



ZADÁNÍ BAKALÁ SKÉ PRÁCE

Název:	Personalizace uživatelských rozhraní pro vizualizaci dat
Student:	Martin Endršt
Vedoucí:	Ing. Pavel Kordík, Ph.D.
Studijní program:	Informatika
Studijní obor:	Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce zimního semestru 2017/18

Pokyny pro vypracování

Cílem práce je navrhnout a implementovat aplikaci, která pom ůže zvolit a parametrizovat vhodný zp sob vizualizace dat pro daného uživatele. P edpokladem je, že stejná data je t eba pro r zné uživatele (technik, manažer) prezentovat v jiné form a jiným zp sobem.

1. Nastudujte techniky vizualizace informace. Soust e te se na metody použitelné k vizualizaci dat p ipravených pro modelování.
2. Navrhn te aplikaci, která umožní data zobrazovat prost ednictvím r zných vizualizací a vizualizace snadno parametrizovat pomocí konfigura ních soubor .
3. Návrh implementujte a otestujte na n kolika sadách dat, které dodá vedoucí práce. Pro samotnou vizualizaci použijte knihovnu D3js. Volba implementa ní platformy aplikace je sou ástí práce.
4. Pro n kolik uživatel zjist te, jaká je jejich preferovaná forma zobrazení a otestujte r zné techniky personalizace uživatelského rozhraní - nap . na základ metainformací nebo pomocí interaktivní evoluce.

Seznam odborné literatury

<http://jmlr.org/proceedings/papers/v29/Peltonen13.pdf>
<http://archive.ics.uci.edu/ml/>
<https://bigml.com/features>

L.S.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrđík, CSc.
říd kan

V Praze dne 29. b ezna 2016

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
KATEDRA SOFTWAREVÉHO INŽENÝRSTVÍ



Bakalářská práce

Personalizace uživatelských rozhraní pro vizualizaci dat

Martin Endršt

Vedoucí práce: Ing. Pavel Kordík, Ph.D.

16. května 2016

Poděkování

Rád bych poděkoval svému vedoucímu Ing. Pavlu Kordíkovi, Ph.D. za pomoc s vymezením tématu této práce a za ochotu i čas, který mi při jejím vypracovávání věnoval.

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 16. května 2016

.....

České vysoké učení technické v Praze
Fakulta informačních technologií

© 2016 Martin Endršt. Všechna práva vyhrazena.

Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí, je nezbytný souhlas autora.

Odkaz na tuto práci

Endršt, Martin. *Personalizace uživatelských rozhraní pro vizualizaci dat*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.

Abstrakt

Tato bakalářská práce se zabývá zkoumáním využitelnosti interaktivní evoluce k personalizaci zobrazení dat. Během řešení byly nejprve prozkoumány některé aplikace využívající interaktivní evoluci a analyzován jejich přístup k uživatelské interakci s evolucí. Na základě poznatků získaných z ostatních aplikací byla vyvinuta webová aplikace demonstrující tři různá řešení, která byla napsána v jazyce JavaScript s využitím knihovny D3.js k vykreslení zobrazení.

Klíčová slova evoluční algoritmus, interaktivní evoluce, vizualizace dat, binární hodnocení

Abstract

This Bachelor thesis explores the possibility of using interactive evolutionary algorithm to personalize data visualisation. First step was to research some applications using interactive evolutionary algorithms and analyse their approach to user interaction with the evolution. Based on research of other applications using the interactive evolution a web application demonstrating three different approaches to this problem was developed using JavaScript as a primary programming language along with D3.js library to render the data visualisations.

Keywords evolutionary algorithm, interactive evolutionary algorithm, data visualisation, binary evaluation

Obsah

Úvod	1
1 Vysvětlení pojmů	3
1.1 Evoluční algoritmus	3
1.2 Interaktivní evoluce	3
1.3 Neuronová síť	3
1.4 Outlier	4
2 Průzkum	5
2.1 Zhodnocení a výběr technik vizualizace	5
2.2 Typy vizualizací	5
2.3 Prozkoumání existujících aplikací využívající interaktivní evoluci	6
2.4 Získané poznatky	9
3 Experimentace	11
4 Návrh	17
4.1 Volba implementační platformy	17
4.2 Návrh evolučního procesu	19
4.3 Návrh aplikace	21
4.4 Návrh tříd	24
4.5 Systémové požadavky	30
4.6 Nasazení	30
5 Realizace	33
5.1 Vykreslování grafů	33
5.2 Pohledy evoluce	33
5.3 Vzdálenost jedinců u evolucí s více pohledy	34
6 Testování a hodnocení	35

6.1	Testovací sady	35
6.2	Testování uživatelů	38
Závěr		41
Literatura		43
A Seznam použitých zkratk		45
B Seznam použitých nástrojů a software		47
C Uživatelská příručka		49
C.1	Obrazovka Data	49
C.2	Panel simulace	50
C.3	Obrazovka Zobrazení	51
C.4	Obrazovka Binární evoluce	52
C.5	Obrazovka Vícepohledová evoluce, obrazovka Nepřímé kódování	53
D Obsah příloženého CD		55

Seznam obrázků

2.1	Obrázek	7
2.2	Obrázek	8
2.3	Obrázek	9
2.4	Obrázek	9
4.1	Obrázek	20
4.2	Obrázek	23
4.3	Obrázek	23
4.4	Obrázek	24
4.5	Obrázek	25
4.6	Obrázek	26
4.7	Obrázek	27
4.8	Obrázek	28
4.9	Obrázek	31
6.1	Obrázek	37
6.2	Obrázek	37
6.3	Obrázek	37
C.1	Obrázek	49
C.2	Obrázek	50
C.3	Obrázek	51
C.4	Obrázek	52
C.5	Obrázek	53

Úvod

Úvod do problematiky

Trendem poslední doby je využívání informačních systémů v různých odvětvích poskytování služeb či prodeje. Téměř každý obchodník provozuje internetový obchod, v knihovnách lze rezervovat knihy přes webové rozhraní a můžeme si dokonce nakoupit potraviny z pohodlí domova. Za každou z těchto služeb stojí většinou nějaký informační systém.

Tyto systémy shromažďují velké množství dat, k nimž přistupuje více uživatelů s různými požadavky na jejich interpretaci. Uvažujme obchod s elektronikou, který provozuje internetový prodej, skladuje a dopravuje své zboží. O vše se stará jeden informační systém. Účetní v takovém obchodě požaduje jinou vizualizaci dat než skladník a dopravce zajímá jiné zobrazení než manažera. Ani jedna z těchto osob však nemusí nutně vědět jak takovou vizualizaci, která by vyhovovala jejich potřebě, vytvořit.

Cíl práce

Cílem práce je prozkoumat možnost využití interaktivní evoluce pro vytvoření potřebné vizualizace dat s co nejjednodušším uživatelským rozhraním a s minimální potřebnou interakcí s uživatelem.

Motivace

Toto téma bakalářské práce jsem si vybral zejména ze zájmu blíže prozkoumat možnosti využití interaktivní evoluce na problému, se kterým jsem se již v praxi setkal, a ohodnotit využitelnost takového řešení.

Vysvětlení pojmů

1.1 Evoluční algoritmus

Evoluční algoritmus, též nazývaný genetický algoritmus, je heuristický optimalizační postup snažící se nalézt řešení problémů, pro něž neexistuje exaktní algoritmus. Využívá techniky inspirované evolučními procesy z přírody – jmenovitě dědičnost, mutaci, přirozený výběr a křížení. Evoluční algoritmus iterativně prohledává prostor možných řešení, kde každá iterace je nazývána generací, a na konci každé generace provede nad množinou právě zkoumaných řešení – zvanou populace – operace výběru, křížení a mutace. Kvalitu řešení udává fitness funkce, což je zobrazení řešení do reálného čísla, a evoluční algoritmus se většinou snaží tuto hodnotu maximalizovat. Algoritmus se zastaví po dosažení dostatečně dobrého řešení nebo po uplynutí stanoveného počtu generací. [1] [2]

1.2 Interaktivní evoluce

Interaktivní evoluční algoritmus je druhem evolučního algoritmu aplikovaného na problém, kde nelze přesně určit fitness funkci a je třeba vstupu od uživatele, aby ohodnotil kvalitu řešení. Většinou se jedná o optimalizaci subjektivně hodnocených řešení – například hudba či obraz. [3]

1.3 Neuronová síť

Neuronová síť je model inspirovaný centrálním nervovým systémem (konkrétně mozem), skládající se z jednotlivých neuronů a propojení mezi nimi. Díky možnosti definování vah na každém propojení a aktivační hranice na neuronu se může neuronová síť přizpůsobit neboli učit. Využívá se například v rozpoznávání řeči nebo rozpoznávání textu. [4]

1.4 Outlier

Outlier je hodnota pozorování, která se výrazně vychyluje od vzoru rozdělení ostatních pozorování. [5]

Průzkum

V této kapitole popisuji první krok v řešení práce – průzkum technik vizualizace a průzkum existujících aplikací využívajících interaktivní evoluci. V rámci tohoto kroku jsem provedl i literární rešerši, na jejíž poznatky odkazuji v kapitole 3.

2.1 Zhodnocení a výběr technik vizualizace

Hlavním cílem vizualizace informace je srozumitelně prezentovat informaci pomocí grafické formy. K dosažení tohoto cíle využíváme většinou grafy, mapy a skládané informační grafiky.

2.2 Typy vizualizací

2.2.1 Lineární zobrazení

Lineární zobrazení zobrazují data v závislosti na jediné vlastnosti – například slova v abecedním pořadí podle počátečního písmena. Jejich schopnost komunikovat informaci je poměrně malá, a proto jsou jen zřídka využívána.

2.2.2 Planární zobrazení

Pravděpodobně nejrozšířenější skupina vizualizací – zobrazují závislosti dvou vlastností v různých formátech. Jejich výhodou je snadná pochopitelnost – velice dobře komunikují informaci. Hodí se pro vizualizaci široké škály dat. Příkladem této skupiny je sloupcový graf či čárový graf.

2.2.3 Prostorová zobrazení

Prostorová zobrazení prezentují závislosti tří vlastností dat. Jsou vhodná pro použití v případě, kdy planární graf nedokáže data dostatečně rozlišit. Jejich

interpretace je však mnohdy složitější než u planárních grafů.

2.2.4 Mapy

Pro zobrazení dat, která jsou závislá na pozici, se nejvíce hodí zobrazení pomocí map. Toto zobrazení nám umožňuje zobrazit informace v kontextu geografickém nebo například v rámci grafického prostředí. Většinou zobrazuje závislost nějaké kvantity na poloze – například počet voličů podle krajů nebo četnost kliků na webové stránce.

2.2.5 Zhodnocení

Lineární zobrazení neposkytují dostatečnou schopnost komunikace informace, proto je z výběru vyřadíme. Planární zobrazení na druhou stranu velice dobře komunikují informaci příjemci, jsou dostatečně jednoduchá pro parametrizaci, aby byla konfigurovatelná pomocí interaktivní evoluce v rozumném počtu potřebných dotázání uživatele. Zároveň však jsou dostatečně obecná, aby zobrazila většinu potřebných souvislostí. Prostorová zobrazení jsou také poměrně snadno parametrizovatelná, jsou však složitější na ohodnocení uživatelem a výběr tří parametrů značně rozšiřuje prostor možných zobrazení. Výsledkem by pak byla pomalá evoluce vyžadující velké množství dotazů na uživatele k dosažení požadovaného zobrazení. Mapy by byly jednoduché pro ohodnocení uživatelem, problémem je však potřeba doplňujících informací k vykreslení základové mapy. Jelikož je aplikace určena převážně pro uživatele s nižší informační gramotností, nelze předpokládat, že uživatel má taková data k dispozici. Z těchto důvodů se ve své aplikaci zaměřím pouze na planární grafy – konkrétně na koláčový, čárový a sloupcový graf.

2.3 Prozkoumání existujících aplikací využívající interaktivní evoluci

Důležitou částí interaktivní evoluce je způsob realizace interakce uživatele s evolucí, proto v této kapitole prozkoumám přístup k interakci s uživatelem některých aplikací využívajících interaktivní evoluci. Vybral jsem aplikace PicBreeder, EndlessForms a Nintendo Wii Mii Creator.

