FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Aplikace pro demonstraci funkce genetických algoritmů
Student:	Bc. Adam Kugler
Vedoucí:	doc. Ing. Petr Fišer, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2018/19

Pokyny pro vypracování

Vytvořte aplikaci pro demonstraci funkce genetických algoritmů (GA). Aplikace bude použita pro výuku předmětu MI-PAA.
Základní pedagogické požadavky jsou:
- Přehledné a snadno použitelné grafické rozhraní, aplikace musí být jednoduše spustitelná, bez nutnosti jakékoliv instalace.
- Možnost výběru z několika problémů k řešení, automatické i ruční generování instancí problémů, možnost práce s více instancemi, vracení se k nim.
- Možnost nastavování parametrů GA, implementace několika typů křížení a selekce.
- Sledování běhu GA (graficky), vytváření statistik.

Nejprve proveďte důkladnou analýzu funkčních i ostatních (nefunkčních) požadavků. Za tímto účelem proveďte uživatelský průzkum mezi absolventy předmětu MI-PAA a jeho cvičícími.

Dále navrhněte vhodnou SW architekturu pro implementaci. Zde mějte na zřeteli univerzálnost a rozšiřitelnost (pro další problémy).
Nakonec nástroj adekvátně otestujte, proveďte důkladné uživatelské testy.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 9. prosince 2017

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Aplikace pro demonstraci funkce genetických algoritmů

Bc. Adam Kugler

Katedra softwarového inženýrství
Vedoucí práce: doc. Ing. Petr Fišer, Ph.D.

8. května 2018

Poděkování

Děkuji vedoucímu mé práce doc. Petru Fišerovi za vedení a pomoc při řešení této práce. Díky jeho radám a připomínkám je výsledek mé práce jistě lepší a kvalitnější. Děkuji kolegům, Jaroslavu Veselému a Michalovi Kluzáčkovi, za aktivní spolupráci při vývoji aplikace, kterou se práce zabývá. V neposlední řadě děkuji všem cvičícím a studentům, kteří se zapojili do uživatelského průzkumu nebo uživatelského testování.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 8. května 2018 .....

Odkaz na tuto práci

Kugler, Adam. *Aplikace pro demonstraci funkce genetických algoritmů.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

Abstrakt

Tato práce se zabývá vývojem výukové aplikace, která má pomoci studentům pochopit, jak funguje genetický algoritmus. Jedná se o optimalizační algoritmus, který je inspirován biologickou evolucí. Hlavními prostředky, které studentům umožní prohloubit znalosti v této oblasti, jsou nastavitelné parametry a vizualizace průběhu algoritmu pomocí grafu. Uživatelé si mohou vyzkoušet práci na různých problémech na vlastních či vygenerovaných instancích.

Teoretická část se zabývá později implementovanými strategiemi genetického algoritmu. Praktická část se zabývá vývojem samotné aplikace. Na přání vedoucího byla vytvořena jedna aplikace pro tři optimalizační algoritmy. Zbylé algorimy implementovali moji kolegové, proto se moje práce zaměřuje na genetický algoritmus a části vývoje aplikace, které jsem vypracoval já. Nově vzniklá aplikace nahradí zastaralé Java applety a bude použita při výuce od příštího semestru.

Klíčová slova Genetický algoritmus, fitness, selekce, křížení, mutace, elitismus, vizualizace, Euklidovský obchodní cestující, problém, generátor.

Abstract

This diploma thesis is about development of the educational application, which should help students to understand how a genetic algorithm works. A genetic algorithm is an optimization algorithm, which is inspired by biological evolution. Adjustable parameters and the visualisation of algorithm run as chart are the main instruments, which help students to deepen the knowledge in this field. Users can try to work on various problems. Problem instances could be created manually or automatically generated.

The theoretical part analyses genetic algorithm strategies, which are later implemented. The practical part is about the application development. One application was created for three different optimization algorithms on request from the supervisor. The two other algorithms were implemented by my colleagues so my thesis focuses on genetic algorithm and developments parts I've been involved with. The new application is going to be used for teaching from next semester instead of old Java applets.

Keywords Genetic algorithm, fitness, selection, crossover, mutation, elitism, visualisation, Euclidean travelling salesman, problem, generator.

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Mnoho oblastí informatiky, kde byly položeny teoretické základy již poměrně dávno, se díky rostoucímu výpočetnímu výkonu a rozmachu problémů, které chce lidstvo řešit, dostává stále více do praxe. To samé platí i pro genetický algoritmus, který spadá do oblasti umělé inteligence. Genetický algoritmus, což je optimalizační algoritmus, dokáže výpočetní výkon řádně využít, a to včetně paralelních výpočtů. Tento algoritmus se v současné době používá například k šlechtění neuronových sítí.

Aby algoritmus fungoval co nejlépe, je nutné nejprve vhodně nastavit jeho parametry pro daný problém, což může být obtížné. Nastavování parametrů bez toho, aniž by člověk věděl, co znamenají, je velmi neefektivní. Proto člověk, který má pracovat s GA, musí znát jeho princip a také chování algoritmu při změně parametrů. Jakmile nastavíme parametry správně, algoritmus už pracuje sám a řeší podobné instance stejného problému.

Na FIT ČVUT se o tomto algoritmu učí v rámci předmětu MI-PAA. Princip algoritmu není težké pochopit, ale jak jsem se sám přesvědčil, někteří studenti nechápou vliv parametrů na algoritmus. Je sice hezké, když se na přednášce vysvětlí, jak parametry ovlivňují průběh, ale nejlepší je, když si studenti vyzkouší práci s parametry sami. K tomu donedávna sloužil zastaralý Java applet, který se však rok od roku podaří spustit méně studentům. Aby tedy studenti měli možnost si tuto práci vyzkoušet, vznikla potřeba vytvořit novou moderní verzi této aplikace, která bude provozuschopná několik dalších let.

Mým úkolem tedy je vytvořit aplikaci, která by studentům pomohla pochopit, jak funguje genetický algoritmus, a především jeho parametry. Tato aplikace se bude používat při výuce MI-PAA, ale mohla by být taktéž dostupná pro širokou veřejnost. V průběhu analýzy padlo rozhodnutí sdružit tuto aplikaci společně s nově vznikajícími aplikacemi pro další optimalizační algoritmy. Konkrétně tedy simulované ochlazování [1] a Tabu prohledávání [2].

Cíle práce

Cílem této práce je vytvořit aplikaci, která by studentům pomohla pochopit, jak funguje genetický algoritmus. Aplikace by měla studentům pomoci nejen s chápáním algoritmu samotného, ale především by jim měla ukázat, jak jednotlivé parametry ovlivňují průběh genetického algoritmu. Předpokládá se, že uživatel o algoritmu již slyšel (na přednáškách) a chápe základní principy a orientuje se v terminologii. Aplikace si tedy neklade za cíl naučit uživatele genetický algoritmus od základů.

Tato aplikace bude použita na hodinách předmětu MI-PAA, kde se o tomto algoritmu učí. Aplikace by měla nahradit původní aplikaci, která je technologicky zastaralá a je velmi obtížné ji spustit. Nová aplikace by také měla být lepší než původní, a to především co se týče uživatelského rozhraní.

Za tímto účelem bude proveden uživatelský průzkum mezi cvičícími a studenty předmětu MI-PAA a já se budu snažit vyhovět jejich přáním. Dalším úkolem je zvolit vhodnou SW architekturu pro implementaci aplikace. Důležité je, aby aplikace byla univerzální a rozšiřitelná. Na závěr bude výsledná aplikace podrobena uživatelským testům.

Kapitola 1

Genetický algoritmus

Genetický algoritmus je optimalizační algoritmus využívající heuristickou funkci. Tyto algoritmy se využívají, pokud chceme získat dostatečně dobré řešení nějakého problému, pro jehož vyřešení neexistuje exaktní algoritmus, který by nám dal nejlepší možné řešení v rozumném čase. Jedná se typicky o prohledávání stavového prostoru, který je velmi rozsáhlý, nebo dokonce neomezený.

Jak název napovídá, GA je společně s dalšími evolučními algoritmy inspirován evoluční teorií. S touto teorií bývá nejčastěji spojován slavný přírodovědec Charles Darwin, který ji popsal ve své knize [3], a je též považován za otce evoluční biologie. GA využívá principy evoluční biologie a přirozeného výběru:

- Organismus zplodí v průměru za život více než jednoho plodného potomka.
- Existuje určitý selekční tlak, čili organismy s určitými vlastnostmi mají větší šanci na přežití než jiné.
- Existuje vnitrodruhová variabilita, jednotliví příslušníci téhož druhu se od sebe liší.
- Existuje dědičnost, potomci se v průměru podobají více svým rodičům než ostatním členům populace.

Evoluce živočišných druhů ale trvala miliony let, a na to nemáme čas. Proto je GA velmi zrychlenou simulací evoluce. Níže rozebírám techniky používané pro genetický algoritmus. Tento výčet není zdaleka kompletní, snažím se pouze stručně popsat mechanismy, které budu později implementovat. Genetický algoritmus není nijak dogmaticky normován. Tím pádem si každý může vymyslet vlastní operátory a pokročilejší techniky nebo může implementaci těch typických trochu pozměnit. Já se snažím zachytit ty nejběžnější, které se považují za funkční a jsou v souladu s českými [4] a anglickými [5] přednáškami MI-PAA. Jako další podstatné zdroje této kapitoly používám [6], [7], [8], [9].

1.1 Jedinec

Jedinec, neboli *fenotyp*, je základním prvkem, se kterým GA pracuje. Jedinec představuje nějakou konkrétní konfiguraci (řešení) daného problému. Uvnitř programu je však tato konfigurace nějakým způsobem zakódována a tomu se říká *genotyp*. Tyto pojmy se často zaměňují. Genotyp se skládá z jednotlivých *genů* (proměnných) a hodnota konkrétní proměnné se nazývá *alela*.

1.1.1 Binární jedinec

Jedná se o jedince, jehož genotyp je zakódován jako binární pole. Toto kódování je pro GA typické, protože se velká řada problémů dá zakódovat jako binární řetězec a snadno se s ním pracuje v rámci genetických operátorů, které rozebírám níže. Význam jednotlivých bitů sděluje až instance problému.

1.1.2 Permutační jedinec

Dalším kódováním, kterým se budu zabývat, jsou permutace. Toto kódování je už poměrně specifické, protože je určeno pro permutační problémy. Nejznámějším permutačním problémem je problém obchodního cestujícího, který má být mým algoritmem řešitelný. Genetické operátory pro permutace jsou o něco složitější než ty binární, protože chceme, aby výsledkem byla opět permutace.

1.2 Hodnotící funkce

Klíčovou roli, zda bude GA fungovat či ne, hraje hodnotící funkce. V případě GA mluvíme o tzv. *fitness/zdatnosti*. Tato heuristická funkce se snaží odhadnout, jak zdatný konkrétní jedinec je, tedy jak kvalitní je konkrétní řešení reprezentované jedincem. Tato funkce nám tedy propojuje zadaný problém a GA, který může být jinak zcela problémově nezávislý.

Fitness funkce se váže přímo na konkrétní problém a je důležité, aby přiřadila lepší hodnotu jedinci, který je (zdá se být) lepší. Především kvůli technikám výběru 1.5 je dobré, aby fitness funkce rozlišovala také o kolik se kvalita řešení liší. Hlavním problémem fitness funkce je, že chceme co nejblíže popsat kvalitu řešení, a zároveň je nutné, aby výpočet funkce byl rychlý, neboť se provádí pro každého nově vzniklého jedince.

Pro některé problémy je poměrně zřejmé co použít pro určení fitness, u jiných problémů je to ale velmi komplikovaná záležitost a většinou musíme přistoupit k nějakému kompromisu. Většinou rozlišujeme maximalizační a minimalizační problémy a podle toho se GA snaží hodnotu fitness funkce buď maximalizovat, nebo minimalizovat. V kontextu této práce se zaměřím na maximalizaci fitness, pokud nebude řečeno jinak, neboť ostatní mechanismy GA se na maximalizaci lépe vysvětlují.

1.2.1 Relaxace

Později se dočtete, že aplikací genetických operátorů může vzniknout jedinec, jehož konfigurace není přípustná vzhledem k problému. Většinou se jedná o nějaké dodatečné omezující podmínky, jak může řešení problému vypadat. Takovýto jedinec se nazývá „nevalidní“. Bohužel u některých problémů nemůžeme zabránit vznikání takovýchto jedinců a musíme s nimi tedy nějak pracovat.

První dvě techniky se zabývají eliminací nevalidních jedinců z populace. První, nazývaná „trest smrti“, jednoduše takové jedince zahodí. Problémem je, že nějak musí vytvořit ty validní, aby se algoritmus někam posunul. Druhá technika se snaží nevalidní jedince opravit tak, aby se stali validními. Potíž tohoto způsobu spočívá v tom, že tato oprava může být velmi náročná a někdy by znamenala vyřešení problému samotného. Takto náročné opravy si však GA nemůže dovolit, protože ho používáme pro problémy, které jsou velmi náročné samy o sobě.

Třetí technikou a zároveň důvodem, proč je tato část umístěna právě sem, je relaxace. Použití relaxace umožňuje nevalidním jedincům žít v populaci v průběhu výpočtu GA. Avšak jejich fitness je značně penalizována, aby algoritmus dával přednost validním jedincům. Na nevalidní jedince se můžeme dívat, jako na postižené nějakou vadou, mohou se v populaci množit, ale pokud se v populaci objeví nějaký validní jedinec, měl by je značně převálcovat a zajistit tak, aby výsledkem bylo validní řešení.

Relaxace je tedy technika přepočítání fitness tak, aby algoritmus umožňoval výskyt nevalidních jedinců v populaci, ale zároveň nedegradoval algoritmus tak, aby se spokojil s nevalidním řešením, které je ve výsledku k ničemu. Vymyslet, jak udělat relaxaci, je mnohdy složitější úkol než vymyslet, jak počítat fitness. Někdy se používá pouhý koeficient, kterým se fitness nevalidního jedince přenásobí, aby došlo k jejímu zmenšení. Jindy se posune fitness nevalidních jedinců o nějakou konstantu. To však ne vždy dovolí zvítězit validnímu jedinci a nějaký nevalidní ho může porazit.

Já osobně mám nejradši způsob, kdy je relaxace udělána tak, že každý nevalidní jedinec má horší fitness než jakýkoli validní. Zjistit nejhorší fitness validního jedince bývá často možné. Většinou postupuji tak, že u validních jedinců počítám skutečnou fitness a u nevalidních počítám, jak daleko mají k tomu stát se validním řešením i za cenu toho, že aspekty skutečné fitness upozadím.

1.3 Populace a generace

Další důležitou vlastností GA je, že pracuje s více řešeními naráz. To otevírá možnost jistému paralelismu. Řešení problému se hledá iterativně v cyklu 1.4. Množině jedinců, se kterou GA pracuje, se říká *populace*. *Generace* je potom

populace v konkrétním čase (iteraci). Když pracujeme s GA můžeme nastavit velikost populace a počet generací, které mají být vytvořeny.

1.4 Cyklus GA

Genetický algoritmus začíná vytvořením počáteční populace. Jedná se většinou o náhodně vygenerované jedince. Počáteční generaci je třeba ohodnotit vypočítáním fitness (případně seřadit). Tvorba nové generace probíhá v několika fázích. První z nich je *selekce* 1.5, kde jsou z aktuální generace vybráni vhodní kandidáti na *rodiče*. Rodiče pak vstupují do *křížení* 1.6, kde jsou z rodičů vytvořeni *potomci*. Potomci následně projdou *mutací* 1.7. Tito potomci jsou opět opatřeni fitness a vytvoří se z nich nová generace 1.8.

Tento cyklus se opakuje do té doby, než jsme s výsledkem dostatečně spokojeni, nebo po vyprodukování zadaného počtu generací. Jako výsledek GA se někdy uvádí jedinec s nejlepší fitness napříč všemi generacemi, někdy pouze jedinec s nejlepší fitness v poslední generaci. Tyto výsledky by však měly být v ideálním případě stejné, protože chceme, aby nám populace zkonvergovala k nejlepšímu řešení.

1.5 Selekce

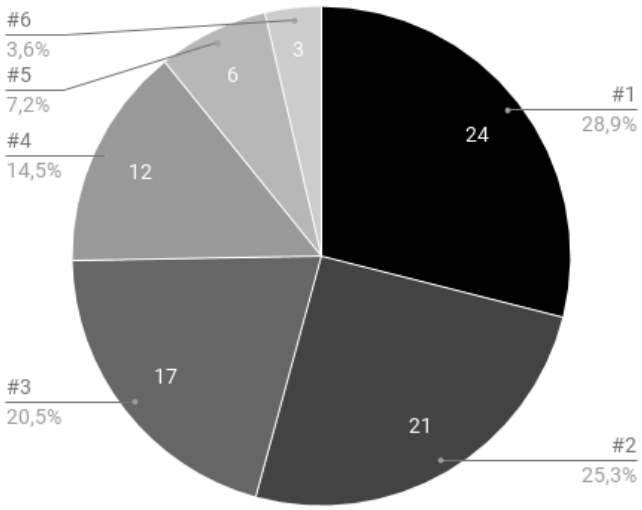
V kontextu evoluční biologie se jedná o onen „přírodní výběr“ a přežití zdatnějšího jedince. Nebavíme se zde o přežití jedince samotného, ale spíše o přežití jeho genetické informace a rozšíření do další generace pomocí křížení 1.6.

Podobně jako jiné iterativní algoritmy potřebuje GA způsob, jak zajistit intensifikaci, tedy posílení kvalitní informace v řešení. Toto zajišťuje právě selekce, která má za úkol zvýšit podíl kvalitních řešení. O tom, jak moc má výběrový mechanismus upřednostňovat kvalitní řešení před nekvalitními, rozhoduje *selekční tlak*. Pokud je selekční tlak nízký, příliš se nerozlišuje mezi kvalitními a nekvalitními jedinci a průběh algoritmu připomíná náhodnou procházku. Naopak pokud je selekční tlak příliš vysoký, algoritmus upřednostňuje nejkvalitnější řešení, které za chvíli zamoří celou populaci a dojde tak k předčasné konvergenci a uváznutí v lokálním optimu. Tomuto jevu se v souvislosti s GA říká *degenerace* populace.

Na konci fáze selekce máme k dispozici rodiče, jejichž počet nejčastěji odpovídá počtu potomků, které chceme vytvořit.

1.5.1 Turnajový výběr

Ač název svádí představit si spíše rytířské klání, tento způsob selekce je inspirován souboji mezi některými druhy zvířat o účast v páření. Ve své podstatě mají tyto souboje s rytířskými turnaji mnoho společného.



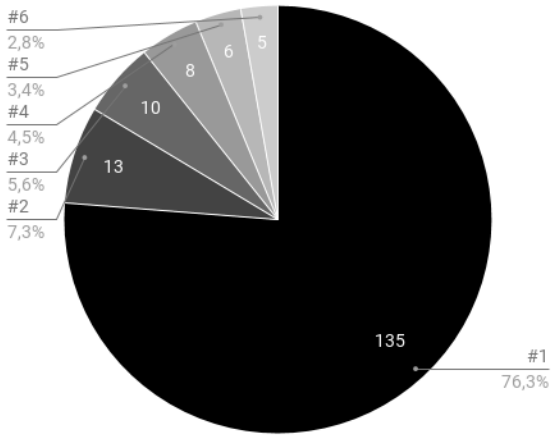
Obrázek 1.1: Ukázka rozdělení podílů na ruletě

Každý jedinec má stejnou šanci účastnit se turnaje bez ohledu na jeho fitness a může se turnajů účastnit opakovaně, dokonce může soupeřit sám se sebou. Tento fakt je velmi důležitý a bývá studenty často nesprávně pochopen. Jedinec v turnaji soupeří s dalšími jedinci a ten s nejlepší fitness vyhrává. Tímto způsobem se vybere jeden rodič, proto se v této fázi obvykle odehraje více turnajů. Počet zápasníků v turnaji určuje velikost turnaje. Velikost turnaje je zároveň parametr, který řídí selekční tlak. Čím větší turnaj, tím více soupeřů musí vítěz porazit, tím větší je tedy selekční tlak.

1.5.2 Ruletový výběr

Ruleta pochází z prostředí kasina. Nicméně tato představa vychází spíše z koláčového grafu, kde mají jedinci svůj podíl a pravděpodobnost výběru jedince odpovídá zastavení kuličky rulety na pozici s jeho podílem. Pokud tedy chceme zůstat u evoluční biologie, můžeme si představit, že ruletový výběr představuje, s jakou pravděpodobností jedinec zaujme partnera. Zdatnější jedinci mají pochopitelně tuto pravděpodobnost vyšší.

Pravděpodobnost výběru jedince je úměrná jeho fitness. Rozdělení těchto pravděpodobností se provádí pomocí přidělování podílů v ruletě. Každý jedinec dostane svůj podíl. Z podílů se vytvoří koláčový graf, který představuje ruletové kolo. Poté, co zatočím ruletou, vyberu toho jedince, kde skončila kulička jako rodiče. Pravděpodobnost výběru jedince přesně odpovídá jeho podílu. Točení ruletou odpovídá vygenerování náhodného úhlu od 0°do 360°. Kam tento úhel ukáže, ten jedinec se vybere. Například podle rulety na obrázku 1.1.



Obrázek 1.2: Ukázka dominance jedince na ruletě

Problémem ruletového výběru je, jak nastavit podíly v ruletě. Pokud nastavíme podíly jako fitness jedinců, nemusí to vždy dobře dopadnout. Na fitness funkci 1.2 se nemůžeme příliš spolehnout, obzvláště v případě, kdy chceme řešit GA obecně a nemáme žádnou záruku, jak bude funkce vypadat. Rozdíly mezi fitness zde totiž hrají klíčovou roli. Může dojít ke dvěma krajním případům.

