Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra Teoretické Informatiky

Bakalářská práce

Efektivní plánování směn pro dohled nad datovými centry

Anna Kučerová

Vedoucí práce: Ing. Jan Černý

8. května 2015

Poděkování

Zde bych ráda poděkovala Ing. Janu Černému, vedoucímu této bakalářské práce, za čas, rady a ochotu odpovídat na všechny otázky.

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, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Praze dne 8. května 2015 .....

Odkaz na tuto práci

Kučerová, Anna. *Efektivní plánování směn pro dohled nad datovými centry.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2015.

Abstrakt

Tato bakalářská práce se zabývá řešením problému plánování směn s využitím evolučních algoritmů a jiných heuristik. Jedná se o NP-obtížný problém, ve kterém potřebujeme přiřadit směny zaměstancům v souladu s množstvím omezení.

V této práci bylo ukázáno, že nejlepších výsledků z implementovaných algoritmů dosahovali koevoluční genetický algoritmus a memetický algoritmus, které dokázali v každém běhu najít optimální řešení.

Klíčová slova plánování směn, evoluční algoritmus, hill climbing, simulované žíhání, rozvrh, nurse scheduling problem, memetický algoritmu

Abstract

Scheduling is known to be NP-hard optimization problem. Which goal is to create a schedule satisfying a set of constraints. Many different approaches can be used to solve this problem, this work focuses mainly on evolutionary algorithms.

Cooperative coevolution and memetic algorithm are presented as the most effective solutions. It has been shown that those two approaches are able to find optimal schedule most efficiently.

Keywords scheduling, evolutionary algorithm, hill climbing, simulated annealing, rostering, nurse scheduling problem, memetic algorithm

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Plánování směn je NP-obtížná úloha, ve které se musí rozdělit směny zaměstnancům podle určitých omezení, čímž vznikne rozvrh směn.

Rozvrh je množina aktivit, u kterých je dopředu známo, jak dlouho budou trvat, a které jsou vykonávány nějakým zdrojem, nebo jej využívají. V tomto případě jsou aktivitami směny a zdrojem zaměstnanci datových center. Směna je rozvržení fondu pracovní doby.

Aby byl ale rozvrh v praxi použitelný, musí splňovat určitá omezení, která dělíme na tvrdá (neporušitelná) omezení a měkká (porušitelná) omezení.

Složitost této úlohy stoupá s počtem zaměstnanců a pracovišť, pro které je třeba směny naplánovat, a proto v dnešní době vznikají algoritmy a studie, snažící se tento problém řešit, zefektivnit a usnadnit tím práci lidí majících plánování na starosti. Tím s sebou přináší zkvalitnění služeb dané instituce, protože plánující osoba bude mít více času, ale i ostatní zaměstnanci budou pracovat efektivněji díky lepšímu rozvrhnutí směn.

První kapitola 1 se věnuje popisu práce a požadavkům zadavatele na tuto práci. V další kapitole 2 je navázáno popisem rozvrhů a vysvětlením problému přiřazení směn zdravotním sestrám z důvodu jeho podobnosti se zadáním. Následující kapitola 3 je věnována analýze problému spolu s popisem algoritmů, které se dají použít k řešení tohoto problému. V další kapitole 4 je poté už popsána realizace jak aplikace, tak i jednotlivých algoritmů a v poslední kapitole 5 jsou zhodnoceny její výsledky spolu s porovnáním kvality s běžně používanými přístupy.

Kapitola 1

Cíl práce

Cílem této práce je seznámit se s problémem plánování směn a požadavky společnosti Seznam.cz a dále studovat evoluční algoritmy a jejich využití pro navrhování a vytváření rozvrhů pro zaměstnance v datových centrech. Následně navrhnout a implementovat evoluční algoritmus, který bude porovnán s běžně používanými přístupy. Výsledný rozvrh ale musí splňovat určité podmínky.

1.1 Požadavky společnosti Seznam.cz

Provoz v datových centrech je rozdělen na dvě dvanáctihodinové směny, denní a noční, ve kterých je vyžadována přítomnost alespoň jednoho zaměstnance.

Výsledný rozvrh **musí** splňovat následující pravidla (tvrdá omezení):

1. Zaměstnanec nesmí mít dvě po sobě jdoucí směny.
2. Zaměstnanec nesmí mít směnu ve dvou různých centrech najednou.
3. Zaměstnanec nesmí mít směnu v centru, ve kterém nepracuje.
4. Směna v centru musí být obsazena alespoň jedním zaměstnancem.

A v optimálním případě bude splňovat i tato (měkká) omezení:

1. Zaměstnanec by neměl mít směnu během dovolené.
2. Zaměstnanec by neměl mít méně než 2, nebo více než 4 směny týdně.

Zadavatel poskytl dva vstupní soubory *CSV*. První obsahuje ID jednotlivých zaměstnanců a číslo centra, ve kterém může pracovat a ve druhém je uvedeno, které dny má jaký zaměstnanec dovolenou a zda se jedná o denní, nebo noční směnu.

Kapitola 2

Rozvrh

Rozvrh můžeme popsat jako množinu aktivit, u kterých víme dopředu, jak dlouho budou trvat, a které jsou vykonávány nějakým zdrojem nebo jej využívají.

Rozvrh se skládá z aktivit, které mají být provedeny. Aktivity trvají dopředu stanovenou dobu a využívají zdroje. Zdroj může být například stroj, na kterém se daná aktivita provede, nebo i pracovník provádějící danou aktivitu. Rozvrhem pak rozumíme alokaci zdrojů a času pro každou aktivitu tak, aby byly splněny všechny podmínky kladené na požadovaný rozvrh. V tomto případě jsou aktivitami směny zaměstnanců.

Při tvorbě správného rozvrhu jsme většinou omezeni takzvanými tvrdými a měkkými omezeními.

- Tvrdá omezení jsou taková, která nesmíme porušit, protože výsledný rozvrh by například porušoval pracovní řád, nebo dokonce zákon. Tato omezení musí být bezpodmínečně splněna.
- Měkká omezení porušit smíme. Pokud by měkká omezení bránila nalezení výsledného rozvrhu vyhovujícího všem, tak je můžeme porušit.

