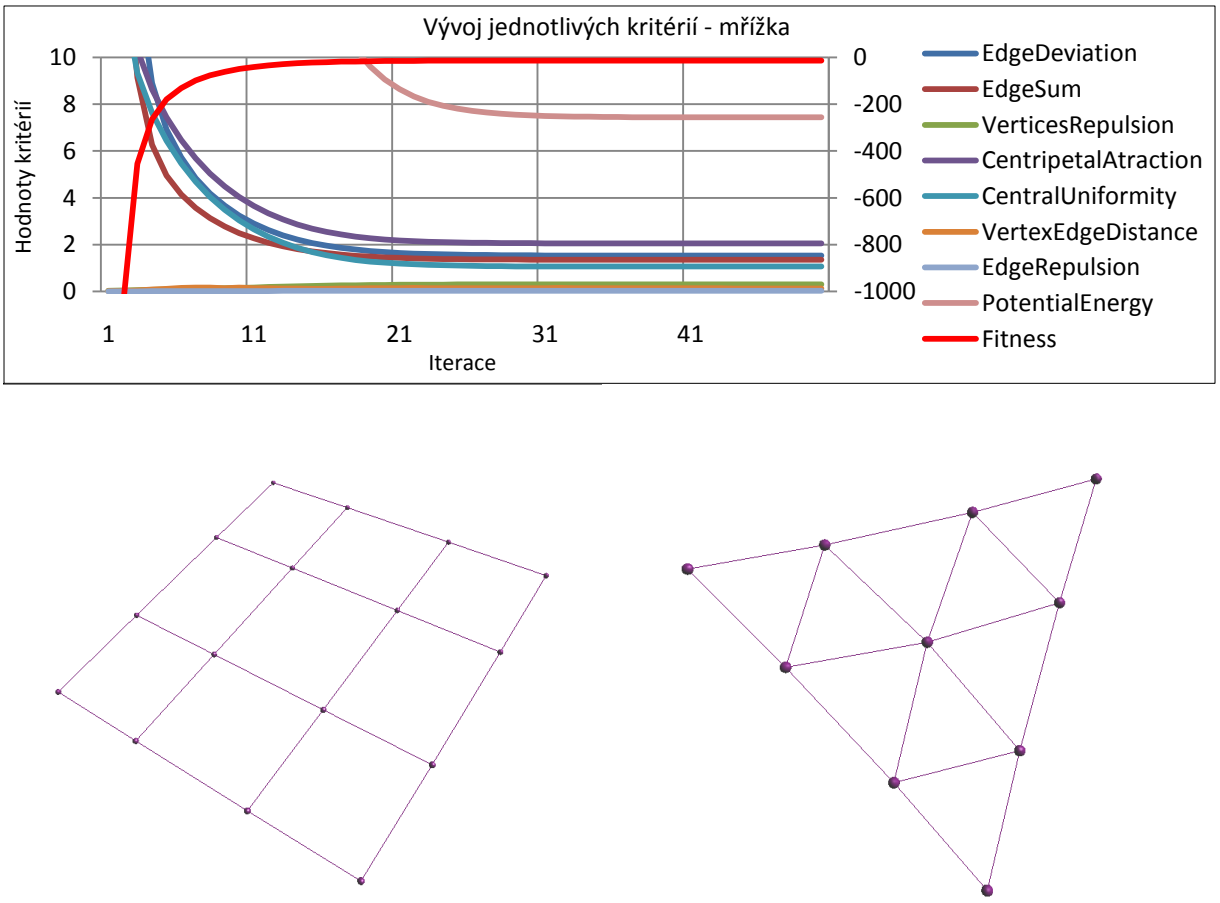
Jako první bylo provedeno rozmístění grafu dodechaedronu převzatého z (12) a testovaného v (10), (13) a (17). Tento graf má velice zajímavé rozmístění v rovině, vzhledem k tomu, že je planární. Jeho nativní podoba je však prostorová. V [Obr. 4.1] je zobrazeno jeho prostorové rozmístění pomocí metody simulovaného pohybu pro 3D. Je zajímavé, že metoda simulovaného pohybu pro 2D najde zobrazení, které je dvourozměrnou projekcí stejného zobrazení. Tyto závěry pak byly spolu s dalšími možnými 2D zobrazeními popsané i v literatuře. Z grafu vývoje hodnot fitness, které jsou během simulace modelu měřeny, je vidět, že model se přibližně po 25 iteracích zastavil, přičemž jednotlivé složky se ustálili v různých hladinách.



Obr. 4.1

Zobrazení grafu dodechaedronu pomocí metody simulovaného pohybu

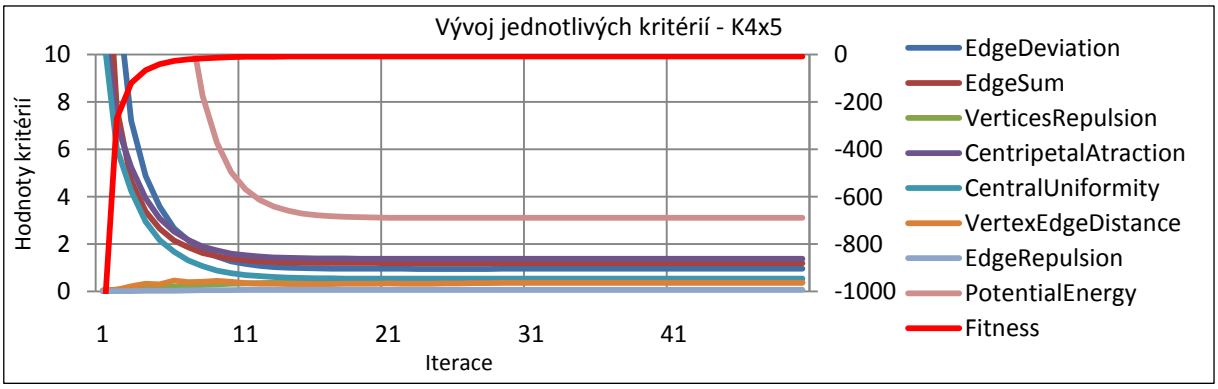
Dalším experimentem je zobrazení mřížky 4x4 vrcholy (případně její trojúhelníková podoba) [Obr. 4.2]. Tento graf je velice často v literatuře používán pro demonstraci zobrazovacích schopností jednotlivých metod. Graf byl opět převzat z (12) a dále prezentován v (10), (13) nebo (18). Doba ustálení tohoto modelu závisí na kvalitě náhodné inicializace a zároveň na velikosti mřížky. Z přiloženého grafu je možné pozorovat ustálení za méně než 40 iterací. Na ilustraci je také vidět, že nalezené zobrazení je sice v prostoru, ale v zásadě je rovinné. Z toho důvodu nepřekvapí, že je řešení shodné s nejlepšími nalezenými vizualizacemi pro 2D v literatuře.