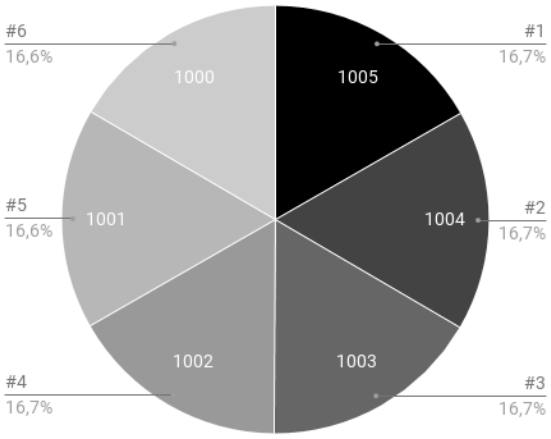
První z nich je, když ruletě dominuje jeden jedinec a ostatní jedinci nemají téměř šanci být vybráni. V takovém případě dojde nejspíše k předčasné konvergenci. Je to stejné, jako kdybychom nastavili vysoký selekční tlak. Je dobré si uvědomit, že takovýto případ není v průběhu GA nic výjimečného. Stačí si představit fitness funkci využívající relaxaci a náhlé nalezení validního řešení. Fitness validního řešení vystřelí nahoru a ostatní nevalidní řešení nechá daleko za sebou. Příklad ilustruje obrázek 1.2. V extrémním případě může dominantní jedinec zabírat celou plochu rulety a mít jistotu výběru.

Druhý případ naopak nastane, pokud jsou rozdíly ve fitness minimální. Ruleta pak považuje jedince za rovnocenné a vybírá je náhodně, téměř podle rovnoměrného rozdělení. Tento případ vlastně odpovídá nízkému selekčnímu tlaku a algoritmus nejspíše nebude schopný zkonvergovat k jednomu řešení. V průběhu GA tato situace nastává poměrně často, hlavně ke konci výpočtu. Obrázek 1.3 ilustruje tuto situaci.

Abychom se těmto situacím vyhnuli a dostali řízení selekčního tlaku do vlastních rukou, používají se následující techniky.

1.5.2.1 Lineární škálování

Lineární škálování je způsob jak přeškálovat fitness na podíly tak, aby byl definován minimální (f_1) a maximální (f_2) podíl a zároveň byly zachovány relativní rozdíly. K přeškálování se používá následující funkce:



Obrázek 1.3: Ukázka vyvážené rulety

$$f' = f_1 + (f - f_{MIN})\frac{f_2 - f_1}{f_{MAX} - f_{MIN}}$$

f původní hodnota fitness

f_{MIN} minimum z původních fitness

f_{MAX} maximum z původních fitness

f_1 nové minimum

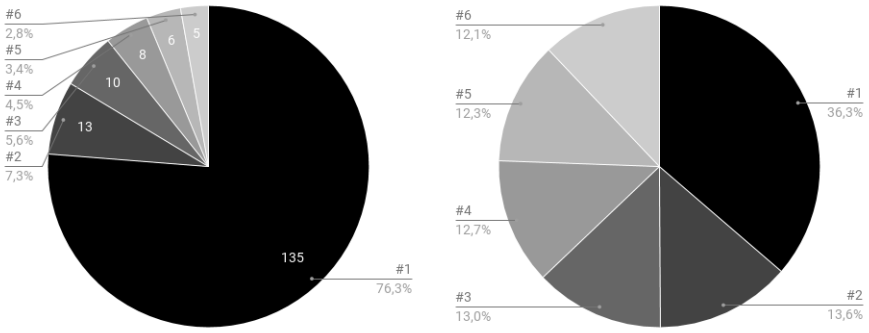
f_2 nové maximum

f' hodnota po přeškálování

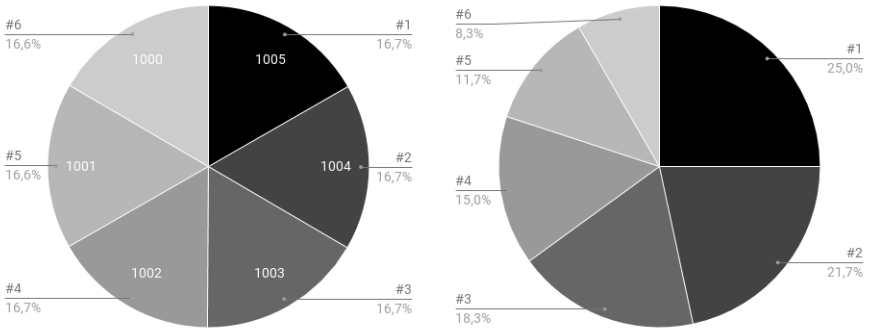
Funkce v podstatě namapuje původní fitness do požadovaného intervalu $\langle f_1; f_2 \rangle$. Na konkrétních hodnotách této funkce příliš nezáleží. Stále totiž řešíme poměry. Pokud například chceme, aby nejhorší jedinec byl vybrán s třikrát menší pravděpodobností než ten nejlepší, nastavíme $f_1 = 1$ a $f_2 = 3$. Na obrázcích 1.4 a 1.5 vidíme, jak si lineární škálování s těmito parametry poradí se zmiňovanými krajními případy. Selekční tlak tedy řídíme nastavováním minimálního (f_1) a maximálního (f_2) podílu.

1.5.2.2 Ranking

Další technikou, jak si poradit s přidělením podílů na ruletě, je tzv. ranking. Ranking v podstatě degraduje hodnotu fitness funkce na klasické porovnávání lepší/horší, podobně jako tomu bylo u turnaje. Ranking typicky funguje tak, že nejhorší jedinec dostane podíl 1 a nejlepší jedinec podíl rovný počtu jedinců v populaci. Každý jedinec tedy dostane podíl odpovídající jeho pořadí.



Obrázek 1.4: Ukázka dominantní rulety před (vlevo) a po přeškálování (vpravo)



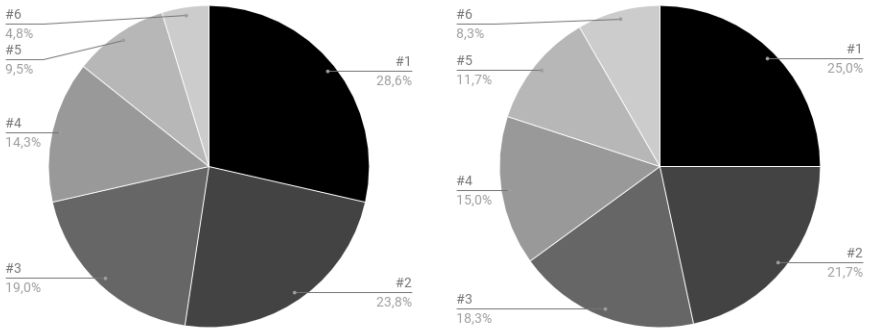
Obrázek 1.5: Ukázka vyvážené rulety před (vlevo) a po přeškálování (vpravo)

Tento základní ranking však nenabízí žádnou kontrolu selekčního tlaku, ač máme zaručený rozsah hodnot. Z tohoto důvodu se výsledný rank často škáluje pomocí lineárního škálování 1.5.2.1, kde volíme jako vstup rank místo fitness. Na obrázku 1.6 opět demonstruji, jak to dopadne pro případy 1.2 a 1.3. Zde je důležité si uvědomit, že na hodnotách fitness nezáleží, ale záleží pouze na pořadí, proto bude ruleta vypadat vždy stejně. Pro lineární škálování byly opět použity $f_1 = 1$ a $f_2 = 3$.

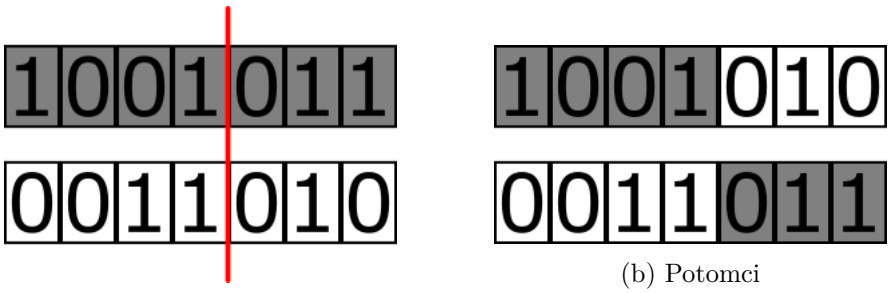
Zdroje už se ale liší v tom, jak si poradit s přidělováním ranku jedincům se stejnou fitness. Některé tvrdí, že každý jedinec má jiný rank, jiné zase, že tito jedinci mají stejný rank. Většina zdrojů se však touto problematikou vůbec nezabývá, protože nelze jednoznačně říci, že jedna varianta je lepší než ta druhá. Tímto problémem se více zabývám v části 7.3.2.1.

1.6 Křížení

Křížení stojí na myšlence, že pokud má jedinec nějakou dobrou vlastnost, tak zkřížením s jiným jedincem se některé vlastnosti přenesou do potomků a



Obrázek 1.6: Ukázka rank rulety bez (vlevo) a s lineárním škálováním (vpravo)



(a) Rodiče

(b) Potomci

Obrázek 1.7: Ukázka jednobodového křížení

doufáme, že to budou ty dobré. Může se také stát, že zkřížením vlastností, které se zdají (podle fitness) jako špatné, vznikne vlastnost, která je naopak dobrá. Křížení z pohledu GA je o dost jednodušší než v přírodě, neřeší se pohlaví ani příbuznost jedinců. Každý se může křížit s každým. Jedinec se tedy může zkřížit i sám se sebou, jeho potomci však budou pouhé kopie onoho jedince a novou informaci může vnést pouze mutace 1.7.

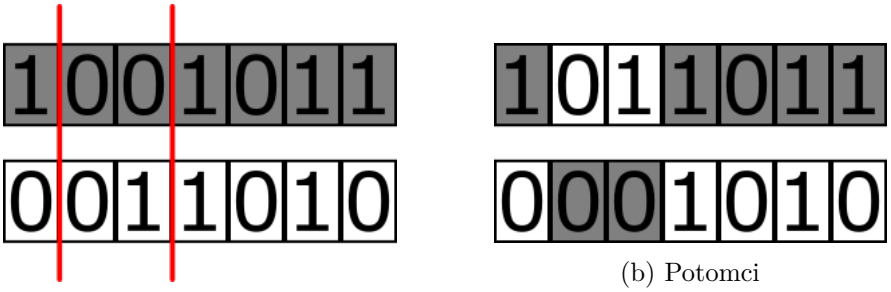
Křížení je důležitým prvkem GA a hlavním operátorem, který tvoří nové jedince a dochází tak k prohledávání prostoru. Existuje mnoho druhů křížení. Níže se budu zabývat křížením pro různá kódování, kde jsou vstupem dva rodiče a výstupem dva potomci.

1.6.1 Pro binární kódování

1.6.1.1 Jednobodové křížení

Nejjednodušší křížení. Náhodně se zvolí bod, kde se binární pole rozdělí. První půlka rodiče se přenese do jednoho potomka a druhá do druhého. Zbytek se doplní z druhého rodiče. Situaci ilustruje obrázek 1.7.

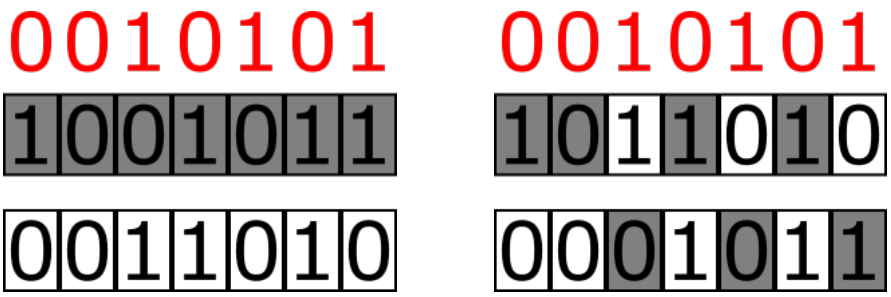
Hlavním problémem tohoto křížení je, že bity, které se nachází na začátku a na konci, s velkou pravděpodobností neskončí ve stejném jedinci. To je v po-



(a) Rodiče

(b) Potomci

Obrázek 1.8: Ukázka dvoubodového křížení



(a) Rodiče s náhodným vektorem

(b) Potomci s náhodným vektorem

Obrázek 1.9: Ukázka uniformního křížení

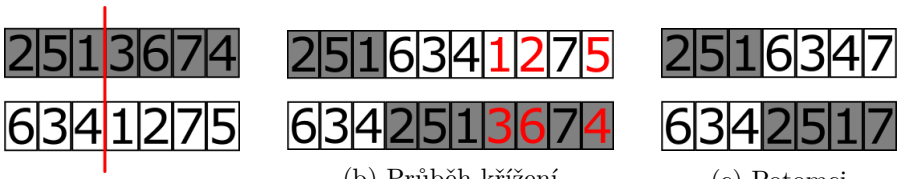
řádku, pokud spolu tyto bity nijak nesouvisí, netvoří tzv. „schéma“, což však nemůžeme vždy zaručit.

1.6.1.2 Dvoubodové křížení

Řeší problém s jednobodovým křížením. Náhodně se určí začátek a konec úseku, který si mají rodiče vyměnit. Existují i vícebodová křížení, postup je analogický, jako vidíme na obrázku 1.8.

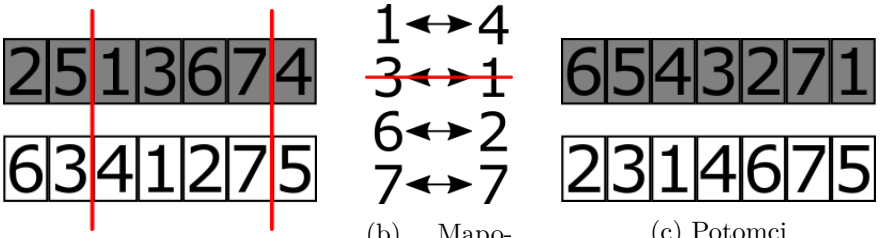
1.6.1.3 Uniformní křížení

Předchozí křížení se snažila zachovat úseky bitů, protože předpokládala, že čím bližší si bity jsou, tím více spolu souvisí. Což nemusí být vůbec pravda. Uniformní křížení toto nepředpokládá a jistým způsobem se na něj dá pohlížet také jako na vícebodové křížení. Uniformní křížení spočívá v tom, že každý bit se s 50% pravděpodobností vymění mezi rodiči a jinak zůstane na místě. Jinými slovy, vygenerujeme náhodný binární vektor stejné délky jako jedinci, a pokud pozice obsahuje 1, vyměníme bity mezi rodiči jako na obrázku 1.9.



(a) Rodiče (b) Průběh křížení (c) Potomci

Obrázek 1.10: Ukázka order crossover



(a) Rodiče (b) Mapovací tabulka (c) Potomci

Obrázek 1.11: Ukázka PMX

1.6.2 Pro permutační kódování

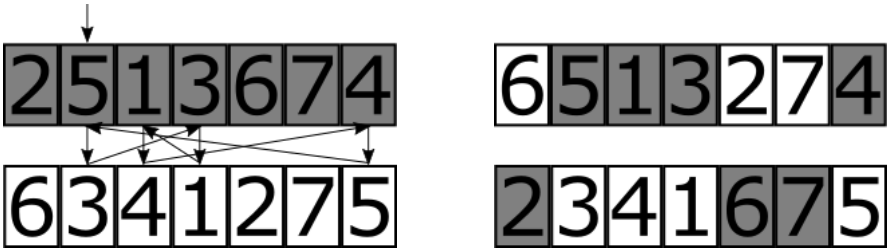
Křížení pro permutační kódování je o něco náročnější, protože operace musí zachovat permutaci. Vybraná křížení jsou však velmi podobná křížením binárním.

1.6.2.1 Order crossover

Podobá se jednobodovému křížení. Náhodně se vybere bod, geny do tohoto bodu se zkopírují z prvního rodiče a poté začneme postupně doplňovat geny z druhého rodiče tak, abychom zachovali permutaci. Vynecháme tedy hodnoty, které jsou v jedinci již obsaženy. Na obrázku 1.10 jsou tyto hodnoty zvýrazněny červenou barvou.

1.6.2.2 Patrially matched crossover

Podobá se dvoubodovému křížení. Náhodně se vybere úsek, pro který se vytvoří mapování, tedy tabulka dvojic, které se mezi sebou mají prohodit. Díváme se vždy na odpovídající geny rodičů, jejichž hodnoty chceme prohodit. Mapování se provádí zleva doprava. Pokud jedna z hodnot již je v tabulce obsažena, pak tyto geny přeskočíme. Na závěr přeházíme hodnoty v rodičích podle mapovací tabulky, jak můžeme vidět na obrázku 1.11.



(a) Rodiče s cyklem (b) Potomci

Obrázek 1.12: Ukázka cycle crossover

1.6.2.3 Cycle crossover

Toto křížení začneme vytvářením cyklu na náhodné pozici v prvním rodiči. Pokračujeme tak, že si zapamatujeme hodnotu na stejné pozici v druhém rodiči. Tuto hodnotu pak hledáme v prvním rodiči, až ji nalezneme, přesuneme se na pozici s touto hodnotou. Opakováním tohoto postupu musí nutně vzniknout cyklus. Cyklus dokončíme, jakmile se vrátíme na startovní pozici. Poté, co máme cyklus zbudovaný, přejdeme k vlastnímu křížení. Pozice, které jsou obsaženy v cyklu, necháme tak, jak jsou, a u ostatních prohodíme hodnoty mezi rodiči. Křížení je vyobrazeno na obrázku 1.12.

1.7 Mutace

Jak jste si asi všimli, křížení pracuje výhradně s informacemi, které obsahuje počáteční generace a ty mezi sebou nějak kombinuje. Může se ale například stát, že všichni jedinci v populaci mají stejný gen nastaven na 1. Pro optimální řešení by se ale hodilo nastavit gen na 0. Křížení toho nemůže nikdy dosáhnout, a proto přichází do hry mutace. K mutaci dochází v přírodě zřídka, jedná se o změnu genotypu. V podstatě to znamená, že potomek dvou rodičů nemusí obsahovat pouze jejich informace, ale do genotypu je přidán nějaký náhodný šum.

Stejně jako ve skutečné evoluci se musí i s mutací u GA velmi opatrně. Mutace sice umožňuje prohledat dosud neprobádané kouty stavového prostoru a zabránit tak předčasné konvergenci. Mutace však má i svou stinnou stránku a tou je, že odvádí algoritmus od hledání dobrého řešení tím, že zahazuje informace a doplňuje je náhodným šumem. V krajním případě může dojít až k divergenci populace, kdy prohledávání prostoru připomíná spíše náhodnou procházku. Mutace je mechanismem pro exploraci a je protipólem selekčního tlaku.

1.7.1 Pro binární kódování

Mutace u binárního kódování se řídí pravděpodobností mutace. Celé bitové pole se postupně projde a každý bit se s touto pravděpodobností invertuje.

1.7.2 Pro permutační kódování

Největším problémem mutace permutace je, že nemůžeme ovlivnit jedinou hodnotu, vždy musíme prohodit alespoň dvě. Existuje celá řada permutačních mutací, které mají své výhody a nevýhody. Já uvádím mutaci, která se nejvíce podobá té pro binární kódování. Jedná se o prohození hodnoty s jinou na jiné pozici. Pravděpodobnost mutace se zde většinou udává jako pravděpodobnost, že se hodnota na dané pozici prohodí s jinou. Toto se provede pro každou pozici v genotypu.

1.8 Náhrada populace

Existuje více způsobů, jak nahradit starou populaci. Typickým způsobem pro GA je náhrada celé populace nově vzniklými jedinci (potomky této generace). Někdy se potomci a rodiče sloučí a z nich se vybere nová generace. Buď jako k nejlepších jedinců, nebo pomocí nějakého selekčního mechanismu 1.5.

1.8.1 Elitismus

Elitismus je pokročilejší technika náhrady populace, kdy se do nové populace přenese určitý počet nejlepších jedinců ze staré populace a zbytek se doplní z potomků. Tato technika zajistí, že neztratíme dosavadní nejlepší řešení a stále ho udržujeme v populaci.