Výsledné řešení poté musí splňovat co nejvíce měkkých omezení a všechny tvrdá omezení. Použití měkkých omezení je jednou z možností jak vyjádřit určité preference výsledného řešení.

Pro školní rozvrh například musí platit:

- Učitel nesmí učit dva předměty ve stejnou dobu.
- Student nesmí mít dva předměty ve stejnou dobu.
- Každý předmět musí mít přiřazenou učebnu, učitele a studenty.

Ale pouze by mělo platit:

- Student nemá volnou hodinu během dne.

- Student nemá tělocvik po obědě.

Dále pak rozvrh musí být možné ohodnotit za účelem vyjádření dalších specifických preferencí. Ve školním rozvrhu může být preferencí například upřednostňování výuky odpoledne, což se musí projevit právě v ohodnocení daného rozvrhu. Úkolem rozvrhovače je poté řešit optimalizační úlohu nalezení rozvrhu s ohodnocovací funkcí blízkou hledanému extrému.

Mezi nejznámější typy rozvrhů patří: školní rozvrh, rozvrh výroby, rozvrh směn zaměstnanců.

2.1 Problém přiřazení směn zdravotním sestrám

Problém přiřazení směn zdravotním sestrám neboli Nurse Scheduling Problem (NSP) či Nurse Rostering Problem (NRP), je NP-obtížný problém, kdy se musí sestrám v nemocnici rozdělit směny podle množiny tvrdých omezení, které musí rozvrh splňovat a množiny měkkých omezení, které rozvrh může porušit. Tento problém je ale možné zobecnit na problém rozdělení směn podle omezení i v jiných oblastech. V literatuře lze nalézt mnoho heuristických metod použitelných pro řešení takovýchto problémů: evoluční algoritmy, simulované žíhání, memetické algoritmy, scatter vyhledávání, iterované lokální vyhledávání, optimalizace hejnem částic, optimalizace mravenčí kolonií, atd.

Entitami, které mají vliv na tvorbu rozvrhu jsou: sestry, směny a určitá časová období. Přesněji řečeno, sestry musí odpracovat určité směny v určitých časech. Přístupů k tomuto problému je podle literatury mnoho a každý z nich má svá vlastní omezení, nicméně ve většině jsou tvrdá omezení následující [1]:

- Musí být naplánovány všechny typy směn.
- Směny musí být obsazeny.
- Každá sestra může odpracovat maximálně jednu směnu denně.

Do tvrdých omezení se také řadí omezení určená Zákoníkem práce.

Mezi měkká omezení která zvyšují kvalitu rozvrhu patří:

- Maximální\minimální počet směn týdně.
- Maximální\minimální počet po sobě jdoucích pracovních dnů.
- Maximální\minimální počet po sobě jdoucích volných dnů.
- Maximální\minimální počet odpracovaných hodin.
- Maximální počet po sobě jdoucích pracovních víkendů.
- Maximální počet pracovních víkendů za měsíc.
- Počet volných dnů po určitém počtu nočních směn v řadě.

- Stejné druhy směny během týdne.
- Směna vyžaduje jiné schopnosti než sestra má.
- Žádost o volný den.
- Žádost o volnou směnu.

Kapitola 3

Heuristické optimalizační algoritmy

Heuristické algoritmy jsou skupinou optimalizačních algoritmů a technik využívajících určitý stupeň náhodnosti. Používají se na problémy velkého rozsahu, kde není dopředu jasné, jak bude vypadat optimální řešení, a kde by prohledávání všech možností nešlo použít nebo bylo příliš pomalé. Díky použití náhodnosti je urychlen proces prohledávání. Ale použitím těchto algoritmů není zaručeno nalezení optimálního řešení ani stejný výsledek po každém průběhu, a to právě kvůli použití náhody. V oblasti heuristických algoritmů se využívají tyto termíny:

Stavový prostor je celý rozsah hodnot parametrů, které představují řešení problému.

Fitness funkce je kvantitativní reprezentace kvality řešení.

Použitelné řešení je řešení, které splňuje požadovaná tvrdá omezení.

Optimální řešení je použitelné řešení, které splňuje nejvíce měkkých omezení.

Kandidát na řešení je použitelné řešení, které může být označeno za optimální řešení.

Okolí jsou všechna řešení lišící se od daného řešení pouze v jednom prvku.

Pravidelné procházení znamená, že okolí bude prohledáno vždy v pořadí určeném v datové sadě.

Náhodné procházení znamená, že okolí bude prohledáno vždy v náhodném pořadí.

Náhodný restart je použit, pokud algoritmus zastaví dříve, než má. Algoritmus je restartován z jiného náhodně vybraného kandidáta na řešení.

První kandidát na řešení je většinou vybrán náhodně z použitelných řešení ze stavového prostoru během inicializační fáze heuristického algoritmu. Následně se zkontroluje kvalita vypočítáním hodnoty jeho fitness (více o fitness zde 3.3.5.14). Poté je prohledáno okolí vybraného řešení ve snaze najít optimální řešení a nastane fáze rozhodování, kde musí být vybráno řešení, které postoupí do další iterace. Hledání se zastaví, když je nalezeno řešení splňující požadavky.

Heuristické algoritmy se dělí do následujících skupin podle přístupu k nalezení řešení:

- Algoritmy založené na populaci nebo na jednom řešení - odvíjí se od počtu řešení zkoumaných najednou.
- Lokální nebo globální prohledávání - záleží zda se algoritmus soustředí na jednu oblast stavového prostoru, nebo se ho snaží pokrýt celý.

Předpokládejme, že stavový prostor S je konečná množina všech řešení a hodnotící funkce f je funkce přiřazující všem řešením nějaké reálné číslo. Úkolem je tedy najít řešení $i \in S$, které se snaží optimalizovat f na S.