2.3.1 PicBreeder

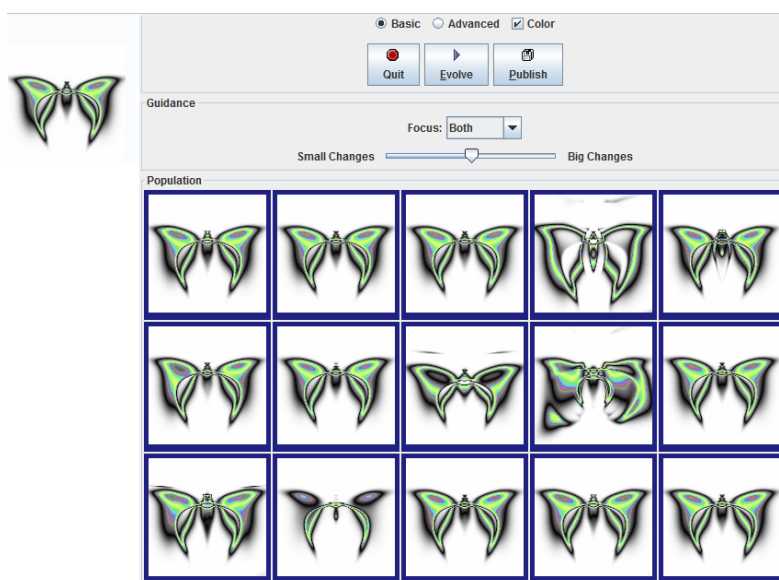
Asi nejznámější aplikací využívající interaktivní evoluce je PicBreeder dostupný na adrese <http://picbreeder.org>. Jeho účelem je kolektivní šlechtění 2D počítačové grafiky. Každý obrázek je vykreslován s využitím neuronové sítě, která je šlechtěna pomocí interaktivní evoluce. Výhodou tohoto řešení je fakt, že s postupující evolucí se šlechtěná neuronová síť stává komplexnější a

2.3. Prozkoumání existujících aplikací využívající interaktivní evoluci

výsledkem může být složitější obrázek. Díky tomu je teoreticky možné vygenerovat libovolný obrázek. Aplikace je realizovaná pomocí webové galerie a Java appletu pro šlechtění.

Uživatel má možnost použít ke šlechtění existující obrázek a pokračovat v jeho evoluci nebo začít novou náhodnou populací. Uživatelské rozhraní appletu je jednoduché – uživateli je nabídnuta mřížka 5x3 obrázků z nichž si vybere ty, které se mu nejvíce líbí. Zároveň je také uživateli umožněno vybrat velikost změn mezi generacemi a zda se má evoluce zaměřovat na barvu, tvar nebo na obojí. Viz. obrázek 2.1

Při výběru jediného obrázku je následující generace vytvořena pomocí různé mutace s vysokou pravděpodobností na tomto jedinci. Při výběru více obrázků je další generace získána pomocí křížení vybraných obrázků a následné mutace s nižší pravděpodobností.



Obrázek 2.1: Uživatelské rozhraní aplikace picbreeder

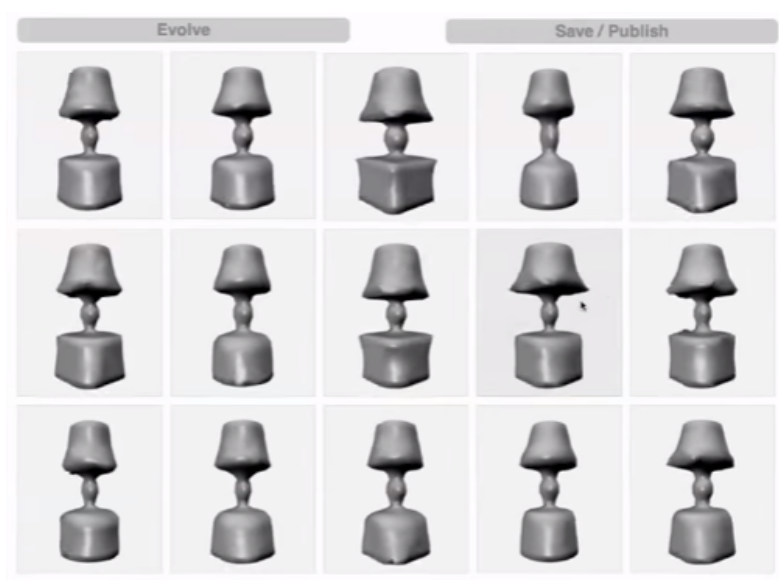
2.3.2 EndlessForms

EndlessForms je webová aplikace inspirována již zmíněným PicBreeder zaměřující se na generování 3D objektů a dostupná na adrese <http://endlessforms.com>. Každý objekt je definován pomocí sítě voxelů, které určují zda je v daném místě objekt plněn materiálem či nikoliv. Narozdíl od PicBreeder je prostor potenciálních výsledků omezen a jeho velikost je závislá na hustotě rozdělení pracovního prostoru do voxelů. Pro šlechtění je opět možné vyjít z existujícího modelu nebo začít s náhodnou populací.

2. PRŮZKUM

Uživatelské rozhraní je velice jednoduché – uživatel má možnost vybrat jeden či více objektů z 5x3 mřížky pro další evoluci. Není zde žádná kontrola nad rozsahem změn mezi evolucemi. Viz obrázek 2.2.

Stejně jako u aplikace PicBreeder vede výběr jednoho objektu k další populaci vytvořené různými mutacemi vybraného jedince, při výběru více objektů se vybrané objekty stanou rodiči pro další populaci.



Obrázek 2.2: Uživatelské rozhraní endlessforms

2.3.3 Nintendo Wii Mii Creator

Interaktivní evoluce našla uplatnění i v herním průmyslu. Aplikace Wii Mii Creator od společnosti Nintendo nabízí uživatelům možnost vytvořit si digitální postavičku, která je reprezentuje v rámci her – tzv. avatar. Hráči si mohou přizpůsobit svůj avatar manuálním výběrem vzhledových vlastností, nebo mohou využít interaktivní evoluce k nalezení takového avataru, který jim vyhovuje.

Nejprve je hráči pro výběr představena velká sada náhodně generovaných jedinců, z nichž si vybere jednoho jako základ dalších generací. (viz. obrázek 2.3) Poté je hráči představena mřížka 3x3 jedinců, ze které si může v každé iteraci vybrat jedince použitého k vytvoření populace pro další generaci. (viz. obrázek 2.4)

Pro nedostupnost informací o tom, jak jsou jedinci reprezentováni, uvádím pouze odhad. Předpokládám, že jedinec je definován pomocí sady čísel nebo řetězců, kde každý z nich udává hodnotu vzhledové vlastnosti (například ["female", "dark", "mikado", "blonde"] by značilo avatar ženského po-



Obrázek 2.3: Uživatelské rozhraní aplikace Wii Mii Creator [zdroj: YouTube, kanál James McDermott, dostupné na adrese <https://www.youtube.com/watch?v=MQpOMiTUz3M>]



Obrázek 2.4: Uživatelské rozhraní aplikace Wii Mii Creator [zdroj: YouTube, kanál James McDermott, dostupné na adrese <https://www.youtube.com/watch?v=MQpOMiTUz3M>]

hlaví s tmavou pletí a blond účesem mikádo). Další generace jsou pravděpodobně vytvořeny mutací vybraného jedince – nedochází ke křížení.

2.4 Získané poznatky

Z analýzy řešení zkoumaných aplikací vyplývají zajímavé poznatky. Přestože se zkoumané aplikace výrazně liší v kódování jedinců, všechny využívají téměř totožného způsobu prezentace populace uživateli – mřížku jedinců s možností

výběru jednoho či více kandidátů pro další generaci.

Výhoda tohoto přístupu je ta, že uživatel najednou hodnotí více jedinců a je schopen mezi nimi vidět souvislosti. Je zřejmé, že při volbě větší mřížky bude potřeba méně generací (tedy méně kliknutí uživatele) k nalezení požadovaného řešení, jelikož uživatel v každé generaci zhodnotí větší část prostoru řešení. Je však třeba mít na paměti, že evaluaci provádí člověk. Člověk má tendenci ztrácet pozornost, je tedy nutné zvolit takovou hustotu zobrazení jedinců, aby jedinci byli stále jednoznačně čitelní/rozlišitelní a aby uživatel neztrácel pozornost z důvodu přílišného množství jedinců. Je důležité, aby byli všichni jedinci zobrazení pohromadě bez nutnosti posouvání pohledu. Při hodnocení řešení komplexnějšího problému, kde uživatel není schopen odvodit kvalitu jedince prvním pohledem, volíme menší hustotu, aby nedocházelo k odrazování uživatele.

Další podobnost je ve způsobu tvoření populace pro další generaci. V běžném evolučním algoritmu je většinou používán postup **selekce** > **křížení** > **mutace** a liší se pouze ve způsobu realizace jednotlivých kroků. Ve zkoumaných aplikacích je však využívána převážně mutace vybraného jedince jako zdroj dalších generací. Tento postup vede k prohledávání pouze okolí konkrétního kandidáta, což by v klasické evoluci bylo nežádoucí a vedlo k suboptimálním výsledkům. Interaktivní evoluce však není využívána k hledání globálně optimálního řešení (v případě interaktivní evoluce většinou není optimální řešení definovatelné), ale právě k prohledávání okolí přípustných řešení.

Experimentace

V této kapitole popisují proces experimentace na základě získaných informací z kapitoly 2.

Nejprve jsem se seznámil s knihovnou D3.js vytvořením jednoduchých skriptů vykreslujících různé grafy nad testovacími daty. Po vytvoření několika zobrazení jsem se rozhodl vyzkoušet různé metody evoluce, se kterými jsem se seznámil v rámci předmětu BI-ZUM a literární rešerše.

Jako první krok jsem vytvořil jednoduchý evoluční proces, který je hodnocen uživatelem binárně. Proces začíná s náhodnou populací o velikosti 20, kódování jedince je realizované pomocí pole čísel určujících index sloupce pro osu x, index sloupce pro osu y, index způsobu zpracování dat a index zobrazení. Do další generace přechází evoluce výběrem pěti nejlepších jedinců turnajem z pěti, poté je každý vybraný jedinec s každým překřížen (operace křížení vrací 2 potomky, kde každý gen je přebrán z náhodného rodiče), čímž vznikne populace pro novou generaci. Nakonec je každý jedinec s pravděpodobností 10% zmutován výběrem náhodného genu a jeho nastavením na náhodnou přijatelnou hodnotu. Každý jedinec uchovává hodnotu fitness, při křížení dostane potomek fitness rodičů v poměru zvolených genů – například dědí-li 3 geny od rodiče A a 1 gen od rodiče B, bude hodnota fitness potomka

$$\frac{3}{4} * f_a + \frac{1}{4} * f_b$$

Při ohodnocení uživatelem je fitness jedince zvýšena / snížena o konstantu a pro celou populaci je stanovena nová hodnota fitness jako

$$f_i = \frac{1}{dist(i, s) + 1} * f_s$$

kde i značí momentálně zpracovávaného jedince, s značí hodnoceného jedince, f je fitness funkce a $dist$ je hammingova vzdálenost definovaná jako počet genů

3. EXPERIMENTACE

s rozlišnou hodnotou. Jak lze snadno předpokládat, tento evoluční proces je velice pomalý – pro testovací sadu je pro náhodnou kombinaci požadovaných atributů potřeba v průměru 158 hodnocení (výsledky průměrů uvádím po eliminaci outlierů pro 1000 opakování).

Takové množství potřebné interakce je samozřejmě nepřijatelné, proto jsem začal postupně aplikovat různé podpůrné metody. Jako první metodu jsem aplikoval fitness sharing.

Fitness sharing

Fitness sharing je metoda podporující divergenci genotypů a tím snižuje riziko uvíznutí v lokálním minimu. Metoda využívá shlukování jedinců na základě podobnosti a penalizaci jedinců v rámci shluků rozdělením fitness mezi členy stejné skupiny – tedy fitness jedince i po aplikaci fitness sharing je

$$f'_i = \frac{f_i}{m_i}$$

kde f_i je původní fitness jedince i a m_i je počet jedinců v odpovídajícím shluku. [6] Aplikací této metody, kdy řešení sdílí fitness, pokud je jejich hammingova vzdálenost menší než 2, došlo k mírnému snížení průměrného potřebného počtu hodnocení na 153 (došlo však k přiblížení negativních outlierů). Další zvolenou metodou je simulované žíhání.

