VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

NÁSTROJ PRO TVORBU A OPTIMALIZACI ROZVRHŮ

TOOL TO DESIGN AND OPTIMIZE SCHEDULE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **JAKUB MÁLEK**
AUTHOR

VEDOUCÍ PRÁCE **Ing. VÍTĚZSLAV BERAN, Ph.D.**
SUPERVISOR

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2021/2022

Zadání bakalářské práce

Student: **Málek Jakub**
Program: Informační technologie
Název: **Nástroj pro tvorbu a optimalizaci rozvrhu**
Tool to Design and Optimize Schedule
Kategorie: Uživatelská rozhraní

Zadání:

1. Prostudujte nástroje pro tvorbu moderních webových aplikací a techniky UX. Seznamte se s optimalizačními metodami a podobnými nástroji.
2. Navrhněte nástroj s GUI , který umožní uživateli zadat omezení a podmínky pro tvorbu rozvrhu a automaticky vytvoří návrhy variant s minimem kolizí.
3. Implementujte navržený systém s využitím relevantních dostupných technologií.
4. Vyhodnoťte vlastnosti výsledného systému na základě experimentů s reálnými uživateli. Doplňte o experimenty prezentující optimalizační vlastnosti při automatické tvorbě rozvrhu.
5. Prezentujte klíčové vlastnosti řešení formou plakátu a krátkého videa.

Literatura:

- Brian Burke. *Gamify: how gamification motivates people to do extraordinary things.* Brookline, MA: Garthner, Inc., 2014. ISBN 9781937134853.
- Joel Marsh. *UX pro začátečníky*. Zoner Press, 2019.
- Steve Krug. *Don't make me think, revisited: a common sense approach to web usability.* San Francisco: New Riders, ISBN 978-0321965516.
- Dále dle pokynu vedoucího.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1, 2 a částečně bod 3.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Beran Vítězslav, Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 11. května 2022
Datum schválení: 1. listopadu 2021

Abstrakt

Tato práce se zabývá tvorbou webové aplikace pro skautské kurzy, která by měla ulehčit práci s generováním rozvrhů pro závěrečné ověřování kompetencí. Součástí práce je studium moderních webový technologií pro vytváření webových služeb, genetických algoritmů, návrhem grafického rozhraní, které umožní uživatelům zadat vstupní data pro generování rozvrhů a závěrečným vyhodnocením vzniklé apliace.

Abstract

This thesis deals with the creation of a web application for scouting courses, which should simplify the work of generating timetables for the final competency verification. The work includes the study of modern web technologies for creating web services, genetic algorithms, the design of a graphical interface that allows users to input data for generating schedules and the final evaluation of the resulting application.

Klíčová slova

genetický algoritmus, evoluční algoritmus, návrh GUI, automatická tvorba rozvrhů, webová aplikace

Keywords

genetic algorithm, evolution algorithm, GUI design, automatic scheduling, web application

Citace

MÁLEK, Jakub. *NÁSTROJ PRO TVORBU A OPTIMALIZACI ROZVRHŮ*. Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Vítězslav Beran, Ph.D.

NÁSTROJ PRO TVORBU A OPTIMALIZACI ROZVRHŮ

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Vítězslava Berana, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Jakub Málek
6. května 2022

Poděkování

Děkuji svému vedoucímu práce Ing. Vítězslavu Beranovi, Ph.D. za čas, motivaci a znalosti, které mi poskytl během psaní mé bakalářské práce

Obsah

Kapitola 1

Úvod

Skautských kurzů máme několik druhů[1]. Jedná se zejména o kurzy Vůdcovské a Čekatelské zkoušky. Uchazeči si po projití tímto kurzem odnesou plno zkušeností pronásledující vedení oddílů, kdy se z nich může stát vedoucí oddílu, či celého střediska. Pro splnění kurzu je účastník povinen splnit všechny kompetence, které jsou stanovené ústředím[2]. Kompetence se ověřují pozorováním, splněním úkolu, rozhovorem anebo testem. Většina skautských kurzů, které ověřují kompetence formou rozhovorů, potřebují jakousi formu rozvrhu, pro určení který účastník jde kdy ověřovat kterou kompetenci. V dnešní době používají nástroje podobné Excelu či tabulkám Google. Tvoření rozvrhu jim v pár lidech trvá i několik hodin a výsledek není ideální. Obsahuje různé kolize ať už účastníků, tak vedoucích. V této práci se tedy budu zabývat systémem, který bude umět vygenerovat tento rozvrh na základě vstupních dat, bude umět upravovat vygenerované rozvrhy, uživatelé budou mít přehled o seznamu vedoucích a účastníků.

Tato práce si klade za cíl vytvořil automatický způsob generování rozvrhů za pomocí evolučních algoritmů promítnutý do webové aplikace, ke které budou mít uživatelé přístup z jakéhokoli počítače.

Dále je potřeba pro uživatele vytvořit grafické rozhraní, kterým budou moci generování rozvrhů spouštět. Jednoduché a přehledné zobrazení rozvrhů by se mělo nacházet na úvodní stránce. Rozhraní by mělo být na první pohled pochopitelné a jasné co jaké tlačítko dělá. Prostředí by mělo být přehledné a informace, které spolu souvisí by se měly zobrazovat na jedné stránce, nejlépe bez různých dialogových oken. Uživatel by měl mít možnost zadávat data potřebná pro generování a následně tyto data editovat či mazat.

V závěru práce se podívám na různé testovací sady a jejich výstupy. Propůjčím ovládání aplikace koncovým uživatelům a zjistím jaký mají názor na aplikaci oni.

[1] https://kurzy.skaut.cz

[2] https://krizovatka.skaut.cz/spisovna/smernice/vzdelavani/smernice-k-obsahu-vybranych-cinovnickych-kvalifikaci

Kapitola 2

Technologie

V současné době máme k dispozici mnoho prostředků (ať již dokončených, nebo stále vyvíjených), které nám umožňují relativně jednoduchým způsobem přizpůsobovat aplikace požadavkům dnešní doby.

2.1 Genetický algoritmus

V této a následující kapitole čerpám informace z knihy Genetické algoritmy a genetické programování [3].

Genetický algoritmus patří mezi základní evoluční algoritmy s rysy se vyvíjet. Je převážně založen na Darwinově teorii evoluce [1] postavených na základech dědičnosti, mutace přirozeného výběru, křížení a další. Ukazuje se, že mnoho praktických problémů lze řešit za pomocí analogie procesů probíhající v biologických systémech. Teorie vychází z předpokladu, že jedinec v aktuální populaci s nějakou dobrou vlastností bude mít více potomstva, kteří tuto vlastnost podědí. V potomstvu bude tedy více jedinců s těmito dobrými vlastnostmi a jedinci v ní bude nejméně stejně tak kvalitní jako jejich předek. Tento vztah popisuje následující vzorec, kde α je jedinec a P_t je aktuální populace. Problém není řešen náhodně, ale pro následující krok využívají znalosti optimalizační funkce, pro kterou se přenesl název fitness funkce. Předností algoritmu je, že na rozdíl od ostatních optimalizačních algoritmů zakládají na optimalizaci celé populace na rozdíl od jednoho objektu (chromozomu).

$$\max_{\alpha \in P_{t+1}} F(\alpha) \geqq \max_{\alpha \in P_t} F(\alpha) \tag{2.1}$$

V mém případě jsou jedinci v populaci jednotlivé varianty rozvrhu. Jejich hodnota fitness funkce určuje jejich kvalitu ta závisí na jejich přežití. Jednotlivé rozvrhy jsou v mém případě reprezentována chromozomy, ze kterých se skládá moje populace. V populaci dochází k reprodukci mezi chromozomy, kde stará populace vymírá a z ní se vytváří nová populace. Tento postup lze chápat jako evoluci populace. Proces reprodukce chromozomů chápeme jako dvě operace, a to křížení a mutace. Během křížení vznikají určitým způsobem noví jedinci z rodičovských chromozomů. Pokud bych použil pouze křížení, tak se může moje řešení dostat do lokálního extrému a z toho důvodu se aplikuje operace mutace. Ta spočívá v tom, že se náhodně zamění jeden nebo více genů vzniklého potomka. Může tak vzniknout ještě lepší jedinec, avšak v mnoha případech vznikají jedinci horší, kteří jsou další evolucí většinou eliminováni.

Vznik nového jedince k vytvoření nové populace probíhá tak, že se z aktuální populace vyberou pseudonáhodně minimálně dva chromozomy na základě jejich fitness funkce a za pomocí genetických operátorů vznikne nový jedince. Nový jedince je buď totožný s původním anebo vznikne křížením vybraných chromozomů. Pokud bylo pro vznik nového jedince použito křížení může být na nový chromozom použit operátor mutace. Je možné se do následující generace se dostanou jedinci z předchozí generace tzv. *Elitní jedinci.* Jedná se o několik jedinců s nejvyšší hodnotou fitness. Díky tomuto neztratíme nejlepší jedince, o které bychom v průběhu evoluce mohli přijít.

Na kódování v obecném genetickém algoritmu záleží následující sestavení fitness funkce. Mým úkolem je najít vhodné řešení, které bude vyhovovat často i protichůdným požadavkům. Jedná se zejména o rychlé vyhodnocení fitness funkce, jednoduché určení platnosti daného řešení, malá paměťová náročnost a jednoduchá konverze genotypu na fenotyp. Mezi základní metody patří kódování binární, grafové, permutační a další. Často bývají kódovací metody speciálně vytvořené pro daný řešený problém (Mohou vznikat úpravou již známých metod).

Binární kódování je jedno z nejběžnější kódování chromozomů. V nejjednodušším případě se jedná o binární reprezentaci vektoru číselných proměnných, se kterou se pracuje bitově. Tyto vstupy mohou být spojité, ale v programu je nutné je diskretizovat. Při použití binárního kódování může dojít na případ, že se dvě vedlejší hodnoty liší ve všech bitech (př. 7 (0111) a 8 (1000))[1]. Toto je pro použití v genetickém algoritmu nežádoucí, protože v případě, že hledané optimální řešení leží za bariérou, je její překročení více než nepravděpodobné. Z tohoto důvodu se používá tzv. *Grayovo kódování.* které zaručí že mezi všemi následujícími hodnotami je rozdíl právě v jednom bitu, tedy Hammingova vzdálenost [7] je 1. Dalším řešením může být zvážit tento problém při implementaci řešení a explicitně ji ošetřit. Lze použít operátor inverzní transformace, který invertuje bity chromozomu.

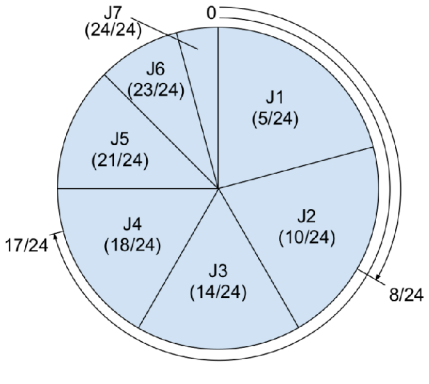
Permutační kódování se využívá zejména pro plánovací úlohy z důvodů zachování pořadí prvků, které se nesmí opakovat. Toto přímo odpovídá mému problému, ve kterém hledáme permutaci jednotlivých bloků v rozvrhu. Permutační úlohy jsou jedny z prvních, kde se ukázalo, že použití genetického algoritmu dává smysl. Řešení bývá většinou velmi těžké, protože při permutaci o velikosti N je N! Různých řešení. Ukázky řešení úloh obchodního cestujícího nebo problém dvou loupežníků jsou popsány v knize Evolučné algoritmy [4].