Kapitola 2

Sběr požadavků

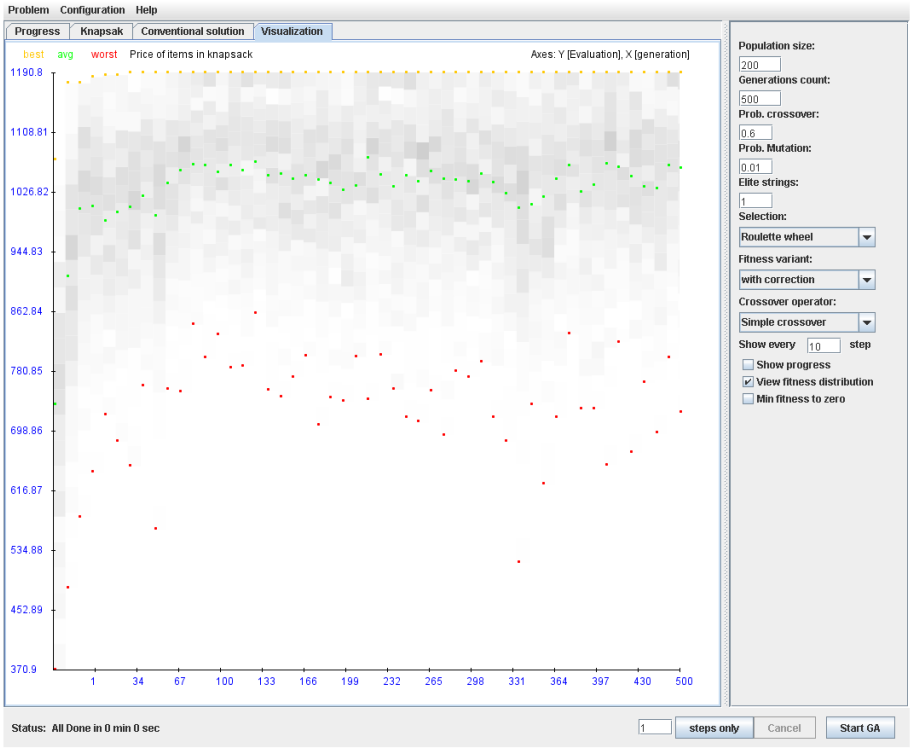
Tato kapitola se zabývá analýzou požadavků pro moji aplikaci. Důležité je zmapovat předešlou aplikaci 2.1 a zjistit, co by se dalo udělat lépe. Klíčové jsou také názory a přání koncových uživatelů, tedy studentů 2.3 a pedagogů 2.2 MI-PAA.

2.1 Analýza původní aplikace

Původní aplikace, která je součástí bakalářské práce [10], je dostupná jako webová aplikace [11]. Největší nevýhodou původní aplikace pro demonstraci evolučních algoritmů je použitá technologie. Aplikace je napsána jako Java applet. Tato technologie je dnes již zastaralá a nese s sebou jistá bezpečnostní rizika. Z těchto důvodů Java applety již nejsou podporovány mnoha prohlížeči a jdou spustit pouze na starších verzích prohlížečů přidáním bezpečnostních výjimek. Toto je docela nepříjemný proces, kterému jsou vystavováni studenti na cvičeních MI-PAA. Z tohoto důvodu je třeba, aby vznikla aplikace nová, která nahradí původní.

Aplikace v podstatě implementuje základní obecný genetický algoritmus tak, jak je popsán na přednáškách předmětu MI-PAA, který umí řešit různé kombinatorické problémy. Student si tedy vybere, jaký problém chce řešit, nahraje instanci problému ze souboru nebo si nechá nějakou náhodně vygenerovat. Instanci poté může upravit. Jakmile je zadán vstupní problém, může se přesunout k hlavnímu úkolu.

Podstatou této aplikace je úvodní okno, ve kterém se nastavují parametry pro genetický algoritmus. Po spuštění můžeme sledovat vývoj nalezeného řešení buď jako vizualizaci nejlepšího jedince, nebo jako vývoj nejlepší, průměrné a nejhorší fitness jedince napříč generacemi (obr. 2.1).



Obrázek 2.1: Ukázka původní aplikace

2.1.1 Nastavitelné parametry

Právě nastavitelné parametry jsou jádrem této aplikace. Díky nim by studenti měli pochopit, jak se chová algoritmus, když parametry změní. Samozřejmě, že pokud student o GA nic neví, tak náhodným měněním parametrů se toho moc nového nedozví. Mezi nastavitelné parametry patří:

Population size nastavuje velikost populace v každé generaci.

Generations count nastavuje počet generací, po kterých algoritmus skončí.

Prob. crossover nastavuje pravděpodobnost křížení.

Prob. mutation nastavuje pravděpodobnost mutace.

Selection umožnuje zvolit mezi ruletovým nebo turnajovým výběrem, kde se navíc udává i velikost.

Fitness variant nastavuje, zda se použije relaxace, nebo budou nevalidní řešení opravována. Druhou možnost však nelze použít pro všechny problémy.

Crossover operator nabízí výběr mezi jednobodovým, dvoubodovým nebo uniformním křížením.

2.1.2 Vizualizace

Aplikace poskytuje dva druhy vizualizací.

Jednou z nich je vizualizace nejlepšího řešení zadané instance problému. Tato vizualizace musí být ušita na míru každému problému zvlášť. Taková vizualizace sice poskytuje studentovi představu o tom, jak vypadá konkrétní řešení, ale pro pochopení práce s parametry má, dle mého názoru, malý, troufl bych si říci až nulový přínos.

Druhá vizualizace je v podstatě graf, kde osa X představuje jednotlivé generace a osa Y reprezentuje hodnoty fitness. Do tohoto grafu se v každé generaci vynese nejhorší, průměrný a nejlepší jedinec dané generace. Navíc je zde ještě reprezentována celá populace pomocí různých odstínů šedi. Tmavší barva představuje větší hustotu jedinců. Tato vizualizace je pro pochopení práce s parametry daleko lepší. Student může jednoduše, za případné asistence cvičícího, pomocí těchto tří pomyslných křivek (Hodnoty jsou diskrétní a mohou na sobě být zcela nezávislé.) pozorovat, zda má parametry nastavené „správně“. Snaží se o to, aby nejlepší fitness byla co nejvyšší, a také aby se rozdíly mezi fitness v následujících generacích postupně snižovaly a docházelo tedy ke konvergenci.

2.1.3 Hlavní nevýhody aplikace

- Java applet.
- Aplikace nemá responzivní design.
- Pokud chce uživatel pracovat s jinou instancí, musí překlikávat mezi záložkami. Není příliš intuitivní, jak spolu jednotlivé záložky souvisí.
- Uživatel nemůže řídit selekční tlak u ruletového výběru.
- Aplikace nemá žádnou nápovědu přímo u parametrů.
- Nápověda v záhlaví nefunkční.
- Vizualizace průběhu pouze pomocí pixelů je špatně viditelná a čitelná.

2.2 Představa cvičících

V této kapitole předkládám, jak by měla aplikace vypadat dle pedagogů podílejících se na výuce předmětu MI-PAA. Opírám se především o názory mého vedoucího docenta Petra Fišera a přednášejícího docenta Jana Schmidta.

Podle nich by studenti v době použití aplikace již měli znát základní princip GA a také, jak fungují jednotlivé jeho kroky. Tudíž není nutné vizualizovat jednotlivé kroky. Naopak vizualizace průběhu je klíčová. Původní applet 2.1 vyhovuje svou funkcionalitou, nikoli však přehledným a snadno použitelným uživatelským rozhraním. Zastaralá technologie je také velký problém. Cvičící by rádi zachovali možnost práce s více problémy a vytváření instancí ručně či pomocí generátoru. Uživatel by měl mít také možnost pracovat s více instancemi najednou a vracet se k nim. Studenti by měli mít aplikaci k dispozici na školních počítačích v učebnách. Instalace aplikace nebo potřebného prostředí do učeben je poměrně náročný úkon, proto vzniká požadavek na spuštění aplikace bez nutnosti jakékoli instalace něčeho, co na školních počítačích není standardně.

Docent Schmidt také navrhl možnost práce se seedem pseudonáhodného generátoru. Což by znamenalo, že při spuštění algoritmu se stejným nastavením a seedem by výsledek byl vždy stejný. Na druhou stranu spouštění se stejným nastavením a různým seedem by vracelo různé výsledky. Nicméně tuto možnost dotazovaní studenti (2.3) nepovažovali za příliš přínosnou.

2.3 Názory studentů

Mým dalším úkolem bylo provést uživatelský průzkum mezi studenty. Abych zjistil názory co nejvíce studentů, kteří studovali MI-PAA a měli by mít zkušenosti s aplikací, kterou popisuji v sekci 2.1, vytvořil jsem dotazník a oslovil s ním bývalé a později, po absolvování příslušného cvičení, i současné studenty MI-PAA. Cílem tohoto dotazníku bylo zjistit, jak studentům vyhovuje vizualizační applet, jak by si ho představovali, a jak vůbec chápou genetický algoritmus. Očekával jsem o něco větší ochotu zapojit se, nicméně nakonec můj dotazník vyplnilo 32 studentů. Naštěstí velká část respondentů byla sdílná v otevřených otázkách.

Studenti vidí největší přínos ve vyzkoušení a pochopení, jaký vliv mají jednotlivé parametry na průběh algoritmu. Vyřešení problému genetickým algoritmem, popřípadě názorné vysvětlení látky z přednášek, pro ně není zase tak přínosné vzhledem k pochopení iterativních algoritmů. Někteří studenti se s genetickým algoritmem seznámili již v bakalářském předmětu BI-ZUM. Pro pochopení jednotlivých kroků GA, kterým neporozuměli na přednáškách používají naučná videa na Youtube. Překvapivým zjištěním bylo, že téměř polovině respondentů se applet nepodařilo vůbec spustit a jeden o něm vůbec nevěděl. Pokud applet spustili, udělali tak především na cvičeních MI-PAA a pár jedinců si ho pustilo doma. Dvě pětiny dotázaných studentů, kteří se pokusili applet spustit, použilo školní počítač, zbytek vlastní.

Téměř 90 % studentů dává přednost webové aplikaci a jako hlavní důvod uvádí, že nemusí nic stahovat a aplikace zvládne běžet v prostředí prohlížeče, který používá každý. Minoritním důvodem je nedůvěra k cizím programům a

neochota si je stahovat/instalovat. Ti, co dávají přednost desktopové aplikaci, vidí výhodu v práci offline a dostupnosti aplikace v počítači.

Dalším zjištěním bylo, že studenti z appletu pochopili spíše nastavení parametrů než samotný princip algoritmu. Toto může být však způsobeno tím, že studenti chápou princip již z dřívějška. Také se ukázalo, že někteří studenti nechápou princip, natož pak nastavení parametrů. Pro takovéto studenty je tedy vizualizace pomocí měnící se fitness naprosto zbytečná, když ani neví, co hodnoty představují.

Jako výhody appletu studenti uvádějí:

- Vizualizace průběhu pomocí grafu
- Nastavitelné parametry
- Generátor instancí

Nevýhody naopak jsou:

- Nepřehledné UI
- Java applet
- Studenti nemohou nahlédnout více do jádra algoritmu (nevidí jednotlivé kroky)
- Chybí vysvětlivky k parametrům

Studenti by dále uvítali možnost porovnávání grafů, krokování a tutoriál či nápovědu, která by jim pomohla správně nastavit parametry.

2.3.1 Můj názor z pohledu studenta

Jelikož jsem v době navštěvování předmětu MI-PAA již znal zadání své diplomové práce, dovolím si zde přidat svůj názor z pohledu studenta MI-PAA. Bohužel jsem byl s genetickým algoritmem seznámen již dříve v několika různých předmětech, proto nedokáži posoudit, zda přednášky jsou kvalitním zdrojem pochopení této látky. Mohu se pouze opírat o další iterativní algoritmy. Již dříve jsem byl okrajově seznámen se simulovaným ochlazováním a o Tabu prohledávání jsem slyšel vůbec poprvé. Myslím si, že základní myšlenku a princip algoritmů jsem pouze na základě přednášek pochopil, ale k hlubšímu pochopení, jak konkrétně tyto algoritmy pracují, jsem musel přednáškové slidy pročíst několikrát a použít i jiné zdroje.

Když se mi poprvé dostalo do rukou zadání této práce, myslel jsem, že budu vizualizovat jednotlivé kroky GA (selekce, křížení, mutace). Nechápal jsem, proč vedoucí chce, abych vizualizoval průběh GA pomocí grafu fitness. Později mi však došlo, že vizualizovat jednotlivé kroky sice pomůže studentům pochopit jádro GA, ale s pochopením nastavení parametrů jim moc nepomůže.

Student informatiky pochopí genetický algoritmus jednou, napíše ho párkrát, ale nastavovat jeho parametry pro různé problémy bude mnohokrát. Proto je tak důležité, aby studenti pochopili, co jednotlivé parametry ovlivňují, a jak jsou vzájemně provázané. A nic lepšího, než že si to vyzkouší v praxi, není.

2.4 Sestavení požadavků

V této části přehledně uvádím požadavky na aplikaci, které byly vytvořeny na základě předchozí analýzy nebo byly uvedeny v zadání práce.

2.4.1 Funkční požadavky

2.4.1.1 Více problémů

Uživatel bude mít možnost vybrat si z palety problémů, které může GA řešit.

2.4.1.2 Generování instancí

Každý problém bude umožňovat automatické vygenerování instance nebo její vytvoření uživatelem.

2.4.1.3 Současná práce s více instancemi

Uživatel bude mít možnost pracovat s více instancemi a nějakým způsobem se k nim vracet.

2.4.1.4 Nastavování parametrů GA

Aplikace bude umožňovat nastavování parametrů genetického algoritmu, dle přání uživatele. GA přijme vstupní parametry od uživatele a bude se jimi řídit.

2.4.1.5 Implementace více typů křížení a selekce

GA bude podporovat několik typů křížení a selekce. Požaduje se implementace jednobodového, dvoubodového a uniformního křížení. Očekává se implementace nějakého křížení pro permutační problémy. GA bude umožňovat výběr turnajem, či ruletou. Na rozdíl od původní aplikace se požaduje také řízení selekčního tlaku pomocí škálování.

2.4.1.6 Graf průběhu GA

Uživatel bude mít možnost graficky sledovat průběh GA. Každý běh bude uchovávat informace o průběhu, výsledcích a nastavení parametrů.

2.4.1.7 Nápověda

Uživatel bude mít k dispozici nápovědu, která mu popíše, co který parametr dělá a jak algoritmus funguje.

2.4.2 Obecné (nefunkční) požadavky

2.4.2.1 UI v anglickém jazyce

Všechny popisky a nápověda budou v anglickém jazyce.

2.4.2.2 Přehledné a snadno použitelné GUI

Uživatelské rozhraní by mělo být přehledné a snadno použitelné v souladu s předmětem MI-NUR a mělo by splňovat Nielsenovu heuristiku.

2.4.2.3 Snadná spustitelnost bez instalace

Aplikace musí být jednoduše spustitelná na školních počítačích v učebnách FIT ČVUT bez nutnosti instalace dalšího softwaru.

2.4.2.4 Univerzálnost

GA by měl být napsán tak, aby si poradil s jakýmkoli problémem, který pro něj bude vytvořen, a nebude tedy spoléhat na určité vlastnosti konkrétních problémů.

2.4.2.5 Rozšiřitelnost

Aplikace by měla umožňovat snadno přidat další problémy k řešení a přidávat další možnosti selekce a křížení.

Kapitola 3

Koncept aplikace

V této kapitole popisuji důležitá návrhová rozhodnutí ohledně architektury celé aplikace. Při návrhu aplikace jsem se řídil dle dříve nasbíraných požadavků 2.4. Některé části návrhu jsou výsledkem společných rozhodnutí, jichž jsme dosáhli s kolegy, kteří se zabývají aplikacemi pro simulované ochlazování [1] a Tabu prohledávání [2], protože jsme v průběhu analýzy/návrhu byli vyzváni vedoucím, abychom svoje aplikace sloučili do jedné.

3.1 Webová nebo desktopová aplikace

Nejprve bylo nutné rozhodnout, zda se bude jednat o webovou, či desktopovou aplikaci. Snažil jsem se především vyhovět požadavku Snadná spustitelnost bez instalace.

Výhody desktopové aplikace jsou zřejmé, uživatel si ji může pustit kdykoli i bez připojení k internetu. Není nutné řešit komunikaci mezi klientem a serverem. Navíc u desktopových aplikací příliš nehrozí, že by po čase přestaly být podporovány/spustitelné, jako se tomu stalo u Java appletů. Problémem naopak může být nutnost instalace a běh na různých operačních systémech, popřípadě omezení na jeden operační systém, na kterém bude aplikace fungovat. Jedním z nápadů bylo vytvořit desktopovou aplikaci v jazyce Java. Tímto by se běh na různých operačních systémech vyřešil. Přinášelo by to však nutnost mít k dispozici JVM na počítači, kde se pokusíme aplikaci spustit. To z hlediska požadavku 2.4.2.3 není problém, protože JVM jsou k dispozici na počítačích v učebnách. Tuto možnost jsem vítal, protože Java je programovací jazyk, ve kterém jsem nejzkušenější.

Zdá se však, že webová aplikace je z hlediska účelu použití vhodnější. Předpokládá se totiž, že studenti si aplikaci spustí na cvičeních MI-PAA a pak možná několikrát doma. Z tohoto důvodu není nutné, aby studenti měli aplikaci ve svých počítačích a zbytečně jim zabírala místo, když ji po skončení předmětu již nejspíše nevyužijí. Další výhodou webové aplikace je, že je nezávislá na operačním systému, stačí ji otevřít v prohlížeči, který podporuje

použité technologie. Zde zároveň narážíme na největší nevýhodu webové aplikace – prohlížeč musí podporovat technologie, které aplikace používá, aby bylo možné aplikaci spustit. Nikdo bohužel nezaručí, že technologie, které prohlížeč podporuje dnes, bude podporovat i za několik let. Nemáme žádnou záruku zpětné kompatibility, jako je tomu například u jazyka Java. Aplikaci tedy může časem potkat stejný osud jako původní Java applety.

Nicméně v dnešní době jsou webové aplikace moderní a dosti vytlačují desktopové aplikace. Proto jsem se rozhodl splnit přání studentů a navzdory tomu, že s webovými aplikacemi nemám velké zkušenosti, jsem se přiklonil k variantě webové aplikace.

Někdy v průběhu tohoto rozhodování padl návrh od vedoucího, abychom společně s kolegy, kteří se zabývají aplikacemi pro simulované ochlazování [1] a Tabu prohledávání [2], spojili své aplikace do jedné. Proto jsme se rozhodli, že uděláme webovou aplikaci, která bude naše algoritmy sdružovat.

3.1.1 Komunikace mezi klientem a serverem

Zvolení webové aplikace přineslo otázku: Jak vyřešit komunikaci mezi klientem a serverem? Protože genetický algoritmus společně s ostatními algoritmy je poměrně náročný na výpočetní výkon, rozhodli jsme se, že veškerá práce bude na straně klienta a server bude pouze prostředníkem, který uživateli umožní přístup k aplikaci. Tímto rozhodnutím se zároveň vyřešil problém se škálovatelností a dostupností serveru. K těmto problémům by jinak mohlo docházet, protože na cvičeních bude mít aplikaci spuštěnou více jak 20 uživatelů současně. A dle mých vlastních zkušeností studenti rádi experimentují, co aplikace vydrží. Použitím tohoto řešení by si student v nejhorším případě „shodil“ vlastní počítač, ale náš server bude stále dostupný.

3.2 Problém a algoritmus

Rozhodnutí spojit mou aplikaci s dalšími dvěma algoritmy značně ovlivnilo další vývoj. Nyní tedy aplikace obsahuje různé problémy a různé algoritmy. Každý algoritmus musí být schopen vyřešit jakýkoli z nabízených problémů. Problémy nemusí vůbec vědět o dalších problémech nebo algoritmech, pouze musí nabízet určité rozhraní, které mohou algoritmy využívat. Stejně tak by algoritmy měly být problémově nezávislé a řešit je pouze pomocí nabízeného rozhraní. Algoritmy by měly existovat nezávisle na ostatních metodách. Představa je taková, že by se do aplikace daly přidávat další algoritmy a problémy.

3.3 Práce v jiném vlákně

Vzhledem k tomu, že výpočet algoritmu je náročná operace, musí být tento výpočet přesunut do samostatného vlákna, aby se nezablokovalo GUI a uži-

vatel mohl s aplikací dále interagovat a případně výpočet zastavit. To samé platí pro generování instancí problémů, což může být také časově náročné především pro velké a složité instance. Použitá technologie tedy musí podporovat více výpočetních vláken.

3.4 Databáze

Jako nejlepší možnost naplnění požadavku 2.4.1.3 se ukázalo zavedení databáze do naší aplikace. Otázkou bylo, zda použít databázi na serveru nebo u klienta. Sdílená serverová databáze nepřicházela v úvahu. Pokud bychom chtěli zřídit databázi na straně serveru, museli bychom požadovat autentizaci. Registrace a přihlašování je pro studenty nepříjemné a zdlouhavé, proto padlo rozhodnutí, vytvořit databázi u klienta za pomoci internetového prohlížeče.

Kapitola 4

Společné uživatelské rozhraní

Abychom nebyli svázáni implementačními detaily, začal jsem společně s kolegy s návrhem uživatelského rozhraní aplikace v rámci předmětu MI-NUR. Tato část byla poměrně náročná, protože každý z nás měl jinou představu o aplikaci. Museli jsme tedy splnit požadavky na všechny tři aplikace a dojít k vzájemné shodě.

4.1 Návrh GUI

Začali jsme s prvotním návrhem webové aplikace bez zkoumání podobných aplikací, abychom nebyli ovlivněni a výsledkem nebylo pouhé okopírování existující aplikace. Poté jsem vypracoval rešerše 4.2 a získané poznatky jsme zapracovali do výsledného prototypu 4.3. Prvotním návrhem se ve své práci [1] detailněji zabývá kolega Kluzáček, a proto zde uvedu pouze stručné shrnutí.