Simulované žíhání

Simulované žíhání je, stejně jako evoluční algoritmus, metoda optimalizace prohledáváním prostoru inspirovaná procesem žíhání kovů. K prohledávání prostoru zavádí parametr teploty, která určuje pravděpodobnost přijetí zhoršujícího řešení. Díky možnosti přijetí zhoršujícího řešení je algoritmus odolnější proti uvíznutí v lokálním optimu. Teplota je na začátku prohledávání nastavena na vysokou hodnotu a postupně se s každou iterací snižuje (ne nutně lineárně – často se používá geometrického snižování). [7] Principu simulovaného žíhání lze využít i v evolučním procesu – například využitím parametru teploty při selekci jedinců pro křížení (čím větší teplota, tím větší pravděpodobnost zvolení horšího jedince). Pokud algoritmus uvízne v lokálním optimu (tedy nepřichází se zlepšujícím řešením), můžeme teplotu opět zvyšovat ve snaze nalezení lepšího řešení.

V mé adaptaci určuje teplota rozsah, z kolika nejlepších řešení je vybírán reprezentant pro ohodnocení. Díky tomu pokryje uživatel ze začátku evoluce větší rozsah různých řešení a algoritmus pak rychleji konverguje. Aplikováním této metody klesl průměrný počet potřebných ohodnocení na 139.

Tabu search

Během měření efektivity jsem si všiml, že se evoluce často vrací do stejných řešení. Tabu search je metoda optimalizace prohledáváním prostoru řešení

fungující na základě generování bezprostředních sousedů současného řešení. Takový způsob prohledávání je velice náchylný k uváznutí v lokálním optimu, a proto Tabu search přijímá i zhoršující řešení, pokud mezi sousedy není žádné zlepšující řešení. To by však patrně vedlo k cyklům, a proto Tabu search zaznamenává již navštívená řešení a už se do nich nevrací. [8] Této vlastnosti Tabu search využívám – zavádím seznam genotypů, které již byly uživateli prezentovány a po provedení operací mutace a křížení kontroluji přítomnost genotypu jedince v seznamu. Pokud naleznu shodu, provádím operaci mutace, dokud je genotyp obsažen v seznamu. Zavedením této metody došlo k poměrně výraznému zlepšení průměrného potřebného počtu hodnocení na 105 (zároveň se neobjevil žádný outlier s výrazně vyšším počtem hodnocení).

Prostředí řešení, prořezávání prostoru řešení

I přes velké zlepšení aplikováním podpůrných metod se množství interakce stále pohybuje na nepřijatelných hodnotách pro lidskou evaluaci. Je zřejmé, že binární hodnocení není vhodným způsobem hodnocení evoluce, jelikož je v každé iteraci prohledávání a hodnocení velice malý prostor. Než však přejdu k vícehledovému hodnocení po vzoru existujících aplikací z kapitoly 2, pokusím se aplikací agresivnějších přístupů snížit počet hodnocení na přijatelnou hodnotu.

Ostrovní model je metoda rozdělující populaci optimalizačního procesu do nezávislých skupin, nad kterými probíhá evoluce v izolovaném prostředí (vhodné pro paralelizaci), kterému se říká ostrov. V určitých intervalech část populace z ostrova migruje do jiného ostrova a stane se součástí tamějšího evolučního procesu. [9] Díky izolovaným prostředím lze na každém ostrově definovat fitness funkci jiným způsobem. Uvažujme situaci, kdy pro složení zkoušky potřebuje být jedinec silný a obratný. Jedinci z ostrova A budou odměňováni převážně na základě síly, kdežto jedinci z ostrova B budou odměňováni na základě obratnosti. Po kombinaci jedinců z ostrova A i B vzniknou jedinci silní a obratní, kteří lépe maximalizují celkovou fitness funkci.

Nechal jsem se inspirovat myšlenkou prostředí evoluce a částečné fitness a zavedl jsem je do evolučního procesu. Prostředí definuje váhy pro každou hodnotu optimalizovaného parametru, fitness jedince je určena jako součet odpovídajících vah. Na začátku evoluce jsou váhy nastaveny shodně pro každou hodnotu a při každém hodnocení se váhy zvyšují/snižují geometrickým způsobem.

Prořezávání prostoru řešení je také optimalizační podpůrná metoda. Její princip spočívá v omezení potenciálních řešení a odstranění těch částí prostoru řešení, o kterých můžeme s jistotou říci, že hledané řešení v nich neleží. Touto metodou zvyšujeme tlak na zlepšení řešení a potenciálně zrychlujeme optimalizační proces. Je však třeba přistupovat k prořezávání opatrně, jelikož

3. EXPERIMENTACE

prořezáváním snižujeme divergenci genotypů, což potlačuje sílu evolučního algoritmu – řešení v odřezaném prostoru mohou obsahovat část genotypu potřebnou k lepšímu řešení a využitím této metody dojde naopak ke zpomalení optimalizace.

Využil jsem myšlenky prořezávání prostoru a pokusil se najít způsob odřezání na základě rozdílů mezi pozitivně a negativně hodnocenými jedinci. Doposud byla řešení hodnocena pozitivně při shodě více než dvou atributů. Problém takového hodnocení je však ten, že potřebujeme velký počet rozdílných kladně a záporně ohodnocených jedinců pro odvození nepotřebných částí prostoru. Rozhodl jsem se tedy snížit počet optimalizovaných parametrů na 3 vypuštěním typu zobrazení z genotypu. Nyní lze z rozdílu mezi posledním kladně hodnoceným jedincem a současně hodnoceným jedincem omezit prostor řešení. Způsob úpravy vah prostředí na základě hodnocení je popsán pseudokódem 4.1 uvedeným v kapitole 4. Je nutno zdůraznit, že tento způsob prořezávání je velice agresivní a spoléhá na hodnocení uživatele v souladu s předpokladem – kladné pro dva shodné atributy, jinak záporné. Při hodnocení jiným způsobem může algoritmus odřezat tu část prostoru, kde leží řešení, a tím znemožnit nalezení výsledku. Po těchto úpravách došlo k prudkému snížení průměrného potřebného počtu hodnocení na 18, což je dle mého názoru přijatelné.

Posledním pokusem zlepšení tohoto evolučního procesu je žádat uživatele o hodnocení pouze v periodách iterace větší než 1. Pro zkoušku jsem žádal uživatele o hodnocení každou desátou iteraci. Na testovacích datech však nedošlo k žádnému zlepšení, ba naopak. Průměrný počet potřebných hodnocení se zvýšil na 27. Snižování periody výsledky takřka neměnilo, až periody o velikosti 3 a 2 se přiblížily číslu 18. Vysvětluji si to tím, že hledané řešení může být přeskočeno na pozadí a trvá déle, aby se do něj proces vrátil (pozn. adaptace Tabu search v tomto případě označuje za prohledaná pouze ta řešení, která byla prezentována uživateli, tudíž pokud bylo žádané řešení prohledáno na pozadí, může se do něj evoluce vrátit).

Dalším krokem je využití hodnocení více jedinců zároveň pomocí více pohledů. Kódování jedinců ponecháme stejné (zahrnující typ zobrazení), ponecháme způsob hodnocení pomocí prostředí, váhy prostředí upravíme vždy podle zvoleného jedince inkrementací vah hodnot shodných s jedincem a dekrementací vah ostatních hodnot. Váhy měníme geometricky s krokem 2. Uživateli zobrazujeme naráz 8 nejlepších jedinců, uživatel z nich vybírá jednoho jedince. Velikost populace je 16, další generaci vytváříme křížením každého jedince se zvoleným jedincem s mutací s pravděpodobností 40% a kontrolou duplicity genotypu v populaci (při duplicitě mutujeme, jedince dokud je duplicitní), počáteční populace je náhodná. Při otestování této evoluce je průměrný počet požadovaných hodnocení 67.

Začnu tedy opět s aplikací Fitness sharing se stejnou definicí vzdálenosti a shlukováním se vzdáleností menší než 2. Počet potřebných hodnocení zůstal téměř nezměněn – 63. Zvětšováním rozsahu shluků jsem dospěl na hodnotu 68

(to je shodné s očekáváním, jelikož při pokrytí celé populace jedním shlukem nemá fitness sharing žádný vliv). Zkusím tedy aplikovat simulované žhání – v tomto případě udává teplota pravděpodobnost výběru jedince z dolní poloviny populace pro zobrazení uživateli. Počet potřebných hodnocení se snížil na 59. Aplikací Tabu search došlo opět k velmi malému snížení potřebného počtu hodnocení – na hodnotu 55.

Z faktu, že tyto metody mají minimální vliv, ačkoliv u předchozí evoluce přinášely značné zlepšení, a z poměrně vysokého výskytu outlierů usuzuji, že návrh samotného procesu evoluce je nevhodný pro tento problém. Nechal jsem se tedy inspirovat aplikací Nintendo Wii Mii creator, která také řeší prohledávání diskrétního prostoru, a upravil způsob přechodu do nové generace. Velikost populace je snížena na 8, tedy shodná s počtem náhledů. Nová generace je tvořena mutováním vybraného jedince, procházíme tedy jeho okolí. Dále zavádím Tabu search tak, že jedinci, kteří byli uživateli zobrazení, již nebudou generováni. Touto změnou se počet potřebných hodnocení snížil na 26. Vypozoroval jsem však, že evoluce se pomalu dostává z nesprávných částí prostoru řešení. Generace nové populace je tedy upravena tak, že jedinci pro novou generaci jsou generováni ve zvětšující se vzdálenosti. V této podobě vychází průměrný počet požadovaných hodnocení na 16.

Poslední zajímavou modifikací je využití nepřímého kódování.

Nepřímé kódování Při použití nepřímého kódování nepopisuje genotyp jedince řešení přímo, ale popisuje postup, jak se k řešení dostat – například s použitím stromů nebo hyperkrychle. To vede k potenciálnímu zrychlení hledání řešení komplexních problémů, jelikož při křížení adaptují potomci různé části postupu řešení od obou rodičů. [10] [11]

Zavedu tedy kódování jedinců v podobě stromu konfiguračních bloků, které vždy provedou jednu operaci konfigurace na konfigurovaném zobrazení (tedy blok například nastaví typ zobrazení či zdroj dat pro osu). Náhodný jedinec vygeneruje strom náhodných konfiguračních bloků náhodné hloubky. Zavádím také validnost jedince, která určuje, zda zpracováním stromu vznikne validní zobrazení (má nastaveny všechny potřebné parametry). Vzdálenost jedinců je určena jako hammingova vzdálenost vzniklých konfigurací. Mutace jedince vytvoří náhodný strom (malé hloubky) a poté nahradí část původního stromu v náhodné hloubce nově vygenerovaným. Operace mutace se provádí, dokud strom negeneruje validní konfiguraci. Následující generaci opět tvoří sousedi generování mutací jedince ve zvětšující se vzdálenosti. Stále využíváme Tabu search a nepovolujeme duplikaci konfigurací. Tato evoluce dochází k výsledku v průměru po 18 hodnoceních.

Návrh

4.1 Volba implementační platformy

Díky specifickému zadání, které určuje způsob vykreslení pomocí D3.js, je volba implementační platformy značně omezená. V úvahu přichází Dart, JavaScript a TypeScript.

4.1.1 Dart

Dart je jazyk vyvinutý společností Google v roce 2011 v rámci Dart projektu, jehož součástí je mimo jiné i virtuální prostředí DartVM a prohlížeč Dartium. Dart je syntaxí blíže jazykům jako C# nebo Java a je dle mého názoru podařeným jazykem, pro programátora mnohem přívětivější než JavaScript. Spouštět Dart kód přímo umí pouze DartVM, který je momentálně integrován jen v prohlížeči Dartium, a nevím o žádných plánech integrovat toto prostředí do jiných prohlížečů. V roce 2013 byl Dart zahrnut do ECMA standardu pod ECMA TC52. Dart je kompilován do JavaScript pomocí kompilátoru dart2js. Více informací o projektu Dart můžete najít na oficiálních stránkách <https://www.dartlang.org>

Tento jazyk mne velice zaujal, jelikož samotný JavaScript mi přijde jako nepřehledný jazyk hlavně při vytváření objektově orientované aplikace. Velkou nevýhodou je však skutečnost, že pouze velice málo knihoven vytvořených pro JavaScript existuje i pro Dart, a pokud ano, pak se většinou jedná o neoficiální překlad, který bývá několik verzí pozadu. Dart nabízí možnost pracovat přímo s JavaScript knihovnamí pomocí Dart.interop, nicméně jeho použitím vzniká dle mého názoru nepěkný mix dvou jazyků. Pro knihovnu D3.js existuje pouze neoficiální překlad pro Dart a to byl důvod, proč jsem se rozhodl Dart nepoužít.