Darwinova teorie o původu druhů [1] popisuje přirozený výběr jedinců k vzniku nového jedince. Tuto teorii se snažím simulovat i v rámci genetického algoritmu. Existuje mnoho strategií výběru chromozomů, kde se všechny snaží docílit propagace dobrých vlastností jedinců do další generace. Zároveň se snaží zachovat dostatečnou pestrost této populace. Pokud chceme zachovat vlastnosti fitness funkce, nesmíme pouze vytvářet nové jedince, ale musíme zajistit, aby se někteří jedinci dostali nezměněni do další populace. Lze to zařídit tak, že některé jedince s určitou mírou pravděpodobnosti nebudeme křížit, ale pouze je zkopírujeme do další populace. Pravděpodobnost se většinou pohybuje v rozmezí 0.05–0.25. Ruletový a Turnajový mechanismus jsou nejběžnějšími metodami výběru. Jsou popsány v následujících kapitolách. Při podrobnějším zkoumání jsou obě metody stejně dobré, avšak při vhodném nastavení parametrů má turnajový mechanismus lepší časovou náročnost.

Ruletový mechanismus vizualizován obrázkem 2.1 si lze představit jako klasickou ruletu, kde jsou jednotlivé výseče různě veliké, na základně fitness hodnoty každého jedince. Tímto způsobem mají jedinci s větší plochou na ruletě (větší fitness hodnota), větší šanci na přežití a vstoupení do další populace. Jednoduchá implementace je znázorněna na obrázku níž. Na

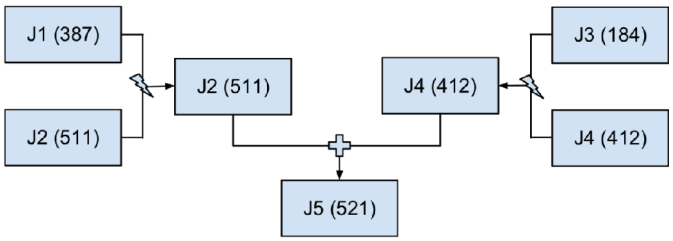
[1]Tento jev se nazývá Hammingova bariéra. V informatice se pro nejmenší počet pozic, na kterých se řetězce stejné délky daného kódu liší, používá název Hammingova vzdálenost

ruletu je rozmístěno sedm jedinců (J1–J7), kteří mají normalizovanou svoji fitness hodnotu tak, aby byl součet roven jedné. Seřazení jedinců, podle hodnoty fitness (jak je zobrazeno na obrázku) není podmínkou. Při spuštění selekce se vygenerují dvě náhodná čísla a podle toho do kterého intervalu jedince spadnou, jsou následně tito dva jedinci vybráni jako favorité. (V ukázce čísla 8/24 a 17/24 vyberou jedince J2 a J4).



Obrázek 2.1: Ruletový mechanismus [2].

Turnajový mechanismus, znázorněný na obrázku 2.2 je silně inspirován procesy z přírody, kde se jedinci navzájem utkávají v boji o přežití. Princip metody je v tom, že se z populace vyberou minimálně dva jedinci, kteří se utkají v boji o přežití, přičemž vyhrává ten s lepším ohodnocením. Tento jedince je poté zařazen do reprodukčního procesu s dalším jedincem, který byl vybrán podobným způsobem. Čím více jedinců v turnaji, tím vzniká větší selektivní tlak, proto se ve většině případů vybírají pouze dva jedinci. Aby metoda neměla tendenci uváznout v lokálním maximu, v případě, že by jeden jedince z populace byl nadprůměrný, přidává se malá pravděpodobnost, že vyhraje slabší jedince. Tento postup dodává rozmanitost řešení a zabraňuje zmíněnému uváznutí v extrému. Následující obrázek znázorňuje princip turnajového mechanismu. Dva jedinci J1 a J2 a dva jedinci J3 a J4, kteří jsou náhodně vybráni z populace, se utkají v souboji. Vítěz, tedy ten s vyšším hodnocením, postupuje do reprodukce a společně vytvářejí jedince J5.



Obrázek 2.2: Turnajový mechanismus [2].

Jedna z dalších důležitých věcí je přechod do další generace a práce s generací předchozí. Touto volbou můžeme ovlivnit rychlost konvergence, rozmanitost dalších populací, nebo přežití nejlepších jedinců do další populace, abychom o nejlepší jedince nepřišly. Nejjednodušší řešení je nechat předešlou populaci umřít. V tuto chvíli ztratíme všechny předešlé potomky, což by nám nevadilo, pokud by platil vztah, že následující potomek je stejně dobrý

nebo lepší než jeho rodič. To v reálném případě bohužel neplatí. Pro bezpečnost, abychom neztratili nejlepší jedince, překopírujeme je do další generace a zároveň se také zúčastní reprodukčního procesu. Tím pádem máme v nové generaci nejlepší jedince z té předchozí a také nové jedince, kteří vznikli křížením. Tato technika se nazývá elitismus. V některých případech můžeme přechod do další populace omezit tak, že budeme filtrovat a mazat duplicitní jedince. Touto technikou se dá oddálit problém předčasné konvergence k lokálnímu extrému.

Poslední důležitá otázka při používání genetického algoritmu je jeho ukončení. Nejjednodušší cestou můžeme konec algoritmu zvolit počtem prošlých generací. Tato hodnota se musí nastavit před spuštěním a nelze ji během chodu měnit. Může tedy nastat předčasné ukončení, kdy algoritmus nenajde správné řešení, nebo naopak algoritmus může uvíznout v lokálním extrému a pak běží zbytečně dlouho. Lepší varianta je sledovat stav a rozmanitost populace a v případě podobnosti jedinců se dá usuzovat, že se algoritmus dostal do lokálního extrému a můžeme jej ukončit. V případě, že víme, jaké podmínky má výsledný stav splňovat, můžeme genetický algoritmus ukončit na základě těchto podmínek.

2.2 Křížení, mutace a výpočet fitness hodnoty

Křížení má v genetice důležitou roli, ze dvou a více rodičů se vyberou ty dobré geny a z nich se vytvoří nový a lepší potomek. Mutace pak může náhodně nějaký gen změnit a tím dodat do nové populace rozmanitost. Je důležité, aby tyto operátory vytvářely pouze korektní řešení a nebylo pak dále nutné nějaké neplatné členy zahazovat. Bez křížení by genetický algoritmus postrádal smysl, jedná se o klíčovou vlastnost. která z několika rodičů vytvoří podobné potoky. Kdybychom vynechali mutaci, tak má genetický algoritmus tendence uváznout v lokálních extrémech. Dále pak existují další individuální operace, které mohou řešit problémy dané úlohy. Následující odstavce vysvětlují binární jednobodové křížení, které je jednoduché a často používané a permutační křížení, které je relevantní k mému problému.

Jednobodový operátor křížení je jednoduchá operace pro vektory pevné délky. Lze ji použít na binární i permutační kódování, avšak nezachovává platnost permutace, a tak je potřeba ještě navrhnout opravný algoritmus. Operátor funguje tak, že zvolí jeden stejný bod v obou chromozomech a části rodičovských chromozomů se vymění na základě tohoto bodu. V případě, že se zvolí více bodů křížení, jedná se o K-bodové křížení. Využívá se zejména pro binární reprezentaci. Vygeneruje se jeden náhodný bod z množiny <0; N-1>, v tomto místě se chromozomy rozdělí a kombinací pravé části jednoho a levé části druhého, vzniknou dva noví jedinci.

$$\begin{array}{c}(1,1,0 \mid 1,0,0,1,0)\\(0,1,0 \mid 0,0,1,1,0)\end{array} \Longrightarrow \begin{array}{c}(1,1,0 \mid 0,0,1,1,0)\\(0,1,0 \mid 1,0,0,1,0)\end{array} \tag{2.2}$$

$$\begin{array}{c}(1,1,0 \mid 1,0 \mid 0,1,0)\\(0,1,0 \mid 0,0 \mid 1,1,0)\end{array} \Longrightarrow \begin{array}{c}(1,1,0 \mid 0,0 \mid 0,1,0)\\(0,1,0 \mid 1,0 \mid 1,1,0)\end{array} \tag{2.3}$$

Permutační křížení nám po skončení musí zachovat permutaci, takže nejjednodušší způsob je použití permutačních operátorů křížení (Každý prvek chromozomu se vždy bude vyskytovat pouze jednou). Lze toho dosáhnout dodatečnou úpravou anebo lze algoritmus přímo takto navrhnout. Jednotlivé algoritmy se liší v tom, jak informace z předků zachovávají.

Nejznámější je operátor křížení s částečným přiřazením (PMX – Partially-mapped crossover). Je založen na dvoubodovém křížení, kde se na závěr provede opravný algoritmus. Byl

vyvinut se záměrem využít uspořádané podmnožiny permutace jednoho z rodičů a přitom zachovat pořadí a pozici co nejvíce genů z druhého rodič. Následující příklad objasní jeho princip.

Mějme dva rodiče p1 a p2 se zvolenými body křížení.

$$p_1 = (1, 2, 3|4, 5, 6, 7|8, 9) \quad p_2 = (4, 5, 2|1, 8, 7, 6|9, 3)$$

Při vytváření potomků se v prvním kroku vymění prostřední části a dle vyměněných prvků se vytváří přepisovací pravidla pro konečný krok algoritmu.

$$o_1 = (\square, \square, \square|1, 8, 7, 6|\square, \square) \quad \dot{o}_2 = (\square, \square, \square|4, 5, 6, 7|\square, \square)$$
$$k = \{(1 \leftrightarrow 4), (8 \leftrightarrow 5), (7 \leftrightarrow 6), (6 \leftrightarrow 7)\}$$

Druhý krok vyplní krajové geny chromozomů, podle toho, které se prozatím v potomku nevyskytují

$$o_1 = (\square, 2, 3|1, 8, 7, 6|\square, 9) \quad a_2 = (\square, \square, 2|4, 5, 6, 7|9, 3)$$

Poslední krok vyplní zbývající místa na základě přepisovacích pravidel z kroku prvního

$$o_1 = (4, 2, 3|1, 8, 7, 6|5, 9) \quad o_2 = (1, 8, 2|4, 5, 6, 7|9, 3)$$

Druhý často používaným algoritmem, který tu zmíním je křížení založené na zachování pořadí (OX – order crossover). Tento algoritmus vyjme z prvního rodiče uspořádanou množinu genů. Tyto geny vymaže z druhého rodiče a poté oba rodiče spojí do nového potomka. Pro představení si to opět ukážeme na příkladu.