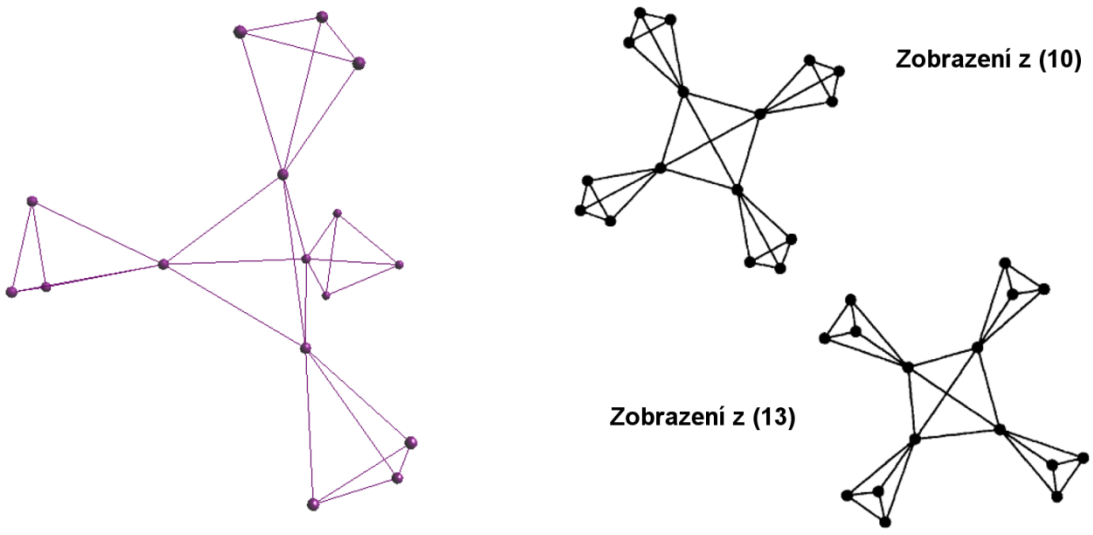


Obr. 4.2
Zobrazení grafu mřížky metodou simulovaného pohybu

Velice oblíbený je pro demonstrační účely graf převzatý z (12) a zobrazovaný v (10), (13) a (17), který je složen z pěti symetricky

spojených úplných grafů K4 (dále jako K4x5). Jeho zobrazení pomocí mého vizualizátoru na [Obr. 4.3] je symetrické a krásně ukazuje vlastnosti každé komponenty. Vzhledem k tomu, že graf K4 je planární, byla v literatuře prezentována také nalezená rozmístění, která zobrazila výběžky K4 v rovině bez křížení. Při zachování proporcí grafu je však nemožné zobrazit bez křížení prostřední komponentu K4. Tato zobrazení pak měla prostřední komponentu jinou než výběžky, přesto že je struktura všech prvků stejná. V tomto ohledu se zobrazení ve 3D vyplatilo, protože mé zobrazení ukazuje shodnou strukturu všech pěti částí grafu.