4.1.2 JavaScript

JavaScript pravděpodobně není třeba představovat. Tento jazyk vyvinutý v roce 1995 společností Netscape je vsoučasnosti nejrozšířenějším jazykem pro tvorbu interaktivních webových rozhraní. Standardizovaná verze JavaScriptu se nazývá ECMAScript a je základem pro různé skriptovací jazyky jako například ActionScript. Osobně nemám JavaScript příliš v oblibě a to převážně kvůli neelegantnímu řešení OOP a problematickému spojování zdrojových souborů. Na druhou stranu je to jednoznačně nejrozšířenější jazyk pro tvorbu interaktivních webových aplikací, což vede k obrovskému množství knihoven pro tuto platformu. Pouze z tohoto důvodu jsem se nakonec rozhodl využít JavaScript pro implementaci aplikace.

4.1.3 TypeScript

TypeScript je nadstavba jazyka JavaScript vyvinutá společností Microsoft v roce 2012, jejíž hlavní výhody jsou v poskytnutí statického typování a přívětivějšího přístupu k OOP pomocí běžných klíčových slov jako například `class` nebo `interface`. Podobně jako u jazyka Dart je i TypeScript kód kompilován do JavaScript. Pro využití JavaScript knihovny v TypeScript je zapotřebí hlavičkových souborů definujících typy, které knihovna využívá. V dnešní době jsou k dispozici hlavičkové soubory pro mnoho známých JavaScript knihoven včetně D3.js. TypeScript je dle mého názoru krok správným směrem a kód v tomto jazyce je jednoznačně lépe čitelný než v JavaScript. Hlavní důvod, proč jsem tuto nadstavbu nepoužil pro implementaci, je ten, že s ní nemám žádné zkušenosti.

4.1.4 Uživatelské rozhraní

K realizaci uživatelského prostředí mi nezbyvá než využít jazyk HTML v kombinaci s CSS styly. Jelikož tvorba uživatelského prostředí v HTML není mou silnou stránkou, rozhodl jsem se využít frameworku Bootstrap a volně dostupné šablony Janux Responsive Dashboard. Bootstrap je framework vyvinutý společností Twitter pro sjednocení grafické identity interních nástrojů v roce 2011. Později byl publikován pod licencí MIT a v současnosti se jedná o jeden z nejsledovanějších projektů na GitHub.

4.1.5 Vývojové prostředí

Na základě velice pozitivních zkušeností s vývojovým prostředím PHPStorm a CLion od společnosti JetBrains jsem bez váhání použil WebStorm od stejné společnosti. Vývojová prostředí z rodiny JetBrains mne zatím nikdy nezklamala. Cena by samozřejmě byla překážkou, nicméně JetBrains poskytuje licenci pro studenty na všechny své nástroje. WebStorm mi usnadnil práci díky výbornému našeptávání v JavaScript a automatickému doplňování v HTML.

4.2 Návrh evolučního procesu

Při navrhování evolučního procesu musíme především stanovit, jak bude řešení kódováno do genotypu jedince a jakým způsobem vytvoříme následující generaci. Dalšími aspekty jsou způsob hodnocení jedince, velikost populace a podpůrné metody řízení evoluce. Při návrhu vycházím z evolučních procesů, ke kterým jsem dospěl v rámci kapitoly 3.

4.2.1 Binární hodnocení

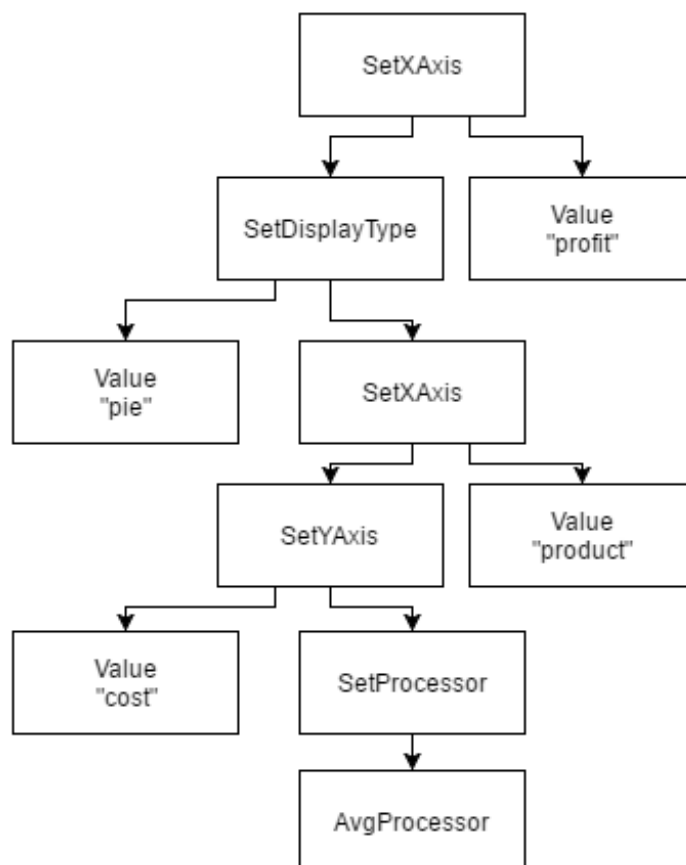
Jak jsem již ověřil experimenty, při použití binárního hodnocení nelze použít komplexní jedince, protože uživatel by musel ohodnotit veliké množství jedinců, než by se evoluce začala blížit požadovanému výsledku. Proto se v této evoluci omezím na optimalizaci tří parametrů – zdroj dat pro osu X, zdroj dat pro osu Y a způsob zpracování dat (myšleno průměr, suma...). Jedinec je kódován pomocí pole v následujícím formátu: `[index_sloupce_x, index_sloupce_y, cislo_zpracovani]`. Definuji také prostředí evoluce, které se přizpůsobuje podle uživatelského hodnocení (viz pseudokód 4.1). Prostředí určuje preferovanou hodnotu každého parametru a v každé generaci je využito pro ohodnocení jedinců – jedinci je přiřazena hodnota fitness podle vah nastavených v prostředí. Pro zesílení tlaku k rychlému nalezení řešení je využito prořezávání prostoru řešení reagující na hodnocení uživatele. Evoluce pracuje s 50 jedinci. Pro vytvoření následující generace se nejprve vyberou rodiče pěti turnaji o pěti jedincích, poté dojde ke křížení a potomci nahradí rodiče v další generaci. Výsledkem křížení dvou jedinců jsou dva potomci s náhodně rozdělenými parametry rodičů. Na závěr se provede kontrola duplicit jedinců – pokud se nalezne jedinec s genotypem, který již je v populaci reprezentován, nahradí se náhodným jedincem.

4.2.2 Vícepohledová evoluce

Kódování jedince zůstává obdobné jako u předchozí evoluce, avšak díky prozkoumávání větší části prostoru řešení najednou můžeme zahrnout parametr typu zobrazení. Genotyp tedy vypadá následovně `[index_sloupce_x, index_sloupce_y, cislo_zpracovani, cislo_zobrazeni]`. U této evoluce nerozlišujeme kvalitu řešení, pouze prohledáváme okolí právě voleného řešení. Následující generace je tvořena různou mutací vybraného řešení. Abych zmírnil pravděpodobnost uvíznutí v nevyhovující části prostoru řešení, generuji jedince pro další generaci ve zvětšující se vzdálenosti od aktuálně zvoleného jedince. Pro zamezení uvíznutí v cyklu zaznamenávám, jaká řešení byla již prozkoumána, a už se do nich nevracím.

4.2.3 Nepřímé kódování

Pro jedince používáme kódování pomocí stromů skládajících se z bloků různých typů – SetXAxis, SetYAxis, SetDisplayType, SetProcessor a Value (viz. obrázek 4.1). Při vytváření zobrazení je strom zpracováván směrem od kořene a jednotlivé bloky použity pro konfiguraci zobrazení. Stejně jako v minulé evoluci použijeme populaci o velikosti 8, generujeme jedince pro další generaci ve zvětšující se vzdálenosti a nevracíme se do již prozkoumaných řešení. Při mutaci se vybere náhodná hloubka a vygeneruje se nový náhodný strom, který nahradí část původního stromu v této hloubce. Při generování náhodných stromů a při mutaci se může stát, že výsledkem stromu je neplatná konfigurace (například nejsou nastavené osy) – takový strom je přegenerován, dokud negeneruje platnou konfiguraci.



Obrázek 4.1: Nepřímé kódování - strom

Listing 4.1: Pseudokód přizpůsobení prostředí evoluce na základě uživatelského hodnocení

```

if (pozitivni){
  if (predchozi pozitivni definovan){
    if (manhatton(soucasny, predchozi pozitivni) == 2){
      nastav vahu shodneho parametru na 10
      pro shodnou hodnotu parametru a na 0
      pro hodnoty ruzne
    } else if (manhatton(soucasny, predchozi pozitivni) == 1){
      for (nezmenene parametry){
        nastav vahu parametru na 10
        pro shodnou hodnotu parametru
        a na 0 pro hodnoty ruzne
      }
      nastav vahu zmeneneho parametru na 0
      pro hodnoty parametru soucasneho a predchoziho pozitivniho
    }
  } else {
    vaha osaX[soucasny.x] += VAHOVY INKREMENT
    vaha osaY[soucasny.y] += VAHOVY INKREMENT
    vaha zpracovani[soucasny.zpracovani] += VAHOVY INKREMENT
  }
  predchozi pozitivni = soucasny
} else {
  if (predchozi pozitivni definovan){
    if (manhatton(soucasny, predchozi pozitivni) == 1){
      nastav vahu zmeneneho parametru na 10
      pro hodnotu predchoziho pozitivniho a na 0
      pro ostatni hodnoty
    } else {
      vaha osaX[soucasny.x] /= VAHOVA PENALIZACE
      vaha osaY[soucasny.y] /= VAHOVA PENALIZACE
      vaha zpracovani[soucasny.zpracovani] /= VAHOVA PENALIZACE
    }
  } else {
    vaha osaX[soucasny.x] /= VAHOVA PENALIZACE
    vaha osaY[soucasny.y] /= VAHOVA PENALIZACE
    vaha zpracovani[soucasny.zpracovani] /= VAHOVA PENALIZACE
  }
}
}

```

4.3 Nárh aplikace

4.3.1 Specifikace požadavků

Aplikace je určena pro prozkoumání a demonstraci možnosti využití interaktivní evoluce při výběru a parametrizaci zobrazení. Aplikace musí umožňovat

import dat a metadat, se kterými uživatel může pracovat při výběru zobrazení. Dále musí uživateli nabídnout snadné rozhraní pro ovládání evolučního procesu. Aplikace také bude umožňovat simulaci evoluce pro snadné zhodnocení kvality evolučního procesu. V poslední řadě musí mít uživatel možnost zobrazení exportovat do souboru či z existujícího souboru zobrazení načíst.