Máme opět dva rodičovské chromozomy následujícího tvaru

$$p_1 = (a, b, c, d, e, f, g, h, i) \quad p_2 = (h, g, d, i, a, b, e, f, c)$$

Zvolíme si dva body křížení, které ohraničí zmíněnou posloupnost genů. Pro náš příklad jsem zvolil body 4 a 7, tedy nám vznikla následující posloupnost s

$$s = (e, f, g)$$

Geny, které jsou shodné v druhém rodiči s množinou s vymažeme. Vznikne pomocná množina $p2'$

$$p'_2 = (h, d, i, a, b, c)$$

Následně nemůže spojením dojít k porušení permutace, a tak výsledné množiny s a $p2'$ spojíme na základě zvolených bodů křížení a vznikne nám výsledný potomek.

$$o = (h, d, i, a, e, f, g, b, c)$$

Poslední metoda, o které se zmíním je křížení založené na zachování pozice (PBX – position-based crossover). Je založené na metodě se zachováním pořadí s rozdílem, že v tomto případě vybrané geny nejsou těsně za sebou, ale jsou vybrány náhodně. Opět si pro ujasnění uvedeme krátký příklad.

Mějme rodiče stejného typu jako doposud

$$p_1 = (a, b, c, d, e, f, g, h, i) \quad p_2 = (h, g, d, i, a, b, c, f, c)$$

Nyní vygeneruje bitovou masku pro náhodný výběr genů z chromozomu

$$l = (0, 1, 0, 1, 1, 0, 0, 1, 0)$$

Dle masky vybereme z prvního rodiče příslušné prvky a uchováme jejich pozici, z druhé rodiče smažeme geny stejné hodnoty

$$o = (\square, b, \square, d, e, \square, \square, h, \square)$$
$$p_2' = (g, i, a, f, c)$$

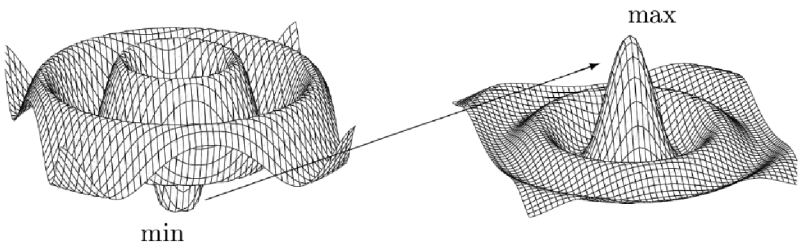
Do volných pozic potomka jsou postupně vloženy geny ze zbytku druhého rodiče. Tímto způsobem nám vznikne kompletní potomek

$$o = (g, b, i, d, e, a, f, h, c)$$

Existují i další permutační metody, kterými se tu ale nebudu zabývat, protože to není v mém případě potřeba.

Mutace je podstatně lehčí metoda než operace křížení. Náš požadavek je, aby jedinec, který projde operátorem mutace byl blízký jedinci před mutací. Většinou stačí pozměnit genotyp jedince. Při běžném kódování se udělá náhodná změna na jednom či více genů. V případě permutačního kódování se dva nebo více genů vymění.

Fitness funkce hodnotí kvantitativní míru schopnosti jedince přežít a vstoupit do reprodukčního procesu. Biologický ekvivalent je zobrazení genotypu na fenotyp, kde chromozom kóduje organismus, jehož schopnost reprodukce je dána fitness daného chromozomu. Fitness funkci lze získat z transformace účelové funkce. Umí ohodnotit kvalitu vybraného chromozomu. Účelová funkce zobrazuje ohodnocení stavového prostoru s nejlepší hodnotou v minimu, kdežto fitness funkce je její transformací a nejlepší řešení se nachází v jejím nejvyšší hodnotě. Vizuálně je toto zobrazeno na obrázku 2.3.



Obrázek 2.3: Zobrazení účelové funkce (vlevo) na fitness funkci (vpravo) v prostoru všech možných chromozomů, které jsou přetransformovány do dvoudimenzionálního prostoru [2].

2.3 Webová aplikace

Vývoj webových aplikací znamená v dnešní době rychlý výsledek a možný přístup bez ohledu na místo nebo zařízení. Díky pečlivě navrženému designu z kapitoly 2.3, a promyšlenému výběru vhodné technologie dokážeme využít potenciál webových aplikací naplno. V dnešní době jsou na vzestupu modernější technologie a aplikace se postupně dostávají na web. Pro vývoj frontendové části aplikace má vývojář k dispozici široké spektrum technologií, mezi které patří například PHP, .NET, JavaScript, Python, Java a další. Pro můj případ jsem zvolil moderní knihovnu React[2]. V tomto směru jsem také mohl zvolit framework Angular anebo Vue.js. Framework Angular je oprati Reactu až moc komplexní a vhodný spíše pro velké a robustní projekty. Udržuje programátory na jedné vlně a také nemají moc možnost se projevit v tom co doopravdy umí. Naopak Vue je zase až moc volné a moc otevřené. Programátor zde píše do souborů s příponou .js. React jako takový umožňuje vytvářet komponenty, které do sebe umí zanořit, má speciální typy souborů s příponou .jsx, které jsou speciálně navrženy pro React. Komponenty Reactu jsou většinou psané pomocí JSX[3]. React zohledňuje skutečnost, že logika vykreslování je neodmyslitelně spjata s další logikou uživatelského rozhraní: jak jsou zpracovávány události, jak se mění stav v čase a jak jsou data připravována k zobrazení. Navíc k tomu je React bezplatný a open-source a také existuje rozšíření zvané Typescript, která doplňuje chybějící typovou kontrolu do javascriptu.

Při výběru technologie použité pro vývoj serverové části máme také pestrou paletu různých programovacích jazyků a jejich knihoven či frameworků. V mém případě jsem si vybral vysokoúrovňový jazyk, který je dobře známý a to Python. Důvodem výběru toho jazyka je základní znalost programování a nalezené knihovny a frameworky, které mi usnadní následný vývoj. V tomto jazyce existuje mikro webový framework Flask[4], který zajišťuje přijímání a odesílání požadavků do aplikace. Dále jsem si v tomto jazyce našel knihovnu PyGAD[5], která mi pomůže s implementací genetického algoritmu. K serverové části neodmyslitelně patří také databázový server a databáze. Zde jsem se rozhodl pro objektově-relační databázový systém PostgreSQL[6], který je zdarma a je open-source. Funkce PostgreSQL zahrnují databázové transakce s atomicitou, konzistencí, izolovaností a trvalostí (ACID), splňující aktualizovatelné pohledy, triggery, cizí klíče a uložené procedury.

2.4 Existující řešení

Pro porovnání, zda již neexistuje nějaký program, který by splňoval potřeby mých uživatelů jsem prostudoval několik již existujících možností. Mohl bych zde zmínit velmi používané softwary, které se běžně používají ve školách, jako jsou Bakaláři, Edupage, Škola online a další. U těchto aplikací je problém, že rozvrh se dá vygenerovat pouze do tabulky s časy a dny v týdnu. V mém případě se jedná o potřeby, které jsou v tomto směru jedinečné a pro vedoucí skautských kurzů byly a jsou nejlepší variantou tabulky Google.

Dlouho jsem za pomocí vyhledávače Google a různých frází, jako např. *"scheduling meetings generator"*, *"automatic schedule generator"*, *"scheduling meetings generator in large team"*, hledal nějaký podobný případ mé aplikace. Bohužel se mi nic nepovedlo najít.

[2]Knihovno javascriptu pro vytváření uživatelských rozhraní. Znalosti čerpány z [8]

[3]Rozšíření Reactu pro syntaxi jazyka javascript, které poskytuje způsob, jak strukturovat vykreslování komponent pomocí syntaxe známé mnoha vývojářům. Svým vzhledem se podobá jazyku HTML

[4]https://flask.palletsprojects.com/en/2.1.x/

[5]Více popsaná v kapitole 4.1

[6]https://www.postgresql.org

Všechny výsledky byly buď zasazeny do týdenního nebo měsíčního kalendáře anebo byly pouze pro jedno člena týmu. Žádný z vyhledaných výsledků neměl možnost generovat tento typ rozvrhů.

Jediné srovnání, které mohu v tomto směru dělat, je použití metod pro generování u již zmíněných aplikací používaných ve školách. Zmínění Bakaláři[7] používají pro generování metodu, kterou nelze srovnávat s řešením této práce. Využívají deterministického algoritmu, který potřebuje asistenci člověka. Další aplikací je EduPage[8], patřící pod aSc Rozvrhy[9]. Jedná se o mezinárnodí firmu vyvíjející software pro generování rozvrhů do škol. Generování rozvrhů je zde velmi rychlé i pro rozsáhlé problémy. Je tu možnost dodatečných úprav již vygenerovaného rozvrhu s asistencí kolizí a případným napovídáním. Ovládání celé aplikace je velmi intuitivní, tím pádem ho zvládne ovládat i nezkušený uživatel. Poslední zmíněný systém, který hojně využívají školy v České republice, je Škola online[10]. Jedná se o podobný software jako Bakaláři, bohužel se mi nepodařilo dohledat jakým způsobem tento program generuje rozvrhy.

[7] Dostupné na https://www.bakalari.cz
[8] Dostupné na https://www.edupage.org
[9] Dostupné na https://www.asctimetables.com/
[10] Dostupné na https://www.skolaonline.cz

Kapitola 3

Návrh rozvrhu pomocí evolučního algoritmu

Pro správně postavený systém, je důležitý podrobný a přesný návrh řešení. V této kapitole se pokusím detailně popsat všechny části sestavovaného systému. Vysvětlím obecnou definici problému, tak jak ji vidí uživatelé. V architektuře systému se podíváme na návrh shora, ukážeme si jak přijímat a reprezentovat data, jak na síťovou architekturu, velikost daného problému.

Hlavní jádro problému je řešeno pomocí genetického algoritmu, na který se podíváme v následujících kapitolách, kde navrhnu klíčovou vlastnost, a to kódování dat pro použití genetického algoritmu. V další kapitole se pokusím definovat vnitřní chování algoritmu, jak by mělo probíhat křížení, mutace a následný výpočet fitness funkce[1]. V kapitole 3.5 shrnu potřeby datového modelu a obecně uchovávání dat. Na závěr této kapitoly vytvořím návrh grafického prostředí pro budoucí uživatele.

3.1 Definice problému

V létě 2021 jsem se zúčastnil letního Vůdcovského kurzu, kde jsem si všiml problému, který byli nuceni řešit skautští vedoucí. Na závěr akce, kdy probíhalo tradiční ověřování kompetencí, museli sestavovat rozvrh za pomocí několika lidí, hodin a tabulek Google. Řekl jsem si, že jim v tomto problému mohu pomoci, a tak jsem se s hlavní představitelkou sestavování rozvrhu potkal a probrali jsme detailně daný problém.

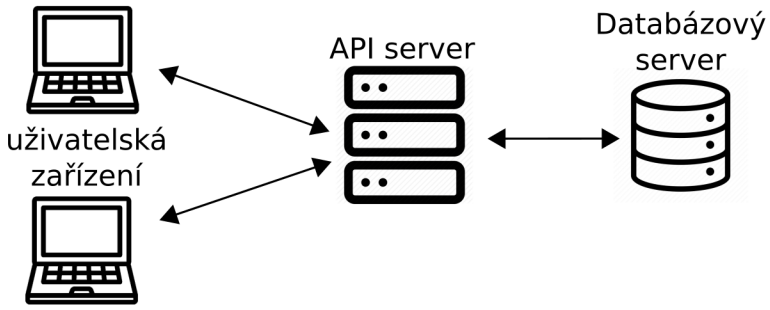
Hlavní potřebou je ze vstupních dat vygenerovat několik verzí různých rozvrhů, kterými si vedoucí bude moci listovat a případně je upravovat. Vstupní data obsahují seznam účastníků, vedoucích, kompetencí a kombinací mezi vedoucími a kompetencemi, tedy jakou kompetenci jaký vedoucí ověřuje s tím, že více vedoucích může ověřovat více kompetencí. U každého účastníka a vedoucího je nutné ukládat pouze jeho jméno, příjmení a případnou přezdívku. Kompetence jsou předem dané a nemění se, skládají se z názvu a detailního popisu. Rozvrh by měl mít uložený název a případné ohodnocení.