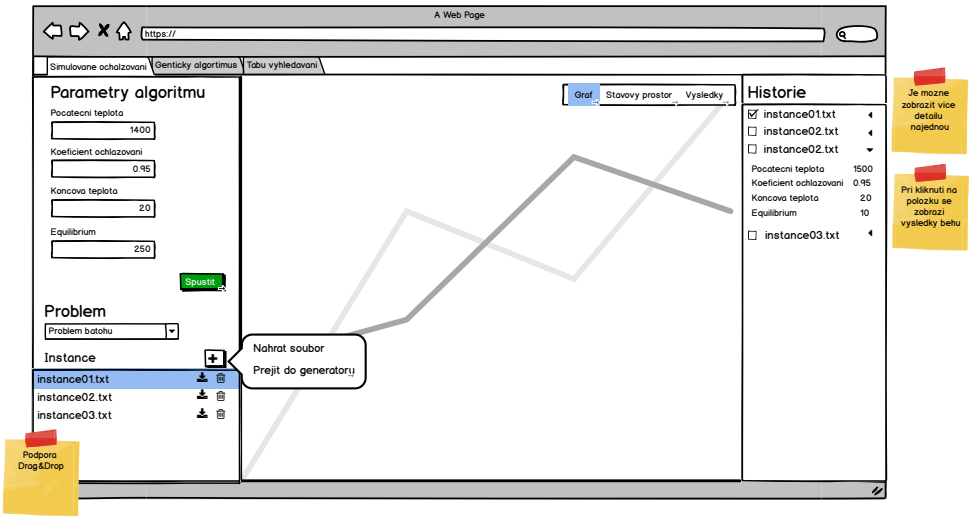
4.1.1 Product statement

Projekt slouží k demonstraci běhu iterativních algoritmů. Bude sloužit studentům k lepšímu pochopení těchto algoritmů v předmětu MI-PAA za pomocí grafického zobrazení průběhu výpočtu.

4.1.2 Business requirements

Návrh se řídil požadavky uvedenými v 2.4. Zde uvádím ještě hlavní požadavky, které se týkají společného GUI.

- Grafická prezentace běhu algoritmů.
- Řešení několika druhů problémů.
- Generátor náhodných instancí problémů.
- Řešení problémů pomocí algoritmů:



Obrázek 4.1: Prvotní návrh UI

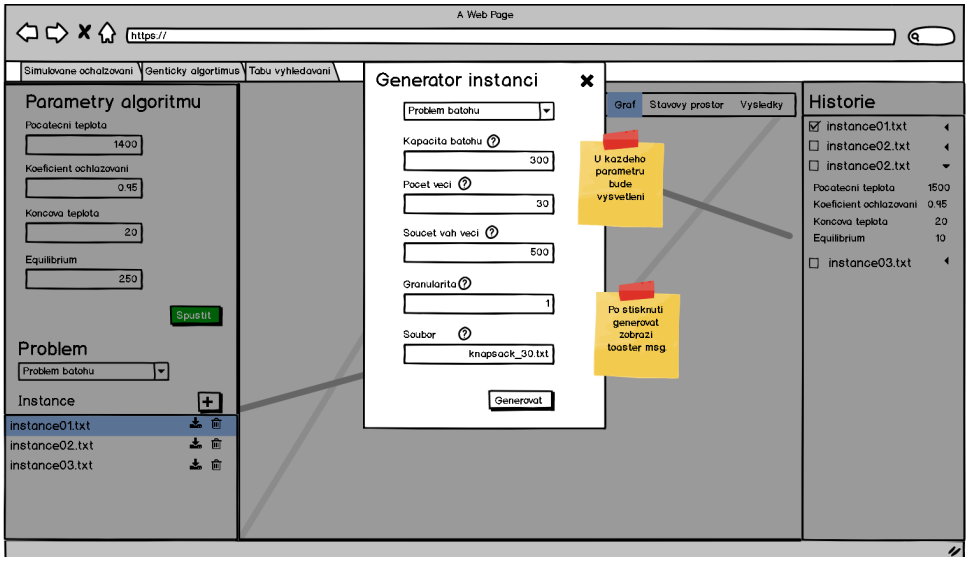
  - Simulované ochlazování
  - Genetický algoritmus
  - Tabu prohledávání

- Možnost nastavení parametrů daného algoritmu.

4.1.3 Popis wireframů

Prvním rozhodnutím, jak koncipovat aplikaci, bylo z jakého pohledu se dívat na algoritmy a problémy. Můžeme zvolit nejdříve problém a nějakou jeho instanci řešit jednotlivými algoritmy. Pro účel této aplikace bylo však důležitější rozlišovat především algoritmy, protože se předpokládá, že student bude pracovat s jedním algoritmem. Až se s ním dostatečně seznámí, bude se věnovat jinému. Proto v horní části aplikace vidíme záložky jednotlivých algoritmů.

Chtěli jsme, aby uživatelské rozhraní bylo přehledné a uživatel viděl vše, co může s aplikací dělat, ale zároveň nebyl zahlcen informacemi. Celou aplikaci jsme vizuálně rozdělili na tři logické části: levý, střední a pravý panel. Levý panel představuje vstupy pro algoritmus. Střední panel zobrazuje výsledky běhu algoritmu. Pravý panel funguje jako historie. Návrh v podobě wireframu můžete vidět na obrázku 4.1.



Obrázek 4.2: Wireframe generátoru

4.1.3.1 Levý panel

Tento panel se skládá ze dvou částí. První z nich jsou nastavitelné parametry algoritmu, které jsou pro každý algoritmus jiné. Tato část hraje klíčovou vzdělávací roli a očekává se, že právě s parametry bude student nejvíce experimentovat, a tudíž i interagovat s touto částí GUI. Abychom studentům usnadnili experimentování, je každá vstupní položka validována tak, aby uživatel nemohl zadat nesmyslnou hodnotu, a zároveň je každý parametr opatřen nápovědou, která studentovi osvětlí, co daný parametr znamená. Pod parametry je také umístěno tlačítko na spouštění algoritmu.

Druhá část se zabývá volbou vstupní instance problému, kterou má algoritmus vyřešit. Uživatel může přepínat mezi druhy problému a vybírat různé instance ze seznamu. Položky ze seznamu může stáhnout nebo smazat. Do seznamu může přidat instance nahráním souboru v definovaném tvaru, nebo použít generátor instancí 4.2. Generátor instancí bude mít také nastavitelné parametry, proto se zobrazí jako popup okno.

Problém k řešení je vlastně první věc, kterou by měl člověk vybírat, když chce použít některý z implementovaných optimalizačních algoritmů. Proto jsme se poměrně dlouho dohadovali, zda ho neumístit ještě před parametry algoritmu, nebo jako první položku do levého panelu. Vzhledem k dynamicky se měnící délce seznamu instancí a zachování rozložení vstup → výstup → historie nám však výsledné řešení přišlo jako nejvhodnější volba.

4.1.3.2 Střední panel

Úkolem tohoto panelu je zobrazovat výsledky běhu algoritmu. Přičemž nejdůležitější z nich je graf průběhu algoritmu. Zde je také možné výsledky porovnávat díky historii 4.1.3.3. Původním záměrem bylo zobrazovat výsledky odděleně a překlikávat mezi nimi pomocí tlačítek „Graf“, „Stavový prostor“, „Výsledky“. Později se nicméně ukázalo, že bude lepší zobrazit vše na jednu stránku pod sebe.

4.1.3.3 Pravý panel

Tento panel představuje historii již vykonaných běhů algoritmu. Díky checkboxům je možné porovnávat různé běhy, protože výsledky zatrhnutých instancí se zobrazí na středním panelu. Uživatel si také může připomenout, s jakými parametry algoritmus pouštěl.

4.2 Rešerše podobných aplikací

Tato kapitola obsahuje několik rešerší podobných již existujících aplikací. Rešerše jsou zpracovávány především z hlediska uživatelského rozhraní. Tyto rešerše nám pomohli zdokonalit prvotní návrh naší aplikace. Aplikací, které by měly stejný účel jako naše aplikace, mnoho není, proto jsem se zaměřil i na webové služby, které se nezabývají pouze iterativními algoritmy.

Některé další rešerše, které se týkají především simulovaného ochlazování jsou obsaženy v [1].

4.2.1 Applet genetického algoritmu

Jedná se o applet rozebíraný v 2.1. Zde uvádím pouze výhody a nevýhody, které nám pomohou navrhnout uživatelské rozhraní.

4.2.1.1 Klady

- Vizualizace algoritmu pomocí grafu vývoje nejlepší, průměrné a nejhorší fitness.
- Možnost volby velkého množství parametrů.
- Řešení čtyř různých problémů.
- Možnost upravit vstupní instance.
- Nahrání vlastní instance ze souboru.
- Přednastavené parametry.
- Umožňuje zastavovat algoritmus po jednotlivých generacích.

4.2.1.2 Zápory

- Zastaralá technologie appletů, která na dnešních prohlížečích nefunguje.
- Nevaruje uživatele už při zadání nesprávného vstupu (nečíselný do číselného), ale až po spuštění, navíc varováni nejsou v čitelné formě.
- Vizualizace u obchodního cestujícího velmi nečitelná.
- Po najetí myší na bod v grafu nezobrazí podrobné údaje (přesná fitness a podobně).
- Není responzivní.
- Dropboxy „Configuration“ a „Help“ jsou matoucí (neznámá funkčnost) a funkce u „Help“ ani nefungují.

4.2.2 Genetic Algorithm Walkers

Genetic Algorithm Walkers [12] je webová aplikace, které se snaží naučit panáčka chodit pomocí genetického algoritmu. Na obrazovce vidíme (obr. 4.3) aktuální generaci snažící se chodit, přehled fitness této generace v tabulce a tabulku nejlepších nalezených fitness.

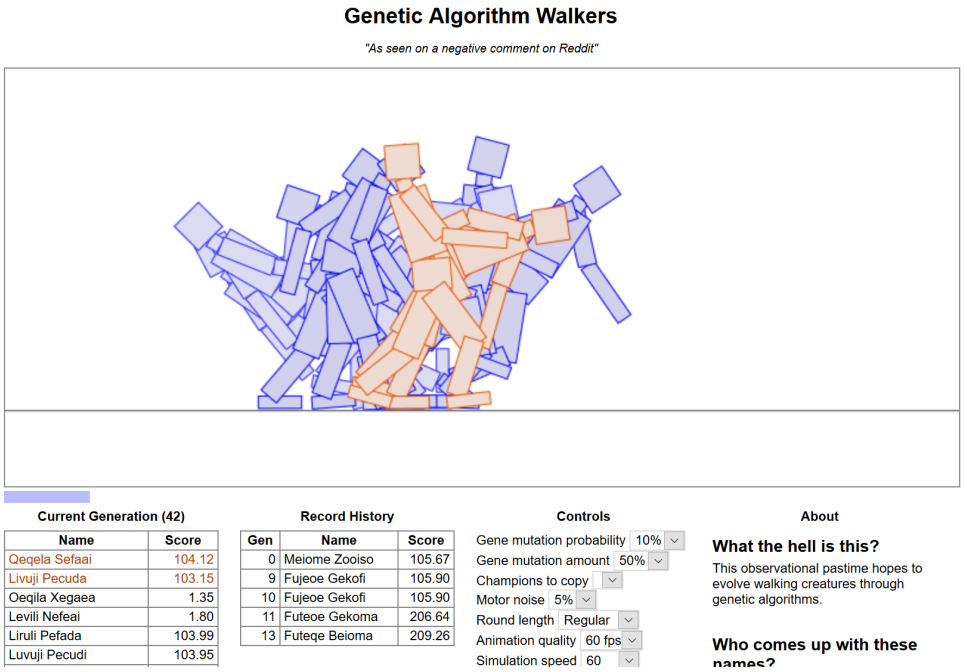
Také zde nalezneme parametry pro genetický algoritmus, které můžeme měnit v reálném čase.

4.2.2.1 Klady

- Zajímavá vizualizace celé generace panáčků (vidíme reálné výsledky).
- Algoritmus začne pracovat ihned po načtení stránky.
- Vše jednoduše a přehledně na jedné stránce.

4.2.2.2 Zápory

- Soustředí se na jediný problém.
- Algoritmus nejde zastavit, ani pustit znovu.
- Omezené množství parametrů (neobsahuje ani všechny základní parametry GA).
- Parametry mohou nabývat pouze několika málo hodnot.
- Nevidíme žádný vývoj, pokud neustále nesledujeme chodící panáčky.



Obrázek 4.3: Genetic Algorithm Walkers

4.2.3 Algovision

Tuto aplikaci jsem zařadil, protože ji opakovaně zmiňovali studenti v dotazníku 2.3. Algovision [13] je projekt profesora Luďka Kučery pro vizualizaci celé řady algoritmů. Tuto aplikaci pan profesor využívá při výuce (např. MI-PAL). Jedná se o webovou aplikaci, která se momentálně přepisuje do JavaScriptu z původního Java appletu, proto funguje pouze omezená množina algoritmů. Potkal ho tedy stejný osud jako předchůdce naší aplikace. Na rozdíl od naší aplikace se Algovision soustředí především na vizualizaci jednotlivých kroků algoritmu. Algoritmy v Algovision bohužel nezahrnují algoritmy naší aplikace.

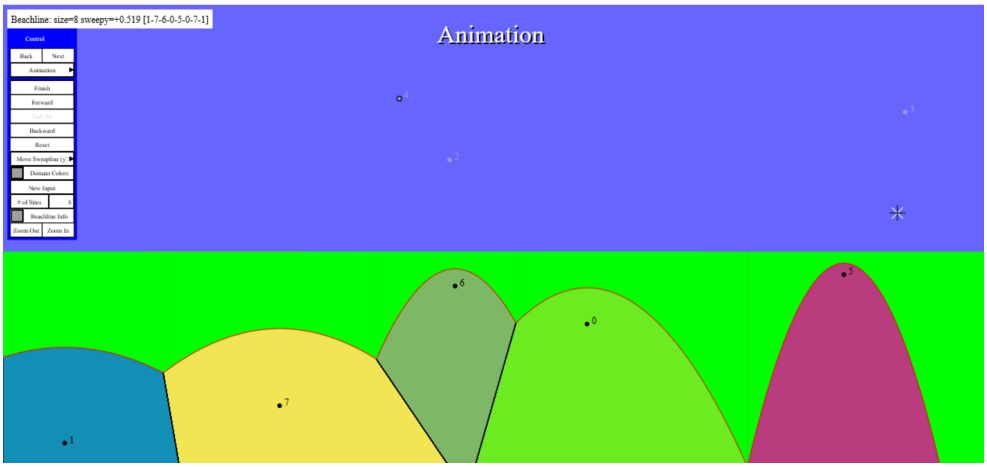
Algovision nás uvítá úvodní obrazovkou a po zahájení „procházky“ nám nabídne menu s kategoriemi a konkrétními typy algoritmů. Po zvolení algoritmu přejdeme na obrazovku s vizualizací a kontrolním panelem.

Běh algoritmu je možné spouštět, pozastavovat a krokovat. Pro každý algoritmus je mnoho možností, jak nastavit vizualizaci. Můžeme zvolit předdefinovaný vstup nebo náhodně vygenerovaný.

Obrázek 4.4 ukazuje práci aplikace, kde tvoří Voroného diagram.

4.2.3.1 Klady

- Přehledné menu pro výběr algoritmu.
- Vizualizace běhu algoritmu několika způsoby – vždy graficky, ale v růz-



Obrázek 4.4: Algovision

ných kontextech algoritmu (např. motivace, představa, povrchní pohled, pohled se všemi detaily).

- Vizualizace přispívá k pochopení vnitřního chování algoritmu.
- Možnost změnit vstup.
- Možnost přemístit ovládací panel kamkoli.

4.2.3.2 Zápory

- Vrátit se na stránku s algoritmy jde pouze tlačítkem „Zpět“ v prohlížeči a návrat přes úvodní obrazovku.
- Některým algoritmům chybí na demonstrační stránce nadpis, tudíž můžeme zapomenout, o co se jedná.
- Nemůžeme vložit vlastní vstup.
- Aplikace je určena pro výuku s učitelem, který algoritmy komentuje, proto je těžké algoritmy pochopit pouze z vizualizace.
- Aplikace se také obtížně ovládá (mnoho kontrolek), pokud jste s ní nebyli dříve obeznámeni.

4.2.4 VisuAlgo

VisuAlgo [14] je webová stránka, která umožňuje vizualizaci několika druhů algoritmů. Od klasických řadících algoritmů přes hash tabulky až po všemožné grafové algoritmy a mnohé další.

Celkově obsahuje stránka 24 různých typů algoritmů pro vizualizaci. Každá z těchto kategorií obsahuje vždy ještě několik druhů zvoleného algoritmu. Kategorie řazení obsahuje např. algoritmy: bubble sort, selection sort, insertion sort atd.

VisuAlgo umožňuje pro algoritmy generovat vstupní data podle různých parametrů nebo je ručně vkládat. Běh algoritmu je možné spouštět, pozastavovat a krokovat. Je také možné pozorovat běh algoritmu ve slovním popisu nebo v kódu. Náhled aplikace je k dispozici na obrázku 4.5.

4.2.4.1 Klady

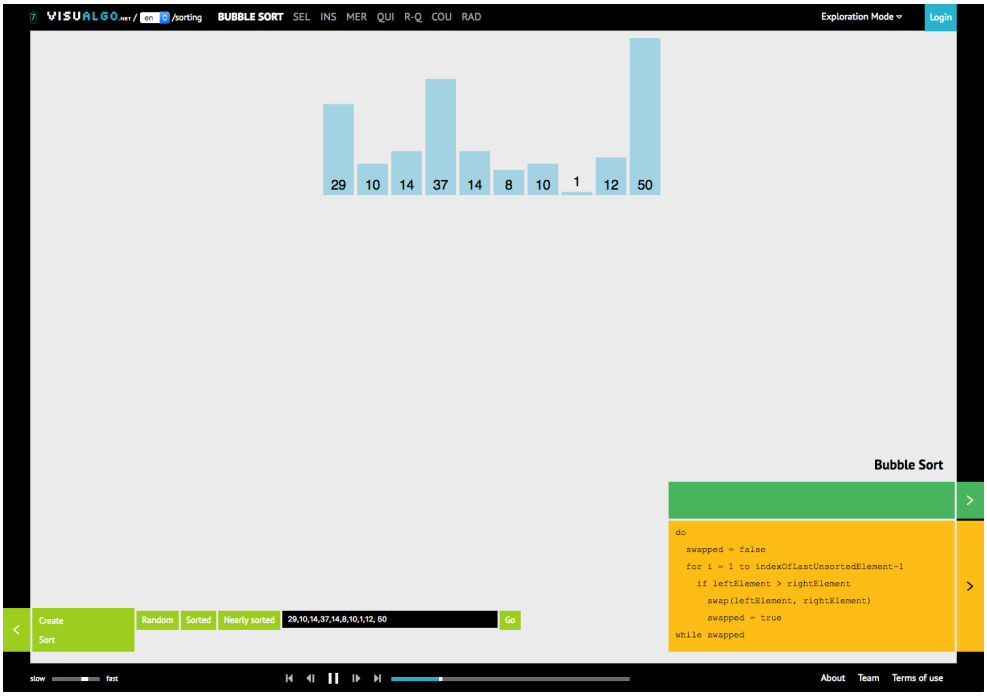
- Vizualizace běhu algoritmu několika způsoby – graficky, textově, v kódu.
- Textový úvod ke každému algoritmu.
- Minimalistická drobečková navigace – i přesto, že se jedná o jednoduchou aplikaci z pohledu průchodu jednotlivými stránkami, je navigace šikovnou funkcí.
- Jednoduché přepínání mezi algoritmy stejného typu.
- Vizuální i animované zobrazení jednotlivých kategorií na hlavní stránce.

4.2.4.2 Zápory

- Před spuštěním algoritmu na stránce vizualizace jsou místa, s budoucím popisem algoritmu a kódem, prázdná a bez jakéhokoliv popisu.
- Slovní popis algoritmu občas přetéká, a tudíž není plně čitelný.
- Při ručním vkládání hodnot není předem definováno, v jaké formě hodnoty vkládat a jaká mají omezení.
- Zobrazované chyby překrývají část se zobrazeným kódem.
- Před zvolením kategorie se uživatel nemůže podívat, jaké algoritmy kategorie obsahuje.

4.2.5 Data Structure Visualizations

Data Structure Visualizations [15] je další učební pomůckou. Aplikace se tentokrát zaměřuje na datové struktury a některé algoritmy. Soustředí se především na jednotlivé kroky, které jsou animované, a je možné proces pozastavit a krokovat. Nabídka datových sktruktur je veliká, já zvolil k detailnějšímu zkoumání vizualizaci AVL stromů, jejíž náhled můžete vidět na obrázku 4.6.



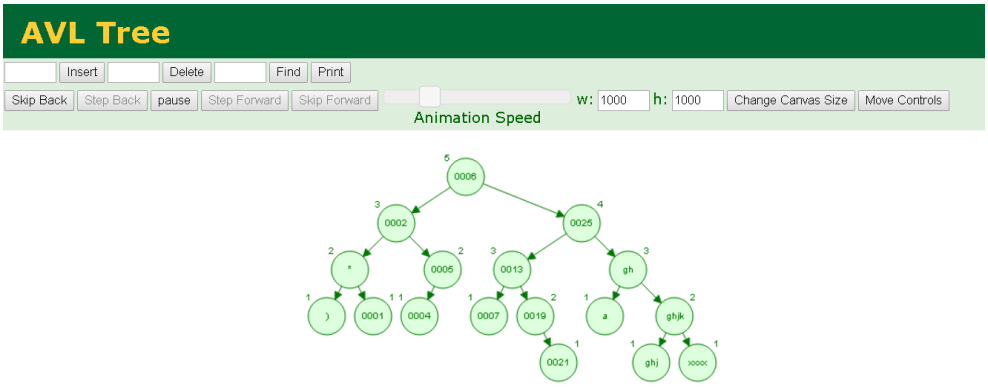
Obrázek 4.5: VisuAlgo

4.2.5.1 Klady

- Rozdělení specifických kontrolek algoritmu (horní panel) a obecných kontrolek (dolní panel).
- Pěkné vizualizace jednotlivých postupů.
- Přehledné a minimalistické.
- Možnost zvolení rychlosti animace.
- Několik možností, jakou formou sledovat algoritmus.