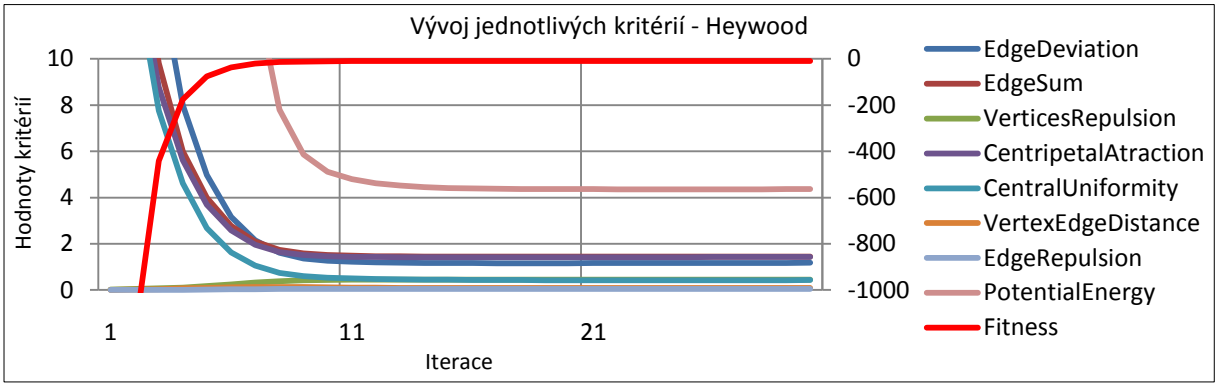


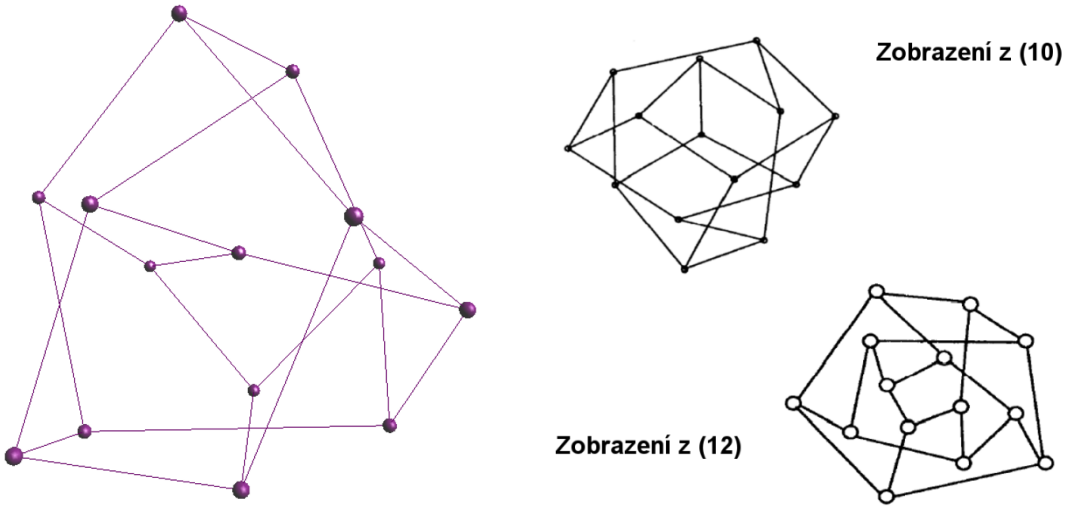


[Obr. 4.3]
Zobrazení grafu K4x5 metodou simulovaného pohybu

Posledním experimentem je pak zobrazení tzv. Heywoodova grafu [Obr. 4.4] převzatého také z (12) a pokusně zobrazeného ve 2D v (10),

(13) a (17). Mnou nalezené řešení promítnuté do obrázku velmi připomíná řešení nalezené v (12), ale v jeho nativní 3D podobě se ukazuje, že řešení krásně zobrazuje strukturu i symetrie. V tomto zobrazení velice připomíná dodechaedron.





[Obr. 4.4]
Zobrazení Heywoodova grafu metodou simulovaného pohybu

4.2 Genetický algoritmus s estetickými kritérii

V této kapitole předvedu a doložím vliv jednotlivých možných nastavení genetického algoritmu a použití různých estetických kritérií na kvalitu a vlastnosti zobrazovaného grafu. Zde je důležité upozornit na fakt, že samotný genetický algoritmus má velké množství parametrů a možných konfigurací. Toto spojené s poměrně velikou škálou představených estetických kritérií způsobuje obrovskou množinu možných kombinací nastavení takovéhoto vizualizátoru. Z tohoto

důvodu se nejdříve zaměřím na testování nastavení jednotlivých operátorů genetického algoritmu. Použiji k tomu graf dodechaedronu, který je dostatečně složitý a z podstaty prostorový. Jako optimalizační kritérium pro tento účel využiji hodnotu potenciální energie, která má vyvážený charakter (jsou v ní zahrnuty jak prvky odpudivosti, tak přitažlivosti vrcholů), oproti ostatním kritériím. Pro zachování jednoduchosti budu po celou dobu testování používat pro mutaci pouze operátor vícenásobné perturbace, který má vhodné vlastnosti lokálního prohledávání a ovlivňuje výsledky rovnoměrně, jelikož je aplikován vždy. Díky tomu, že je tento operátor natolik specifický, nebude při testech vadit ani poměrně vysoká pravděpodobnost mutace. Jako selekční operátor, jsem pak pro všechny následující experimenty použil ruletový výběr nad Linear-Rank, protože umožňuje vhodným způsobem regulovat selekční tlak.

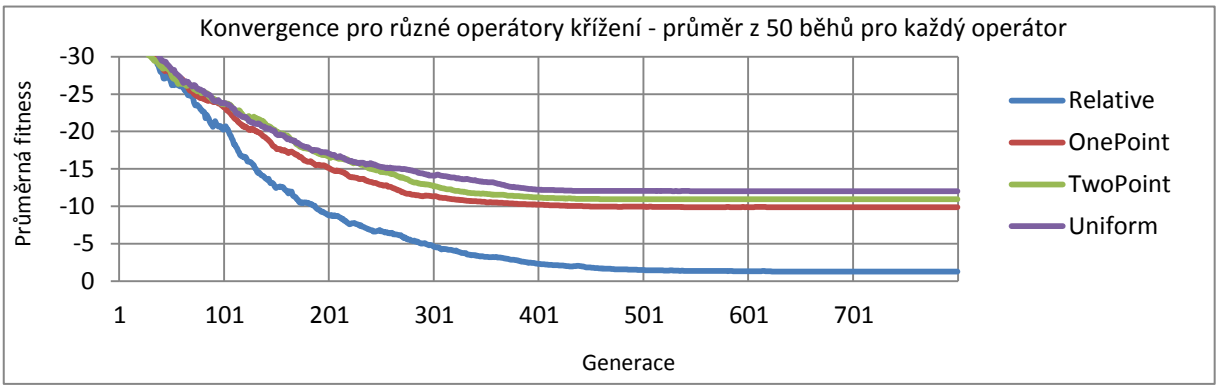
Tabulka 4.1

Nastavení GA pro testy operátorů

Velikost populace	200
Počet generací	800
Selekční tlak	1,05
Pravděpodobnost křížení	85%
Pravděpodobnost mutace	65%