4.3.1.1 Funkční požadavky

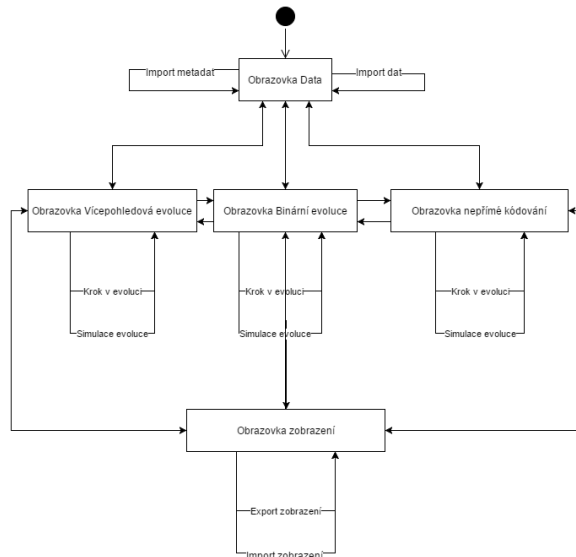
- Import dat
- Import metadat
- Ovládání evolučního procesu
- Simulace evolučního procesu
- Import zobrazení
- Manuální parametrizace zobrazení
- Export zobrazení

4.3.1.2 Nefunkční požadavky

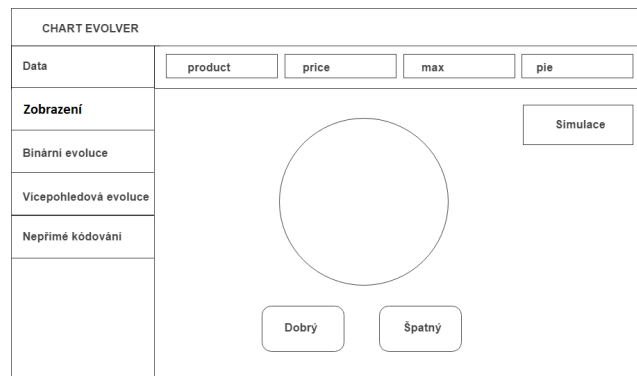
- Implementace v prostředí webového klienta
- Dostupnost bez nutnosti připojení k internetu
- Snadná rozšiřitelnost
- Jednoduché uživatelské prostředí

4.3.2 Uživatelské prostředí

Hlavní záměr je vytvořit aplikaci uživatelsky co možná nejjednodušší. Obrázek 4.2 zobrazuje diagram uživatelské interakce s aplikací. Aplikace je rozdělena do pěti obrazovek – obrazovka pro import dat a metadat, obrazovka pro náhled současného zobrazení a pro jeho import a export, obrazovka pro evoluci s binárním hodnocením, obrazovka pro vícehledovou evoluci a obrazovka pro evoluci s nepřímým kódováním. Po otevření aplikace je uživateli zobrazena obrazovka s daty. Pro demonstrační účely je předem načtena testovací datová sada a její metadata. Při přechodu na obrazovky evoluce je inicializováno prostředí evoluce a uživateli je představena náhodná populace či náhodný jedinec. Na obrazovce s daty je uživateli prezentováno, kolik je načteno řádek dat a tabulka ukazující načtené sloupce s jejich typem.



Obrázek 4.2: Diagram uživatelské interakce



Obrázek 4.3: Wireframe obrazovky Binární evoluce

4.3.3 Prezentační vrstva

Jelikož je hlavička a postranní nabídka neměnná napříč obrazovkami, implementuji šablonovací systém. Šablonové soubory mají sekci pro kód HTML ohraničenou pomocí páru `{html}{/html}` a kód JavaScript ohraničenou pomocí páru `{script}{/script}`. Šablony jsou načítány při přechodu na odpovídající obrazovku, sekce skriptu spuštěna pomocí příkazu `eval()`. Díky tomu aplikace funguje bez načítání stránek. Bohužel musí být spuštěna v prostředí webového serveru, jelikož načítání souborů z lokálního disku není prohlížeči povoleno.

4.3.4 Vstup

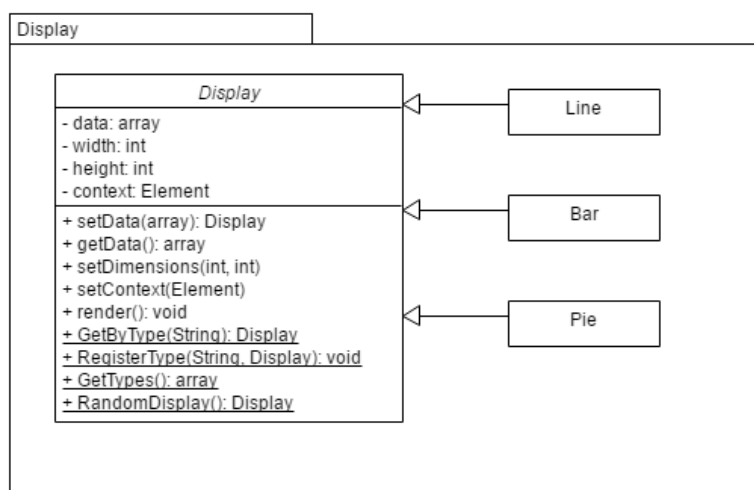
Aplikace potřebuje pro fungování vstupní data a metadata. Data jsou načítána ze souboru ve formátu .csv, kde první řádek udává názvy sloupců a další řádky obsahují samotná data. Metadata jsou též načítána ze souboru ve formátu .csv, první řádek musí být `index,name,type` a další řádky obsahují příslušné informace. Typ je rozlišován pouze na `label` a `value`. Pokud uživatel neposkytne metadata, jsou typy sloupců odvozeny podle prvního řádku dat.

4.3.5 Výstup

Výstupem aplikace jsou konfigurační soubory zobrazení ve formátu .csv. Výstup lze získat z obrazovky Zobrazení kliknutím na Export. Soubory v tomto formátu je poté možné na stejné obrazovce importovat.

4.4 Návrh tříd

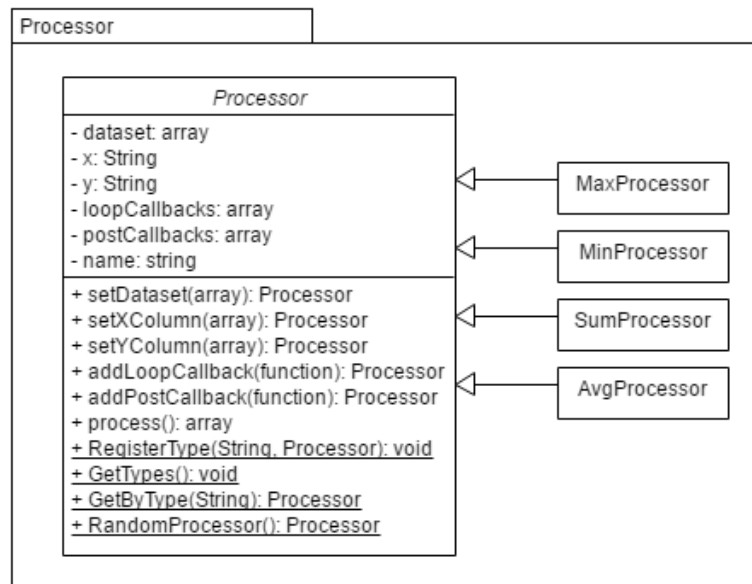
Podle požadavků a návrhu z předchozí sekce vyplynul následující návrh tříd. Názvy proměnných a metod jsou uvedeny v angličtině, návrh je rozdělen do souvisejících celků. Třída `ChartEvolver` není uvedena, jelikož není z návrhového hlediska nijak zajímavá – její hlavní funkcí je držení referencí na ostatní třídy a uchovávání současného stavu aplikace.



Obrázek 4.4: Návrhový model tříd pro abstraktní třídu `Display` a její asociované třídy

4.4.1 Třída Display

Třída Display je abstraktní třídou definující společnou logiku a rozhraní jednotlivých typů zobrazení. Třída si uchovává referenci na HTML Element, do kterého se má zobrazení vykreslit pomocí metody `render()`. Při vykreslení jsou nastaveny rozměry cílového elementu uchované v atributech `width` a `height` s použitím dat uložených v atributu `data`. Metoda `render` je abstraktní, o její implementaci se starají dědící třídy. Aby mohly evoluce pracovat s konkrétními implementacemi, evidují se třídy implementující třídu Display pomocí statické metody `RegisterType`. Statické metody `GetType`, `RandomDisplay` a `GetTypes` poskytují přístup ke konkrétním implementacím třídy Display – v aplikaci se jedná o třídy `Line` realizující čárový graf, `Pie` realizující koláčový graf a `Bar` realizující sloupcový graf.



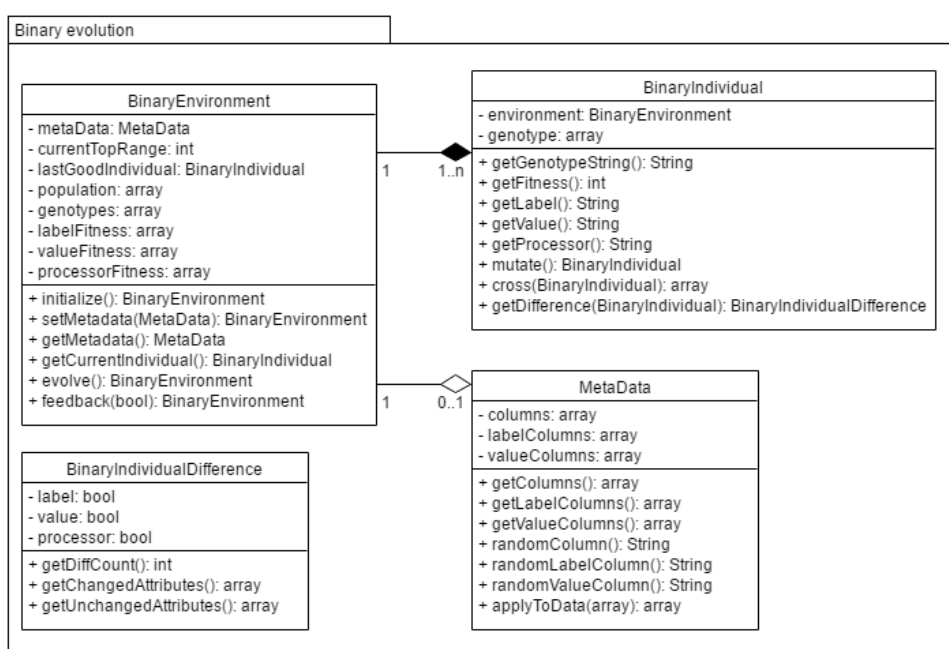
Obrázek 4.5: Návrhový model tříd pro abstraktní třídu Processor a její asociované třídy

4.4.2 Třída Processor

Třída Processor je základní třídou pro jednotlivé způsoby zpracování dat. Hlavní metodou třídy je metoda `process`, která zpracuje data v atributu `data`. Ačkoliv je třída Processor primárně určena jako rodičovská třída pro konkrétní implementace zpracování, není třída navržena jako abstraktní. Pro specifikaci chování procesu zpracování je využito metod `addLoopCallback` a `addPostCallback`. Při běhu metody `process` je na každý řádek dat volána každá z funkcí registrovaných pomocí `addLoopCallback` v pořadí FIFO. Na

4. NÁVRH

konci zpracování jsou zavolány funkce registrované pomocí `addPostCallback` v pořadí FIFO. Tento přístup nám umožňuje dynamické vytváření procesů zpracování bez nutnosti deklarace dědičích tříd. Obdobně jako třída `Display` definuje i třída `Processor` statickou metodu `RegisterType`, určenou pro evidování tříd implementující konkrétní způsob zpracování, a statické metody `GetTypes`, `GetByType` a `RandomProcessor` pro přístup k jednotlivým implementacím způsobů zpracování – v aplikaci se jedná o třídy `MaxProcessor` získávající maximum z dat, `MinProcessor` získávající minimum z dat, `SumProcessor` získávající sumu z dat a `AvgProcessor` získávající průměr z dat.



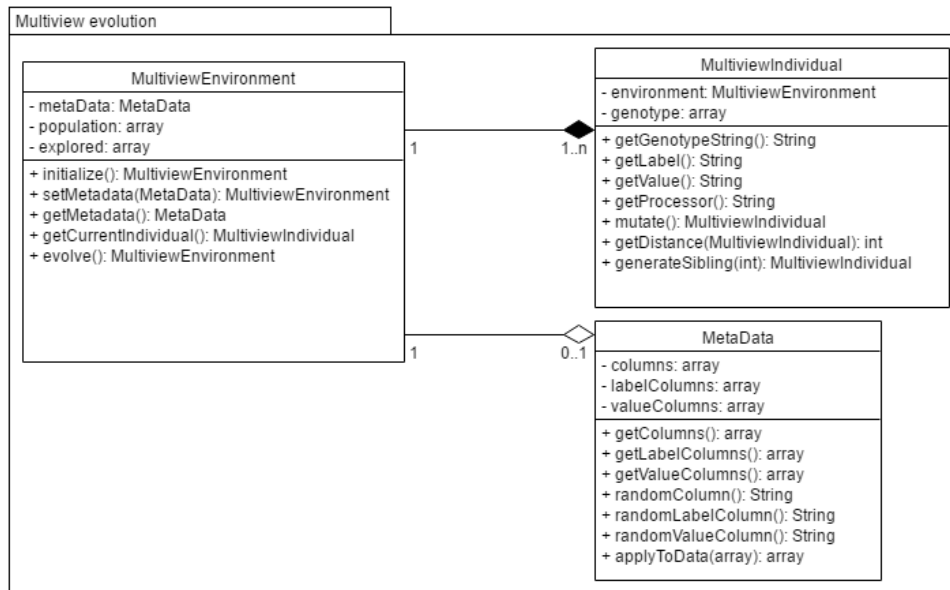
Obrázek 4.6: Návrhový model tříd pro třídy zajišťující evoluci s binárním hodnocením

4.4.3 Třídy evoluce s binárním hodnocením

Hlavní třídy pro evoluci s binárním hodnocením jsou třídy `BinaryEnvironment` a `BinaryIndividual`. `BinaryEnvironment` uchovává momentální stav evoluce – populaci, váhy fitness a posledního kladně hodnoceného jedince. Obstarává také logiku tvorby nové generace pomocí metody `evolve` a logiku reakce na hodnocení od uživatele pomocí metody `feedback`.