4.2.5.2 Zápory

- Změna velikosti canvasu někdy nefunguje, bez jakéhokoli upozornění.
- Po změně canvasu nedojde k překreslení struktury.
- Většinou je nutné datovou strukturu naplnit ručně a postupně po jednom prvku.
- Po „Skip Back“ není možné použít „Skip Forward“.
- „Skip Forward“ tlačítko neintuitivně pouze urychlí spuštěnou animaci.



Obrázek 4.6: Data Structure Visualizations

- Některá pole nereagují na vstup z numerické klávesnice.
- Chybí nápověda nebo jasné označení, kde začít.

4.3 Implementace prototypu

Implementací prototypu se zabýval kolega Veselý ve své práci [2], proto zde opět uvedu jen stručné shrnutí. Prototyp v tomto případě znamená funkční uživatelské rozhraní. Tedy, že jednotlivé komponenty UI (tlačítka, editboxy, . . .) fungují, nicméně vnitřní logika ještě není implementována, takže se pouze simuluje správné chování. Průběh algoritmu tedy vrací pouze náhodnou posloupnost fitness nezávisle na vstupních parametrech a generátor instancí vrací stále stejnou předdefinovanou instanci.

Prototyp vychází z předchozího návrhu 4.1 a zároveň se inspiruje rešeršemi 4.2, navíc jsme doladili několik nedostatků, na které upozornil náš cvičící a konzultant v rámci MI-NUR Ing. Jiří Hunka.

Prototyp se od původního návrhu příliš neliší, jak můžete pozorovat na obrázku 4.7. Nejpodstatnější změnu prodělal střední panel, kde již není přepínání mezi jednotlivými druhy výsledků. Je zde umístěn graf, pod ním charakteristiky běhů a pod nimi výsledné konfigurace běhů, tedy reálné výsledky pro problém.

K parametrům v levém panelu byly doplněny otazníčky, které po najetí myší zobrazí nápovědu. Pro vložení, generování a smazání všech instancí byla přidána samostatná tlačítka.

V pravém panelu byla přidána možnost vybrat první (naposledy spočítanou) položku z historie a také smazat celou historii.

Obrázek 4.7: Ukázka prototypu algoritmu

4.4 Heuristická analýza prototypu

Heuristickou analýzu jsem provedl podle deseti pravidel Jakoba Nielsena [16], které jsou prakticky popsány v [17]. Závažnosti byly vyhodnoceny od 1 do 5, kdy 5 indikuje velice závažný problém a 1 zanedbatelný problém. Výsledky jednotlivých bodů jsou v přehledové tabulce 4.1, každý bod odpovídá jedné níže uvedené kapitolce. Většina nalezených problémů byla postupně odstraněna. Ty zbylé zanikly odstraněním problémů odhalených uživatelským testováním 4.5.

4.4.1 Viditelnost stavu systému

Uživatel musí být informován o tom, co systém právě dělá, s vhodnou odezvou v přijatelném čase.

Vyhodnocení

Déle trvající akce je pouze výpočet pomocí optimalizačního algoritmu. Uživatel vidí průběh na grafu, do kterého průběžně přibývají data z aktuálního výpočtu. Ví tedy, v jaké fázi výpočet je a jak dlouho bude ještě trvat.

4.4.2 Shoda mezi systémem a realitou

Zachování konvencí z reálného světa. Systém by měl hovořit jazykem uživatele.

Nalezené problémy

- V historii vyřešených instancí parametry odpovídají proměnným v kódu, nikoliv opravdovým názvům parametrů. (Závažnost 4)

- Nadpis „Computed instances“ by měl být „History“. (Závažnost 3)
- „Input files“ by měl být přejmenován např. na „Input instances“. (Závažnost 2)

4.4.3 Minimální zodpovědnost (a stres)

Navodit uživateli dojem, že nemůže nic pokazit a že on ovládá systém, nikoli naopak. Umožnit uživateli, aby mohl jednoduše vrátit akci, kterou omylem spustil.

Nalezené problémy

- Není možné smazat pouze jednu položku historie řešení. (Závažnost 2)
- Při „select first“ se nedá vrátit zpátky. (Závažnost 1)

4.4.4 Shoda s použitou platformou a obecnými standardy

Uživatelé by neměli přemýšlet, jestli různá slova, situace a akce znamenají stejné věci. Řídit se standardy.

Vyhodnocení

Aplikace používá standardní komponenty a styl Bootstrap.

4.4.5 Prevence chyb

Lepší než dobré chybové hlášky je, když k chybě nemůže vůbec dojít. Snažte se předcházet chybám uživatele.

Nalezené problémy

- Uživatel má možnost zadat chybné parametry (omezení je pouze na čísla), pole však může být prázdné nebo mít nesmyslnou hodnotu, např. zápornou. (Závažnost 5)
- Tlačítko Start by mělo být stav disabled pokud nejsou všechny parametry správně nastaveny. (Závažnost 4)

4.4.6 Kouknu a vidím

Nezatěžovat uživatelovu paměť. Akce a informace, které uživatel momentálně potřebuje, by měly být viditelné a snadno dosažitelné.

Vyhodnocení

Uživatel vidí vše, co může v danou chvíli dělat. Okno generátoru překryje hlavní okno. Zapamatovatelné ikonky jsou použity tam, kde to bylo možné.

4.4.7 Flexibilita a efektivita

To, co zkušený uživatel pravidelně potřebuje, běžný kolikrát ani nepotřebuje vědět. Skrýt pokročilá nastavení pro běžné uživatele a umožnit zkušeným uživatelům pracovat efektivně.

Vyhodnocení

Naši aplikaci budou používat především studenti jakožto běžní uživatelé. Většině nastavení by měli rozumět a měli by mít možnost si je vyzkoušet. Za pokročilé uživatele můžeme považovat cvičící, kteří budou aplikaci prezentovat studentům. Nebudou tedy používat nic, co by nechtěli po studentech. Z těchto důvodů nepotřebujeme UI šít na míru běžným a pokročilým uživatelům zvlášť.

4.4.8 Estetika a minimalizmus

Zobrazovat pouze to, co je aktuálně důležité, stručně a jasně.

Nalezené problémy

- U potvrzovacích oken je navíc nadpis „Are you sure?“. (Závažnost 1)

4.4.9 Smysluplné chybové hlášky

Chybové hlášení by mělo v běžném jazyce popsat, co se stalo špatně, jak se to stalo a jak tomu příště předejít.

Vyhodnocení

Chybová hlášení jsou stručná a jasná v přirozeném jazyce.

4.4.10 Nápověda a dokumentace

Systém by měl obsahovat nápovědu, i když by se měl dát používat i bez ní. Potřebné informace by měly být snadno dohledatelné a zaměřené na úkon, který chce uživatel provést.

Nalezené problémy

- Systém obsahuje nápovědu ve formě tooltips, nicméně neobsahuje nápovědu k vysvětlení práce s aplikací. (Závažnost 2)

Bod	Rozpor	Max. závažnost
1	✗	-
2	✓	4
3	✓	2
4	✗	-
5	✓	5
6	✗	-
7	✗	-
8	✓	1
9	✗	-
10	✓	4

Tabulka 4.1: Celkové vyhodnocení heuristické analýzy prototypu

	Jméno	Student FIT ČVUT
1	„Eva“ Nguyen	Ano
2	Marek Dostál	Ano
3	Andrej Kozlov	Ano
4	Barbora Jančovičová	Ano
5	Veronika Larionova	Ano
6	Radek Nevyhoštěný	Ne
7	Pavel Večeřa	Ne

Tabulka 4.2: Seznam testerů

- Systém neobsahuje popis formátu vstupních souborů pro problémy. (Závažnost 4)

4.5 Uživatelské testování prototypu

Kromě heuristické analýzy jsem provedl také uživatelské testování. Testování probíhalo ve školní učebně. Testeři byli převážně studenti FIT ČVUT s absolvovaným předmětem MI-PAA. Zapojil jsem i pár laiků, aby zhodnotili aplikaci z hlediska běžných uživatelů (ne studentů IT). Přehled testerů je uveden v tabulce 4.2. Nejprve jsem testerům vysvětlil, o jakou aplikaci jde a co je jejím účelem. Následně jsem jim položil několik úvodních otázek 4.5.1. Poté jsem nechal testery splnit úkoly podle testovacích scénářů 4.5.2. V průběhu plnění úkolů jsem do procesu nijak nezasahoval a obrazovka na testovacím stroji byla nahrávána. Závěrem jsem prodiskutoval plnění scénáře s každým testerem zvlášť. Zaměřil jsem se především na úkony, kde měl tester nějaké problémy. Uživatelské testování pomohlo odhalit několik dalších nedostatků, jejichž náprava přispěla ke zlepšení použitelnosti této aplikace.

Tester	1.	2.	3.	4.	5.
1	Studuji ho	GA	jen Tabu	Ne	Proklikání záložek
2	Ano	Nevím	Ano	Ne	Proklikání záložek
3	Ano	SA	Ano	Ne	Kliknutí „Start“
4	Ano	GA	Ano	Ne	Proklikání záložek
5	Ano	SA	Ano	Ano jeden	Proklikání záložek
6	Ne	–	–	Ne	Kliknutí „Start“
7	Ne	–	–	Ne	Kliknutí „Start“

Tabulka 4.3: Odpovědi testerů

4.5.1 Předtestovací otázky

1. Absolvovali jste úspěšně předmět MI-PAA?
2. Jaký algoritmus jste si vybrali pro svou závěrečnou práci v předmětu MI-PAA?
3. Znáte i zbylé dva algoritmy?
4. Znáte bývalé applety, které sloužily k demonstraci běhu iterativních algoritmů.
5. Prohlédněte si aplikaci a sdělte nám, jak byste aplikaci začali používat.

Odpovědi testerů jsou uvedeny v tabulce 4.3.

4.5.2 Testovací scénáře

1. Jste znovu studentem předmětu MI-PAA a chcete se něco dozvědět o iterativních algoritmech. Začněte spuštěním řešení problému SAT pomocí Tabu search.
2. Chcete zjistit, jak parametry algoritmu změní běh. Upravte parametry algoritmu a pusťte řešení znovu.
3. Porovnejte výsledky obou běhů.
4. Přestalo vás bavit spouštět pouze jednu instanci problému. Vložte do aplikace vlastní soubor (k nalezení na ploše) a spusťte běh znovu s tímto souborem.
5. Došly vám instance SAT problému, vygenerujte si vlastní s libovolnými parametry.
6. Zjistili jste, že Tabu search není pro vás, zkuste spustit vámi vygenerovanou instanci i pomocí genetického algoritmu.

7. Chcete vyzkoušet, co dál aplikace umí. Změňte typ problému na problém batohu, vygenerujte vlastní instanci a znovu spusťte řešení.
8. Již se nevyznáte ve všech instancích a v historii řešení je také moc záznamů. Smažte všechny vstupní soubory a celou historii.

4.5.3 Nalezené problémy

1. Tlačítko „Select first“ není intuitivní.
   - Závažnost: 4
   - Řešení: Tlačítko bude nahrazeno checkboxem, který bude vybírat vše a rušit výběr všeho.
2. Ze souborů není možné určit jejich parametry.
   - Závažnost: 3
   - Řešení: Bude opraveno správným pojmenováním souborů a zobrazením parametrů instance po najetí myší.
3. Generátor se nezavře po vygenerování instance.
   - Závažnost: 1
   - Řešení: Z důvodu možnosti vygenerování více instancí při jednom otevření generátoru bude možné zatrhnout checkbox nezavírat generátor.
4. Lze nastavit nevalidní parametry algoritmu.
   - Závažnost: 5
   - Řešení: Přidání validace.
5. „Clear history“ maže historii napříč celým programem, nikoliv pouze u vybraného algoritmu.
   - Závažnost: 3
   - Řešení: Oprava na mazání pouze u vybraného algoritmu.
6. Plus tlačítko pro generátor není intuitivní.
   - Závažnost: 2
   - Řešení: Zvážení změny ikonky.
7. Pro porovnávání výsledku není zatrhávání nejvhodnější.
   - Závažnost: 1
   - Řešení: Nenalezeno lepší řešení.

Kapitola 5

Architektura a použité technologie

Poté co jsem popsal koncept aplikace 3 a bylo navrženo uživatelské rozhraní 4, mohu popsat výslednou architekturu aplikace, která byla částečně použita při implementaci prototypu 4.3 a detailněji ji popisuje kolega Veselý v [2].

Aplikace je napsána v programovacím jazyce JavaScript, který umožňuje, aby běžela na straně klienta. Uživatelské rozhraní se skládá z jednotlivých Vue.js komponent a správa stavu aplikace je zajišťována pomocí Vuex. Pro vizualizaci grafů byla použita knihovna dc.js. Aby náročné výpočty neblokovaly UI, byl použit Worker Loader, který umožňuje vytvářet Web Workery, kteří mohou spouštět skripty na pozadí v odděleném vlákně a komunikují s UI vláknem pomocí předávání zpráv. Pro ukládání výsledků byla použita ZangoDB jako klientská databáze v prohlížeči.

5.1 JavaScript

JavaScript [18] je interpretovaný, objektově orientovaný programovací jazyk s možností předávat funkce jako parametry ostatním funkcím. Je to nejznámější skriptovací jazyk pro webové stránky. JavaScript je založen na prototypové dědičnosti, je dynamický a podporuje objektově orientovaný, imperativní a funkcionální programovací styl. JavaScript běží na straně klienta, což může být použito pro naprogramování toho, jak se bude webová stránka chovat v reakci na nějakou událost.

5.2 Bootstrap

Bootstrap [19] je open source knihovna pro vytváření webových stránek a aplikací. Obsahuje obrovské množství HTML a CSS šablon pro často používané

UI prvky. Styl této knihovny je považován za standard na webu a zároveň je vzhled prvků moderní.

5.3 Vue.js

Vue.js [20] je framework pro budování uživatelského rozhraní. Uživatelské rozhraní skládá z jednotlivých komponent a zaměřuje se především na prezentační vrstvu (to, co uživatel vidí).

5.4 Vuex

Vuex [21] se stará o správu stavu a zároveň je knihovnou pro Vue.js aplikace. Slouží jako centralizované úložiště pro všechny komponenty v aplikaci s pravidly, které zajišťují, že stav aplikace se mění pouze předvídatelným způsobem. Umožňuje, aby více komponent mohlo sdílet společný stav.

5.5 dc.js

dc.js [22] je grafová knihovna pro vizualizaci a analýzu dat. Tato knihovna využívá knihovnu d3 k vykreslování grafů ve formátu SVG. Grafy vykreslené pomocí dc.js jsou reaktivní a řízena daty, proto mohou uživateli poskytnout okamžitou odezvu na jeho akce.

5.6 ZangoDB

ZangoDB [23] je rozhraní ve stylu MongoDB pro HTML5 IndexedDB. Jedná se tedy o databázi, kterou můžeme použít přímo v prohlížeči.

Kapitola 6

Implementace problému

Než začneme s implementací problému, je důležité si uvědomit, že všechny implementované optimalizační algoritmy fungují jako maximalizační. V této kapitole popíši, jak do systému přidat nějaký problém k řešení. Zároveň uvedu, jaké rozhraní musí problém implementovat. Jednotlivé kroky budu demonstrovat na problému obchodního cestujícího, konkrétně jeho Euklidovské variantě v ploše. Tento problém jsem vybral proto, že jsem chtěl aplikaci rozšířit o další permutační problém a původní applety tento problém také obsahovaly.

6.1 Problém obchodního cestujícího

Tento problém patří mezi nejznámější kombinatorické problémy a je důležitý nejen z teoretického hlediska, ale má i své praktické využití. Problém obchodního cestujícího (TSP) je definován následovně:

- Je dána množina měst (vrcholů).
- Jsou dány nezáporné vzdálenosti (ohodnocené hrany) mezi každými dvěma městy (úplný graf).
- Úkolem je najít nejkratší cestu (posloupnost měst) tak, abychom navštívili každé město a vrátili se tam, odkud jsme vyrazili.

6.1.1 Euklidovský TSP

Jedná se o variantu předchozího problému, kde nejsou vzdálenosti zadány explicitně. Vzdálenosti jsou zadány implicitně, protože každé město má svoje souřadnice. Z těchto souřadnic se poté vypočte Euklidovská vzdálenost. Máme tedy pevné $d \in \mathbb{N}$ a města jsou zadána jako body v $\mathbb{R}^d$. Pro města x a y, které mají d souřadnic, spočteme vzdálenost jako:

$$\sqrt{\sum_{i=1}^{d}(x_i - y_i)^2}$$

Moje implementace je omezená pouze na dvě souřadnice (jedná se tedy o plochu), protože je to jednoduše představitelné a zároveň praktické pro zadávání měst z mapy (pokud pomineme zakřivení Země). Počítáme tedy vzdušnou vzdálenost mezi městy:

$$\sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$$

Z teoretického hlediska se jedná o jednodušší problém než obecný TSP. V knize [24] je popsán PTAS algoritmus, který tento problém řeší.

6.2 Obecný problém

Hlavním úkolem jak definovat nový problém je rozšířit třídu `Problem` nacházející se v souboru `Problem.js`. Ač Javascript nezná pojem abstraktní třídy, přesně takto se můžeme na uvedenou třídu dívat.

6.2.1 Konfigurace

Nejprve bych představil důležité objekty, se kterými problém pracuje. Jedná se o konfigurace. Více problémů může používat stejné konfigurace, proto jsou v samostatné třídě. V době psaní tohoto textu aplikace obsahovala dva typy konfigurací: binární pole (`BitArray`) a permutaci (`Permutation`). Tyto objekty se umí náhodně vygenerovat, zkopírovat a podobně. Dále umožňují nalézt sousední stav, což je pro účely GA nepodstatné, ale zbylé algoritmy tuto možnost využívají, protože pracují přímo s touto konfigurací. V implementaci GA naopak používám pouze data této konfigurace, která zapouzdřím do příslušného jedince 7.2. `BitArray` obsahuje samozřejmě pole bitů potřebné délky a `Permutation` obsahuje permutaci potřebné délky uloženou v poli a začínající od 0.

6.2.2 Implementované metody

constructor() Zakazuje vytvářet instance třídy `Problem`.

createNewIndividual() Vytvoří jedince, který odpovídá typu příslušného problému `getType()` a náhodné konfiguraci `getConfiguration(random)`. Tuto metodu používá pouze GA.

evaluateIndividual(individual) Přidělí vstupnímu jedinci fitness pomocí `evaluateMaximizationCost(configuration)`, pokud ji nemá již přidělenou. Tímto se předchází zbytečnému počítání fitness, pro nezměněné jedince. Tuto metodu používá pouze GA.

6.2.3 Abstraktní metody

evaluateMaximizationCost(configuration) Toto je cenová funkce, která ohodnocuje konfigurace problému, a je tedy nezbytná pro správný běh všech algoritmů. Bez ohledu na to, zda je problém maximalizační nebo minimalizační, musí být tato funkce definována tak, aby se algoritmy snažily její hodnotu maximalizovat. U minimalizačních problémů je jednou z možností prosté otočení znaménka.

transformMaximizationToRealCost(maxCost) Tato metoda je prostá kompenzace předchozí funkce. Do výsledného grafu chceme uvádět nějaké smysluplné hodnoty, které se přímo váží k problému. Vstupem této metody je výstup té předchozí a výsledkem je hodnota, kterou chceme zobrazit v grafu. Ve většině případů konkrétních problémů tato metoda vrací nezměněnou vstupní hodnotu, nebo hodnotu s opačným znaménkem. Tato transformační funkce je zde z důvodu efektivity, protože je rychlejší upravit výsledek ceny pro maximalizaci než znovu počítat cenovou funkci pro zobrazení v grafu.

getType() Určuje typ problému z hlediska používané konfigurace 6.2.1.

getConfiguration(random) Vrací konfiguraci problému. Parametr určuje, zda bude náhodná, nebo se použije výchozí.

getResult(configuration) Zrekonstruuje skutečný výsledek problému ze vstupní konfigurace.

isInvalidInstance(instanceContent) Zkontroluje, zda je vstupní řetězec ve správném formátu pro vytvoření instance problému.

resolveInstanceParams(instanceContent) Získá parametry o instanci, která je zadaná jako textový řetězec.

6.2.4 Implementace Euklidovského TSP

Nyní tedy mohu naimplementovat problém tak, jak jsem ho definoval v 6.1.1.

6.2.4.1 Formát vstupu

Nejprve definuji vstupní formát textových souborů, které reprezentují instance problému. Na prvním řádku je číslo, které udává počet měst. Na každém dalším řádku jsou souřadnice jednoho města oddělena mezerou. Počet těchto řádků odpovídá počtu měst a města jsou očíslována od 0 dle pořadí řádků. Vše, co je za požadovaným vstupem je ignorováno. Vstup může vypadat například takto:

```
5       //5 cities
15 46   //city 0
72 56   //city 1
216 45  //city 2
23 15   //city 3
78 32   //city 4
```

V souladu s tímto formátem implementuji metodu `constructor(data)`, která převede textový řetězec na objekt konkrétní instance problému. Dále implementuji metody `isInvalidInstance(instanceContent)`, která zkontroluje tento formát, a `resolveInstanceParams(instanceContent)`, která zjistí velikost instance z prvního parametru.

6.2.4.2 Formát výstupu

Jak vyplývá z definice problému 6.1, jedná se o permutační problém, proto budu využívat permutace jako konfigurace. Výstupem je tedy pořadí měst. Nezáleží na tom, ve kterém městě začínám, protože chci najít cyklus. Cyklicky posunuté permutace jsou tedy z hlediska tohoto problému ekvivalentní. Musím tedy zajistit, aby `getType()` identifikoval problém jako permutační a `getConfiguration(random)` vracela permutace. `getResult(configuration)` zde pouze vytáhne vlastní permutační pole.