Vedoucí většinu času používají na kurzu pro plánování programu a dalších záležitostí notebooky. Aplikace by měla mít možnost se synchronizovat napříč různými zařízeními. Pro tyto potřeby jsem se rozhodl pro webovou aplikaci.

[1] Více popsáno v kapitone 2.1

3.2 Architektura systému

Pro tvorbu aplikace jsem se rozhodl pro síťovou architekturu klient-server,znázorněnou na obrázku 3.1, právě z důvodů oddělení klienta a serveru pro synchronizaci dat napříč několika zařízeními. Na klientské straně upřednostňuji webovou aplikaci, která není potřeba instalovat a je k dispozici téměř kdekoliv. Bude umět komunikovat se serverem, od kterého bude získávat potřebná data, a naopak bude i umět data na server zasílat. Server bude mít za úkol obsluhovat klienty za pomocí předem sestaveného API, komunikovat s databázovým serverem pro získávání a zapisování dat a zejména bude obsahovat modul, který bude za pomocí genetického algoritmu generovat nové rozvrhy.



Obrázek 3.1: Architektura klient-server.

3.3 Kódování chromozomu

Tvorba jakéhokoli rozvrhu v dnešní době je velmi komplexní a těžko ji dělat pomocí přímočarých algoritmů. Pokud bychom se bavili o všech možných řešení, tak je lze spočítat, ale projít všechna a najít nejoptimálnější výsledek není téměř možné. V mém případě existují 2 položky (účastníci a ověřované kompetence), které je třeba rozdělit mezi vedoucím a jednotlivé časové bloky. Pro všechny tyto entity existují pravidla, jak musí být umístěny, případně jak by bylo optimální je umístit. Velikost stavového prostoru se dá v mém případě odhadnout následující rovnicí, kde V představuje počet vedoucích, K počet kompetencí, $Č$ je počet časových bloků a $Ú$ je počet účastníků.

$$(\mathrm{KU})^{\mathrm{VČ}} = 1.2 * 10^{625}$$

Tato rovnice zahrnuje všechna řešení, která mohou nastat, bez ohledu na řešení,která nedávají smysl. Pokud bychom v rovnici počítali s logickými řešeními, dostali by jsme se na nižší počet řešení. I tak by se ale jednalo o počet, který není v praxi v konečném čase možné projít.

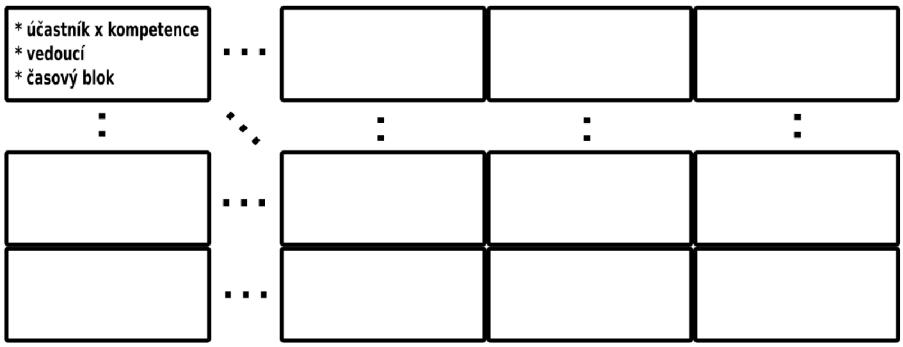
Položka	**Počet položek v tradičním kurzu**
V – Vedoucí	15
K – Kompetence	20
Č – Časové bloky	15
Ú – Účastníci	30

Tabulka 3.1: Tabulka reálných hodnot

Snahou genetického algoritmu je rozmístit všechny požadované skupiny do prostoru rozvrhu tak, aby mezi jednotlivými bloky nevznikly žádné kolize a aby byl rozvrh co nejvíce optimalizován.

Vhodné kódování chromozomu je jedním z nejtěžších úkolů při použití genetického algoritmu. Kódování ovlivňuje další části návrhu, ať už jde o křížení, mutaci nebo výpočet fitness hodnoty. Naším úkolem je tedy najít takové kódování chromozomu, kde se dá rychle vypočítat fitness hodnota, provádět genetické operace a má nízký nárok na paměť.

Je důležité si uvědomit, jaká data potřebujeme zpracovat, v jakém formátu je předáme genetickému algoritmu a v jaké podobě se data vrátí z genetického algoritmu zpět. Kódování chromozomu ukazuje obrázek 3.2, který je lehce zjednodušen. Chromozom představuje jeden celý rozvrh a jednotlivé geny v něm představují jednotlivá ověřování. Chromozom je pevné délky, který má velikost podle kartézského součinu všech kompetencí a účastníků. Každý gen pak nabývá tří hodnot, a to položky z kartézského součinu (účastník x kompetence), vedoucího a daného časového bloku. Tyto tři hodnoty pak umožní zasadit daný gen do rozvrhu.



Obrázek 3.2: Kódování chromozomu.

3.4 Funkce pro křížení, mutaci a výpočet fitness

Důležitou částí je, jak se algoritmus bude chovat uvnitř. Před spuštěním algoritmu je potřeba vygenerovat počáteční populaci, ze které bude algoritmus vycházet. Následovné chování nám určují tři základní funkce, a to funkce pro křížení, funkce pro mutaci a funkce pro výpočet hodnoty fitness.

Fitness funkce by se měl skládat z několika dílčích funkcí, které budou hodnotit jednotlivé kolize a optimalizaci. Problém jsem rozdělil na 3 funkce, které řeší kolize a 2 funkce, které řeší optimálnost. Všechny funkce budou vracet normalizovanou hodnotu v rozmezí 0 a 1, kde 1 znamená, že v daném řešení není žádná kolize, nebo že je nejoptimálnější. Všechny vrácené hodnoty následně sečtu s koeficientem důležitosti[2] tak, aby celková fitness hodnota byla opět normalizovaná.

$$\text{fitness} = \sum w_i * f_i$$

$$\begin{gathered} w_i \ldots \text{Koeficient důležitosti} \\ f_i \ldots \text{Jednotlivé části fitness} \\ \text{Pro } w_i \text{ platí, že } \sum w_i = 1 \end{gathered} \tag{3.1}$$

[2]Mnou definovaná hodnota, určující důležitost jednotlivých částí fitness funkce

Následující odstavce popisují jednotlivá kritéria :

- Stejné geny – jedna položka z kartézského součinu má unikátní kombinaci vedoucího a časového bloku. Toto kritérium je velmi důležité, a proto je zde aplikována exponenciální funkce. Výsledná fitness hodnota je založena na následující rovnice, která vyjadřuje průměr exponenciálního ohodnocení kolizí všech položku z kartézského součinu (účastník x kompetence).

$$f_1 = \frac{1}{n} * \sum_{i=1}^{n} 0.5^{M_i}; w_1 = 0.33$$
$$n \ldots \text{počet genů}$$
$$M \ldots \text{počet kolizí genu} \tag{3.2}$$

- Kolize účastníka – jeden účastník má v jeden časový blok nejvýše jedno ověřování. Také se jedná o důležité kritérium, z toho důvodu je zde použito exponenciální rozdělení. Rovnice vyjadřuje průměr kolizí každého účastníka

$$f_2 = \frac{1}{n} * \sum_{i=1}^{n} 0.5^{M_i}; w_2 = 0.3$$
$$n \ldots \text{počet účastníků}$$
$$M \ldots \text{počet kolizí genu} \tag{3.3}$$

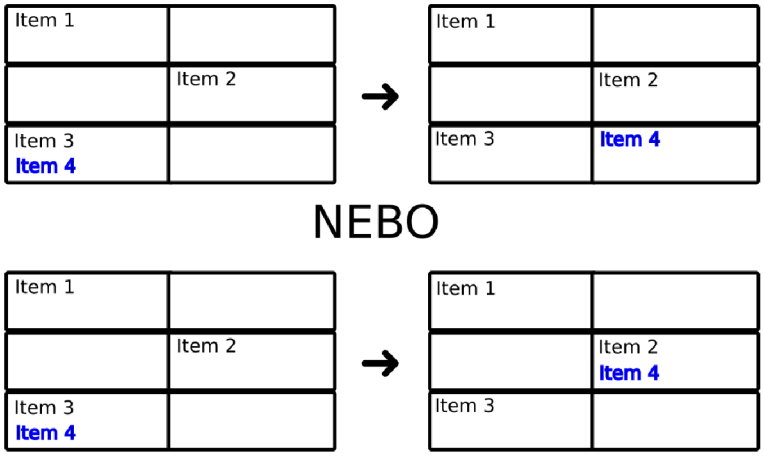
- Kombinace vedoucí – kompetence – Každý vedoucí má kvalifikaci na ověřování určitých kompetencí, které může ověřovat, jiné nelze. Toto kritérium, je hodnoceno taktéž exponenciálně, z důvodu přísného hodnocení.

$$f_3 = 0.5^{M}; w_3 = 0.33$$
$$M \ldots \text{počet nesprávných kombinací} \tag{3.4}$$

- Hustota účastníka a vedoucího – Určuje míru zaplnění mezi prvním a posledním blokem každého účastníka / vedoucího. Tato položka není až tak podstatná, proto je pro její ohodnocení použito lineární rozdělení.

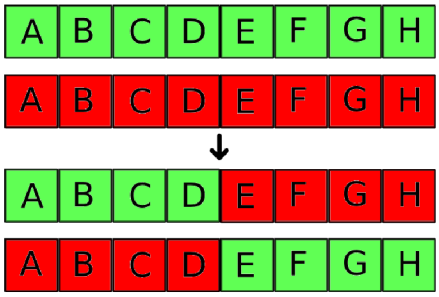
$$f_4, f_5 = \frac{1}{n} * \sum_{i=1}^{n} V_n; w_4 = 0.02; w_5 = 0.2$$
$$n \ldots \text{počet genů}$$
$$V \ldots \text{počet neobsazený bloků} \tag{3.5}$$

Další fází genetického algoritmu je mutační funkce, která řeší náhodnou změnu genu v některých chromozomech. V mém případě bych rád preferoval ty geny, které způsobují nějaké kolize. V první řadě kolizi stejného genu, kdy se dvě položky nacházejí na jednom místě a ve druhé řadě kolize kombinace vedoucího a kompetence, kdy je k vedoucímu přiřazena špatná kompetence. Samotná mutace je řešena tak, že se náhodně zvolenému genu, z předem vybrané množiny, přiřadí nový náhodný vedoucí nebo časový blok anebo obojí. Mutace není ošetřena před vznikem nové kolize, jak ukazuje obrázek 3.3. Tím pádem může vytvořit nový chromozom, který má špatné vlastnosti, ale i takový, který bude lepší než jeho předek.



Obrázek 3.3: Příklad mutací.