Třída `BinaryIndividual` definuje jedince v evolučním procesu. Obsahuje referenci na evoluční prostředí a genotyp uložený v poli. Hlavními metodami jedince jsou metody `mutate` zajišťující logiku mutace a `cross` zajišťující logiku

křížení. Třída `BinaryIndividualDifference` je pomocnou třídou reprezentující rozdíl mezi dvěma jedinci využívána metodou `getDifference` třídy `BinaryIndividual`.



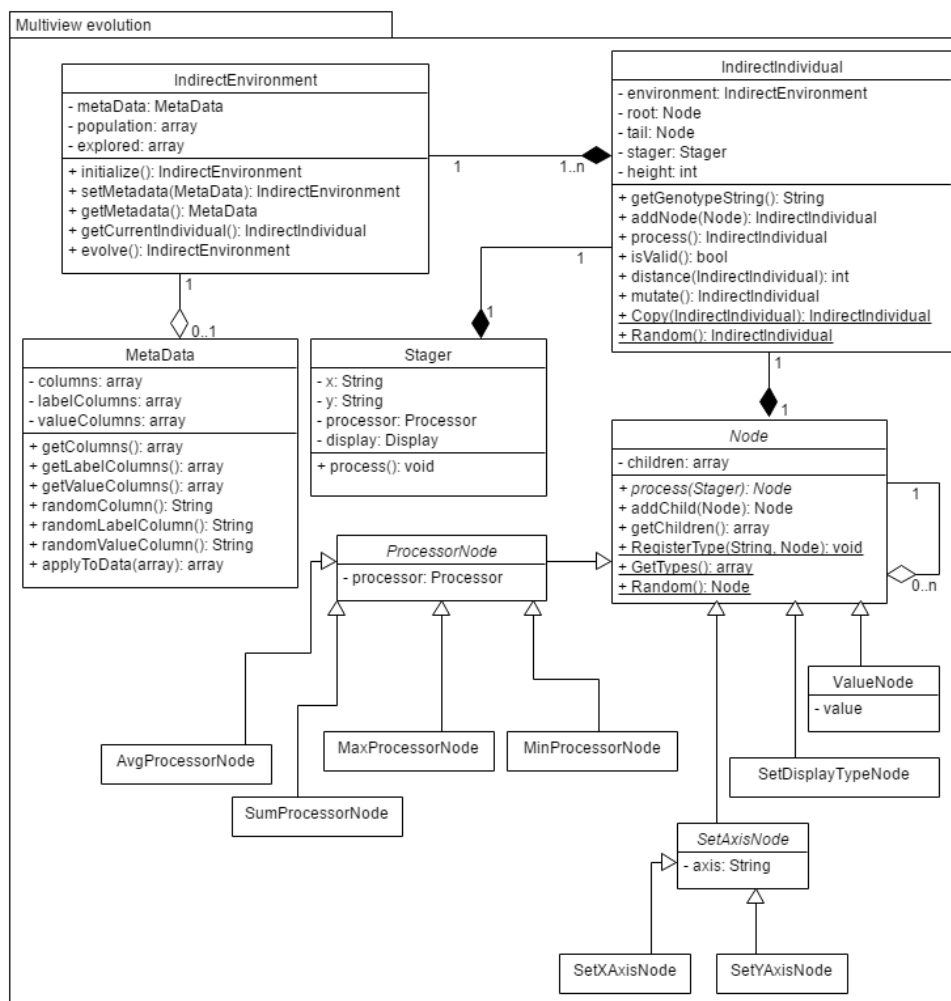
Obrázek 4.7: Návrhový model tříd pro třídy zajišťující vícehledovou evoluci

4.4.4 Třídy vícehledové evoluce

Hlavní třídy vícehledové evoluce jsou třídy `MultiviewEnvironment` a `MultiviewIndividual`. Podobně jako třída `BinaryEnvironment` uchovává `MultiviewEnvironment` současný stav evoluce – populaci, posledně zvoleného jedince a prozkoumané genotypy. Zajišťuje také logiku pro vytvoření další generace pomocí metody `evolve`.

Třída `MultiviewIndividual` reprezentuje jedince v evolučním procesu. Stejně jako třída `BinaryIndividual` obsahuje referenci na evoluční prostředí a genotyp uložený v poli. V evoluci jsou použity hlavně metody `getDistance` získávající vzdálenost od jiného jedince, `mutate` zajišťující logiku mutace a `generateSibling` vytvářející podobného jedince v určené vzdálenosti.

4. NÁVRH



Obrázek 4.8: Návrhový model tříd pro třídy zajišťující evoluci s nepřímým kódováním

4.4.5 Třídy evoluce s nepřímým kódováním

Stejně jako v předchozích evolucích jsou středobodem třídy prostředí a jedince – `IndirectEnvironment` a `IndirectIndividual`. Rozhraní třídy `IndirectEnvironment` je shodné s třídou `MultiviewEnvironment` a není je tedy třeba rozvádět. Třída `IndirectIndividual` opět uchovává odkaz na evoluční prostředí a genotyp. Genotyp je však realizován pomocí stromu konfiguračních bloků. Hlavní třídou konfiguračních bloků je abstraktní třída `Node` definující společnou logiku a rozhraní pro jednotlivé implementace bloků. Třída `Node` uchovává seznam potomků, její hlavní metodou je abstraktní metoda `process`, která

provede potřebnou operaci a vrátí odkaz na následující blok. Podobně jako třídy `Display` a `Processor` má třída `Node` statické rozhraní pro registraci implementací a přístup k nim pomocí metod `RegisterType`, `GetTypes` a `Random`. V rámci aplikace jsou implementovány následující typy bloků:

- `ValueNode`
 - blok obsahující konstantní hodnotu
- `SetDisplayTypeNode`
 - blok konfiguruje typ zobrazení
 - jako prvního potomka očekává blok `ValueNode` s hodnotou odpovídající jménu zobrazení
- `SetAxisNode`
 - abstraktní blok nastavující osy
 - `SetXAxisNode` – blok nastavující osu X
 - `SetYAxisNode` – blok nastavující osu Y
- `ProcessorNode`
 - abstraktní blok nastavující zpracování
 - `MaxProcessorNode` – nastaví zpracování na `MaxProcessor`
 - `MinProcessorNode` – nastaví zpracování na `MinProcessor`
 - `AvgProcessorNode` – nastaví zpracování na `AvgProcessor`
 - `SumProcessorNode` – nastaví zpracování na `SumProcessor`

Třída `Stager` je konfigurována konfiguračním stromem a uchovává zobrazení, data a způsob zpracování dat. Její metoda `process` nastaví zobrazení a vykreslí jej do nastaveného elementu.

Třída `Metadata`

V diagramech byla několikrát znázorněna třída `MetaData` bez vysvětlení. Jedná se o pomocnou třídu, která je využívána při generování jedinců – poskytuje rozhraní pro získání informací o sloupcích pomocí metody `getColumnns`, `getLabelColumns`, `getValueColumns`, `randomColumn`, `randomLabelColumn` a `randomValueColumn`.

4.5 Systémové požadavky

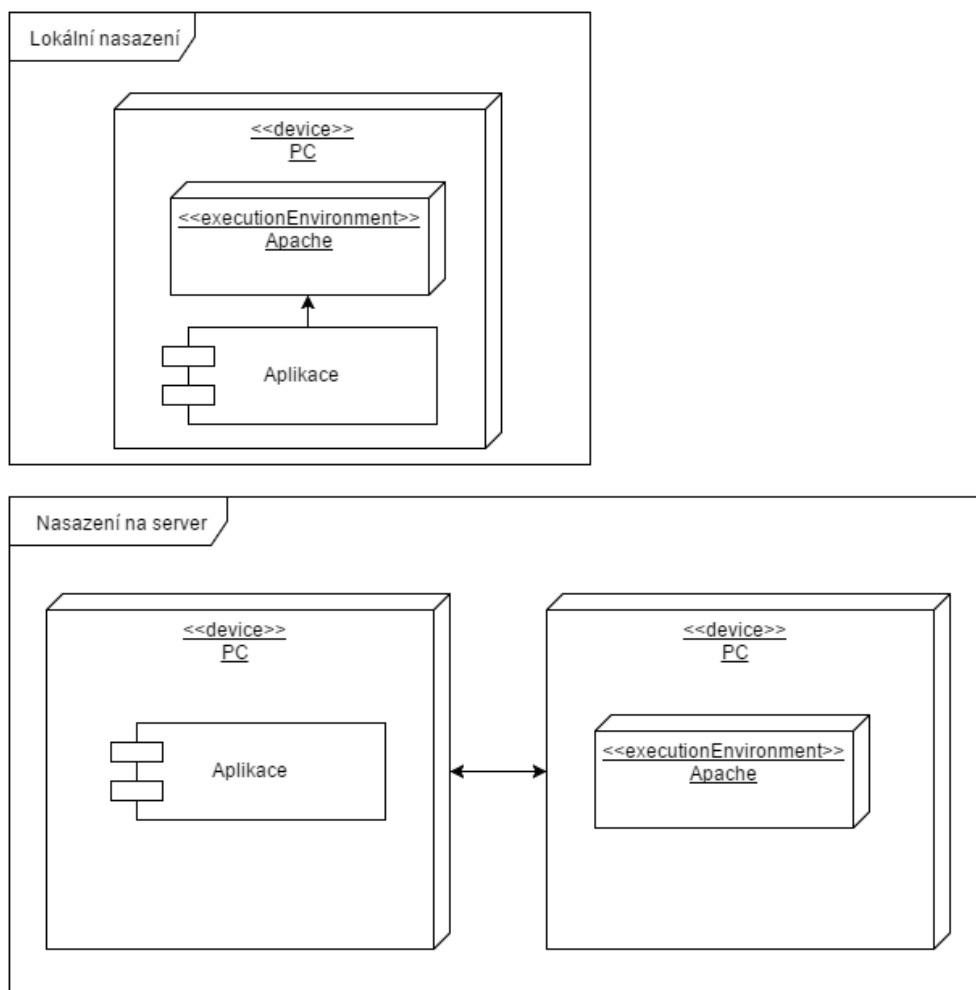
Z důvodu návrhu aplikace jako single-page webové aplikace a omezení XHR požadavků je nutné aplikaci spustit v rámci webového serveru. Pravděpodobně nejrozšířenějším webovým serverem je Apache. Bohužel nejsou dostupné žádné oficiální požadavky kromě minimálního místa na disku 50MB. Z osobních zkušeností vím, že Apache lze provozovat na zařízení Raspberry Pi s procesorem o kmitočtu 700MHz a RAM o velikosti 512MB. Aplikace je vyvíjena primárně pro webový prohlížeč Google Chrome verze 13 a vyšší. Jelikož požadavky pro aplikaci Google Chrome verze 13 jsou vyšší než pro server Apache, uvedu je jako minimální požadavky pro aplikaci.

- Operační systém: Windows 7 a vyšší / Ubuntu 14.04+, Debian 8+, open-SUSE 13.1+, FedoraLinux 21+
- Procesor: Intel Pentium 4 nebo vyšší
- Operační paměť: 2 GB
- Místo na disku: 100 MB

Aplikace není určena pro mobilní zařízení z důvodu chybějící podpory FileAPI ve většině mobilních prohlížečů.

4.6 Nasazení

Aplikace může být nasazena lokálně, je však zapotřebí instalace webového serveru a přístup k aplikaci přes tento server. Alternativně lze aplikaci provozovat na serverovém zařízení a přistupovat k ní přes HTTP protokol.



Obrázek 4.9: Diagram nasazení

Realizace

V této kapitole uvádím zajímavé poznatky získané v průběhu realizace, změny oproti návrhu či nečekané problémy vzniklé během vývoje aplikace.