6.2.4.3 Cenová funkce

Cenová funkce pro TSP je poměrně přímočará. Chci minimalizovat délku cesty. Pro účely naší aplikace tedy budu maximalizovat délku cesty v záporné hodnotě jako `evaluateMaximizationCost(configuration)`. Délku cesty spočítám jako vzdálenosti mezi městy, které spolu sousedí v permutaci. Nesmím zapomenout připočíst i vzdálenost z posledního města do prvního. Jelikož vzdálenosti mezi městy nejsou explicitně zadány a musejí se netriviálně počítat, přidal jsem do problému paměť, která si zapamatuje spočítanou vzdálenost mezi městy. Tato vzdálenost se již znovu nepočítá, ale načte se z této paměti. Protože záporná délka trasy není příliš intuitivní, musím pomocí `transformMaximizationToRealCost(maxCost)` překlopit znaménko, aby graf vypadal jako pro minimalizační problém.

6.2.4.4 Informace o problému

Na závěr ještě sváži problém s identifikátorem, pod kterým bude v aplikaci vystupovat. Musel jsem tedy upravit `services/classResolver.js` a `store/_enums.js`, kde jsou zároveň uvedeny základní informace o problému, které potřebuje aplikace znát. Aby měli studenti přístup k informacím o problému, je potřeba ho řádně popsat do sekce v nápovědě. Nejdůležitější je popis

problému, aby studenti věděli, co vlastně řeší. Dále je nutné uvést způsob výpočtu fitness funkce a formát vstupních instancí.

6.3 Generátor instancí

Požaduje se také vytvoření generátoru instancí problému. `ETSPGenerator.js` přidám tedy k ostatním generátorům. Generátory jsou jednoduché třídy s jedinou metodou `generate()`, která vygeneruje instanci podle zadaných parametrů. Jako parametry generátoru jsem zvolil:

Počet měst Počet dvojic souřadnic, které se mají vygenerovat.

Rozsah osy X Souřadnice x se nachází mezi nulou a touto hodnotou.

Rozsah osy Y Souřadnice y se nachází mezi nulou a touto hodnotou.

Vzhledem k tomu, že euklidovské vzdálenosti jsou většinou desetinná čísla, rozhodl jsem se neomezovat souřadnice na celá čísla. Díky nastavování různých rozsahů os mohu generovat různě „obtížné“ instance. Generátor tedy vygeneruje soubor v odpovídajícím formátu 6.2.4.1 a náhodně vybírá souřadnice ze zadaných rozsahů.

Pro zadávání parametrů přes uživatelské rozhraní musím upravit komponentu `Generator.vue` včetně patřičných kontrol parametrů. Posledním úkonem je přidání generátoru do `GeneratorWorker.js`, který se stará o to, aby generování probíhalo v samostatném vlákně a nedošlo tak k zamrznutí UI.

Kapitola 7

Návrh a implementace genetického algoritmu

7.1 Hlavní smyčka

Hlavní logika genetického algoritmu se nachází ve třídě `GeneticSolver` a její metodě `solve(problem, params)`. Nejdříve se provede nastavení parametrů dle vstupních hodnot. Pro selekci a křížení je zde použit návrhový vzor Strategy [25]. Tento vzor se zde přímo nabízí, neboť se používá, když chceme definovat rodinu algoritmů (způsoby selekce a křížení), zapouzdřit je a umožnit jejich výměnu v kontextu (smyčka GA). Nicméně jeho potenciál zde není zcela využit, protože parametry se nastavují pouze na začátku a v průběhu výpočtu se už nemění. Pokud bychom však chtěli tuto funkcionalitu, je velmi jednoduché jí dosáhnout.

Vytvoření počáteční populace (`initGeneration(populationSize)`) je prvním krokem GA. Necháme si tedy vstupním problémem 6.2 vygenerovat počet jedinců odpovídající velikosti populace. Počáteční generaci následně vyhodnotíme.

Vyhodnocení generace (`evaluateGeneration(generation)`) spočívá v přiřazení fitness jedincům 7.2. Toto opět obstarává daný problém. Kvůli některým selekčním mechanismům a získání informací o fitness se generace seřadí vzestupně dle fitness. Dále získám průměr, nejlepší, nejhorší a medián fitness. Tyto hodnoty jsou po transformaci na skutečnou hodnotu optimalizačního kritéria, kterou zajišťuje problém, poslány do grafu 7.6, který pomocí těchto hodnot monitoruje průběh GA. Poslední věcí ohledně vyhodnocení generace je změna nejlepšího řešení, pokud bylo nalezeno lepší.

Poté se pro zadaný počet generací provádí následující postup. Selekcí se vyberou potencionální rodiče, jejichž počet odpovídá velikosti populace po odečtení elitních jedinců. Z potencionálních rodičů se tvoří nová generace. Nejprve se na základě pravděpodobnosti křížení rozhodne, zda potencionální rodič vstoupí do křížení. Rodiče, kteří vstoupili do křížení, jsou napárováni.

Každý pár vytvoří dva potomky dle mechanismu křížení. Pokud jedinec nemá partnera, nebo nevstoupil do křížení, je přidán (jako kopie) do nové generace. Nová generace poté provede mutaci všech jedinců s danou pravděpodobností mutace. Nová generace nahradí tu starou tak, že se spojí s elitními jedinci. Tato generace je opět vyhodnocena.

Výsledkem metody `solve(problem, params)` je konfigurace s nejlepší nalezenou hodnotou optimalizačního kritéria.

7.2 Jedinec

Jedinec je implementován jako abstraktní třída `Individual` a má dvě datové položky fitness a rank. Fitness jedincům přiděluje problém (viz 6.2) pomocí `evaluateMaximizationCost(configuration)`, kde `configuration` bude jedinec. Pro správnou funkčnost přidělování fitness musí konkrétní jedinci implementovat metodu pro získání čisté konfigurace. Tato metoda musí mít shodný název jako metoda pro příslušný konfigurační typ (viz 6.2.1). Rank jedincům přidělují některé mechanismy selekce 7.3. Implementoval jsem binárního a permutačního jedince.

7.2.1 Abstraktní metody

getGenotype() Vrací genotyp/konfiguraci jedince.

copy() Zkopíruje jedince bez přiděleného ranku a fitness.

mutate(mutationProb) Provede mutaci (viz 7.2.2) jedince s danou pravděpodobností.

7.2.2 Mutace

Mutace je důležitou metodou jedince. Pro abstraktního jedince je tato metoda abstraktní, ale konkrétní jedinci již musí metodu implementovat.

Mutace je implementována tak, jak je popsáno výše v 1.7. U mutace permutace je navíc zajištěno, aby výměna hodnoty proběhla s jinou než aktuální pozicí.

Přemýšlel jsem také o implementování postupně se snižující mutace, což je pokročilejší, ale poměrně běžná technika, inspirovaná teplotou u simulovaného ochlazování. Vedoucí si však tuto možnost nepřál, protože by byla pro studenty příliš komplikovaná a přinesla by více problémů než přínosu.

Také jsem zvažoval, že nabídnu uživatelům více druhů mutací. Nicméně v MI-PAA se přednáší pouze o jedné mutaci pro binární kódování a jedné pro permutační, proto jsem tuto možnost prozatím zavrhl.

7.3 Selekce

Fáze selekce (1.5) je implementována tak, že se v této fázi vybere vždy takový počet jedinců, který odpovídá velikosti populace po odečtení elitních jedinců. Tímto zajistím, abych některé jedince netvořil zbytečně a poté je nezahazoval.

7.3.1 Turnaj

Turnajový výběr (1.5.1) je poměrně oblíbený a jednoduchý způsob selekce. Jeho hlavní výhodu vidím v tom, že z implementačního hlediska se nemusím příliš zabývat konkrétní hodnotou fitness. Jde pouze o to, aby lepší jedinec měl vždy lepší fitness než horší, ale není vůbec důležité o kolik.

Hlavní nevýhodou turnaje je, že selekční tlak se řídí velikostí turnaje, a to musí být celé číslo. U větších turnajů toto není problém, protože rozdíl mezi turnajem velikosti 10 a 11 není zase tak skokový. Problém nastává u malých turnajů. Především u turnaje velikosti 1, kde se jedná pouze o náhodný výběr jedince podle rovnoměrného rozdělení, a turnaje velikosti 2, kde už náhodně (rovnoměrně) vyberu dva jedince a ten lepší z nich je vybrán do další fáze. To znamená, že nejhorší jedinec může být vybrán pouze v případě, že bude soupeřit se sebou samým.

Tento problém se však dá obejít jednoduchým trikem. Trik spočívá v tom, že ve fázi selekce obvykle probíhá více turnajů. Pokud by mi vyhovovala velikost turnaje mezi 3 a 4 mohu přimět algoritmus, aby polovinu turnajů udělal s velikostí 3 a druhou polovinu s velikostí 4. Moje aplikace je výuková a měla by se snadno používat, proto nechci používat nějaké složité mechanismy, které toto nastavení uživateli umožní. Jednoduše tedy umožním zadávat velikost turnaje jako desetinné číslo. Číslo před desetinnou čárkou značí velikost menšího turnaje. Číslo za desetinnou čárkou udává procentuální poměr větších turnajů. Uvědomte si, že větší a menší turnaje se liší pouze o 1 a v případě, že velikost turnaje je celé číslo, se provádějí turnaje pouze této velikosti.

Nastavení hodnot, kde vstupními parametry jsou $velikostTurnaje$, který udává velikost turnaje jako desetinné číslo, a $pocetJedincu$, který udává kolik jedinců se má vybrat do další fáze, se provede takto:

Velikost menšího turnaje $= \lfloor velikostTurnaje \rfloor$

Velikost většího turnaje $= \lceil velikostTurnaje \rceil$

Poměr větších turnajů $= velikostTurnaje - velikostMensihoTurnaje$

Počet malých turnajů $\doteq pocetJedincu - (pomerVetsichTurnaju \cdot pocetJedincu)$

Počet velkých turnajů $= pocetJedincu - pocetMalychTurnaju$

Jako příklad uvedu velikost turnaje 3,63 a potřebuji vybrat 20 jedinců.

Velikost menšího turnaje 3

Velikost většího turnaje 4

Poměr větších turnajů 63 %

Počet malých turnajů Vyjde 7,4. Zaokrouhlím tedy na 7.

Počet velkých turnajů Vyjde 12,6. Zaokrouhlím tedy na 13.

Pro výběr 20 jedinců tedy použiji 7krát turnaj o velikosti 3 a 13krát turnaj o velikosti 4.

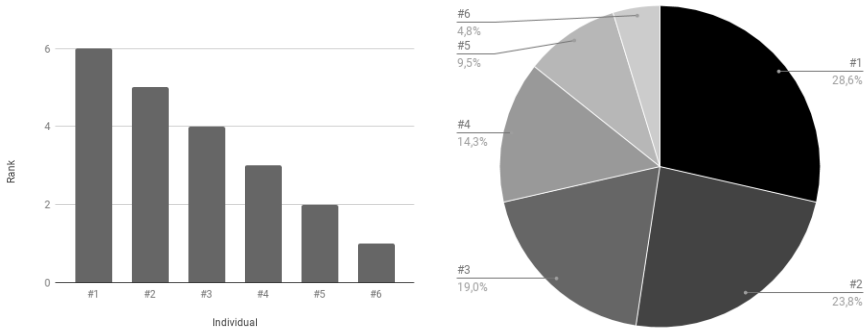
Všimněte si, že nyní používám místo jednoho celočíselného parametru celočíselných parametrů více, ale ty jsou před uživatelem skryty a nastavují se elegantně pomocí jednoho neceločíselného parametru. Tímto způsobem jsem schopen zajistit plynulejší přechod pro nastavování velikosti turnaje. Nicméně stále používám ve finále celá čísla (jinak tomu u turnaje ani nejde) a dochází k zaokrouhlování, takže skokovosti jsem se zcela nezbavil, ale je nyní jemnější. Z výše uvedeného je zřejmé, že uživatel nemůže předpokládat, že nepatrnou změnou parametru dojde i ke změně ve velikosti turnajů, ke skoku buď dojde, nebo ne. Další poznámkou je, že skoky jsou menší s rostoucí populací, neboli počtem turnajů, proto je toto vylepšení přínosné především pro větší populace.

7.3.2 Ruleta

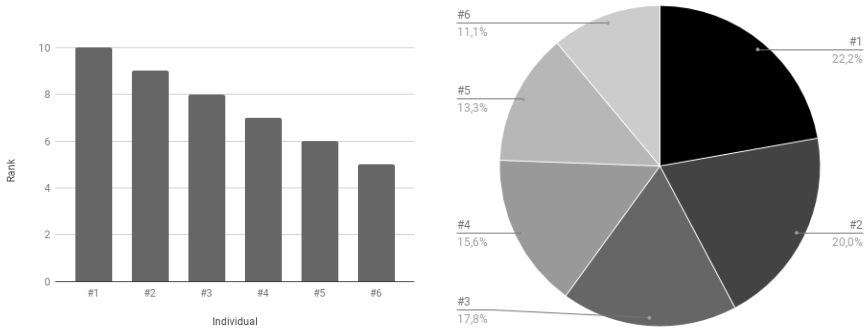
Dalším typický způsobem výběru pro GA je ruletový výběr (1.5.2). Hlavní výhodu tohoto typu výběru vidím v tom, že se dá lépe odhadnout, jak výběr dopadne. Pravděpodobnost výběru jedince prostě odpovídá podílu jedince na „ruletovém kole“. Naopak hlavní nevýhodou je, dle mého názoru, že musíme nějakým způsobem nastavit podíly jednotlivých jedinců v ruletě. Ruleta je také na rozdíl od turnaje poměrně silně svázaná s hodnotou fitness, což není vždy žádoucí.

Vzhledem k tomu, že můj GA by měl být co nejobecnější a měl by si umět poradit s nejrůznějšími problémy, které bude možné doplňovat, nemůže se spoléhat na nějaký zaručený formát fitness funkce. Již v době implementace ruletového výběru existovaly v této aplikaci problémy, které využívaly například zápornou fitness pro nevalidní řešení. Z tohoto důvodu jsem vypustil nejjednodušší variantu nastavit podíl jedince na ruletě přímo podle jeho fitness a rozhodl jsem se implementovat ranking a lineární škálování.

Vlastní ruleta je implementována tak, že všem jedincům aktuální generace se přidělí nějaký podíl (číslo). Tyto podíly se pak sečtou a vygeneruje se pseudonáhodné číslo z rozmezí 0 až součet podílů. Od tohoto čísla se postupně odečítají podíly jedinců, a jakmile číslo klesne pod 0, je vybrán poslední odečítaný jedinec. Pro lepší efektivitu se odečítají postupně jedinci od nejlepšího



Obrázek 7.1: Ukázka rankingu s parametrem 1



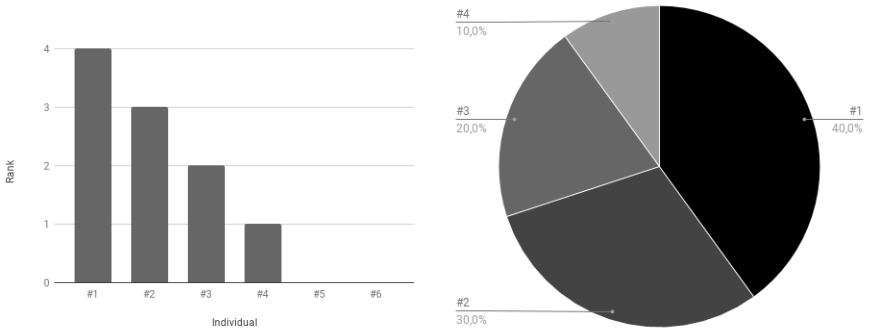
Obrázek 7.2: Ukázka rankingu s parametrem 5

s největším podílem po nejhoršího. Tímto se minimalizuje počet odčítání potřebných k výběru jedince.

7.3.2.1 Ranking

Snažil jsem se vymyslet, jak uživateli umožnit co nejjednodušeji řídit selekční tlak u rankingu (1.5.2.2). Můj nápad byl, že ranking se bude nastavovat parametrem podílu pro nejhoršího jedince. Čím vyšší hodnota tohoto parametru bude, tím vyšší dostane každý jedinec základní podíl na ruletě. To znamená, že jedinci budou mít pravděpodobnost výběru s menšími rozdíly (obr. 7.2). Pokud tedy nastavím tento parametr na 1, dopadne to jako v případě rankingu bez škálování (obr. 7.1). Pokud nastavím parametr na stejnou hodnotu jako počet jedinců v populaci, bude mít nejhorší jedinec pravděpodobnost výběru pouze dvakrát horší než ten nejlepší. V případě záporného parametru naopak vyřadím špatné jedince z výběru úplně a selekční tlak zvýším (obr. 7.3). Změna hodnoty parametru v podstatě znamená posouvání osy X nahoru nebo dolu v grafu podílů.

Dalším problémem k zamyšlení bylo, jak si poradit s jedinci se stejnou



Obrázek 7.3: Ukázka rankingu s parametrem -1

Tabulka 7.1: Ukázka převodu fitness na rank pomocí různých verzí rankingu

Fitness	4	4	12	15	15	15
Původní Rank	1	2	3	4	5	6
Rank A	1	1	2	3	3	3
Rank B1	1	1	3	4	4	4
Rank B2	2	2	3	6	6	6
Rank B3	1,5	1,5	3	5	5	5

fitness. Původní implementace byla taková, že pořadí, a tedy i rank jedince, je dáno umístěním v setřízeném poli, které je setřízeno pomocí řadícího algoritmu. Tudíž měl každý jedinec různý rank bez ohledu na to, že mohl mít stejnou fitness. Původně jsme se s vedoucím práce shodli na tom, že bude lepší a názornější, aby jedinci se stejnou fitness měli stejný rank. Nabízelo se několik možností, které budu demonstrovat na příkladu populace s fitness 4; 4; 12; 15; 15; 15 a rankingem bez škálování. Výsledky jednotlivých možností převodu fitness na rank můžete vidět v tabulce 7.1 včetně původní implenetace.

A Rank budu zvyšovat o 1, až když se zvýší fitness.

B Rank budu napočítávat dle počtu jedinců. V případě, že bude více jedinců se stejnou fitness, všichni dostanou rank, který by měl dostat některý z nich, jako kdyby měli všichni jedinci fitness různou.

   B1 Udělím jedinci nejnižší možný rank.

   B2 Udělím jedinci nejvyšší možný rank.

   B3 Udělím jedinci průměrný rank.

Možnost A se mi příliš nezamlouvá. Sice rank pěkně plynule poroste, ale nemám žádnou záruku, kam se rank vyšplhá. Jsem omezen pouze shora počtem

jedinců v populaci. Navíc tento způsob ovlivňuje rank jedinců, kteří mají jedinečnou hodnotu fitness. Pro tuto možnost se selekční tlak špatně řídí pomocí mého parametru, protože nevím, jaký rank má nejlepší jedinec. Rank může být v každé generaci jiný a špatně se bude nastavovat poměr mezi nejlepším a nejhorším jedincem.

Možnosti B už dávají větší smysl v zachování poměrů napříč populací. Tyto možnosti na rozdíl od A zohledňují počet stejných jedinců. Také víme, že rank nejlepšího jedince bude blízce odpovídat počtu jedinců v generaci a rank nejhoršího jedince bude blízký zadanému parametru. Možnosti B1 a B2 používají extrémy, tudíž mírně vychylují ranking pro nadržování slabším B1, nebo silnějším B2. Nejschůdnější variantou je, dle mého názoru, možnost B3, kde k tomuto vychýlení nedochází.

Po přehodnocení vedoucím práce a docentem Janem Smidtem jsme se nakonec shodli, že bude nejlepší řídit selekční tlak rankingu stejně jako u lineárního škálování. Bude sice obsahovat dva parametry, takže pro studenty bude obtížnější je správně nastavit, ale jakmile si na ně zvyknou, budou moci porovnávat výsledky rankingu a lineárního škálování se stejnými parametry. Podle docenta Schmidta má být ranking úplným uspořádáním, i když podle [9] stačí, aby přidělování ranku byla nerostoucí funkce, pokud přidělujeme od nejlepšího. Problém se stejnými hodnotami fitness tedy odpadá a ponechám původní implementaci přidělování ranku, kde každý jedinec má jiný rank. Tato varianta je v tomto případě lepší, protože v kombinaci s lineárním škálování, vím, že ruletové kolo bude vypadat stejně pro každou generaci.

7.3.2.2 Lineární škálování

Dalším způsobem rozdělení podílu, který na rozdíl od rankingu zachovává rozdíly mezi fitness je lineární škálování (1.5.2.1). Jak už z názvu vyplývá, hodnoty fitness funkce jedinců se pomocí lineární funkce přeškálují, tak aby spadaly do zadaného rozmezí a zároveň zachovávaly relativní rozdíly mezi hodnotami fitness.

Poměrně dlouho jsem si lámal hlavu nad tím, jaký UI prvek použiji pro nastavování minima a maxima. Od standardního textového vstupu jsem se dostal až k různým posuvníkům minima a maxima (viz 7.4). Podobně jako u rankingu jsem přemýšlel nad jedním parametrem, který jsem nakonec zavrhl. Výběrem UI prvku jsem chtěl studentům naznačit, že nastavené minimum a maximum spolu silně souvisí a nezáleží na konkrétních hodnotách, ale spíše na jejich poměru.

Posuvník se zdál jako dobrá volba, problém byl však v tom, že vyžaduje omezit rozsah hodnot. Po poradě s vedoucím jsme dospěli k závěru, že bych neměl studenty omezovat v zadaném maximu. Ať si klidně experimentují i za cenu toho, že jim GA přestane fungovat.