3.1 Hill Climbing

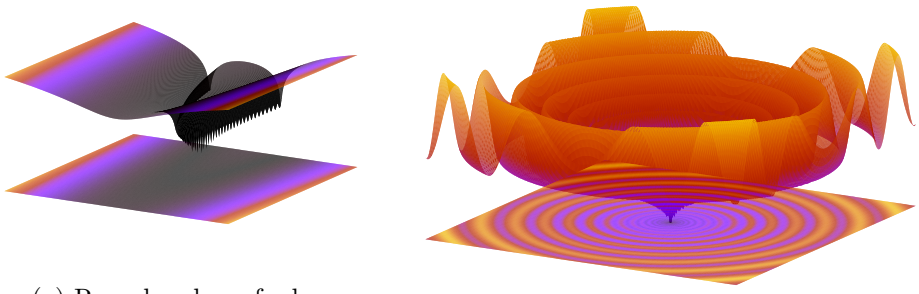
Hill Climbing je heuristický algoritmus, využívající lokální prohledávání pro jedno řešení. Jeho hlavní myšlenkou je začít s kandidátem na řešení, *pod kopcem*, a opakovaně ho vylepšovat, *jít do kopce*, dokud nedosáhneme optima, *vrcholu kopce*.

Algoritmus začíná výběrem náhodného řešení, které se dále snaží zlepšovat lokálními transformacemi. Vyhledá v okolí nejlepší možný posun a pokud takové řešení najde, pak ho přijme a vytvoří takto nového jedince. Jinak zůstane jedinec stejný. Tento proces se opakuje, dokud se dokáží najít lepší řešení. Díky této jednoduchosti je výpočetně velmi nenáročný a činí ho to velmi populární první volbou mezi optimalizačními algoritmy. Velmi často má lepší výsledky než ostatní algoritmy pokud je čas, po který můžou prohledávat prostor, omezený. Ale Hill Climbing má tyto tři nevýhody:

1. Hill Climbing je dobrý pro nalezení lokálního optima, ale negarantuje nalezení globálního optima ve stavovém prostoru, obzvláště pokud je zde mnoho lokálních optim.
2. Pokud je v částech stavového prostoru nulový gradient, pak celé okolí má stejné hodnoty a algoritmus se tak nikam neposouvá.
3. Pokud je stavový prostor hřebenitý, pak je skoro nemožné *vylézt* nahoru díky výběru prohledávání. Protože Hill Climbing zlepšuje řešení pouze v jednom prvku, tak každý krok, který udělá, bude zarovnaný podle osy. To ale u hřebenovitých prostorů může být nevýhoda, jelikož příkré srázy

můžeme zdolat pouze velmi malými kroky po směru os, zatímco zde by byly výhodnější kroky nezarovnané do os. Což může ve výsledku zabrat nepřiměřený čas[2][3].

Mezi příklady nevhodných stavových prostorů patří 3.1. V 3.1b má funkce mnoho lokálních optim, ve kterých algoritmus může uváznout. Oproti tomu v 3.1a Hill Climbing rychle najde údolí, ale v něm už obtížně hledá globální optimum.



(a) Rosenbrockova funkce

(b) Stretched-V funkce

Obrázek 3.1: Ukázky stavových prostorů ve kterých Hill Climbing selhává.

3.2 Simulované žíhání

Simulované žíhání je dalším algoritmem využívajícím lokální prohledávání pro jedno řešení. Jeho jméno stejně jako princip jsou inspirované žíháním v metalurgii, kde se pojmem žíhání rozumí zahřívání a kontrolované ochlazování materiálu pro zvýšení velikosti jeho krystalů a snížení jejich defektů. Tato představa postupného ochlazování je implementována v simulovaném žíhání jako pravděpodobnost přijmutí horšího řešení při prohledávání stavového prostoru. Právě přijímání horších řešení je základní vlastností této heuristiky, díky které může být provedeno mnohem rozsáhlejší hledání pro optimální řešení a zároveň snižuje pravděpodobnost uváznutí v lokálním optimu.

Jednoduchá forma lokálního prohledávání začíná s výběrem počátečního řešení. V okolí tohoto řešení je vygenerováno nové řešení, u kterého je spočítána ohodnocovací funkce. Pokud je nové řešení lepší, pak nahradí řešení staré. Jinak zůstává staré řešení nezměněno a proces se opakuje dokud hodnota ohodnocovací funkce již nedokáže být optimalizována. Z čehož vyplývá, že se prohledávání zastaví v lokálním extrému, které nemusí být zároveň globální extrém. Ale v simulovaném žíhání se algoritmus snaží vyhnout uváznutí v lokálním extrémech tím, že někdy přijme i horší řešení. Přijmutí a odmítnutí horších řešení se řídí pravděpodobnostní funkcí. Pravděpodobnost přijmutí řešení, které způsobí zvýšení δ v ohodnocovací funkci f se nazývá přijímací

funkce. Tato funkce je normálně nastavena na $exp(-\delta/T)$, kde T je kontrolní parametr, který koresponduje s teplotou v analogii s opravdovým žíháním v metalurgii. Tato přijímací funkce naznačuje, že malé zvýšení f bude pravděpodobněji přijato než velké zvýšení f. Když je hodnota T vysoká, pak je přijata většina horších řešení, ale jak se T blíží k nule, tak většina horších řešení bude zamítnuta. Proto začíná simulované žíhání s vysokou teplotou za účelem vyhnutí se uvíznutí v lokálních extrémech. Algoritmus dále pokračuje zkoušením určitého počtu kroků při každé teplotě, která se s postupem iterací snižuje.

Fáze simulovaného žíhání jsou tedy:

1. Inicializuj teplotu.

2. Začni s náhodně vygenerovaným vektorem řešení, X, který ohodnoť ohodnocovací funkcí.

3. S náhodnou odchylkou od vektoru X vygeneruj nový vektor řešení Y z okolí aktuálního vektoru řešení X a ohodnoť nový vektor řešení.

4. Pokud je vygenerované řešení lepší, X přiřaď Y, čímž bude z vektoru Y aktuální řešení. Aktualizuj existující optimální řešení a pokračuj krokem 6.