Poslední funkcí, která je součástí vnitřní stavby genetického algoritmu je funkce pro křížení. Hlavní funkcí křížení je z aktuálních potomků vytvořit novou populaci. Při vytváření funkce musíme klást důraz na jednoduchost, ale zároveň pestrost vytvořených potomků. Jelikož budou geny v chromozomu uspořádané, podle položek z kartézského součinu (účastník x kompetence), lze použít jedno nebo více bodové binární křížení, kde lehce kombinovat začátek jednoho a konec druhého chromozomu. V případě obrázku 3.4 vytvářím ze dvou rodičů dva potomky. V případě, že bych chtěl vytvářet více potomků ze dvou rodičů, stačí chromozom rozdělit na více částí a různě nakombinovat.



Obrázek 3.4: Příklad křížení.

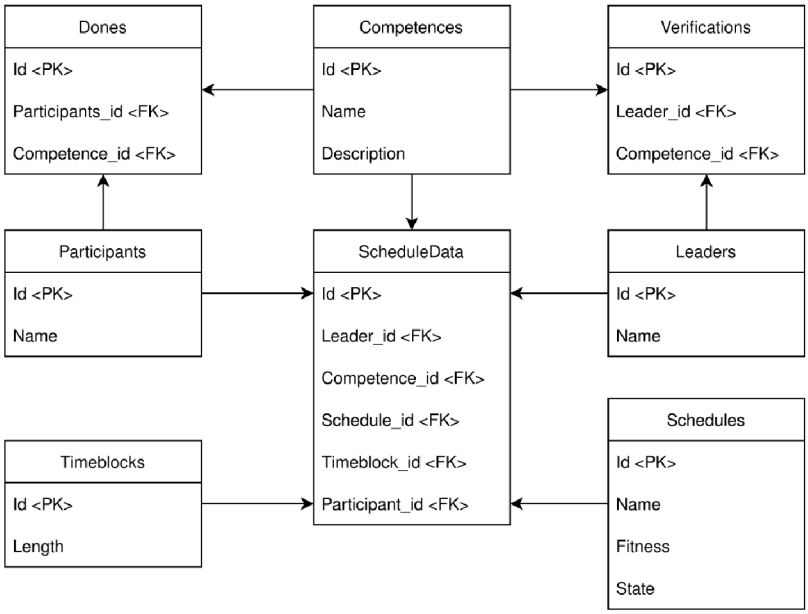
3.5 Datový model

Důležitým bodem pro dobrou aplikaci je mít navržený správný datový model. V případě, špatného nebo chybějícího modelu se může stát, že během implementace budeme nuceni předělávat již funkční databázi. K tomuto jsem použil vizualizaci za pomocí ER-diagramu[3], který se používá pro abstraktní a konceptuální znázornění dat a ze kterého pak následně vychází databázový model.

Obrázek 3.5 ukazuje návrh ER-diagramu, který obsahuje tabulky vedoucích (leaders) a účastníků (participants), které jsou vztahem M:N spojení s tabulkou kompetencí (competences), kde každý účastník může mít splněné nějaké kompetence zaznamenané v pomocné

[3]entitně vztahový model se v softwarovém inženýrství používá pro abstraktní a konceptuální znázornění dat

tabulce splněno (done) a vedoucí ověřující nějaké kompetence zaznamenané v tabulce ověřování (verifications). Další důležitou tabulkou je seznam vytvořených rozvrhů v tabulce rozvrhy (schedules) a tabulka časových bloků, která definuje jednotlivé řádky rozvrhu. Poslední tabulkou, která je centrem všeho je tabulka dat rozvrhu (schedule data), která je vztahem N:1 propojena se všemi ostatními tabulkami. Každý záznam v tabulce reprezentuje jedna položka rozvrhu.



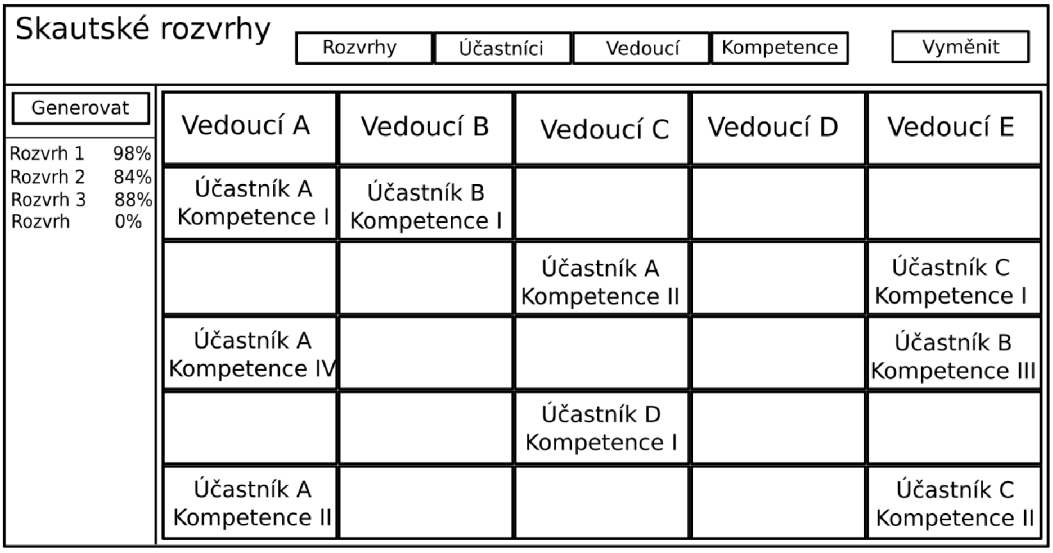
Obrázek 3.5: ER-diagram

3.6 Grafické uživatelské rozhraní

"*Pravděpodobně je mnoho zkušeností, nekončících rozhovorů o filosofickém "prožitku", ale nejsem kvalifikovaný k tomu, abych vás učil filosofii, takže to neudělám. V oblasti UX potřebujeme praktické odpovědi.*" [5]

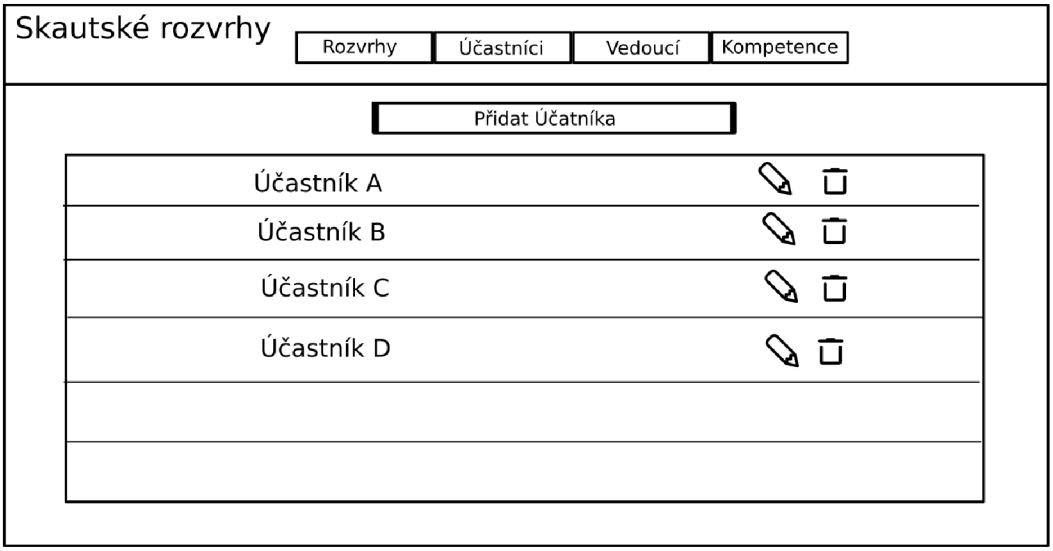
Pro tuto práci bude potřeba udělat jednoduché grafické rozhraní, které uživateli umožní nechat si vygenerovat sadu rozvrhů, zobrazit si vygenerované rozvrhy, zadávat vstupní data (seznam účastníků, seznam vedoucích a jednotlivá ověřování). Na základě těchto parametrů jsem navrhl prototyp grafického rozvržení stránek, které jsou popsané níže. Jedná se o jednoduché 4 stránky, které umožní obsloužit celý systém.

Na obrázku 3.6 je vyobrazena hlavní stránka, která by měla obsahovat nejdůležitější položku problému a tou jsou rozvrhy. Rozhodl jsem se pro rozdělení do dvou sloupců, kdy v levém užším sloupci bude seznam vygenerovaný anebo právě generovaných rozvrhů s hodnotou fitness převedenou do procent a pravý sloupec bude zobrazovat vybraný rozvrh. Velikost sloupců bych doporučil v poměru cca 1:5. Levý sloupec bude také ve své hlavičce obsahovat možnost nechat si vygenerovat nový rozvrh. U zvoleného rozvrhu bude možnost daný rozvrh smazat. Následně v části, kde se zobrazuje rozvrh bude nutné umožnit uživateli provádět na rozvrhu úpravy. Moje představa je, že se vyberou dvě položky a ty se za pomocí tlačítka vymění. V tu samou chvíli se zobrazí aktuální vzniklé kolize a přepočítá se hodnota fitness.



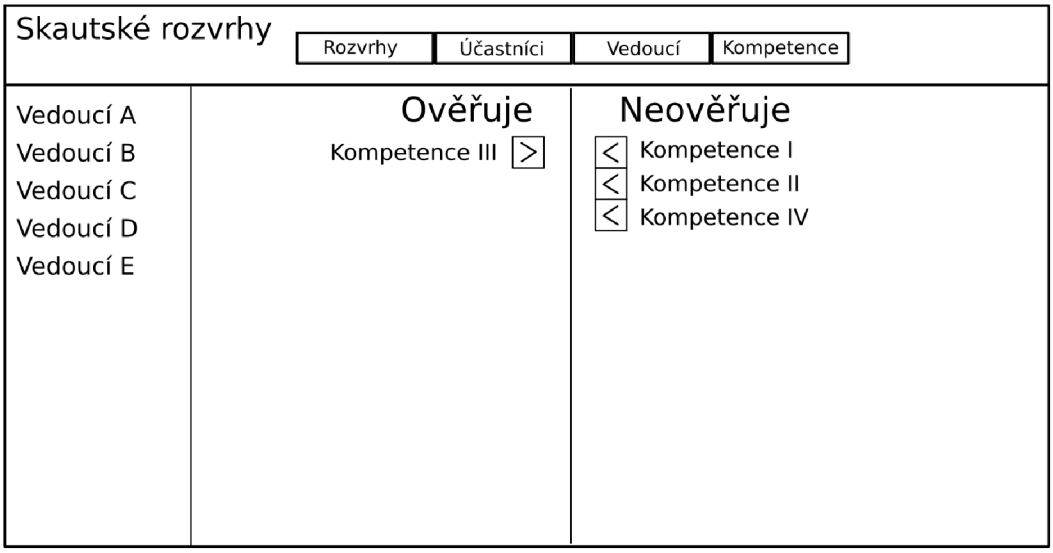
Obrázek 3.6: Návrh hlavní obrazovky – zobrazení rozvrhů.

Obrázek 3.7 obsahuje stránku pro správu účastníků a vedoucích, které budou stejné. Stránka bude obsahovat seznam již přidaných účastníků/vedoucích, které bude uživatel moci upravit nebo smazat. Následně za pomocí dialogového okna bude moci přidat nového účastníka.