Neomezeným maximem tedy varianta posuvníku padla. Proto abych zdůraznil souvislost parametrů, jsem se rozhodl, že je na rozdíl od ostatních umís-

Obrázek 7.4: Ukázka řešení GUI prvku pro škálování (vlevo) a posuvník (vpravo)

tím do jednoho řádku do tzv. `input-group`, což je stále poměrně standardní ovládací prvek (viz 7.4). Nakonec jsem omezil minimální hodnotu minima na 0 a minimální hodnotu maxima na 1.

7.4 Křížení

Implementoval jsem šest druhů křížení, které uvádím v 1.6. Tři pro binární kódování a tři pro permutační. Důležité je, že binární křížení pracuje pouze s binárními jedinci, které také vytváří, a permutační pracuje zase pouze s permutačními. Musel jsem tedy zajistit, aby uživateli byly nabídnuty vždy správné druhy křížení (viz 7.5).

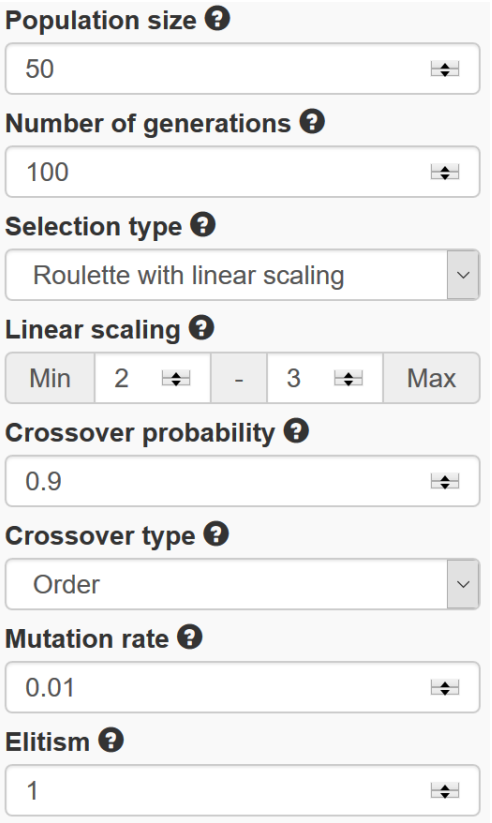
7.5 Parametry

Nyní se přesouvám od vnitřní implementace GA k uživatelskému rozhraní. Konkrétně k zadávání parametrů uživatelem. Veškerá validace, ošetření vstupů a interakce s uživatelem jsou zajištěny komponentou `methodParams/Genetic.vue`, jejíž náhled můžete vidět na obrázku 7.5. Pořadí parametrů je dáno logickým pořadím, jak jsou parametry používány uvnitř GA.

Všechny parametry obsahují nápovědu a validaci s označením a popisem neplatného vstupu. Všechny parametry jsou nastaveny na výchozí hodnotu, která je poměrně rozumná, ale zároveň nemusí být optimální pro danou instanci problému. Ideální hodnoty parametrů se vztahují k instanci a problému, musel bych tedy pro každý problém a ukázkovou instanci najít ideální hodnoty parametrů, které by byly pravděpodobně různé. Problémy by tímto způsobem nesdílely nastavení parametrů GA, což není příliš logické. Druhým důvodem je, že hledání ideálních hodnot parametrů je úkolem studentů. Níže rozebírám omezení jednotlivých parametrů, jejich význam je popsán výše. Na rozdíl od ostatních algoritmů GA nepoužívá všechny parametry pro každý běh, protože některé jsou specifické pouze pro určitý typ selekce, proto bylo nutné odfiltrovat parametry, které se nevztahují ke konkrétnímu běhu, z výsledků běhu.

7.5.1 Population size

Udává velikost populace. Je to přirozené číslo s minimem 1.



Obrázek 7.5: Ukázka GUI pro parametry GA

7.5.2 Number of generations

Udává počet generací. Je to přirozené číslo s minimem 1.

7.5.3 Selection type

Vybírá selekční mechanismus. Každá možnost má navíc vlastní parametr pro řízení selekčního tlaku.

- Roulette with linear scaling

  Linear scaling pomocí minima a maxima nastavuje lineární přeškálování fitness. Minimum je desetinné číslo s minimální hodnotou 0. Maximum je desetinné číslo s minimální hodnotou 1. Maximum musí být větší nebo rovno minimu.

- Roulette with ranking

  Rank scale pomocí minima a maxima nastavuje lineární přeškálování ranku. Minimum je desetinné číslo s minimální hodnotou 0. Ma-

ximum je desetinné číslo s minimální hodnotou 1. Maximum musí být větší nebo rovno minimu.

- Tournament

  Tournament size udává velikost turnaje. Je to desetinné číslo s minimální hodnotou 1 a maximální hodnotou, která odpovídá velikosti populace.

7.5.4 Crossover probability

Udává pravděpodobnost, že vybraný jedinec vstoupí do křížení. Je to desetinné číslo od 0 do 1.

7.5.5 Crossover type

Vybírá mechanismus křížení s následujícími možnostmi:

a) Je vybrán binární problém.

- One-point
- Two-point
- Uniform

b) Je vybrán permutační problém.

- Order
- Partially matched
- Cycle

Musel jsem zajistit, aby při přepnutí problému z binárního na permutační a naopak se přepnul i typ křížení na vyhovující typ.

7.5.6 Mutation rate

Udává pravděpodobnost mutace. Je to desetinné číslo od 0 do 1.

7.5.7 Elitism

Udává počet nejzdatnějších jedinců, kteří se přenesou do nové generace. Je to celé číslo s minimem 0 a maximem, které odpovídá velikosti populace.

7.6 Graf průběhu

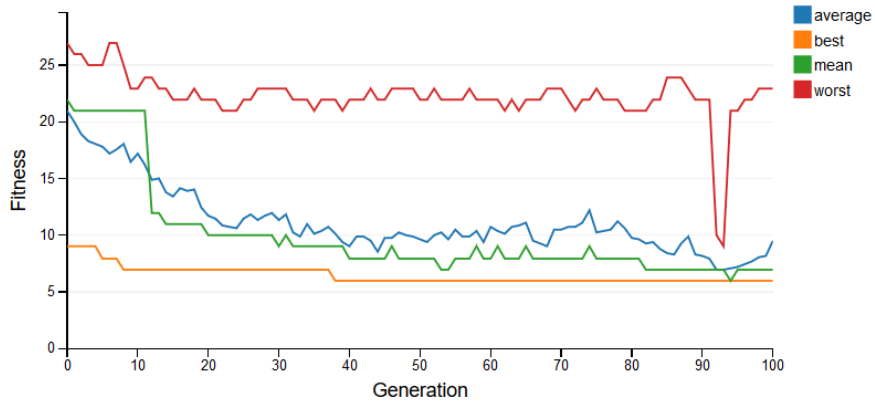
Asi nejdůležitějším prvkem a důvodem vzniku celé této aplikace je graf, který vizualizuje průběh GA. Tento graf byl obsažen i v původním appletu 2.1.2. Appletový graf se mi příliš nelíbil, protože byl špatně viditelný a jeho průběžné načítání vypadalo podivuhodně (graf se postupně smršťoval). K vytvoření grafu jsem použil knihovnu dc.js a základ grafu, který vytvořil kolega Veselý pro Tabu prohledávání a simulované ochlazování. Na rozdíl od těchto dvou algoritmů, které používají jednu hodnotu, je totiž graf pro GA složitější, protože používá hodnoty více typů.

Rozložení os grafu je velmi intuitivní. Osa X čísluje generace, osa Y potom označuje hodnotu fitness. Poměrně dlouho jsem přemýšlel, jaké hodnoty nanášet do grafu, abych co nejlépe vystihl podstatu jednotlivých generací. Věděl jsem, že tam chci nejlepší a nehorší fitness. Dále jsem váhal, zda použít průměrnou fitness či medián. Viděl jsem v praxi použité obě varianty. Nakonec jsem se rozhodl, že použiji vše a dám uživateli na výběr, co chce vidět. Na rozdíl od původního appletu používám spojnicový graf, abych zdůraznil tendenci algoritmu, zároveň je vše lépe vidět. Pozorný čtenář si nejspíše všiml, že jsem vynechal zobrazení „hustoty populace“, které bylo v původním appletu. Jak jsem již uvedl v 2.1.2, nevidím v tom velký přínos, protože se zde zřídka děje něco zajímavého, a průměr společně s mediánem je většinou schopen strukturu populace vyjádřit. Zároveň by tento šum měl velmi neblahý vliv na porovnávání různých běhů.

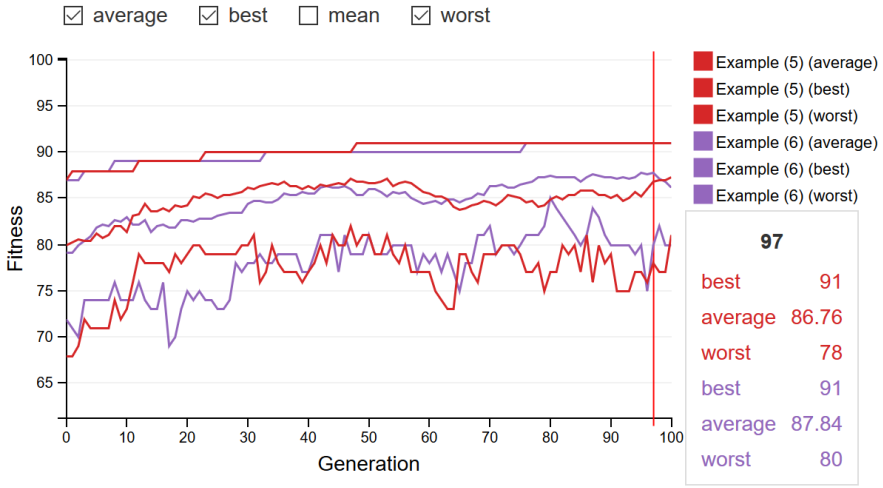
V kontextu mojí aplikace hovoříme vlastně o dvou grafech. Prvním z nich je graf, který zobrazuje výsledky aktuálně běžícího algoritmu. Tento graf slouží zároveň k informování uživatele o tom, za jak dlouho algoritmus dokončí práci, protože má předem daný počet generací. Vzhledem k tomu, že jsem se snažil napsat GA co nejefektivněji, tento graf není příliš dlouho vidět, ale můžete si ho prohlédnout na obrázku 7.6. Jakmile se totiž výpočet dokončí, aplikace se přepne do srovnávacího módu.

Graf ve srovnávacím módu vypadá trochu jinak, jak můžete pozorovat na obrázku 7.7. Hlavním rozdílem je, že všechny čáry náležící k jednomu běhu mají stejnou barvu. To je velice důležité pro to, abychom od sebe dokázali odlišit jednotlivé běhy. Stejnobarevné čáry nepředstavují problém, protože kromě průměru a mediánu fitness se nikdy nekříží. Předpokládal jsem, že uživatel bude používat buď průměr, nebo medián v závislosti na situaci. S tím souvisí možnost vybrat si hodnoty, které chce uživatel vidět. Jako výchozí nastavení jsou zatrženy nejlepší, průměrná a nejhorší fitness. Při testování nástroje jsem totiž přišel na to, že někdy se pro určení konvergence lépe hodí průměr a jindy medián. Medián se hodí především v případě, že je velký rozdíl mezi validním a nevalidním řešením, které pak značně vychyluje průměr. Srovnávací graf navíc disponuje zobrazením přesných hodnot po najetí na graf.

Srovnávací graf společně s historií tvoří skvělý nástroj pro zjišťování vlivu změny parametrů na průběh GA. Osobně tuto část považuji za největší přínos



Obrázek 7.6: Graf živého průběhu GA



Obrázek 7.7: Srovnávací graf GA

této aplikace.

7.7 Nápověda

Kromě tooltipů přímo u parametrů jsem sepsal popis mojí implementace GA, kde uvádím, jak konkrétně můj algoritmus pracuje. Tento popis je na samostatné stránce v nápovědě aplikace s vysvětlením všech nastavitelných parametrů včetně obrázků.

Kapitola 8

Závěrečné testování

Závěrečné testování jsme pojali dvěma způsoby. Poslali jsme odkaz na poslední verzi aplikace cvičícím předmětu MI-PAA, aby nám dali zpětnou vazbu. Také jsme si pozvali testery na uživatelské testování dle scénářů. Uživatelské testování jsme prováděli s kolegy s tím, že si každý otestuje svou část. Testování opět probíhalo v prostorách FIT ČVUT na notebooku. Testování se zúčastnilo 7 testerů. Všichni byli bývalí studenti MI-PAA a někteří se účastnili již testování prototypu. Testování se zaměřovalo na nové prvky, které nebyly v prototypu obsaženy a na „reálnou“ práci s aplikací.

8.1 Otázky na testery

Před, během a po testování jsme se testerů ptali na tyto otázky. Otázky jsou společnou částí tohoto testování. Já si navíc přidal otázku ohledně práce s grafem GA.

8.1.1 Předtestovací otázky

1. Absolvovali jste úspěšně předmět MI-PAA?
2. Jaký algoritmus jste si vybrali pro svou závěrečnou práci v předmětu MI-PAA?
3. Znáte i zbylé dva algoritmy?
4. Znáte bývalé applety, které sloužily k demonstraci běhu iterativních algoritmů?
5. Zúčastnili jste se testování prototypu naší aplikace?
6. Prohlédněte si aplikaci a sdělte nám, jak byste aplikaci začali používat.

8.1.2 Otázka GA

7. Jak se vám pracovalo s grafem u GA?

8.1.3 Otázky na konec

8. Co se vám na aplikaci líbilo?
9. Co se vám nelíbilo nebo co vám přišlo složité / neintuitivní?
10. Co si o této aplikaci myslíte v porovnání s původními applety?
11. Myslíte si, že by vám tato aplikace pomohla pochopit jednotlivé algoritmy a vliv parametrů na jejich běh?

8.1.4 Odpovědi na otázky

- „Eva“ Nguyen
  1. Absolvovali jste úspěšně předmět MI-PAA?
     - Ano.
  2. Jaký algoritmus jste si vybrali pro svou závěrečnou práci v předmětu MI-PAA?
     - Simulované ochlazování.
  3. Znáte i zbylé dva algoritmy?
     - Pouze jsem o nich slyšela.
  4. Znáte bývalé applety, které sloužily k demonstraci běhu iterativních algoritmů?
     - Neznám.
  5. Zúčastnili jste se testování prototypu naší aplikace?
     - Ano.
  6. Prohlédněte si aplikaci a sdělte nám, jak byste aplikaci začali používat.
     - Začala bych proklikáním záložek a poté bych spustila běh jednoho z nich.
  7. Jak se vám pracovalo s grafem u GA?
     - Hodnoty po najetí na graf jsou nepřehledné a špatně se v nich orientuji, zvolila bych spíše jeden sloupec pro každý typ hodnoty.
  8. Co se vám na aplikaci líbilo?
     - Pěkné UI, grafová reprezentace.

9. Co se vám nelíbilo nebo co vám přišlo složité / neintuitivní?
   - Instance se špatným formátem je těžké poznat. U stejně pojmenovaných instancí v historii by měla být stejná čísla jako v legendě.
10. Co si o této aplikaci myslíte v porovnání s původními applety?
    - Původní applety jsem neviděla.
11. Myslíte si, že by vám tato aplikace pomohla pochopit jednotlivé algoritmy a vliv parametrů na jejich běh?
    - Spíše ano.

- Marek Dostál

1. Absolvovali jste úspěšně předmět MI-PAA?
   - Ano.
2. Jaký algoritmus jste si vybrali pro svou závěrečnou práci v předmětu MI-PAA?
   - Simulované ochlazování.
3. Znáte i zbylé dva algoritmy?
   - Znám.
4. Znáte bývalé applety, které sloužily k demonstraci běhu iterativních algoritmů?
   - Znám.
5. Zúčastnili jste se testování prototypu naší aplikace?
   - Ano.
6. Prohlédněte si aplikaci a sdělte nám, jak byste aplikaci začali používat.
   - Spustil bych Start.
7. Jak se vám pracovalo s grafem u GA?
   - Je složitější než grafy pro ostatní algoritmy. Ze začátku se nezdá být tolik přehledný. Jakmile si však na něj člověk zvykne a uvědomí si, co hodnoty znamenají, je to jednoduché.
8. Co se vám na aplikaci líbilo?
   - Vzhled, validace. Neseká se.
9. Co se vám nelíbilo nebo co vám přišlo složité / neintuitivní?
   - Nevím, co znamená výsledné řešení.
10. Co si o této aplikaci myslíte v porovnání s původními applety?
    - Lepší, snadno spustitelné.

11. Myslíte si, že by vám tato aplikace pomohla pochopit jednotlivé algoritmy a vliv parametrů na jejich běh?
    - Spíše ano.

- Petra Krnáčová

1. Absolvovali jste úspěšně předmět MI-PAA?
   - Ano.
2. Jaký algoritmus jste si vybrali pro svou závěrečnou práci v předmětu MI-PAA?
   - Simulované ochlazování.
3. Znáte i zbylé dva algoritmy?
   - Ano, ale už si je moc nepamatuji.
4. Znáte bývalé applety, které sloužily k demonstraci běhu iterativních algoritmů?
   - Ano.
5. Zúčastnili jste se testování prototypu naší aplikace?
   - Ne.
6. Prohlédněte si aplikaci a sdělte nám, jak byste aplikaci začali používat.
   - Prohlédla bych si tooltipy u jednotlivých parametrů, pak je nastavila a spustila běh.
7. Jak se vám pracovalo s grafem u GA?
   - Na graf je třeba si zvyknout, ale potom je práce s ním jednoduchá.
8. Co se vám na aplikaci líbilo?
   - Přehledné rozhraní. Spousta vysvětlivek, validace.
9. Co se vám nelíbilo nebo co vám přišlo složité / neintuitivní?
   - Ze začátku jsem si nevšimla historie, ale nemyslím si, že to je problém.
10. Co si o této aplikaci myslíte v porovnání s původními applety?
    - Přehlednější, novodobé a snadno spustitelné.
11. Myslíte si, že by vám tato aplikace pomohla pochopit jednotlivé algoritmy a vliv parametrů na jejich běh?
    - Ano, dobré na učení. Člověk si algoritmy může vyzkoušet, aniž by je musel programovat.

- Danny Dočekalů

1. Absolvovali jste úspěšně předmět MI-PAA?
   - Ano.
2. Jaký algoritmus jste si vybrali pro svou závěrečnou práci v předmětu MI-PAA?
   - Simulované ochlazování.
3. Znáte i zbylé dva algoritmy?
   - Z toho, co jsem se učil na zkoušku.
4. Znáte bývalé applety, které sloužily k demonstraci běhu iterativních algoritmů?
   - Ano, ale šly mi spatně spustit.
5. Zúčastnili jste se testování prototypu naší aplikace?
   - Ne.
6. Prohlédněte si aplikaci a sdělte nám, jak byste aplikaci začali používat.
   - Proklikal bych algoritmy a pak stiskl „Start“.
7. Jak se vám pracovalo s grafem u GA?
   - Graf je poměrně přehledný i přes to, že obsahuje mnoho informací.
8. Co se vám na aplikaci líbilo?
   - Graf. Porovnání instancí.
9. Co se vám nelíbilo nebo co vám přišlo složité / neintuitivní?
   - Práce s historií byla na začátku poněkud matoucí, časem jí však používám už bez problému.
10. Co si o této aplikaci myslíte v porovnání s původními applety?
    - Lehce spustitelné a funkční. Lépe se s tím pracuje.
11. Myslíte si, že by vám tato aplikace pomohla pochopit jednotlivé algoritmy a vliv parametrů na jejich běh?
    - Ano, i při řešení domácích úloh.

- Jan Nováček

1. Absolvovali jste úspěšně předmět MI-PAA?
   - Ano.
2. Jaký algoritmus jste si vybrali pro svou závěrečnou práci v předmětu MI-PAA?
   - Simulované ochlazování.

3. Znáte i zbylé dva algoritmy?
   - Ano.
4. Znáte bývalé applety, které sloužily k demonstraci běhu iterativních algoritmů?
   - Ne.
5. Zúčastnili jste se testování prototypu naší aplikace?
   - Ne.
6. Prohlédněte si aplikaci a sdělte nám, jak byste aplikaci začali používat.
   - Proklikal bych algoritmy a pak stiskl „Start“.
7. Jak se vám pracovalo s grafem u GA?
   - Práce s grafem je příjemná a pohodlná.
8. Co se vám na aplikaci líbilo?
   - Rozložení GUI. Interaktivní grafy.
9. Co se vám nelíbilo nebo co vám přišlo složité / neintuitivní?
   - Závorkování při stejných jménech. U Tabu prohledávání chyběl popis hodnoty po najetí na graf.
10. Co si o této aplikaci myslíte v porovnání s původními applety?
    - Applety jsem neviděl.
11. Myslíte si, že by vám tato aplikace pomohla pochopit jednotlivé algoritmy a vliv parametrů na jejich běh?
    - Ano, zároveň bych mohl ověřovat a ladit parametry pro domácí úkoly a semestrální práci v MI-PAA.

- Jindřich Máca

1. Absolvovali jste úspěšně předmět MI-PAA?
   - Ano.
2. Jaký algoritmus jste si vybrali pro svou závěrečnou práci v předmětu MI-PAA?
   - Genetický algoritmus.
3. Znáte i zbylé dva algoritmy?
   - Ano, teoreticky.
4. Znáte bývalé applety, které sloužily k demonstraci běhu iterativních algoritmů?
   - Nevěděl jsem, že existují.