5. Pokud není lepší pak přijmi Y s pravděpodobností:

$$P = exp(-\Delta s/T)$$

   kde $\Delta s = Z(Y) - Z(X)$. Pokud je řešení přijato, prohoď X a Y.

6. Periodicky snižuj teplotu.

7. Opakuj kroky 2-6 dokud nebude splněno zastavovací kritérium.

Nevýhodou tohoto přístupu je, že k některým řešením se může přistupovat vícekrát.

3.3 Evoluční algoritmy

Evoluční algoritmy jsou inspirovány evolucí a přirozeným výběrem, představenými v roce 1859 Charlesem Darwinem, založenými na přežití nejsilnějšího jedince a genetickou dědičností představenou Johannem Gregorem Mendelem v roce 1865. Tyto algoritmy hledají řešení pomocí „křížení“ jedinců, které spolu kombinují a mění a vytvářejí tak nové generace a následně přiřazují ohodnocení (tzv. fitness) jedincům, nebo jejich částem. Evoluční algoritmy dokáží vytvořit dobrá řešení ve všech typech problémů, proto jsou vhodné pro

optimalizaci problémů typu black box. Díky této obecnosti mají evoluční algoritmy široké užití pro mnoho vědeckých odvětví jako např.: biologie, genetika, robotika, fyzika, chemie a podobně.

Protože si tato metoda hodně vypůjčuje z genetiky, buněčné biologie a evoluční teorie, odpovídá tomu také její názvosloví. Proto je jedno možné řešení nazýváno *jedincem*, zatímco celá množina řešení je označena pojmem *populace*. Ve speciálních případech je možné rozdělit populaci do menších *subpopulací*. Reprezentace jedince se nazývá *genom* nebo *chromozom*. Každý chromozom obsahuje sekvenci *genů*, tzn. atributů, které popisují jedince. Hodnotě genu se říká *alela*. Pro tvorbu nových jedinců používáme *křížení* a nově vytvořené jedince označujeme pojmem *potomci*. Celý tento proces nazýváme *evoluce* [5].

Objekty tvořící možná řešení v prostoru řešení jsou nazýváni *fenotypy*, zatímco jejich kódování, jedinci v evolučním algoritmu, jsou takzvané *genotypy*. V prvním kroku, vybírání reprezentace jedinců, je snaha specifikovat, jak se budou mapovat fenotypy na množinu genotypů reprezentující fenotypy. Pokud například bude úkolem optimalizovat problém v oblasti celých čísel, pak daná množina celých čísel bude tvořit množinu fenotypů. Když budou poté reprezentováni binárně, pak například číslo 18 bude fenotyp a 10010 bude jeho reprezentující genotyp. Celý prostor genotypů může být velice odlišný od fenotypů a celý evoluční algoritmus se bude zabývat pouze genotypy[7].

Dle [8] můžeme většinu implementací evolučních algoritmů rozdělit na tři podobné, ale nezávisle vzniklé přístupy:

- Genetické algoritmy
- Evoluční programování
- Evoluční strategie

3.3.1 Hollandův teorém o schématech

Pokud je vyžadován důkaz pro funkčnost evolučních algoritmů, pak se většinou uvádí, že v jedincích jsou identifikovány kladné rysy, které jsou kombinovány s jinými, zatímco záporné rysy během evoluce odumírají[9].

Představou kladných vlastností se formalizuje koncept stavebních bloků, což jsou schémata, představující dobře přizpůsobené sady vlastností, které můžeme najít u většiny dobrých řešení. Neboli lze vypozorovat mezi podobně dobrými jedinci v populaci podobnost v obsazení některých pozic stejnými bity. Je tedy možné říci, že kvalitu jedince ovlivňuje obsazení určitých konkrétních pozic, zatímco na ostatních pozicích záleží méně. A právě podobnostmi mezi řetězci a vlivem genetických operátorů na schémata se zabývá Hollandův teorém o schématech.

V genetických algoritmech operujících nad binárními řetězci je schéma řetězec symbolů abecedy {0, 1, #}. Znak # bude interpretován jako nezajímavý symbol. Proto například schéma #1#0 může představovat čtyři řetězce (0100,

0110, 1100, 1110). Počet znaků 0 a 1 se nazývá stupeň $\mathcal{O}(H)$ schématu H. Vzdálenost mezi nejvzdálenějšími znaky 0 nebo 1 je označena jako *určující délka* $\mathcal{L}(H)$ schématu.

Holland vytvořil schéma teorém, který předpovídá, jak se počet řetězců v populaci (odpovídající schématu) bude lišit v různých generacích. Teorém můžeme přeformulovat také následovně:

$$E\left[\frac{m(H,t+1)}{M}\right] \geq p(H,t) \cdot (1-p_m)^{\mathcal{O}(H)} \cdot \left[1 - p_{xo}\frac{\mathcal{L}(H)}{N-1}(1-p(H,t))\right]$$

kde p_m je pravděpodobnost mutace na bit, p_{xo} je pravděpodobnost křížení, N je počet bitů v řetězci, M je počet řetězců v populaci, $m(H,t+1)$ je počet řetězců odpovídajících schématu H v generaci $t+1$, $E[]$ je operátor očekávání a $p(H,t)$ je pravděpodobnost výběru schématu H. Při výběru ruletovou selekcí je pravděpodobnost:

$$p(H,t) = \frac{m(H,t)f(H,t)}{M\overline{f}(t)}$$

kde $m(H,t)$ je počet řetězců odpovídajících schématu H v generaci t, $f(H,t)$ je průměr hodnot fitness řetězců odpovídajících H a $\overline{f}(t)$ je průměrná fitness řetězců v populaci[10].