Obrázek 3.7: Návrh obrazovky pro přidání osoby.

Poslední obrázek 3.8 vyobrazuje stránku, která bude spravovat ověřování jednotlivých vedoucích. Zachovávám dva sloupce ve stejném poměru jako na hlavní stránce. Levý sloupec zobrazuje seznam vedoucích a v pravém sloupci po vybrání vedoucího lze měnit jeho ověřované kompetence.



Obrázek 3.8: Návrh obrazovky pro editaci ověřování.

Kapitola 4

Uživatelská aplikace pro tvorbu rozvrhu

Dle návrhu popsaného v kapitole 3 jsem sestavil webovou a serverovou aplikaci s využitím technologií zmíněných v kapitole 2.

4.1 Knihovna PyGAD

V této kapitole čerpám z oficiální dokumentace knihovny PyGAD[1].

Hlavní částí mé aplikace je generování rozvrhu za pomocí knihovny PyGad v jazyce Python. Původně jsem chtěl implementaci zapouzdřit do třídy a všechny metody do ní skrýt, bohužel se ukázalo, že knihovna PyGAD, při přijímání funkcí jako parametr, kontroluje počet argumentů funkce a při třídních metodách toto nefungovalo z důvodů třídního parametru self, který se nachází na prvním místě každé třídní metody. Celá implementace se tedy nachází v odděleném souboru, kde jsou pro ni zajištěny vlastní proměnné a funkce.

V první fázi se z databáze získají všechna potřebná data pro následnou inicializaci, následně se zkontroluje, zda není velikost rozvrhu malá pro potřebná data. V případě malého rozvrhu se celý proces generování zastaví a skončí s chybou, která se zapíše do databáze. Počáteční populace se následně vytváří na základě náhody, kdy se ke každé položce z kartézského součinu účastník x kompetence přiřadí náhodně časový blok a vedoucí. Tento proces se opakuje pro každý jeden chromozom. V mém řešení jsem se rozhodl pro 500 chromozomů pro původní populaci. Takto vzniklá populace se následně předá knihovně PyGAD.

Knihovně PyGAD se při inicializaci předává velká spousta parametrů, které jsou potřeba pro správný chod algoritmu. Všechny parametry, které používám ve svém řešení, mám sepsané zde:

- *num_generations* nastavené na 500 – udává maximální počet generací
- num_parents_mating nastavené na 100 – udává počet řešení, která budou zvolena jako rodič
- fitness_func – přiřazená funkce obstarává vyhodnocování hodnoty fitness
- num_genes nastavené na délku chromozomu – udává počet genů v chromozomu
- initial_population – přiřazuje se počáteční populace

[1] https://pygad.readthedocs.io/

- crossover_type – přiřazená funkce obstarává křížení chromozomů
- mutation_type – přiřazená funkce obstarává mutaci
- gene_type nastavené na int – typ genu
- stop_criteria nastavené na "saturate_30"– zadává podmínku pro předčasné ukončení algoritmu.

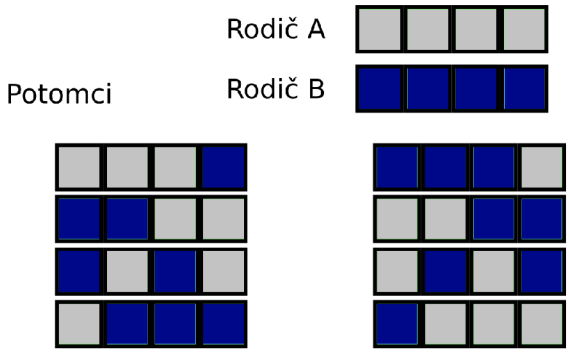
Pojďme se podívat na jednotlivé části implementace genetického algoritmu. V první řadě začněme nejdůležitější částí, a to kódováním chromozomu. Knihovna umožňuje zvolit typ pro každý gen, bohužel ale gen může nabývat typu pouze celého nebo desetinného čísla (a jim podobným). Z tohoto důvodu jsem musel můj gen, který se skládá ze tří čísel, zakódovat do jednoho čísla za pomocí binárních operací. Jednoduše jsem tyto tři čísla v bitové hodnotě doplnil na stejnou délku, poskládal je za sebe a výslednou hodnotu jsem přečetl jako číslo v desítkové soustavě. Tímto způsobem se mi povedlo zakódovat data pro knihovnu PyGAD, aby je zvládla zpracovat. Jelikož všechny operace nad daty jsem prováděl já ve svých funkcí, nebyl s tímto řešením problém. Operace pro kódování a dekódování chromozomu je použita v každé funkci, vzniká tím nevýhoda tohoto řešení, a to navýšení složitosti a s tím i zvýšení časové náročnosti programu.

První funkcí, kterou jsem předával knihovně, byla funkce pro počítání fitness hodnoty. Samotná funkce volá další funkce, které ohodnocují zmíněné kolize v kapitole 3.4. Dle návrhu tyto funkce ohodnocují daný chromozom v normalizované stupnici 0–1.

- Kritérium hodnotící kolize více bloků v jedné položce rozvrhu jsem nazval *MoreBlocksInOneCell*. Tato funkce projde každou položku v rozvrhu a pokud narazí na více položek na jednom místě, tak za každou položku navíc vynásobí dílčí fitness hodnotu jednou polovinou (výchozí hodnota dílčí fitness je 1), konečná fitness hodnota, což je součet všech dílčích fitness, se pak vydělí počtem položek rozvrhu, aby se získal průměr.
- Funcke *ParticipantCollision* počítá kolize účastníka. Pro každého účastníka zkontroluju celý rozvrh, zda nenajdu ve stejný čas více položek. Obdobně jako u předchozí funkce sčítám mocniny jedné poloviny, které následně sečtu a vydělím počtem.
- Kolize mezi správným přiřazením vedoucího ke kompetenci řeším ve funkci *WrongLeaderCompetenceCombination*. Tato funkce je velmi prostá, pro každý gen v chromozomu zkontroluje, zda daný vedoucí ověřuji přiřazenou kompetenci. V případě že ne, přičte k počítadlu špatných kombinací jedničku. Na závěr umocní číslo 0.5 na napočítaný počet špatných kombinací a toto číslo vrátí.
- Posledními funkcemi jsou *SpaceBetweenLeader* a *SpaceBetweenLeader*, které zkoumají volný prostor mezi prvním a posledním blokem v rozvrhu dané osoby. Fungují tak, že pro každou osobu zkontrolují daný gen, zapamatují si všechny položky, kde se daná osoba vyskytuje a spočítají počet nezaplněných položek. Rozdíl poslední a první položky pak vydělí počtem prázdných bloků. V případě 0 prázdných bloků je dílčí fitness hodnota automaticky 1, aby nedocházelo k chybě dělení nulou. Na závěr pak sečtou dílčí fitness hodnoty a vydělí počtem osob k získání průměru.

Funkce řešící křížení chromozomů, vyobrazená na obrázku 4.1, je velmi jednoduchá. Jelikož u křížení nepotřebuji přistupovat k datům jednotlivých genů, stačí mi pouze jejich zakódovaná reprezentace. V této funkci pro každou po sobě jdoucí dvojici v seznamu chromozomu

aplikuji křížení, k vytvoření 8 nových potomků. Dané chromozomy rozdělím na čtvrtiny a tyto části následně různě spojuji k vytvoření 8 nových jedinců. Přesné spojení ukazuje následující obrázek.



Obrázek 4.1: Křížení použité v implementaci.

Funkce mutace, která na základě pseudonáhodného generování vytváří nové geny přiřazováním nového vedoucího nebo časového bloku (nebo obojího). Pokud se v chromozomu nachází nějaká položka, která má stejného vedoucího a čas s jinou položkou, tedy má kolizi, je vybírána přednostně, stejně tak jsou do přednostního výběru řazeny položky, kde je vedoucí přiřazen ke špatné kompetenci. Existují zde dvě funkce, které zjišťují kolize pro přednostní mutaci. V případě žádné kolize je gen pro mutaci vybírán pseudonáhodně.

Po ukončení genetického algoritmu následuje ukládání výsledného rozvrhu do databáze. V první řadě se chromozom dekóduje a každá položka rozvrhu uloží jako samostatný řádek do tabulky "scheduleData", která je vázána na ostatní tabulky, tudíž se ukládají pouze cizí klíče ostatních tabulek. Následně se aktualizuje záznam v tabulce "schedule", kde se doplní stav "Hotovo" a přiřadí hodnota fitness, které odpovídá procentuální míře kvality rozvrhu.

4.2 Server aplikace a API

Jak jsem zmínil, v kapitole 2, pro implementaci serveru používám mikro webový framework Flask. Za pomocí toho frameworku a jeho dekorátorů[2], jsem schopen jednoduše implementovat každé rozhraní mého API. Dekorátor mi zajistí, aby se při zavolání určité url, spustila právě funkce, kterou očekávám. Navržené API není rozděleno do více souborů z důvodů jeho jednoduchosti a podrobněji je popsáno v příloze B. Každý interface z API musí nějaký způsobem komunikovat s databází. Toto je zprostředkováno pomocí knihoven jednotlivých tabulek databáze. Každá tabulka má svůj soubor a vlastní třídu, která obsahuje právě ty metody, které v aplikaci používám. Většinou se jedná o metody typu getAll, create, delete, update, ale najdou se i složitější metody, které se používají pro získávání dat pro generování. Tyto metody pak obsahují přímo zapsané SQL dotazy, které se odesílají na databázový server. Každá třída má instanci třídy *Database*, která zprostředkovává přímé spojení s databázovým serverem.

[2]slouží k ovlivnění chování funkcí či metod, aniž bychom je přímo modifikovali.

4.3 Grafické uživatelské rozhraní

Webovou část aplikace jsem psal v typescriptu za pomocí knihovny React. Snažil jsem se o co nejvíce jednoduché řešení s co nejméně vyskakovacími okny a co nejvíce přehledné. Pro potřebu zadání vstupních hodnot stačí uživateli zadat seznam účastníků a seznam vedoucích, kterým přiřadí kompetence, které ověřují. Pro tyto soubory hodnot slouží tři oddělené stránky, které umožňují přidávat, editovat a mazat jednotlivé osoby a na třetí stránce je možné editovat přiřazené kompetence určitým vedoucím. React si zakládá na oddělování jednotlivých částí webu do tzv. *komponent*. Pro tyto komponenty se vytvářejí nové třídy nebo funkce, které je pak kdekoliv v kódu TSX možné zavolat jako HTML tag.