5.1 Vykreslování grafů

Po vytvoření skriptů pro vykreslení testovacích grafů jsem nebyl spokojen s jejich realizací z důvodu špatné abstrakce do obecnějších tříd. Kód vytvoření zobrazení byl příliš specifický pro konkrétní zobrazení a každý typ vyžadoval speciální úpravu vstupu. Jelikož jedním z nefunkčních požadavků je snadná rozšiřitelnost, bylo potřeba kód vytvoření zobrazení zabalit do tříd se společným rozhraním. Po krátkém hledání na internetu jsem našel rozšiřující knihovnu pro D3.js zvanou NVD3.js, která tento problém řeší – balí rozhraní knihovny D3.js do rozhraní s jednotnou jmennou konvencí a přidává jednotný způsob definice dat pro různá zobrazení. Knihovna NVD3 byla vyvinuta pro interní potřeby společnosti Novus Partners a později publikována pod licencí Apache License verze 2.0. Více informací a odkazy pro stažení lze nalézt na adrese <http://nvd3.org>

5.2 Pohledy evoluce

Po implementaci vícepohledové evoluce a evoluce s nepřímým kódováním jsem narazil na výkonnostní problémy na pomalejším zařízení, pokud došlo u více grafů k nastavení osy x na sloupec s velkým množstvím různých hodnot. Zároveň jsem na sobě pozoroval, že při hodnocení generace občas ztrácím pozornost. Rozhodl jsem se tedy pro snížení počtu pohledů na 6. Důsledkem bylo zvýšení průměrného potřebného počtu generací k dosažení požadovaného výsledku o 2, avšak zlepšil se můj pocit z ovládání evoluce. Bohužel na slabším zařízení stále docházelo k zasekávání, i když v menší míře.

5.3 Vzdálenost jedinců u evolucí s více pohledy

Při testování aplikace jsem si všiml, že evoluce při využití hodnocení více pohledy neprochází prostor tak, jak by uživatel předpokládal. Je to způsobeno definicí vzdálenosti mezi jedinci. Hammingova vzdálenost nerozlišuje, které geny jsou mezi jedinci rozdílné. Pro uživatele jsou však jednotlivé atributy prezentovány s různým důrazem. Typ zobrazení je například z vizuálního pohledu velice dominantní vlastností jedince a pokud evoluce generuje následující generaci s minimální shodou typu zobrazení s vybraným jedincem, získá uživatel pocit, že evoluce nefunguje správným způsobem. Zavedl jsem tedy váhy parametrů do vzdálenosti jedinců u evolucí s hodnocením více pohledy a vzdálenost dvou jedinců je pak definovaná jako součet vah rozdílných parametrů. Výsledkem je předpověditelnější evoluční proces a počet potřebných hodnocení se nezměnil.

Testování a hodnocení

V této kapitole uvádím výsledky testování konečné aplikace. Testování proběhlo na třech sadách dat pomocí automatického testovacího skriptu a za pomoci tří dobrovolníků bez znalosti aplikace.

6.1 Testovací sady

Testovací sady jsou náhodně generované pomocí PHP skriptů. Simulují data z informačních systémů tří různých subjektů – objednávky v půjčovně aut, zapůjčené knihy v knihovně a objednávky internetového obchodu s elektronikou. Každá datová sada má 1000 řádek dat. Testovací skript simuluje počínání uživatele. Po spuštění vygeneruje 100 náhodných sad požadavků a každou sadu testuje ve všech třech evolucích.

6.1.1 Půjčovna aut

Datová sada půjčovny aut obsahuje 1000 záznamů o objednávkách imaginární půjčovny aut. Každý řádek obsahuje jméno a příjmení zákazníka, pokud je zákazníkem osoba, nebo název firmy, pokud je zákazníkem firma. Dále obsahuje název a značku auta, typ zákazníka (osoba nebo firma), město, kde si zákazník auto přebírá, město, kde zákazník auto vrací, způsob platby, cenu, dobu trvání pronájmu v hodinách, výnos pro majitele auta a pomocný sloupeček `count` s konstantní hodnotou 1. Soubor metadat k této datové sadě není k dispozici, jelikož aplikace správně odhadne typy sloupců podle prvního řádku.

Pro touto datovou sadou potřebuje evoluce s binárním hodnocením v průměru 10 hodnocení, evoluce s vícehledovým hodnocením 5 hodnocení a evoluce s nepřímým kódováním 12 hodnocení.

6.1.2 Knihovna

Datová sada knihovny obsahuje 1000 záznamů o propůjčení knih. Každý řádek obsahuje jméno a příjmení čtenáře, název knihy, jméno autora, název vydavatelství, rok vydání, kategorii knihy, způsob ručení za knihu, dobu trvání zapůjčení v dnech a pomocný sloupeček `count` s konstantní hodnotou 1. V adresáři zdrojových souborů je vložen soubor metadat pro tuto datovou sadu, jelikož aplikace by chybně označila sloupeček roku jako `value`.

Pro touto datovou sadou potřebuje evoluce s binárním hodnocením v průměru 7 hodnocení, evoluce s vícehledovým hodnocením 5 hodnocení a evoluce s nepřímým kódováním 6 hodnocení.

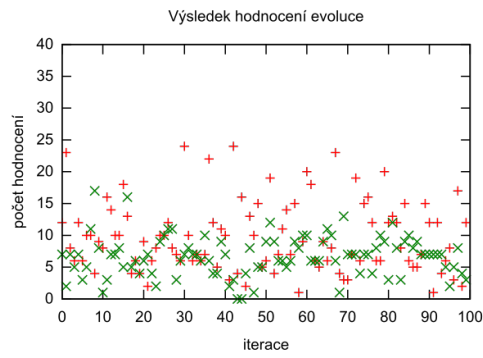
6.1.3 Obchod s elektronikou

Datová sada obchodu s elektronikou obsahuje 1000 záznamů o prodaném zboží. Každý řádek obsahuje jméno a příjmení zákazníka, název produktu, název výrobce, kategorii produktu, jméno a příjmení zaměstnance, který prodej zpracoval, cenu, cenu s DPH, hodnotu DPH, počet prodaných kusů a způsob platby. Tato datová sada nepotřebuje soubor metadat, jelikož aplikace správně odhadne typy sloupců podle prvního řádku.

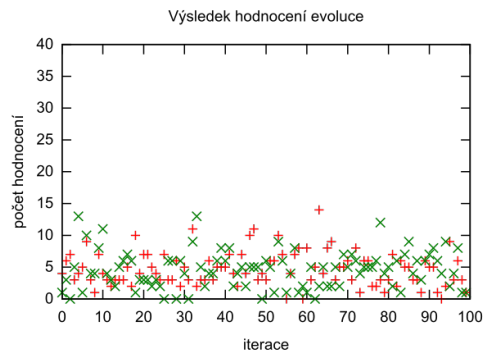
Pro touto datovou sadou potřebuje evoluce s binárním hodnocením v průměru 10 hodnocení, evoluce s vícehledovým hodnocením 6 hodnocení a evoluce s nepřímým kódováním 14 hodnocení.

6.1.4 Zhodnocení

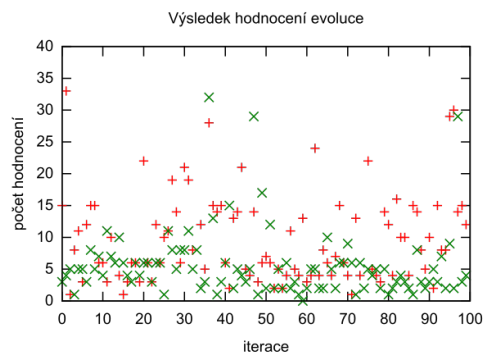
Z provedených testů je zřetelné, že nejlepších výsledků dosahuje evoluce s vícehledovým hodnocením, následuje evoluce s binárním hodnocením a nejméně konverguje evoluce s nepřímým kódováním jedinců. Je nutno zdůraznit, že prostor, který prohledává evoluce s binárním hodnocením, je výrazně menší z důvodu vypuštění atributu typu zobrazení z procesu evoluce. Naopak evoluce s nepřímým kódováním jedinců prohledává větší prostor, jelikož konfigurace osy `X` v zobrazeních, na rozdíl od ostatních evolucí, povoluje sloupce s typem `value`. Datová sada knihovny představuje očividně nejmenší prostor potenciálních řešení, ostatní datové sady se zdají být srovnatelné. Obrázky 6.1, 6.2 a 6.3 znázorňují měření sad knihovny (zelené body) a autopůjčovny (červené body). Měření s datovou sadou obchodu s elektronikou není zahrnuto, jelikož jeho výsledky jsou příliš podobné výsledkům na datové sadě autopůjčovny.



Obrázek 6.1: Testování - binární hodnocení



Obrázek 6.2: Testování - vícehledové hodnocení



Obrázek 6.3: Testování - nepřímé kódování

6.2 Testování uživatelů

V této fázi proběhlo testování aplikace třemi uživateli. Přizvaní dobrovolníci nemají žádnou předchozí zkušenost s touto aplikací, ani s žádnou jinou aplikací využívající interaktivní evoluci. Dobrovolníci byli instruováni, aby se zaměřili převážně na srozumitelnost uživatelského rozhraní, shodu průběhu evoluce s jejich očekáváním a rychlost dosažení požadovaného zobrazení. Uživatelům byl vysvětlen význam datových sad, nebylo jim však vysvětleno ovládání aplikace ani způsob hodnocení jednotlivých evolučních procesů.

První uživatel

První uživatel uvedl, že nemá žádné výtky k uživatelskému rozhraní aplikace. Nejlépe hodnotil evoluci s binárním hodnocením. Zdálo se mu, že uživateli poskytuje nejvíc informací. Jeho způsob hodnocení odpovídal předpokládanému způsobu. Nejhůře hodnotil evoluci s nepřímým kódováním, jelikož byla nejpomalejší z možností a zobrazovala výsledky, které nemají žádné reálné využití.

Druhý uživatel

Druhý uživatel uvedl, že jeho jedinou výtkou k uživatelskému rozhraní je rozmístění pohledů jedinců u evolucí hodnocených více pohledy, jelikož netvoří jednotné těleso, ale pohledy jsou zobrazeny čtyři na první řádce a dva na druhé. Z evolucí nejlépe hodnotil druhou evoluci, u evoluce s binárním hodnocením hodnotil způsobem, který byl v rozporu s předpokladem, a proto se mu zdál algoritmus nefunkční (ve dvou případech odřezal algoritmus tu část prostoru, kde leželo jím hledané řešení). Třetí evoluci hodnotil negativně ze stejného důvodu jako první uživatel. Uvedl také, že mu zprvu nebylo jasné, jakým způsobem se zobrazení z evolucí přeneso do obrazovky zobrazení, nicméně že během užívání aplikace na to přišel a že jej nenapadá lepší řešení.

Třetí uživatel

Třetí uživatel měl stejnou výtku k uživatelskému prostředí jako druhý uživatel, neměl však problém s exportováním vyšlechtěného zobrazení. Jako nejlepší také hodnotil druhou evoluci. Hodnocení v první evoluci prováděl shodně s předpokladem a byl spokojený s rychlostí dosažení požadovaného výsledku. Preferoval však schopnost porovnání různých řešení, kterou mu nabízela druhá evoluce. Třetí evoluce mu přišla zajímavá, ne však natolik využitelná jako ostatní dva způsoby.

Zhodnocení

Na základě hodnocení uživatelů jsem opravil zobrazení pohledů tak, aby se na menších obrazovkách zobrazovaly jako mřížka 2x3 a na větších obrazovkách jako řádek šesti pohledů. Zvážil jsem výhrady k použitelnosti některých

výsledků nabízených evolucí s nepřímým kódováním jedinců a rozhodl jsem se pro demonstrační účely ponechat evoluci v jejím současném stavu.

Závěr

Cílem práce bylo prozkoumat možnost využití interaktivní evoluce pro vytvoření požadované vizualizace s co nejjednodušším uživatelským rozhraním a s minimální potřebnou interakcí s uživatelem. Dále bylo cílem vytvořit aplikaci umožňující uživateli parametrizaci zobrazení buď manuálně, nebo s pomocí interaktivní evoluce. Požadavky na tuto aplikaci zahrnují jednoduché uživatelské rozhraní, dostupnost bez nutnosti připojení k internetu a snadnou rozšiřitelnost.

Všechny cíle práce byly naplněny. Výsledek práce byl otestován testovacím skriptem a třemi uživateli. Jejich reakce byly využity pro zlepšení uživatelského prostředí. Aplikace byla navržena tak, aby byla snadno rozšiřitelná v souladu s požadavky. Toho je docíleno díky implementaci šablonového systému a vytvořením srozumitelného rozhraní pro klíčové části evoluce.