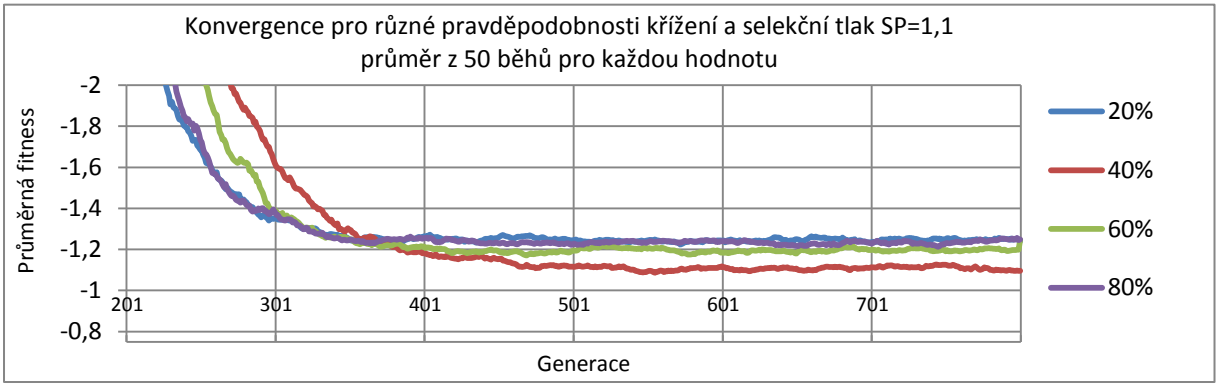
4.2.1 Nastavení operátoru křížení

Nejdříve srovnám použití různých operátorů křížení. Porovnání budu provádět s nastavením podle [Tabulka 4.1]. Tyto hodnoty byly zvoleny na základě osvědčeného nastavení při podobné implementaci v (17). Provedl jsem pro každý operátor křížení 50 rozmístění a sledoval jsem hodnotu průměrné fitness. Získaná data jsem následně zprůměroval a vynesl do grafu v [Obr. 4.5]. Z grafu je vidět, že operátor nazývaný jako relativní křížení má nejlepší vlastnosti, protože ostatní operátory mají omezenou konvergenci. Z tohoto důvodu budu dále používat pro všechna měření a testy jen operátor relativního křížení.



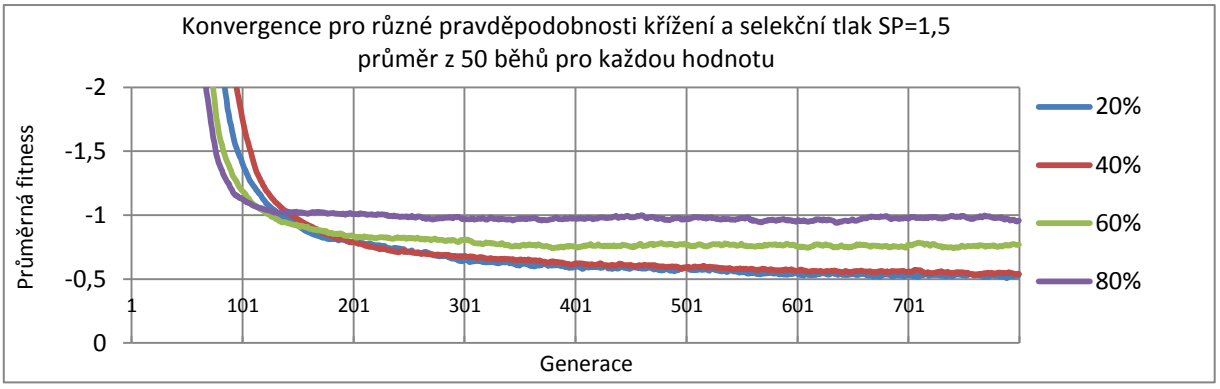
Obr. 4.5

Nyní budu pro genetický algoritmus testovat vliv různých nastavení pravděpodobnosti aplikace operátoru křížení na rychlost konvergence. Opět budu fixovat ostatní hodnoty podle [Tabulka 4.1] a použiji relativní operátor křížení, jak bylo řečeno výše. Vzhledem k tomu, že na účinnost operátoru křížení má zásadní vliv selekční tlak, budu provádět dvě sady měření. První sada zobrazená v [Obr. 4.6] byla naměřena pro selekční tlak $SP = 1{,}1$. Z grafu je patrné, že obě extrémní hodnoty zrychlují konvergenci, ale za cenu horšího výsledku. Nejlépe pro tento selekční tlak dopadla hodnota pravděpodobnosti 40%.



Obr. 4.6

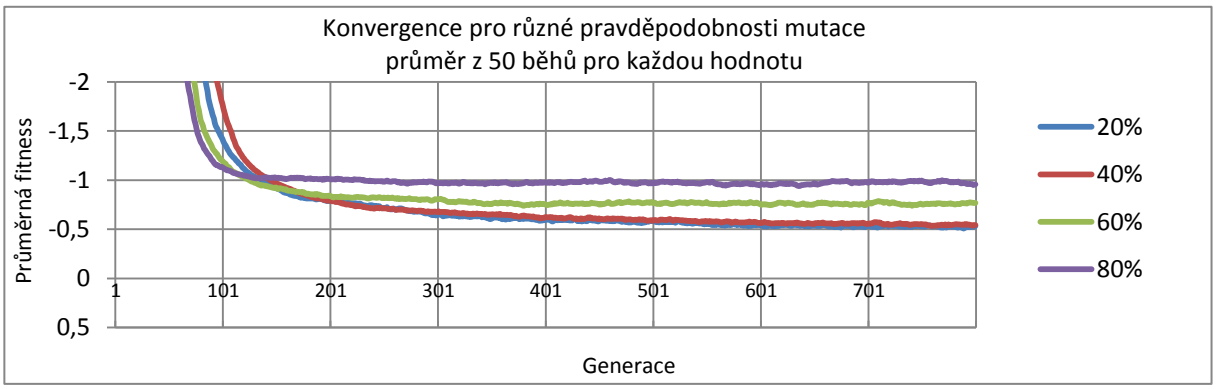
Druhé měření, jehož výsledky jsou zobrazené v [Obr. 4.7] bylo prováděno se selekčním tlakem $SP = 1{,}5$. Naměřené hodnoty přesvědčivě ukazují, že při téměř stejné rychlosti konvergence zde dosahují nízké hodnoty pravděpodobnosti aplikace operátoru křížení mnohem lepších výsledků. Z těchto měření jsem usoudil, že vhodná hodnota pravděpodobnosti aplikace operátoru křížení bude 40%.