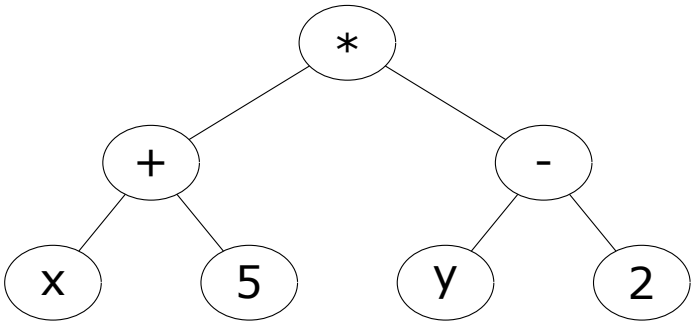
3.3.2 Genetické algoritmy

Zájem o genetické algoritmy se probudil na začátku šedesátých let, kdy je jako první představil John Henry Holland. Jeho cílem bylo porozumět adaptaci, tak jak probíhá v přírodě, a přijít na způsob, jak tento mechanismus použít v počítačových systémech. Jeho kniha z roku 1975 *Adaptation in Natural and Artificial Systems* prezentovala genetické algoritmy jako abstraktní biologickou evoluci a položila základy k úspěšné tvorbě genetických algoritmů.[11]

Jeho algoritmus se běžně nazývá Simple Genetic Algorithm, neboli SGA. Pracuje se v něm s populací binárních řetězců, kde každý řetězec představuje jedno konkrétní řešení problému. Použitím operátorů křížení a mutace algoritmus vytváří z řetězců nové jedince.

3.3.2.1 Genetické programování

Genetické programování je podmnožinou genetických algoritmů. V roce 1990 bylo představeno Johnem Kozou. Při použití tohoto způsobu je prováděna evoluce programů (jedinců) stejně jako v genetických algoritmech je vyvíjeno řešení problému. Většinou jsou tyto programy reprezentovány stromovými strukturami, ve kterých probíhá křížení jako výměna podstromů rodičů. V obrázku 3.2 je vidět typický jedinec, který představuje aritmetický výraz $(x+5)*(y-2)$.[12]



Obrázek 3.2: Ukázka typického jedince v Genetickém programování

3.3.3 Evoluční programování

Představeno Lawrence Jerome Fogelem v roce 1966 v knize *Artificial Intelligence Through Simulated Evolution* [13]. V této knize je vyvíjen konečný automat, který předpovídá řetězce generované Markovovými procesy. Přístup evolučního programování je podobný genetickým algoritmům, ale s tím rozdílem, že se vyvíjí pouze parametry programu. Na celou populaci jedinců se pohlíží spíše jako na populaci jiných druhů, než jednoho druhu. I proto je hlavním operátorem mutace, která mění pouze jednoho jedince.

3.3.4 Evoluční strategie

Technologie evoluční strategie byla vytvořena v šedesátých letech Ingo Rechenbergem a Hans-Paul Schwefelem, za účelem řešení složitých problémů optimalizace diskrétních a spojitých parametrů.

3.3.5 Fáze evolučních algoritmů

Běh evolučního algoritmu můžeme rozdělit na následující fáze:

- Inicializuj populaci (většinou náhodně).
- Opakuj následující kroky, dokud nejsou splněny ukončující podmínky.
  - Selekce - neboli výběr „rodičů“, může a nemusí být orientovaná podle fitness
  - Reprodukce - tvorba potomků z rodičů, typicky křížení
  - Mutace - malá změna jedince
  - Ohodnocení jedinců nebo generace funkcí fitness

- Náhrada - nahrazení staré generace novou (úplná nebo částečná)
- Kontrola ukončujících podmínek

3.3.5.1 Inicializace

Velikost populace závisí na typu problému a velikosti stavového prostoru. První generace jedinců je často vygenerována náhodně, aby lépe pokryla celou velikost prostoru možných řešení. Pokud nám to ale prostor řešení dovolí, může být první generace vybrána přímo ze zvolených jedinců.

3.3.5.2 Selekce

Ve stádiu selekce se evoluční algoritmus snaží vybrat jedince pro další křížení nebo mutaci. Výběr se přitom může, nebo nemusí, orientovat podle hodnoty fitness jedinců. Jedinci také mohou být z celé populace, nebo jen například z části populace s lepší hodnotou fitness.

3.3.5.3 Ruletová selekce

Mezi nejběžnější operátory selekce patří Ruletová selekce, kde pravděpodobnost, že jedinec bude vybrán, je přímo uměrná hodnotě jeho fitness, neboli:

$$P_i = \frac{f_i}{\sum_{j=1}^{\mu} f_j}$$

kde μ je počet jedinců v populaci a f_i je fitness jedince i. Což ale znamená, že jedinci s větší hodnotou fitness mají vyšší pravděpodobnost být vybráni, zatímco jedinci s horší fitness mají nižší šanci.

3.3.5.4 Turnajová selekce

Turnajová selekce náhodně vybere k jedinců, mezi kterými uspořádá turnaj, neboli vybere jedince s nejlepším ohodnocením fitness. Na rozdíl od ruletové selekce ale není závislá na konkrétních hodnotách fitness. Selekční tlak můžeme jednoduše upravit změnou velikosti turnaje. Pokud má turnaj více účastníků, pak jedinci s horší fitness mají menší šanci na vítězství. Při velikosti turnaje jedna je výsledek stejný jako při náhodné selekci.

3.3.5.5 Oříznutí

Při selekci oříznutím seřadíme jedince podle jejich hodnoty fitness a vybereme například nejlepší polovinu nebo třetinu pro další křížení, nebo pokud máme větší populaci než potřebujeme. Tato metoda ale není tak sofistikovaná, ani úspěšná jako jiné metody, a proto se v praxi moc nepoužívá.

3.3.5.6 Stochastické univerzální vzorkování

SUS je variantou ruletové selekce, kde má sice fitness jedince svojí váhu, ale pro procházení prostoru jedinců používáme stejné intervaly o velikosti:

$$step = \frac{\frac{f_i}{\sum_{j=1}^{\mu} f_j}}{\mu}$$

kde μ je počet jedinců v populaci a f_i je fitness jedince i, což dává jedincům s nižší hodnotou fitness větší šanci na vybrání, než při použití ruletové selekce. Tato metoda může být výhodná, obzvláště pokud nějaký jedinec má o mnoho větší fitness než ostatní.