Výsledek práce dle mého názoru dobře demonstruje využitelnost interaktivní evoluce pro řešený problém. Během realizace práce jsem získal řadu cenných znalostí v oblasti optimalizačních algoritmů, seznámil jsem se s interaktivní evolucí a získal další zkušenosti s programovacím jazykem JavaScript.

V rámci budoucího rozvoje bych se zaměřil na využití evolučního procesu s nepřímým kódováním tak, aby parametrizoval primárně vizuální stránky zobrazení, které by zobrazovalo data podle uživatelského nastavení na základě metadat.

Literatura

- [1] Eiben, A.; Smith, J.: *Introduction to Evolutionary Computing*. Springer, první vydání, ISBN 3-540-40184-9.
- [2] Poli, R.; Langdon, W. B.; McPhee, N. F.: *A field guide to genetic programming*. <http://lulu.com>, 2008. Dostupné z: <http://www.gp-field-guide.org.uk>
- [3] Takagi, H.: *Interactive Evolutionary Computation: System Optimization Based on Human Subjective Evaluation [online]*. 1998. Dostupné z: <http://alife.tuke.sk/kapitola/846/ines98.pdf>
- [4] Gurney, K.: *An Introduction to Neural Networks*. London: Routledge, 1997, ISBN 1-85728-503-4.
- [5] Renze, J.: Outlier. From MathWorld—A Wolfram Web Resource, created by Eric W. Weisstein. Dostupné z: <http://mathworld.wolfram.com/Outlier.html>
- [6] Sareni, B.; Krahenbühl, L.: Fitness Sharing and Niching Methods Revisited. Dostupné z: <http://jeffclune.com/courses/media/courses/2016-EvoRobot/readings/sareni-ReviewFitnessSharingAndNichingMethods.pdf>
- [7] S. Kirkpatrick, M. P. V., C. D. Gelatt: Optimization by Simulated Annealing. *Science*, ročník 220, č. 4598, 1983: s. 671–680, ISSN 00368075, 10959203. Dostupné z: <http://www.jstor.org/stable/1690046>
- [8] Pintér, J.; Weisstein, E. W.: Tabu search. From MathWorld—A Wolfram Web Resource, created by Eric W. Weisstein. Dostupné z: <http://mathworld.wolfram.com/TabuSearch.html>
- [9] Darrell Whitley, R. B. H., Soraya Rana: The Island Model Genetic Algorithm: On Separability, Population Size and Convergence. 1998. Dostupné z: <http://neo.lcc.uma.es/Articles/WRH98.pdf>

LITERATURA

- [10] Verbanics, P.; Stanley, K. O.: *Transfer Learning through Indirect Encoding*. 2010. Dostupné z: http://eplex.cs.ucf.edu/papers/verbanics_gecco10.pdf
- [11] Jeff Clune, R. T. P., Kenneth O. Stanley; Ofria, C.: *On the Performance of Indirect Encoding Across the Continuum of Regularity*. 2011. Dostupné z: <http://jeffclune.com/publications/2011-CluneEtAl-IndirectEncodingAcrossRegularityContinuum-IEEE-TEC.pdf>

Seznam použitých zkratek

GUI Graphical user interface

HTML Hypertext markup language

CSS Cascading style sheet

FIFO First in first out

BI-ZUM předmět Základy umělé inteligence

XHR XMLHttpRequest

OOP Objektivě orientované programování

PHP PHP: Hypertext preprocessor

Seznam použitých nástrojů a software

JetBrains WebStorm Primární vývojové prostředí

UMLetino Nástroj pro tvorbu UML diagramů, dostupný z <http://umletino.com>

GIMP Grafický editor, využit pro tvorbu příručky

draw.io Nástroj pro tvorbu flowchartů, dostupný z <https://draw.io>

Uživatelská příručka

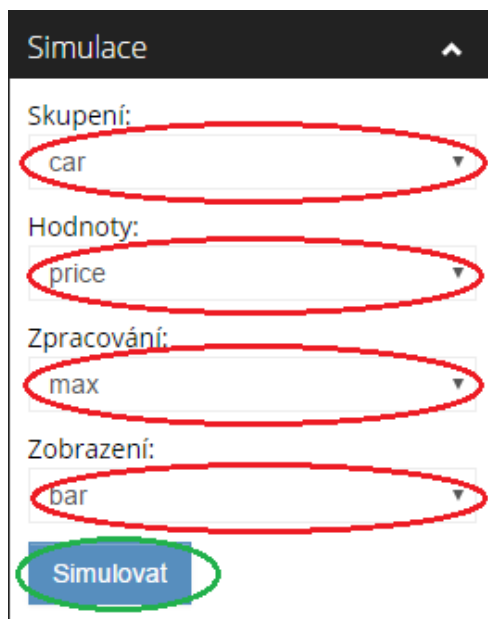
C.1 Obrazovka Data



Obrázek C.1: Obrazovka Data

Tato obrazovka slouží k nahrání dat a metadat, se kterými chcete pracovat ve zbytku aplikace. Pro nahrání souborů využijte vstupních polí vyznačených červeně. Levé pole slouží pro nahrání dat ve formátu .csv, pravé pole pro nahrání metadat ve formátu .csv. Po vybrání souboru s daty se zobrazí informace o nahrané sadě. Zkontrolujte, že počet záznamů zobrazený v modře vyznačené oblasti souhlasí s počtem záznamů v sadě a že názvy atributů v prostředním ze sloupců označených zeleně odpovídají názvům atributů v sadě. Pokud jste nevybrali soubor metadat, budou hodnoty v pravém sloupci odvozeny automaticky podle hodnot prvního řádku datové sady. Pokud nahrajete soubor metadat, změní se tyto hodnoty v souladu s nastavením uvedeným v souboru.

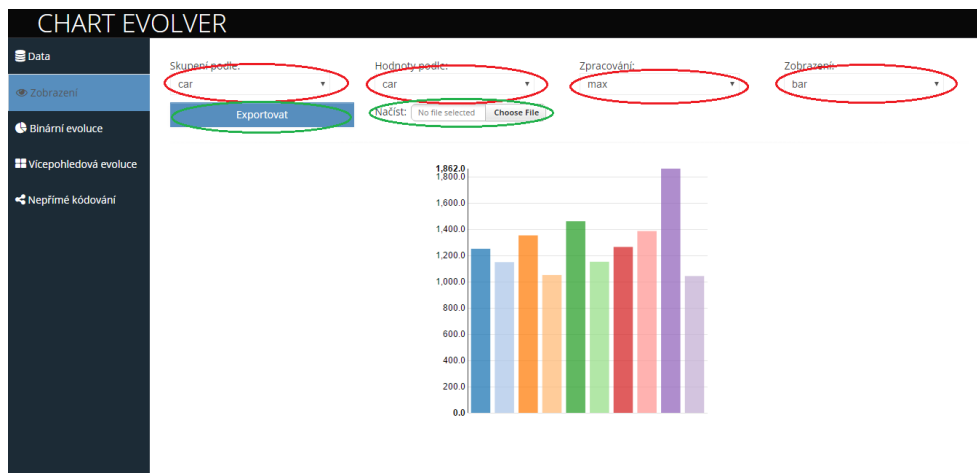
C.2 Panel simulace



Obrázek C.2: Panel simulace

Panel simulace slouží k nastavení a spuštění simulačního procesu. Panel zobrazíte kliknutím na box Simulace na obrazovkách evoluce. Na panelu naleznete čtyři výběrová pole (na obrázku vyznačena červeně). Navolte požadované řešení a klikněte na tlačítko Simulovat (na obrázku vyznačeno zeleně). Po spuštění simulace se panel zavře. V případě obrazovky Binární simulace není na panelu simulace typ zobrazení. Simulace se zastaví po nalezení odpovídajícího řešení. Na obrazovkách s vícehledovým hodnocením simulace vždy nejprve označí vybrané řešení červeným ohraňčením a poté na něj klikne. Po nalezení hledaného řešení je odpovídající pohled ohraňčen zeleně na dobu 4 sekund.

C.3 Obrazovka Zobrazení



Obrázek C.3: Obrazovka Zobrazení

Obrazovka zobrazení slouží k manuální parametrizaci, exportu a importu zobrazení. Pro manuální parametrizaci vyberte požadované vlastnosti zobrazení z červeně vyznačených výběrových polí. K exportu a importu zobrazení slouží ovládací prvky označené zeleně. Jakmile jste se současným zobrazením spokojeni, můžete jej vyexportovat tlačítkem Exportovat. Po stisknutí se automaticky stáhne soubor s názvem display.csv. Chcete-li importovat již uložené zobrazení, využijte vstupního pole Načíst.

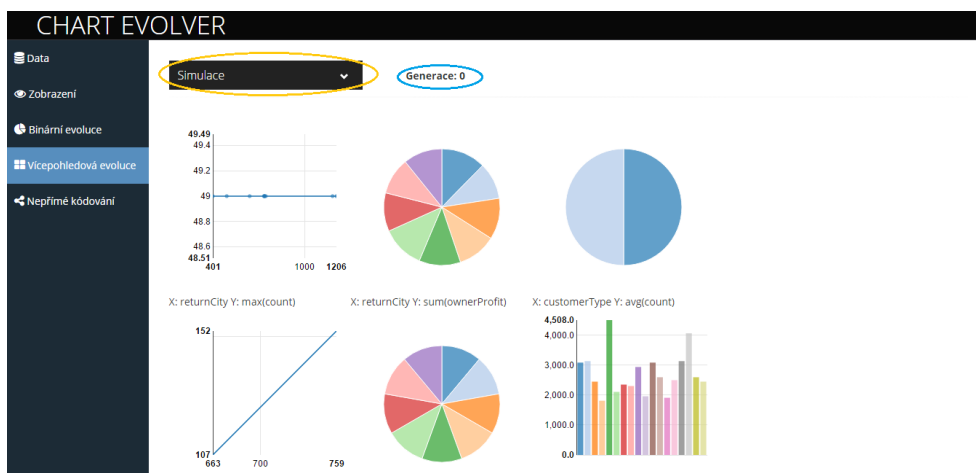
C.4 Obrazovka Binární evoluce



Obrázek C.4: Obrazovka Binární evoluce

Tato obrazovka slouží k parametrizaci zobrazení s použitím interaktivní evoluce s binárním hodnocením. Evoluční proces ovládejte pomocí tlačítek vyznačených zeleně. Vlastnosti aktuálního jedince jsou zobrazeny ve výběrových polích zvýrazněných červeně. Evoluční proces nezahrnuje typ zobrazení, můžete jej manuálně zvolit výběrovým polem označeným oranžově. Tlačítko **Dobrý** použijte, pokud jste spokojeni s většinou vlastností současného jedince. V opačném případě použijte tlačítko **Špatný**. Modře označené oblasti poskytují informace o stavu evolučního procesu. V horní části je uvedeno číslo současné generace, v pravé části jsou uvedeny váhy hodnot jednotlivých atributů. Po kliknutí na žlutě označený prvek **Simulace** se rozbalí panel simulace popsáný v sekci C.2. Při přechodu na obrazovku **Zobrazení** se přednastaví vlastnosti zobrazení podle zobrazeného jedince.

C.5 Obrazovka Vícepohledová evoluce, obrazovka Nepřímé kódování



Obrázek C.5: Obrazovka Vícepohledová evoluce, obrazovka Nepřímé kódování

Tyto obrazovky slouží k parametrizaci zobrazení s využitím interaktivní evoluce s hodnocením více pohledy. Uživatelské rozhraní obou obrazovek je totožné. Pohledy na řešení jsou uspořádány v mřížce 2x3. V případě dostatečného rozlišení pro zobrazení všech pohledů vedle sebe jsou pohledy uspořádány v jedné řadě. V modře označené oblasti je uvedeno číslo současné generace. Pro zvolení jedince pro další evoluci klikněte na příslušné zobrazení. Kliknutím na žlutě označený prvek Simulace zobrazíte panel simulace popsany v sekci C.2. Při přechodu na obrazovku **Zobrazení** se přednastaví vlastnosti zobrazení podle poslední zvoleného jedince.

Obsah přiloženého CD

license.txt	licenční ustanovení
src	
├─ impl	zdrojové kódy implementace
├─ thesis	zdrojová forma práce ve formátu \LaTeX
text	text práce
├─ thesis.pdf	text práce ve formátu PDF