Obr. 4.7

4.2.2 Nastavení operátoru mutace

Stejně jako v předchozím odstavci provedu nyní testování vlivu pravděpodobnosti mutace na rychlost konvergence. Měření začnu na hodnotě 20% a budu ji vždy o stejnou hodnotu zvětšovat. Ostatní nastavení zachovám tak, jak bylo popsáno v [Tabulka 4.1]. Opět budu provádět pro každé nastavení 50 měření a sledovanou hodnotou bude průměrná fitness. Toto měření nám ukáže, jakým způsobem pravděpodobnost mutace ovlivní konvergenci algoritmu. Výsledky měření prezentované v [Obr. 4.8] ukazují, že vysoké hodnoty pravděpodobnosti vedou k rychlejší konvergenci ovšem následně k horší ustálené hodnotě. Nízké hodnoty naproti tomu konvergují o několik málo generací později, zato však do mnohem lepších hodnot. Ze závěru tohoto měření plyne, že použiji hodnotu 35%.



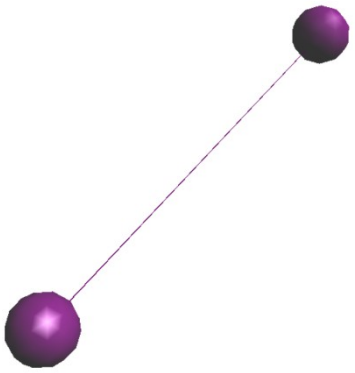
Obr. 4.8

4.2.3 Testování estetických kritérií

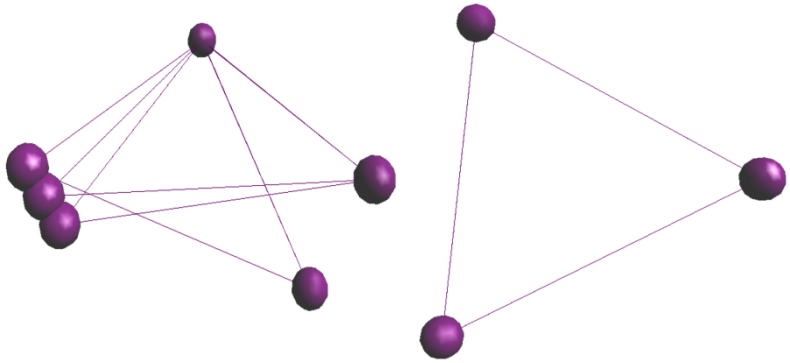
Nyní, když jsem odvodil korektní nastavení genetického algoritmu, mohu se pokusit popsat a ukázat funkčnost jednotlivých estetických kritérií. Na začátku popíši, jak se chovají jednotlivá kritéria sepsaná v [Tabulka 3.1], pokud jsou aplikovaná samostatně. Následně vyberu některé vhodné kombinace a na příkladech popíši jejich chování.

Minimalizace délek všech hran stlačí dle očekávání každou komponentu grafu do jednoho bodu, přičemž vzájemná pozice těchto bodů bude náhodná.

Minimalizace odchylek všech hran od stanoveného optima vede na deformovaný graf. Tento graf se složí do minimální velikosti, jakou mu dovolí jeho symetrie. Velice zajímavé je to například u mřížky, která se složí na jeden segment [Obr. 4.9], a u dodechaedronu, jež se ustálí v pozici rovnostranného trojúhelníku [Obr. 4.10].



Obr. 4.9
Výsledek aplikování samostatného kritéria deviace hrany na graf mřížky

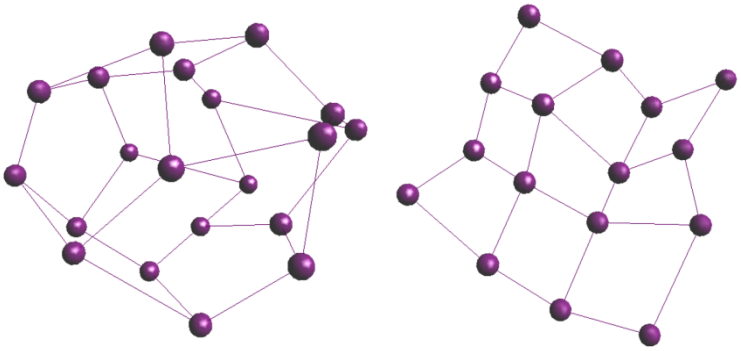


Obr. 4.10
Výsledek aplikování samostatného kritéria deviace hrany na graf dedechaedronu

Minimalizace vzdálenosti od středu stlačí veškeré body do středu kreslící plochy, bez ohledu na cokoliv. Oproti tomu minimalizace odchylky od ideální vzdálenosti od středu umístí veškeré vrcholy grafu na povrch sféry okolo středu o daném poloměru. Standardně by pak body na této sféře měli být umístěny náhodně, ovšem nativní chování operátoru křížení způsobí samovolné umístění všech vrcholů do jednoho bodu na povrchu sféry.

Konečně veškeré multiplikativní inverze vzdáleností, které působí jako repulze mezi body či hranami navzájem, způsobí při samostatném použití doslova rozstřelení grafu do nekonečna. V tomto případě pak pochopitelně selže zobrazovací nástroj. Zde je důležité zdůraznit fakt, že kromě kritérií optimalizujících deviaci, byla všechna předchozí kritéria nějakým způsobem slučující, přičemž repulsivní kritéria tvoří pravý opak. Jak bude popsáno dále, je právě vhodná kombinace kritérií z obou skupin účinným prostředkem vykreslování grafů.