3.3.5.7 Reprodukce

Nové jedince můžeme vytvářet stejně jako v přírodě mnoha způsoby. Pravděpodobně nejčastějším z nich je křížení. Křížení může nastávat mezi libovolným počtem rodičů, záleží jen na strategii výběru dat do potomka. Stejně tak i potomků může být vytvořeno více než dva z jednoho křížení. Běžně se ale používají dva až tři rodiče, ze kterých vznikají jeden až dva potomci. V evolučních algoritmech máme několik možností, jak křížit jedince.

3.3.5.8 Jednobodové křížení

Jednobodové křížení je nejzákladnější operátor křížení evolučních algoritmů. Jsou k němu potřeba většinou dva rodiče a vzniká jeden a více potomků. Algoritmus používaný k vytvoření potomka může být popsán v následujících krocích 3.3a:

1. Vyber bod křížení v chromozomu rodiče.
2. Oba rodiče rozděl na dvě části podle vybraného bodu.
3. Vytvoř potomka kombinací první části z prvního rodiče a druhé části z druhého rodiče.
4. Vytvoř druhého potomka kombinací první části z druhého rodiče a druhé části z prvního rodiče.

3.3.5.9 Dvoubodové křížení

Velice podobné křížení jednobodovému křížení, pouze s rozdílem, že jsou vybírána dvě místa a potomek má data střídavě z obou rodičů. Algoritmus může být popsán následovně 3.3b:

1. Vyber dva body křížení.

2. Rozděl oba rodiče do tří částí podle vybraných dvou bodů.

3. Vytvoř prvního potomka z první a třetí části prvního rodiče a prostřední části druhého rodiče.

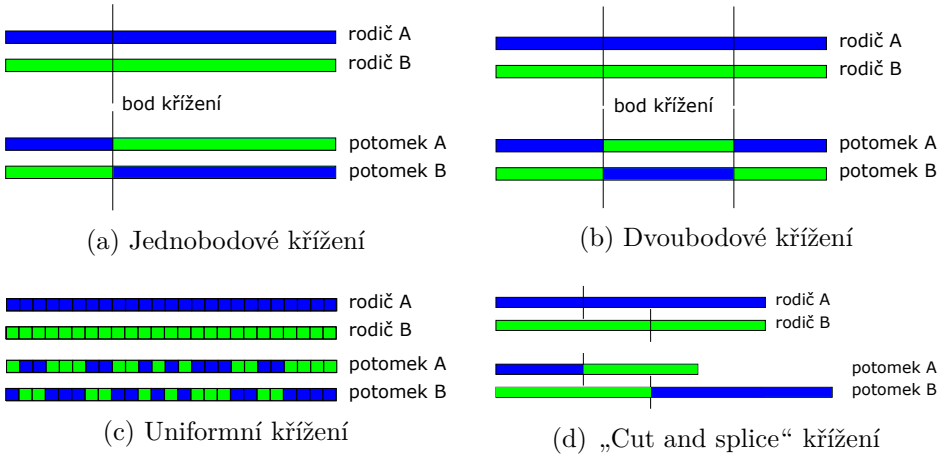
4. Vytvoř druhého potomka z první a třetí části druhého rodiče a prostřední části prvního rodiče.

3.3.5.10 *N*-bodové křížení

Opět varianta jednobodového křížení, ale s výběrem N bodů, ve kterých se střídají data, která potomek zdědí od rodičů.

3.3.5.11 Uniformní křížení

Typ křížení, ve kterém postupujeme po jednotlivých genech rodičů. V každém kroku je pak pravděpodobnost $p = 0.5$, že vybereme data z prvního rodiče. Tato metoda může dosahovat velkých zhoršení i zlepšení hodnot fitness potomků, ale dokáže se s ní lépe projít větším množstvím dat v prostoru možných řešení 3.3c.



(a) Jednobodové křížení

(b) Dvoubodové křížení

(c) Uniformní křížení

(d) „Cut and splice“ křížení

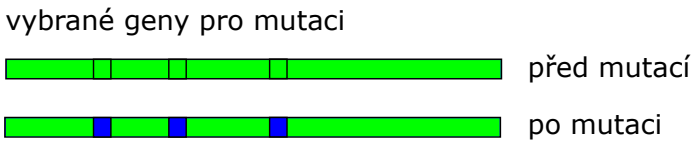
Obrázek 3.3: Druhy křížení

3.3.5.12 „Cut and splice“

Tato varianta křížení zvolí místo v genech rodičů u každého rodiče zvlášť, čímž se liší od normálních přístupů ke křížení, protože díky ní vznikají potomci, kteří nemají stejnou délku chromozomu jako rodiče. Jako přístup v této práci se nehodí, protože zde budou vytvářeni jedinci s pevně danou délkou a to stejnou, jako mají rodiče 3.3d.

3.3.5.13 Mutace

Pomocí mutace dokážeme lépe udržet diverzitu populace tím, že s malou pravděpodobností vybereme jedince, kterému následně změníme malou část genu, čímž se dokážeme posunout do oblasti prostoru řešení, ve které jsme nemuseli mít žádné jedince. Tím se může algoritmus vyhnout uváznutí v lokálním minimu, ale také se vyhne zpomalení nebo i zastavení celé evoluce. Pokud má ale mutace velkou pravděpodobnost, tak se evoluce změní v pouhé náhodné hledání. Speciální případ mutace je změna jedince zpět na jedince samotného, což ale nevadí, protože v biologické mutaci se může stát totéž 3.4.