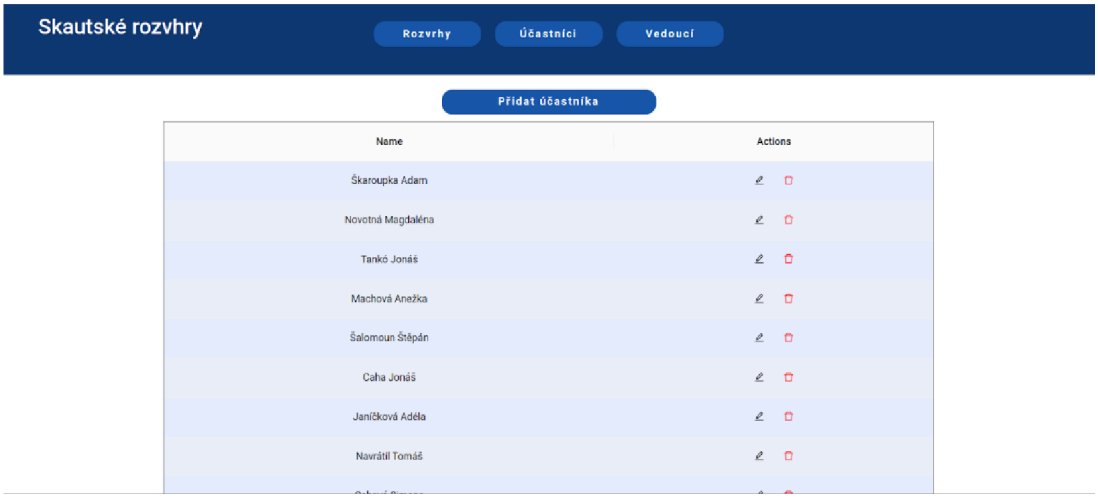
Navrh jsem si základní rozložení pro všechny stránky za pomocí layout systému z knihovny ANT design[3]. Do části, kde se budou jednotlivé stránky zobrazovat jsem si přidal navigaci, která rozhoduje o tom, která stránka se zobrazí na základě url. Do hlavičky jsem pak přidal název aplikace a 4 tlačítka pro orientaci po stránce.

Hlavní stránka zobrazená na obrázku 4.2, je za pomocí grid systému rozdělena na dva sloupce, kdy levý sloupec má šířku 4 a pravý sloupec má šířku 20. Levý sloupec je naimplementován přímo v kódu hlavní stránky. Bylo by možné jednotlivé dílčí části naimplementovat do jednotlivých funkcí, ale v tomto případě se jedná o velmi jednoduchou část, u které se nepočítá s duplikací na další stránky. Pro pravou část, ta, kde se zobrazuje vygenerovaný rozvrh, je vytvořena speciální komponenta. Rozvrh se sestavuje na základě přijatých dat do klasické HTML tabulky, která je pomocí CSS nastylována. Pro tuto stránku jsem musel také přidat tlačítko na výměnu, které je umístěno v hlavičce. Abych nemusel složitě řešit propojování tlačítka se zobrazeným rozvrhem, je tlačítko součástí stránky s rozvrhy a pomocí absolutní pozice a záporných hodnot pro posun je tlačítko umístěno do hlavičky. Pro vybrání položky v rozvrhu pro výběr využívám dvou hodnot každé buňky, a to id vedoucího a časový blok. Tato kombinace je pro každou položku individuální.

Obrázek 4.2: Vzhled GUI pro úvodní stránku.

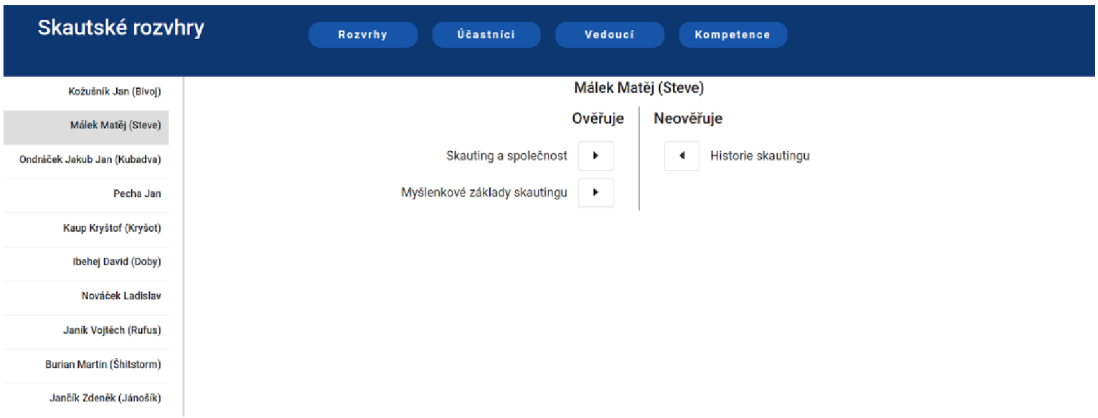
[3]Dostupné na https://ant.design/components/layout/

Další stránka, která je zobrazená na obrázku 4.3, umožňuje uživateli spravovat vedoucí nebo účastníky. Zde je tvorba řešena za pomocí jednoduché tabulky o dvou sloupcích, které zobrazují osoby a akce pro editaci a smazání. Pro vytvoření a editaci slouží dialogové okno, které umožňuje změnit jméno osoby. Před smazání osoby je uživatel dotázán, zda si opravdu přeje smazat zadaného uživatele.



Obrázek 4.3: Vzhled GUI pro stránku editaci osob.

Poslední stránkou, vyobrazenou na obrázku 4.4, je stránka na přiřazování kompetencí. Stránka má stejné rozložení jako hlavní stránka. V levém sloupci jsou zobrazeni vedoucí a pravý sloupec umožňuje přiřadit nebo odebrat danou kompetenci z ověřování vedoucího. Tato jednoduchá stránka také není rozdělena na komponenty, i když by to v pár věcech ulehčilo práci. Aby stránka dlouho nenačítala, jsou změněná data pouze odesílána na server, ale na webu se mění pouze hodnota proměnné.

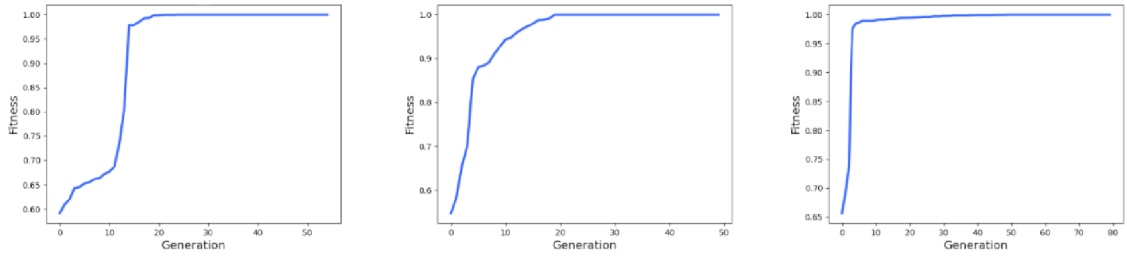


Obrázek 4.4: Vzhled GUI pro stránku editaci ověřování.

Pro stylování jednotlivých komponent a částí stránek jsem používal speciální styly zvané "Syntactically Awesome Style Sheets" zkráceně sass[4], které mi do klasických kaskádových stylů, které známe z tvorby webových stránek, přidají zanořování různých bloků do sebe. Tento .scss soubor se následně kompiluje do klasického .css souboru.

4.4 Testování genetického algoritmu

Malá testovací sada obsahuje 10 účastníků, 10 vedoucích, kde každý ověřuje 2 kompetence a celkem 3 kompetence k ověření. Do výsledného rozvrhu se tedy musí vydat 30 položek (účastníci x kompetence). Průměrná doba dosažení výsledku je 30–60 sekund a výsledný rozvrh má většinou fitness hodnotu větší než 0.95. Na obrázcích 4.5 je vidět vývoj fitness hodnoty v závislosti na počtu generací. Každý graf má jiný počet časových bloků. Zleva 3, 4 a 5 časových bloků.



Obrázek 4.5: Grafy vývoje fitness hodnoty pro 3, 4 a 5 časových bloků.

Pro malé sady je tento postup ideální. V poměrně krátkém čase dostane uživatel ideální řešení i v případě, že je zvolen minimální počet časových bloků.

Střední testovací sada obsahuje 20 účastníků, 15 vedoucích, kde každý ověřuje 3 kompetence a celkem 6 kompetence k ověření. Do výsledného rozvrhu se tedy musí vydat 120 položek (účastníci x kompetence). Průměrná doba dosažení výsledku je 5–10 minut a výsledný rozvrh má různou fitness hodnotu závislou na počtu časových blocích.

Na obrázcích 4.6 je vidět vývoj fitness hodnoty v závislosti na počtu generací. Každý graf má jiný počet časových bloků. Zleva 30 a 40 časových bloků.

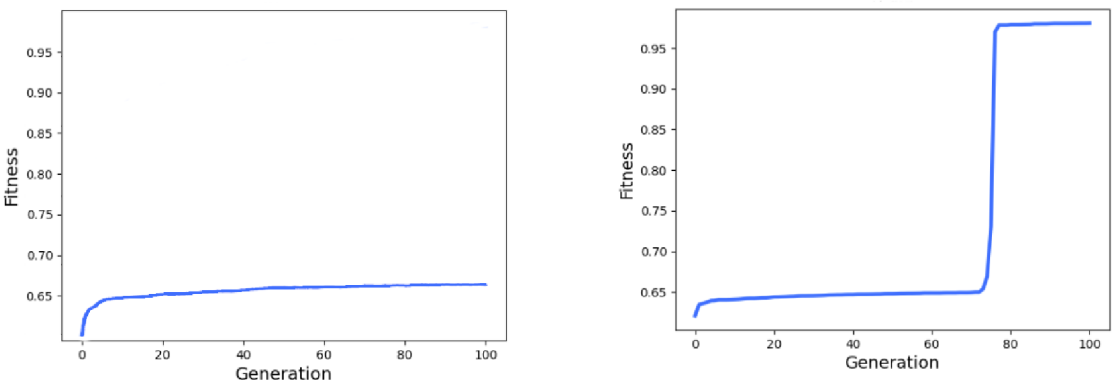
U této testovací sady záleží na počtu časových bloků. Z grafu lze vidět, že při 30 časových bloků se hodnota fitness dostane lehce na 0.65, kdežto při 40 časových bloků vystřelí hodnota nad 0.95.

Velká testovací sada obsahuje 30 účastníků, 20 vedoucích, kde každý ověřuje 3 kompetence a celkem 10 kompetencí k ověření. Do výsledného rozvrhu se tedy musí vydat 300 položek (účastníci x kompetence). Průměrná doba dosažení výsledku je více jak 15 minut a výsledný rozvrh má většinou fitness hodnotu kolem 0.65.

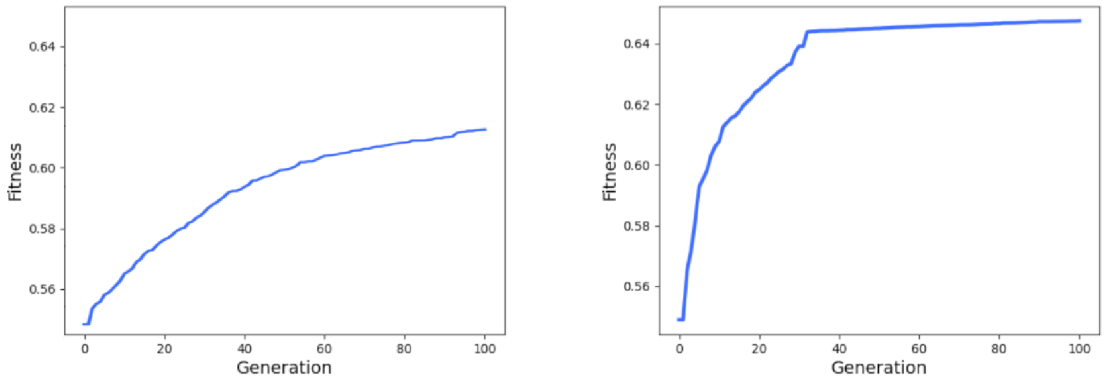
Na obrázcích 4.7 níže je vidět vývoj fitness hodnoty v závislosti na počtu generací. Každý graf má jiný počet časových bloků. Zleva 3, 4 a 5 časových bloků.