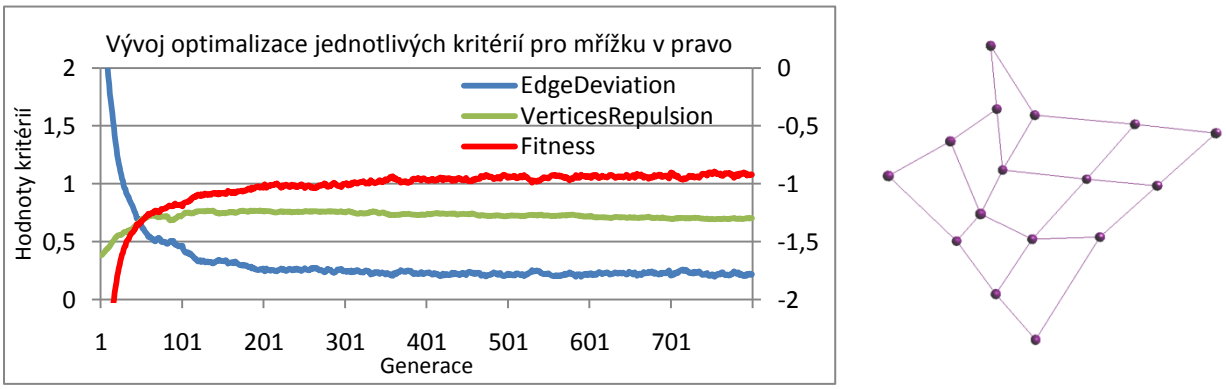
Důvod proč bylo právě kritérium potenciální energie používáno při ladění genetického algoritmu, tkví v jeho jedinečné vlastnosti, protože působí zároveň repulsivně i pojivě. Tato vlastnost je dána konstrukcí popsanou výše. Při jeho samostatném použití bylo nalezeno například zobrazení dodechaedronu či mřížky v [Obr. 4.11].



Obr. 4.11
Nalezená zobrazení pro graf mřížky a dodechaedronu pomocí kritéria potenciální energie

Konečně ukážu některé skupiny estetických kritérií, které vhodným nastavením vah vytváří hezké nakreslení některých grafů. První takovou skupinou je doplnění deviace hrany o repulzi uzlů. Tato dvojice se snaží o vykreslení hran stejně dlouhých a přitom díky repulzi nedochází ke skládání, které bylo popsáno výše. Tímto způsobem bylo získáno zobrazení mřížky z [Obr. 4.12]. Na grafu vývoje jednotlivých kritérií k tomuto zobrazení můžeme pozorovat, jakým způsobem nejprve vysoká hodnota kritéria deviace hran donutí odpudivost vrcholu růst. Obě kritéria se pak ustálí v odpovídajícím rovnovážném stavu. Při

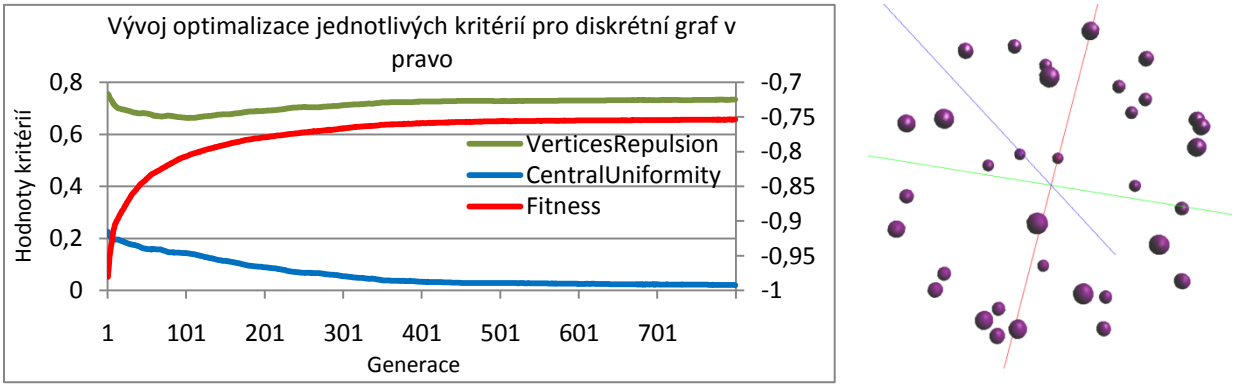
bližším pohledu si však můžeme všimnout, že po ustálení deviace délky hran dochází k jemnému doladění kritéria repulze.



Obr. 4.12

Nalezené zobrazení pro graf mřížky pomocí kritéria deviace hrany a repulze vrcholů spolu se záznamem vývoje jednotlivých kritérií a fitness

Další zajímavou dvojicí kritérií je například repulze vrcholů doplněná o deviaci vzdálenosti od středu kreslící plochy. Tato kombinace funguje výborně pro rozmístění vrcholů diskrétního grafu v prostoru. Jak bylo naznačeno v předchozích odstavcích, způsobí tato kombinace kritérií aproximaci rovnoměrného rozdělení vrcholů po povrchu sféry kolem středu kreslící plochy. Výsledné zobrazení spolu se záznamem průběhu fitness můžete vidět v [Obr. 4.13].



Obr. 4.13

Nalezené zobrazení pro diskrétní graf pomocí kritéria deviace vzdálenosti od středu a repulze vrcholů spolu se záznamem vývoje jednotlivých kritérií a fitness

4.3 Zhodnocení experimentů

Na základě provedených experimentů, bylo zjištěno, že metoda simulovaného pohybu funguje velmi dobře na širokou škálu různých grafů. Výsledky byly konfrontovány s jejich protějšky z literatury, což kvalitu mích zobrazení potvrdilo. Během experimentů jsem také zjistil, že existují také grafy, pro které metoda selže. Jedná se především o grafy, které nejsou souvislé.

Dále bylo zjištěno vhodné nastavení pro operátory genetického algoritmu. S pomocí takto naladěného genetického algoritmu se mi následně povedlo předvést chování jednotlivých estetických kritérií, jež odpovídalo předpokladům. Následně jsem vhodně kombinoval estetické operátory a dosáhl jsem tak hezkého zobrazení prezentovaných grafů. Kromě toho jsem ukázal, že estetická kritéria obstojí i tam kde metoda simulovaného pohybu selhala.

5 Další práce

Tato bakalářská práce se zabývala velmi širokým tématem, a proto je jasné, že nebylo možné v dostatečném rozsahu probádat všechna zákoutí, která téma skýtá. V průběhu samotné práce šlo především o seznámení se základními užívanými postupy na poli zobrazování grafových struktur, stejně jako se slepými uličkami. Podstatným výsledkem je pak implementovaná knihovna, kterou využiji v maximální míře při dalším výzkumu.