5. Zúčastnili jste se testování prototypu naší aplikace?
   - Ne.
6. Prohlédněte si aplikaci a sdělte nám, jak byste aplikaci začali používat.
   - Postupně bych spustil všechny algoritmy.
7. Jak se vám pracovalo s grafem u GA?
   - Práce s grafem je pohodlná.
8. Co se vám na aplikaci líbilo?
   - Jednoduchost, přímočarost. Vše v jednom okně.
9. Co se vám nelíbilo nebo co vám přišlo složité / neintuitivní?
   - Problém pod parametry. Set u SA změnit na Auto.
10. Co si o této aplikaci myslíte v porovnání s původními applety?
    - Applety jsem neviděl.
11. Myslíte si, že by vám tato aplikace pomohla pochopit jednotlivé algoritmy a vliv parametrů na jejich běh?
    - Určitě.

- Jakub Koza

1. Absolvovali jste úspěšně předmět MI-PAA?
   - Ano.
2. Jaký algoritmus jste si vybrali pro svou závěrečnou práci v předmětu MI-PAA?
   - Genetický algoritmus.
3. Znáte i zbylé dva algoritmy?
   - Ano z MI-PAA.
4. Znáte bývalé applety, které sloužily k demonstraci běhu iterativních algoritmů?
   - Ano.
5. Zúčastnili jste se testování prototypu naší aplikace?
   - Ne.
6. Prohlédněte si aplikaci a sdělte nám, jak byste aplikaci začali používat.
   - Proklikal bych si to a spustil.
7. Jak se vám pracovalo s grafem u GA?
   - Práce s grafem je příjemná. Možná bych uvítal zoom, pokud bych se chtěl více zabývat konkrétním úsekem průběhu.

8. Co se vám na aplikaci líbilo?
   - Dynamičnost.
9. Co se vám nelíbilo nebo co vám přišlo složité / neintuitivní?
   - Problém pod parametry. Tlačítko na generátor.
10. Co si o této aplikaci myslíte v porovnání s původními applety?
    - Na rozdíl od appletů funguje.
11. Myslíte si, že by vám tato aplikace pomohla pochopit jednotlivé algoritmy a vliv parametrů na jejich běh?
    - Ano.

8.2 Testovací scénáře

Testovací scénář pro GA jsem udělal poměrně náročný a komplexní. Snažil jsem přimět testery, aby aplikaci používali podobným způsobem, jak si představuji práci studentů s aplikací na cvičeních. Zadání jsem udělal tak, aby nebylo návodné a zároveň jsem používal terminologii GA. Což se občas projevilo jako menší problém, protože někteří testeři se v této terminologii již neorientovali. Zaměřil jsem se především na testování vstupních parametrů a grafu průběhu. Sledoval jsem práci testerů s grafem a diskutoval s nimi, zda se jim dobře používá a je přehledný. Úkoly pro práci s grafem jsem vymyslel tak, abych si ověřil, že je graf dostatečně přehledný i pro porovnávání více instancí najednou.

8.2.1 Scénáře GA

1. Spusťte GA s výchozími parametry pro problém batohu.
2. Zjistěte, ve které generaci bylo poprvé dosaženo maximální fitness a ve které generaci došlo k případnému ustálení populace.
3. Upravte parametry lineárního škálování a zvolte jiný typ křížení. Spusťte.
4. Zvolte Euclidean TSP, změňte typ selekce, mutaci, a vypněte elitismus.
5. Vraťte se k problému batohu a spusťte výpočet se stejnými parametry, jako měl poslední běh pro batoh.
6. Ověřte, že jsou parametry shodné.
7. Zjistěte rozmezí fitness počáteční (nulté) generace pro každý běh batohu.
8. Porovnejte o kolik se změnila průměrná fitness na začátku a na konci běhu pro 2 různé běhy.

8.3 Závěr testování

Závěrečné testování dopadlo velmi dobře. Odhalil se nedostatek s výběrem instance při změně problému a rozhození checkboxů při scrollování historie. Testeři se báli kliknout na rozbalovací šipku v historii, protože směřovala přímo na koš. Nebylo také jasné, co znamenají výsledky na středním panelu. Zbytek byly pouze drobnosti. Nejen testeři, ale i cvičící byli s aplikací spokojeni. Posbírali jsme řadu nápadů, jak aplikaci ještě vylepšit. Ty jednodušší nápady jsme zapracovali ihned. Ty náročnější nápady a nápady, které jsou ke zvážení, uvádím v 9.3.

Kapitola 9

Finální aplikace

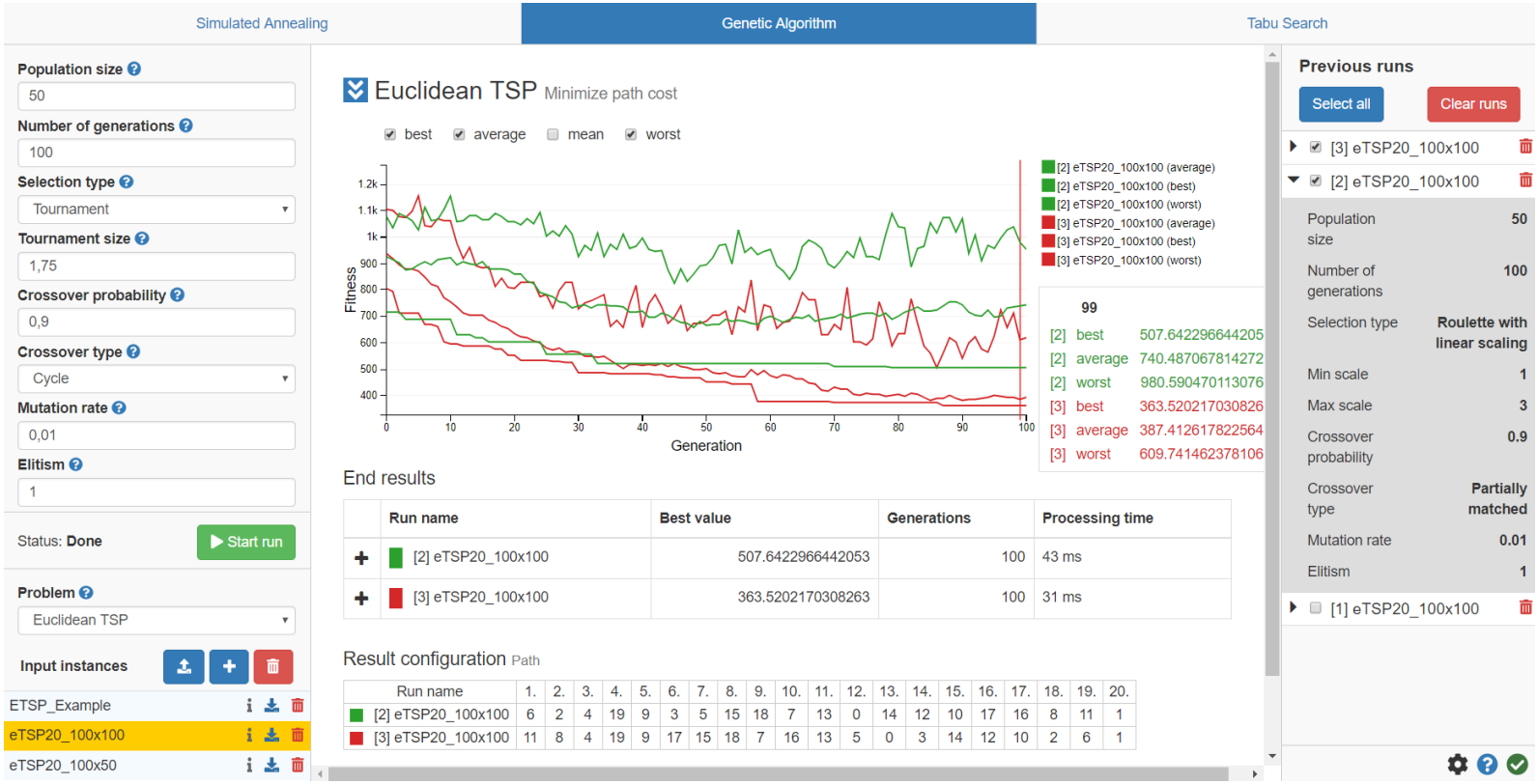
Od prvotního návrhu prošla aplikace mnoha změnami, proto zde uvádím její finální podobu 9.1. Nicméně až na pár drobných detailů zůstává v souladu s původním návrhem 4.1. Jak si můžete všimnout, došlo k přejmenování historie a zavedení pojmu běh („run“) do UI aplikace. Tyto běhy jsou navíc konzistentně uváděny se svým identifikátorem, který je jednoznačně odlišuje. Nemůže se tedy stát, aby se dva běhy jmenovaly stejně, i když běží nad stejnou instancí problému. V centrální části je popsáno, co výsledky znamenají. Ač jsou téměř všude umístěny tootipy s nápovědou, aplikace obsahuje samostatnou stránku s detailní nápovědou, pro uživatele, který je naprosto ztracený nebo chce aplikaci porozumět více do hloubky. Nápovědu můžeme najít pod otazníčkem vpravo dole nebo po zmáčknutí klávesy F1.

9.1 Nabízené problémy

Pro uživatele je připravena pestrá paleta problémů 9.1, které mohou pomocí algoritmů řešit. Problémy se liší jak typem optimalizace, tak typem konfigurace a aproximační třídou [24], [26].

Problém	**Typ optimalizace**	**Typ konfigurace**	**Aprox. třída**
MAX-SAT	Maximalizace	Bitové pole	APX-C
TSP	Minimalizace	Permutace	NPO-C
Batoh	Maximalizace	Bitové pole	FPTAS
MVC	Minimalizace	Bitové pole	APX-C
ETSP	Minimalizace	Permutace	PTAS

Tabulka 9.1: Přehled implementovaných problémů



Obrázek 9.1: Náhled finální aplikace

	M. Kluzáček	A. Kugler	J. Veselý
Návrh UI	✓	konzultant	konzultant
Implementace prototypu	konzultant	konzultant	✓
Testování prototypu	asistent	✓	asistent
Implementace spol. UI	konzultant	konzultant	✓
Simulované ochlazování	✓		
Genetický algoritmus		✓	
Tabu prohledávání			✓
Graf průběhu		GA modifikace	✓
MAX-SAT	generátor		problém
TSP	✓		
Problém batohu	✓	problém	problém
MVC	✓		
ETSP		✓	
Závěrečné testování	SA	GA	Tabu
Nasazení aplikace			✓

Tabulka 9.2: Přehled rozdělení práce na aplikaci

9.2 Rozdělení práce

Finální aplikace je produktem tří diplomových prací, proto zde přikládám přehledovou tabulku 9.2, kde je znázorněno, kdo se zabýval kterou částí systému.

✓ znamená, že tento člověk má za danou část zodpovědnost.

Konzultant znamená, že tento člověk zasahoval do vývoje svými názory.

Asistent znamená, že tento člověk asistoval u uživatelského testování.

Cokoli jiného znamená, že tento člověk vypracoval tuto konkrétní část.

9.3 Náměty na vylepšení

- Nahrávání parametrů z předchozích běhů.
- Přidat problémově specifické křížení a mutace.
- Odlišit jednolivé křivky grafu u GA.
- Možnost přiblížit si určitý úsek grafu.
- Posuvná legenda pro zobrazování velkého počtu výsledků.

- Reset parametrů na defaultní hodnotu tlačítkem a ne pouze obnovením stránky.
- Odladit responzivní design pro mobilní zařízení.
- Možnost upravovat vstupní instance přímo v aplikaci.
- Ušít vizualizaci výsledků na míru jednotlivým problémům.
- Vizualizovat jednotlivé kroky GA.

9.4 Nasazení aplikace z DVD

Tato kapitola byla převzata z [2].

Jediným požadavkem na prostředí, ve kterém lze sestavit aplikaci, je instalace softwaru *node.js*. Pro nasazení na server musíme v konfiguraci nastavit relativní nebo absolutní cestu k umístění souborů z hlediska url. Tato cesta je nastavena pomocí proměnné `build.assetsPublicPath` v konfiguračním souboru `/src/impl/config/index.js` na DVD. Například hodnota této proměnné pro testovací prostředí je: `'/dp-advanced-iterative-methods/'` pro následující url.

https://veselj43.github.io/dp-advanced-iterative-methods/

Kromě konfigurace už se jedná jen o sérii příkazů spuštěných v adresáři `impl`.

- Instalace (resp. stažení) závislostí `npm install` (lze spustit před úpravou konfigurace).
- Sestavení `npm run build`.
- Výsledné soubory jsou k nalezení v adresáři `impl/dist/`.
- Pro nasazení je zapotřebí zkopírovat obsah adresáře `impl/dist/` na server.

Jakmile jsou soubory sestavené s odpovídající konfigurací přístupné, aplikace je nasazena.

Závěr

Cílem této práce bylo vytvořit aplikaci pro demonstaci funkce genetických algoritmů, která bude použita pro výuku předmětu MI-PAA. Začal jsem analýzou předchůdce této aplikace a uživatelským průzkumem mezi studenty a učiteli. Díky tomu jsem mohl sestavit požadavky na tuto aplikaci. Během této fáze vedoucí práce rozhodl, že vznikne společná aplikace pro genetický algoritmus, simulované ochlazování a Tabu prohledávání. Společně s kolegy jsme tedy navrhli a vytvořili uživatelské rozhraní, které jsem otestoval pomocí uživatelského testování a podrobil heuristické analýze.

Poté jsem implementoval problém Euklidovského obchodního cestujícího, který splňuje rozhraní, na kterém jsme se s kolegy domluvili. Instance každého problému může uživatel vkládat jako soubory v daném formátu nebo si může tyto soubory vygenerovat pomocí generátoru, který jsem rovněž implementoval. Následně jsem narvhl a implementoval genetický algoritmus, který je velice modulární a široce parametrizovatelný. Implementace GA obsahuje tři druhy selekce, šest druhů křížení a spousty dalších parametrů. Algoritmus dokáže pracovat s binárním a permutačním kódováním. Největším přínosem této aplikace je vizualizace průběhu GA pomocí grafu fitness napříč generacemi. Uživatel může do grafu zanést více průběhů najednou a lehce je porovnat. Experimentování s nastavením parametrů a sledování průběhu by mělo studentům pomoci pochopit vliv parametrů na průběh algoritmu.

Výsledná aplikace, která vznikla v rámci tří diplomových prací, obsahuje tři optimalizační algoritmy a pět různých problémů. Aplikace je navržena tak, aby bylo možné snadno přidávat další problémy nebo dokonce algoritmy. Jedná se o webovou aplikaci, proto studentům stačí prohlížeč a nemusejí nic složitě instalovat. Aplikace má příjemné a snadno použitelné uživatelské rozhraní, jak se ukázalo v závěrečném testování, které prováděli nejen studenti, ale i cvičící předmětu MI-PAA.

Aplikace bude v brzké době umístěna na školní server, kde bude dostupná nejen studentům, ale i široké veřejnosti. Největší zkouškou pro tuto aplikaci bude však příští semestr, kdy se dostane do rukou novým studentům MI-PAA.

Já však věřím, že jim pomůže lépe se seznámit s touto problematikou.

Možností, jak navázat na tutu práci a rozšířit tak aplikaci, je mnoho, ať už jde o přidávání nových problémů nebo algoritmů. Jednou z možností je také vizualizace jednotlivých kroků GA. Genetický algoritmus je velmi obsáhlý a existuje spousta modifikací. Já se snažil naprogramovat genetický algorimus tak, aby předpokládal obecné problémy a byl v souladu s přednáškami a zároveň nebyl příliš komplikovaný. Existuje však mnoho dalších technik, které GA může použít. Nejvíce jsem zvažoval přidat další druhy mutace nebo křížení, kde jsou mnohé ušity na míru konkrétnímu problému. Tím by GA mohl dosáhnout mnohem lepších výsledků při použití pro vhodné problémy.

Literatura

[1] Kluzáček, M.: *Aplikace pro demonstraci funkce simulovaného ochlazování.* Diplomová práce, České vysoké učení technické v Praze, Fakulta informačních technologií, Praha, 2018.

[2] Veselý, J.: *Aplikace pro demonstraci funkce Tabu prohledávání.* Diplomová práce, České vysoké učení technické v Praze, Fakulta informačních technologií, Praha, 2018.

[3] Darwin, C.: *On the origin of species by means of natural selection, or, The preservation of favoured races in the struggle for life.* Londýn: John Murray, 1859.

[4] Schmidt, J.: Problémy a algoritmy, 9. přednáška: Simulovaná evoluce I. 2017, České vysoké učení technické v Praze, Fakulta informačních technologií, ZS 2017/2018.

[5] Fišer, P.; Schmidt, J.: Problems and Algorithms, 9th lecture: Simulated Evolution I. Genetic Algoritms. 2017, Czech technical university in Prague, Faculty of Information Technology, WS 2017/2018.

[6] Mitchell, M.: *An Introduction to Genetic Algorithms.* Cambridge, MA, USA: MIT Press, 1998, ISBN 0262631857.

[7] Whitley, D.: A genetic algorithm tutorial. *Statistics and Computing*, ročník 4, č. 2, Červen 1994: s. 65–85, ISSN 1573-1375.

[8] Baker, J. E.: Adaptive Selection Methods for Genetic Algorithms. In *Proceedings of the 1st International Conference on Genetic Algorithms*, Hillsdale, NJ, USA: L. Erlbaum Associates Inc., 1985, ISBN 0-8058-0426-9, s. 101–111.

[9] Goldberg, D. E.; Deb, K.: A Comparative Analysis of Selection Schemes Used in Genetic Algorithms. Elsevier, 1991, s. 69 – 93.

[10] Benhák, P.: *Vizualizace běhu genetických algoritmů*. Bakalářská práce, České vysoké učení technické v Praze, Fakulta elektrotechnická, Praha, Červen 2006.

[11] Benhák, P.: Vizualizace běhu genetických algoritmů [software]. [cit. 2017-6-17]. Dostupné z: `http://ddd.fit.cvut.cz/PAA/GA/`

[12] Matsunaga, R.: Genetic Algorithm Walkers [software]. [cit. 2017-11-21]. Dostupné z: `http://rednuht.org/genetic_walkers/`

[13] Kučera, L.: Algovision [software]. [cit. 2017-11-21]. Dostupné z: `http://www.algovision.org/`

[14] Halim, S.: VisuAlgo [software]. [cit. 2017-11-21]. Dostupné z: `https://visualgo.net/en`

[15] Galles, D.: Data Structure Visualizations [software]. [cit. 2017-11-21]. Dostupné z: `https://www.cs.usfca.edu/~galles/visualization/`

[16] Nielsen, J.: 10 Usability Heuristics for User Interface Design [online]. Leden 1995, [cit. 2018-4-17]. Dostupné z: `https://www.nngroup.com/articles/ten-usability-heuristics/`

[17] Žižkovský, P.: Návrh uživatelských rozhraní, 3. přednáška. 2017, České vysoké učení technické v Praze, Fakulta informačních technologií, ZS 2017/2018.

[18] Mozilla: JavaScript documentation. [cit. 2018-5-3]. Dostupné z: `https://developer.mozilla.org/en-US/docs/Web/JavaScript`

[19] Otto, M.: Bootstrap documentation. [cit. 2018-5-3]. Dostupné z: `https://getbootstrap.com/`

[20] You, E.: Vue.js documentation. [cit. 2018-5-3]. Dostupné z: `https://vuejs.org/v2/guide/`

[21] You, E.: Vuex documentation. [cit. 2018-5-3]. Dostupné z: `https://vuex.vuejs.org/en/`

[22] Zhu, N.; Woodhull, G.: Dimensional Charting Javascript Library. [cit. 2018-5-3]. Dostupné z: `https://dc-js.github.io/dc.js/`

[23] Olson, E.: ZangoDB. [cit. 2018-5-3]. Dostupné z: `https://github.com/erikolson186/zangodb`

[24] Vazirani, V. V.: *Approximation Algorithms*. New York, NY, USA: Springer-Verlag New York, Inc., 2001, ISBN 3-540-65367-8.

[25] Gamma, E.; Helm, R.; Johnson, R.; aj.: *Design Patterns: Elements of Reusable Object-Oriented Software*. Pearson Education, 1994, ISBN 9780321700698.

[26] Bazgan, C.; Escoffier, B.; Paschos, V. T.: Poly-APX- and PTAS-Completeness in Standard and Differential Approximation. In *Proceedings of the 15th International Conference on Algorithms and Computation*, ISAAC'04, Berlin, Heidelberg: Springer-Verlag, 2004, ISBN 3-540-24131-0, 978-3-540-24131-7, s. 124–136.

Příloha A

Seznam použitých zkratek

APX-C Approximable complete

BI-ZUM Základy umělé inteligence (bakalářský předmět na FIT)

ČVUT České vysoké učení technické

ETSP Euclidean travelling salesman problem

FIT Fakulta informačních technologií

FPTAS Fully polynomial-time approximation scheme

GA Genetic algorithm

GUI Graphical user interface

JVM Java Virtual Machine

NPO-C Nondeterministic polynomial time optimization complete

MAX-SAT Maximum satisfiability problem

MI-NUR Návrh uživatelského rozhraní (magisterský předmět na FIT)

MI-PAA Problémy a algoritmy (magisterský předmět na FIT)

MVC Minimum vertex cover

PTAS Polynomial-time approximation scheme

SA Simulated annealing

SVG Scalable Vector Graphics

TSP Travelling salesman problem

UI User interface

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
- `documentation` ..... dokumentace zdrojových kódů
- `questionary` ..... výsledky dotazníku