V této testovací sadě, se hodnota fitness dostane málo kdy na hodnotu 0.65. U zvolených časových bloků lze pouze vidět strmější nárůst při zvolení více časových bloků. Pro větší sady není toto řešení bohužel ideální.

[4]https://sass-lang.com



Obrázek 4.6: Grafy vývoje fitness hodnoty pro 30 a 40 časových bloků.



Obrázek 4.7: Grafy vývoje fitness hodnoty pro 25 a 40 časových bloků.

4.5 Testování uživatelského rozhraní

Testovací skupinu pro tento systém tvoří dospělí skautští vedoucí. Při práci s aplikaci se uživatel vžije do uživatele, který tento systém ovládá pro svoji potřebu. Testovací subjekty jsou 3 vedoucí z kurzů, kteří mají zkušenosti s plánováním rozvrhů v tabulkách Google.

Způsob pozorování probíhal přímo. Uživateli byl zadán seznam úkonů, které musí splnit. Tento seznam vypadal následovně:

1. Vytvořit, editovat, smazat účastníka
2. Vytvořit, editovat, smazat vedoucího
3. Přiřadit/Odebrat vedoucímu kompetenci
4. Nechat vygenerovat rozvrh
5. Zobrazit si rozvrh
6. Smazat rozvrh
7. Vyměnit dvě položky v rozvrhu

Po podání tohoto zadání byla načtena úvodní stránka s přehledem rozvrhů a test mohl začít. Bylo sledováno zorientování se v tomto prostředí bez žádných předešlých zkušeností → rychlost, počet špatně provedených úkonů. Ačkoliv přímé pozorování může být dotěrné, jednotlivé položky navigace pro splnění daných úkolů by měly být zřetelně na očích a nemělo by být nutné nad jednotlivými položkami přemýšlet. Uživatel mohl během testu popisovat svoje myšlenkové pochody pro další zpětnou vazbu. Test nebyl zdlouhavý na to, aby uživatel ztratil pozornost a díky jeho komentáři byly odhaleny i nedostatky, které by jinak zůstaly skryté.

Po splnění všech úkonů následovala krátká diskuze, rozvíjející upomínky z komentářů během testování. Pak byl podán dotazník s hodnocením u jednotlivých bodů s možnostmi ne (5), spíše ne (4), nevím (3) spíše ano (2), ano (1).

1. Vidíte využitelnost tohoto systému?
2. Je pro vás těžké se v systému zorientovat?
3. Je zobrazení jednotlivých rozvrhů jasné?
4. Jsou funkce systému v této fázi dostatečné?

Vyhodnocení z těchto testů bylo téměř okamžitě. V tabule 4.1 jsou výsledky zkoumaných osob.

Úkony	**Otázky**
1. 0:38 min	1. 2,3
2. 0:25 min	2. 5
3. 0:14 min	3. 1,3
4. 0:08 min	4. 4,3
5. 0:02 min	
6. 0:03 min	
7. 0:23 min	

Tabulka 4.1: Tabulka výsledků uživatelských testů

Na základě testů mohu říci, že navigace napříč webem není problém. U jednoho testovacího subjektu se nám podařilo odhalit implementační chybu, která byla před odevzdáním ještě opravena.

Samotná funkčnost projektu byla hodnocena kladně, i když někdy byla nedostatečnost funkcí. Aplikaci lze považovat pro běžného uživatele za intuitivní.

Prostory pro zlepšení jsem se ze strany testovaných osob dozvěděl následující:

- Chybí možnost změnit název rozvrhu
- Výměna políček v rozvrhu by byla lepší metodou drag and drop
- Dlouhá doba výpočtu větších rozvrhů, a navíc s malou kvalitou
- Možnost vytvářet kopie rozvrhů a ty následně upravovat
- Export do PDF

Kapitola 5

Závěr

Cílem práce bylo vytvořit webovou aplikaci pro skautské kurzy, která by měla ulehčit práci se závěrečným ověřováním komeptencí. V Práci jsem provedl studii evolučních algoritmů, zjistil jsem, jak fungují, jaké metody se používají pro křížení, pro mutaci, pro výběr potomků, nebo pro výpočet fitness funkce. Seznámil jsem se s návrhem a vytvářením moderních webových aplikací. Naučil jsem se používat knihovnu PyGAD jazyka python, která obstarává režii při práci právě s genetickým algoritmem a za pomocí této knihovny a freamworku Flask jsem implementoval serverovou část aplikace pro vytváření rozvrhů. Pro webovou část jsem se seznámil s knihovnou React a jazykem Typescript, za pomocí kterého jsem napsal aplikaci pro koncové uživatele.

Existující systémy zmíněné v kapitole 2.4 vyvíjejí firmy, o velikosti i desítek lidí vzdělaných v tomto směru, několik let. Pokud budu brát v potaz tuto realitu, tak mohu svoji práci ohodnotit kladně, zejména získané zkušenosti při zkoumání genetického algoritmu a následného použití v praxi. Pokud bych se zaměřil na aktuální stav aplikace, tak je aplikace ve funkčním stavu. Pro malé rozvrhy je toto řešení ideálním přístupem. V následující etapě bych se zaměřil zejména na lepší použití evolučních algoritmů při generování rozvrhů, řádné testování, ošetření případných chybových stavů a vylepšení z poznatků testování na uživatelích.

Do budoucna by se dalo podívat na rozšíření aplikace, jako je například možnost přihlašování a přístupu pro více kurzů, přihlašování přes skautský informační systém, možnost editovat kompetence, možnost zadávat omezení jednotlivým vedoucím, dostupnost pro více typů kurzů, zadávat splněné kompetence účastníkům, aby se s nimi nepočítalo při generování rozvrhu, přístup pro účastníky, zobrazení individuálního rozvrhu a seznam splněných kompetencí, export dat do souboru, ať už PDF či jiného formátu.

Literatura

[1] Darwin, C. *O vzniku druhů přírodním výběrem.* Vyd. 3., V nakl. Academia 2., rev. Praha: Academia, 2007. ISBN 978-80-200-1492-4.

[2] Horký, A. *Tvorba rozvrhů pomocí genetických algoritmů.* VUT, Brno, 2011. Bakalářská práce. Vysoké učení technické, Fakulta informačních technologií.

[3] Hynek, J. *Genetické algoritmy a genetické programování.* 1. vyd. Praha: Grada, 2008. ISBN 978-80-247-2695-3.

[4] Kvasnička, V. *Evolučné algoritmy.* 1. vyd. Bratislava: Vydavateľstvo STU, 2000. ISBN 80-227-1377-5.

[5] Marsh, J. *UX pro začátečníky: (rychlokurz - 100 lekcí).* Brno: Zoner Press, 2019. ISBN 978-80-7413-397-8.

[6] Studnička, V. *Genetické algoritmy – Multi-core cpu implementace.* VUT, Brno, 2010. Diplomová práce. Vysoké učení technické, Fakulta strojního inženýrství.

[7] Waggener, B. *Pulse Code Modulation Techniques.* 1. German: Springer, 1995. ISBN 978-04-420-1436-0.

[8] Wieruch, R. *The road to learn React.* 2018. ISBN 9781720043997.

Příloha A

Slovník pojmů genetického algoritmu přejatých z biologie

Tato příloha seskupuje pojmy vyskytující se v této práci z biologie z oblasti evolučních algoritmů. Jejich popis vychází z přílohy [6].

Chromozom	reprezentuje způsob kódování jednice. Složen z množiny genů. Každý jedince je právě jeden chromozom.
Fenotyp	Pozorovatelné vlastnosti jedince. Vytvořen působením prostředí na jedince.
Fitness funkce	určuje kvantitativní míru schopnosti jedince přežít a vstupovat do reprodukce. Lze ji získat transformací účelové funkce
Genotyp	Soubor všech genetických informací jedince
Jedinec	Nositel genetické informace, reprezentuje jedno ohodnocené řešení. V mém případě se jedná o jeden možný rozvrh.
Křížení	Proces vytváření nového potomka z vybraných předků
Mutace	Náhodná změna jednoho nebo více genů
Populace	množina jedinců, která může být mezi sebou křížena a tak mohou vytvářet nové jedince
Účelová funkce	Reprezentuje vnější prostředí, a umí hodnotit kvalitu jedince v něm

Tabulka A.1: Slovník pojmů přejatých z biologie

Příloha B

API serverové aplikace

V tabule lze vidět seznam procedur, které aplikace umožňuje volat. API určuje, jakým způsobem jsou jednotlivé funkce volány.

API	Typ	Detail
SCHEDULES		
/get_schedule_names	GET	Vrací seznam všech rozvrhů a jejich fitness
/get_schedule/<id>	GET	Vrací data pro daný rozvrh
/generate_schedule	POST	Zahájí generování rozvrhu
/change_schedule	POST	Vymění v rozvrhu dvě položky
/delete_schedule/<id>	DELETE	Smaže rozvrh a jeho data
PARTICIPANT		
/get_participants	GET	Vráti všechny účastníky
/create_participant	POST	Vytvoří nového účastníka
/update_participant	POST	Aktualizuje již existujícího účastníka
/delete_participant/<id>	DELETE	Smaže účastníka
LEADERS		
/get_leaders	GET	Vráti všechny vedoucí
/create_leader	POST	Vytvoří nového vedoucího
/update_leader	POST	Aktualizuje již existujícího vedoucího
/delete_leader/<id>	DELETE	Smaže vedoucího
VERIFICATIONS		
/competences	GET	Vrátí všechny kompetence
/verifications/<id>	GET	Vrátí ověřování daného vedoucího
/create_verification	POST	Přidá vedoucímu ověřování
/delete_verification	POST	Odebere vedoucího ověřování

Tabulka B.1: Přehled všech API

Příloha C

Obsah SD karty

- **bp_doc** – zdrojové texty této bakalářské práce ve formátu LATEX s přiloženým makefile
  - **pictures** – Obrázky v této bakalářské práci
- **bp_web** – zdrojové soubory pro webovou aplikaci napsané v Reactu
  - **src** – Obsahuje zdorjové kódy pro webovou aplikaci
  - **public** – Obsahuje základní html strukturu pro běh aplikace a případné obrázky a fonty
  - **README.md** – Soubor popisující způsob instalace a spuštění app
  - **package.json** – Obsahuje seznam použitých knihoven, zároveň se využívá pro instalaci
  - **tsconfig.json** – Základní nastavení react apliakce
- **bp_api** – zdrojové soubory pro serverovou část aplikace napsané v Pythonu
  - **Database** – Obsahuje třídy pro spojení s databází
  - **GeneticAlgorithm** – Obsahuje funkční kód pro běh genetického algoritmu
  - **README.md** – Soubor popisující způsob instalace a spuštění app
  - **app.py** – Soubor obsahující celé API
- **tests** – Obsahuje 3 testovací sady a výsledky zmíněné v této bakalářské práci. Sady jsou ve formátu SQL určené pro databázi PostgreSQL.
- **bb_xmalek17.pdf** – tato bakalářská práce ve formátu pdf