V další práci na tomto tématu se budu především zabývat použitím dalších technik na principu estetických kritérií pro prostorové zobrazování grafů. Především to bude opravdu multikriteriální optimalizace problému. S touto formulací bude také souviset návrh a testování účinnějších operátorů křížení a mutace pro prostorovou variantu problému kreslení grafů. Jak bylo ukázáno v předchozí kapitole, jsou operátory dobře fungující pro případ 2D velmi neúčinné ve 3D variantě problému.

Dalším směrem pro výzkum bude použití metody POEMS (19) a to jak její jednokriteriální, tak multikriteriální varianty. Tento výzkum je inspirován především velice dobrými, ale uniformními řešeními, která dává postup založený na pružinovém modelu. Chci se pokusit o primární rozmístění grafu pomocí tohoto algoritmu a jeho následné ladění pomocí estetických kritérií. Tento postup bych také rád implementoval pomocí vícestupňového optimalizačního procesu. Zde bude především záležet na kvalitním operátoru mutace, který umožní vytvořit na základě jednoho řešení celou populaci s dostatečnou diversitou.

Kromě nových operátorů pro genetické algoritmy, bych se také rád zabýval vývojem nových estetických kritérií, která lépe využijí informace získané přímo z grafové struktury. Mezi takováto kritéria budou patřit především ta, založena na úhlech, které mezi sebou svírají sousední hrany, a kritéria rozmisťující primárně středy grafu. Kromě toho bych

rád provedl revizi již implementovaných kritérií, založených na geometrických výpočtech, abych je zrychlil pomocí nejnovějších postupů na poli výpočetní geometrie pro 3D.

Poslední výzvou pro více interaktivní uživatelské rozhraní bude implementace grafového metody pro zobrazování grafů v samostatném vlákně. Tato implementace umožní, jak zobrazování průběžných výsledků během výpočtu, tak zásahy uživatele do běhu rozmisťování za běhu. Tento postup se velice osvědčil v článku (17).

6 Závěr

V předkládané bakalářské práci jsem se zabýval vizualizací grafových struktur. K tomuto problému jsem přistoupil jako k optimalizační úloze. Zadání optimalizační úlohy jsem pak formuloval v prvé řadě pomocí estetických kritérií. Následnou optimalizaci jsem prováděl především prostředky genetických algoritmů.

Během vypracovávání jsem do hloubky nastudoval, jak literaturu zaměřenou na zobrazování grafů pomocí klasické metody pružinového modelu, tak nově studované a implementované přístupy založené na estetických kritériích a stochastických optimalizačních metodách. Podrobně jsem se seznámil s metodikou rozboru pojmů estetiky a vizuálního vnímání grafových struktur.

Na základě těchto znalostí jsem navrhl a implementoval modulární knihovnu pro zobrazování grafových struktur, která je především snadno rozšiřitelná o nové prvky a umožňuje rychlé testování nových postupů. Předkládaná knihovna obsahuje metodu využívající pružinový model, který je optimalizován postupem zjednodušené diskrétní simulace. Kromě tohoto přístupu byl také implementován genetický algoritmus se širokou škálou genetických operátorů a sadou osmi estetických kritérií. Současně s důrazem na modularitu celého řešení jsem kladl při implementaci stejný důraz na její výkonnost.

Knihovnu jsem pomocí existujícího vizualizačního nástroje testoval na grafových strukturách převzatých z literatury a podrobně jsem popsal získané výsledky. U estetických kritérií jsem také popsal jejich konkrétní chování a následně navrhnul a předvedl některé jejich vhodné kombinace. Na základě výsledků testování všech částí jsem specifikoval směry budoucích možných vylepšení a úprav, kterými bych se chtěl dále zabývat.

Citovaná literatura

1. **Di Battista, Giuseppe, et al.** *Algorithms for Drawing Graphs: an Annotated Bibliography.* June 1994. Computational Geometry: Theory and Applications.
2. **Demel, J.** *Grafy a jejich aplikace.* Praha : Academia, 2002. ISBN 80-200-0990-6.
3. **Chartrand, Gary.** *Introductory Graph Theory.* Dover : Dover Publications; Unabridged edition, 1984. ISBN 0486247755.
4. **Cormen, H. Thomas, a další.** *Introduction to algorithms.* Massachusetts : MIT Press, 2001. 9780262032933.
5. **Kircpatrick, S., Gellat, C. D. a Vecchi, M. P.** Optimization by Simulated Annealing. *Science.* New Series, May 1983, 220, stránky 671-680.
6. **Koza, J. R.** *Genetic programming: on the programming of computers by means of natural selection.* Massachusetts : MIT Press, 1992. 9780262111706.
7. **Steuer, R. E.** *Multiple Criteria Optimization: Theory, Computations, and Application.* New York : John Wiley & Sons, 1986. 047188846X.
8. **Eades, Peter.** A heuristic for graph drawing. *Congressus Numerantium.* 1984. 42, stránky 149-160.
9. **Kamada, Tomihisa a Kawai, Satoru.** *A spring modelling algorithm.* místo neznámé : Elsevier Science B.V., 1989.
10. **Fruchterman, Thomas M. J. a Reingold, Edward M.** Graph Drawing by Force-directed Placement. 11 *Software - practice and experience.* New York : John Wiley & Sons, 1991. 21, stránky 1129-1164.
11. **Kumar, Aruma and Fowler, Richard H.** *A Spring Modeling Algorithm to Position Nodes of an Undirected Graph in Three Dimensions.* [Technical report] Edinburg, Texas : University of Texas-Pan American, University of Texas - Pan American. 78539.
12. **Davidson, R. a Harel, D.** Drawing graphs nicely using Simulated Annealing. *ACM Trans. Graph.* 1996. stránky 301-331.
13. **Coleman, Michael K. a Parker, Scott D.** Aesthetics-based Graph Layout for Human Consumption. *Software: Practice and Experience.* místo neznámé : John Wiley & Sons, Ltd., 1996. stránky 1415-1438.