Obrázek 3.4: Ukázka mutace

3.3.5.14 Fitness

Pokud chceme vytvořit fungující model evolučního algoritmu, musíme zvolit nějaký způsob, jak ohodnotit jedince v populaci. Mezi možné způsoby patří:

- **Objektivní ohodnocení** - Ohodnocení jedince je **objektivní**, pokud posuzuje jedince nezávisle na ostatních jedincích a nepoužívá jeho normalizaci ani jinak upravenou reprezentaci.
- **Subjektivní ohodnocení** - Ohodnocení jedince je **subjektivní**, pokud není objektivní.
- **Vnitřní ohodnocení** - Ohodnocení jedince je **vnitřní**, pokud nějak zasahuje do dalšího vývoje evoluce.
- **Vnější ohodnocení** - Ohodnocení jedince je **vnější**, pokud nemůže nijak zasahovat do dalšího vývoje evoluce[14].

Ohodnocení neboli fitness jedince určuje jeho kvalitu vůči hledanému řešení. Právě díky tomuto hodnocení algoritmus ví, jaké řešení má hledat, protože méně validní jedinci mají horší fitness než ostatní a tím pádem i například menší šanci na reprodukci.

Díky těmto vlastnostem můžeme fitness evolučních algoritmů zařadit do kategorie vnitřních ohodnocení.

Existuje mnoho přístupů k ohodnocení jedinců pomocí fitness. Za porušená omezení můžeme odčítat „body“, a následně hledat jedince, který bude mít minimálně určitý počet bodů, nebo naopak body můžeme přičítat za splněné podmínky a o porušení kritérií se nestarat. Při použití některých selekčních

metod je vhodné normalizovat hodnoty fitness, protože jinak může například selekce pomocí rulety vracet pouze náhodné jedince.

3.3.5.15 Ukončující podmínky

Mezi podmínky ukončující evoluční algoritmus patří například:

- Nalezení vyhovujícího řešení.
- Vytvoření požadovaného počtu generací.
- Nalezené řešení se už nemůže dále zlepšovat.

3.3.6 Elitismus

Elitismus je metoda používaná při tvorbě nové generace jedinců, kdy nejlepší jedince (většinou jednoho až dva) z předešlé generace (podle hodnoty fitness) přeneseme přímo do nově tvořené generace, ještě před fází křížení. Takto se nemůže stát, že by nejlepší jedinec například zmutoval nebo nějak jinak vypadl z generace vlivem náhodných procesů a tím jsme o něj přišli. Kvalita řešení se tak může pouze zvyšovat, ovšem také se může tímto narušit běh evoluce, pokud by takových jedinců bylo příliš mnoho.

3.4 Koevoluční algoritmy

Pro zlepšení výkonu evolučních algoritmů v problémech s vysokým počtem dimenzí bylo vynalezeno mnoho technik, mezi jinými například koevoluční algoritmy. Problémem evolučních algoritmů totiž může být provádění evoluce na jediné populaci stejného druhu jedinců, protože v přírodě probíhá evoluce mnoha populací jedinců současně a jedinci z různých populací se také navzájem ovlivňují. Tím se zvyšuje selekční tlak a urychluje se tak vývoj jednotlivých populací. Proto vznikly koevoluční algoritmy, jako snaha najít řešení problému pomocí více populací stejného nebo různého druhu jedinců, kteří spolu buď soupeří nebo spolupracují.

Pomocí koevolučních algoritmů se snažíme zajistit stabilitu evoluce a urychlit konvergenci fitness[15].

3.4.1 Kooperativní koevoluce

V kooperativní koevoluci je implementována idea „Rozděl a panuj“, která rozdělí problém o velkém počtu dimenzí na více podproblémů s menším počtem dimenzí, kde každý z nich tvoří podprostor řešení. Na každý z těchto podprostorů je aplikován nějaký typ evolučního algoritmu a následně se menší řešení spojí do celku, který je poté ohodnocen [16]. Jako příklad může být uvedena symbióza.

3.4.2 Kompetitivní koevoluce

Někdy může být užitečné nechat jedince v populaci soupeřit navzájem. Například v modelu lovec-kořist, kde se jedna strana snaží vytvořit rychlejšího, zdatnějšího lovce, zatímco druhá strana se snaží například o vytvoření menší, chytřejší kořisti, která se zvládne lépe před lovcem schovat. Postupem generacemi se zlepšují pouze vlastnosti obou generací, které se jim hodí pro lepší soupeření. Tyto algoritmy se nazývají kompetitivní koevoluční algoritmy.

3.5 Memetické algoritmy

Memetické algoritmy jsou metaheuristické prohledávací přístupy založené na populaci. Jsou inspirované Neo-Darwinistickým principem přírodní evoluce a Dawkinsovou teorií memů (anglicky meme), definovanou jako jednotku kulturní evoluce, která je schopná provést lokální zlepšování. Definice podle Dawkinse:

> „Příklady memů jsou melodie, nápady, fráze, móda, způsob vytváření keramiky nebo stavění oblouků. Jako se geny rozmnožují v genofondu přeskakováním z jedince na jedince pomocí spermií a vajíček, tak se i memy rozmnožují v *memefondu* přeskakováním z mozku do mozku pomocí procesu, který v širokém slova smyslu můžeme nazvat napodobování.“[17]

Na memetické algoritmy můžeme pohlížet jako na spojení mezi globálními prohledáváními založenými na populaci a lokálně zlepšujícími metodami, což znamená, že, jakmile je tato technika správně vyvinuta, můžeme získat řešení vyšší kvality mnohem efektivněji.

V evolučních algoritmech je snaha napodobit biologickou evoluci, zatímco memetické algoritmy slouží k napodobení kulturní evoluce. Lze si to představit jako jednotky informací, které se předávají zatímco si spolu lidé vyměňují názory. V memetických algoritmech je populace složená pouze z lokálně optimálních řešení. Základní kroky kanonického memetického algoritmu pro jednoduchou nelineární optimalizaci založenou na genetickém algoritmu jsou znázorněny v následujícím pseudokódu 3.1 [18].