14. **Holland, J. H.** *Adaptation in natural and artificial systems.* Ann Arbor : The University of Michigan Press, 1975.
15. **Rechenberg, I.** *Evolutionsstrategie - Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Stuttgart : Frommann-Holzboog, 1973.
16. **Fogel, L. J., Owens, J. a Walsh, M. J.** *Artificial Intelligence through Simulated Evolution.* New York : John Wiley & Sons, 1966.
17. **Barreto, André M.S. a Barbosa, Helio J.C.** *Graph Layout Using a Genetic Algorithm.* Petropolis - RJ, Brasil : Proceedings of the Sixth Brazilian Symposium on Neural Networks, 2000.
18. **Branke, Jürgen, Bucher, Frank and Schmeck, Hartmut.** *A genetic algorithm for drawing undirected graphs.* Helsinky-Finland : Proceedings of 3NWGA, 1997.
19. **Kubalík, Jiří a Faigl, Jan.** Iterative Prototype Optimisation with Evolved Improvement Steps. *EuroGP.* Heidelberg : Springer-Verlag Berlin, 2006. stránky 154-165.

Příloha

Package graphdrawer

Interface Summary	
IGraphDrawerInstance	Rozhraní, které musí implementovat každý zobrazovač grafu implementovaný v této knihovně.

Class Summary	
JGraphDrawerAbstractGAGraphDrawer	Základní implementace zobrazovače grafu založeného na optimalizaci pomocí genetického algoritmu.
JGraphDrawerAbstractInstance	Abstraktní implementace rozhraní zobrazovače grafu.
Test	Testovací třída pro ověřování funkčnosti jednotlivých komponent knihovny

Package graphdrawer.exceptions

Exception Summary	
JGAOperatorMisuseException	Výjimka vyvolaná při chybném použití genetických operátorů
JGraphDrawerException	Obecná výjimka této knihovny.
JInvalidInputException	Výjimka vyvolaná při pokusu vytvořit grafovou strukturu z dat, která nemají správný formát.

Package graphdrawer.instance

Class Summary	
JAesthetic3DContinousGraphDrawer	Základní implementace prostorového zobrazovače grafů založená na estetických kritériích a využívající operátor relativního křížení.
JBasic2DContinousGraphDrawer	Dvojdimenzionální varianta jednoduchého zobrazovače grafů založeného na kritériu potenciální energie
JBasic3DContinousGraphDrawer	Implementace prostorového zobrazovače

	založeného na kritériu potenciální energie.
JIterative2DEnergyGraphDrawer	Implementace metody simulovaného pohybu pro dvourozměrné zobrazení grafu.
JIterativeEnergyGraphDrawer	Základní prostorová varianta zobrazovače založeného na metodě simulovaného pohybu.
JOnePointGAAestheticGraphDrawer	Varianta zobrazovače pomocí estetických kritérií s použitým operátorem jednobodového křížení
JTwoPointGAAestheticGraphDrawer	Varianta zobrazovače pomocí estetických kritérií s použitým operátorem dvoubodového křížení
JUniformConvergenceGAAestheticGraphDrawer	Varianta zobrazovače pomocí estetických kritérií s použitým operátorem uniformního křížení.
JUniformGAAestheticGraphDrawer	Varianta zobrazovače pomocí estetických kritérií s použitým operátorem uniformního křížení

Package graphdrawer.utils

Class Summary	
JGraphDrawerFileReader	Třída umožňující zpracovávání textových souborů s definicí grafu ve správném formátu.
JGraphDrawerGraph	Třída instance grafu
LinearSolver	Jednoduchá implementace Gaussovy eliminační metody pro řešení soustavy lineárních rovnic.

Package graphdrawer.utils.GABasics

Interface Summary	
IGraphDrawerGAIndividum	Rozhraní, které implementují jedinci optimalizovaní v genetickém algoritmu.

Class Summary	
JGraphDrawerGAFitnessEvaluator	Základní implementace sekvenčního ohodnocovače populace.

JGraphDrawerGAIndividum	Konkrétní implementace rozhraní jedince použitého v genetickém algoritmu.
JGraphDrawerGAOperators	Třída obsahující statické metody genetických operátorů křížení a mutace
JGraphDrawerGAParallelFitnessEvaluator	Implementace paralelního ohodnocovače populace

Package graphdrawer.utils.GAfitness

Class Summary	
JGraphDrawerAestheticFitness	Základní implementace fitness funkce založené na sadě estetických kritérií.
JGraphDrawerBasicEnergyFitness	Základní implementace funkce fitness založená na výpočtu potenciální energie pružinového systému.
JGraphDrawerConvergenceAestheticFitness	Modifikace funkce fitness využívající váženou sumu estetických kritérií na variantu konvergující do definovaných hodnot.
JGraphDrawerGAAbstractFitness	Abstraktní implementace funkce fitness

Package graphdrawer.utils.GASampling

Interface Summary	
IGraphDrawerGASampler	Rozhraní, které implementují jednotlivé selekční operátory genetického algoritmu.

Class Summary	
JGraphDrawerAbstractRouletteSampler	Abstraktní implementace výběru pomocí ruletového kola.
JGraphDrawerAbstractSampler	Abstraktní implementace rozhraní selekčního operátoru, která poskytuje základní metody pro výběr více jedinců najednou a údržbu.
JGraphDrawerAbstractSUSampler	Abstraktní implementace stochastického univerzálního výběru.
JGraphDrawerGALinearRankSampler	Standardní implementace výběru podle Linear Rank využívající výběr pomocí ruletového kola.
JGraphDrawerGALinearRankSUSampler	Implementace výběru metodou univerzálního

	stochastického výběru nad hodnotami Linear Rank
JGraphDrawerGARouletteSampler	Základní implementace metody ruletového kola proporcionální vůči fitness
JGraphDrawerGASUSampler	Standardní implementace univerzální stochastické výběrové metody proporcionální vůči fitness.
JGraphDrawerGATournamentSampler	Implementace operátoru turnajové selekce