3.5.1 Použití memetických algoritmů

Mezi nejtypičtější problémy pro využití memetických algoritmů patří NP-úplné problémy jako například: Problém obchodního cestujícího, problém batohu, hledání nezávislé množiny, součet podmnožiny, vrcholové pokrytí, a podobně. Ale své místo si najdou i v jiných odvětvích například robotice, elektronice, medicíně, ekonomice, oceánografii, matematice a tak dále. [19]

Algorithm 3.1 Kanonický memetický algoritmus

BEGIN
Inicializace: Vytvoř počáteční generaci GA.
while Nejsou splněny ukončující podmínky **do**
  Ohodnoť všechny jedince v populaci
  for každého jedince v populaci **do**
    Proveď lokální vylepšení a nahraď genotypy a/nebo fenotypy v generaci lepším řešením v závislosti na výběru Lamarckistického nebo Baldwinistického učení.
  end for
  Vytvoř novou generaci standardními operátory GA, tj. křížení, mutace a selekce.
end while
END

Kapitola 4

Realizace

Pro výslednou implementaci programu byl vybrán jazyk Python 3 ve verzi 3.4.

4.1 Spuštění

Program je možné spustit pomocí příkazové řádky zadáním souboru *scheduling.py* a přidáním cesty k *CSV* souboru obsahujícím seznam ID zaměstnanců s uvedenými centry, kde mohou pracovat, a druhé cesty k *CSV* souboru se seznamem ID zaměstnanců a směnami, kdy mají dovolenou. Dále mohou následovat nepovinné přepínače a to:

-s [datum] - Pro zadání data od kterého se má rozvrh vytvořit. Pokud není zadáno implicitně se použije datum spuštění. Formát data je očekáván následující: YYYY-MM-DD.

-l [int] - Udává délku rozvrhu zaměstnanců ve dnech. Výchozí hodnota je nastavena na 28.

-u [person/center] - Vybírá, zda se budou rozvrhy tvořit z pohledu center nebo zaměstnanců. Výchozí hodnotou je center.

Výsledný rozvrh je po dokončení programu zapsán do souboru *result.txt* nacházejícího se ve stejné složce jako zdrojové soubory.

4.2 Reprezentace informací o centrech a zaměstnancích

Informace o centrech a zaměstnancích jsou uloženy ve slovnících *people* pro zaměstnance a *centers* pro centra, které jsou vytvořeny ve funkci *remake* v souboru *remake.py* z obou vstupních souborů.

Ve slovníku *people* jsou lidé rozděleni podle ID, kde ke každému ID patří list, ve kterém je další list dvojic [den, směna], kde směna je typu boolean a pokud jde o denní směnu má hodnotu *True*, a list center, ve kterých zaměstnanec s daným ID může pracovat.

Ve slovníku *centers* je seznam center, které jsou uvedeny v souboru s přiřazenými zaměstnanci k centrům. Pokud nějaké centrum nemá přidělené zaměstnance, pak takové centrum není zpracováno.

4.3 Evoluce

Evoluce celá probíhá ve třídě *Evolution*, v souboru *evolution.py*, jak bylo nastíněno v kapitole 3.3 a využívající kooperativní koevoluci popsanou v sekci 3.4.1. Pro reprezentaci jedinců používá třídu *Schedule*.

V souboru *schedule.py*, je implementována třída *Schedule* reprezentující jednotlivce celé evoluce, to znamená jeden rozvrh. Tento rozvrh je ale složen z menších jednotek neboli třídy *Unit* ze souboru *unit.py*, které reprezentují „malé“ rozvrhy jednotlivých lidí nebo center. Ve třídě *Schedule* můžeme vytvářet nové rozvrhy z jednotek (*units*), rozvrhy nechávat ohodnocovat funkcí fitness a porovnávat rozvrhy mezi sebou podle jejich hodnoty fitness.

Kvůli možnosti vytvářet rozvrhy podle lidí nebo center je třída *Unit* realizována jako abstraktní třída, ze které dědí dvě další třídy:

- Center - třída reprezentující rozvrh pro jedno centrum. Rozvrhem se zde rozumí list dvojic, kde indexem listu je rozuměn den a pozicí v dvojici ranní, nebo noční směna a na každé této pozici je uloženo ID zaměstnance, který má směnu přidělenou.
- Person - třída reprezentující rozvrh pro jednoho člověka. Rozvrhem zde je list o takové délce, aby zaměstnanci vycházely tři směny týdně. Na každé pozici v listu je trojice ve složení [centrum, den, 0/1], kde 0 znamená denní směnu a 1 znamená noční směnu.

V obou třídách jsou navíc implementovány částečné fitness (protože zde není k dispozici celý rozvrh můžeme zkontrolovat pouze část podmínek) a hill climbing, který se právě podle hodnot částečných fitness snaží jednotku vylepšit.

Algoritmus Hill Climbingu je navrhnut tak, aby postupoval od opravování „dražších“ chyb k „levnějším“ a tím bylo ušetřeno prohledávání celého okolí, protože pomocí částečné fitness může být odhadován nejlepší posun a tím může být Hill Climbing podstatně zkrácen.

Dále jsou v souboru *unit.py* implementovány funkce křížení z důvodu potřeby křížit právě jednotky koevoluce. Nakonec byly vybrány tři typy křížících algoritmů a to: jednobodové křížení, dvoubodové křížení a uniformní křížení, které z daných rodičů vybraných pomocí selekce vytvoří právě dva potomky.

Ve třídě *Evolution* jsou implementovány následující algoritmy selekce: turnaj, SUS a ruletová selekce, pomocí nichž jsou vždy vybíráni právě dva rodiče.

Aby se dosáhlo větší úspěšnosti ruletové selekce a SUS, byla implementována normalizace hodnot fitness, protože rozdíly mezi jedinci byli tak malé, že se zmíněné operátory chovaly jako náhodný výběr.

Samotná evoluce pak probíhá v metodě *evolve*, kde nejlepší jedinec z minulé generace je rovnou zkopírován do nové populace (elitismus), a poté se tvoří nová generace, která je o dvacet procent větší než je povolená velikost generace. Na konci iterace se ale populace *ořízne* podle hodnoty fitness zpět na správný počet, čímž se zbaví nejhorších jedinců.

Jako aplikace memetického algoritmu je zde implementován Hill Climbing, který každou pátou generaci vybere pět náhodných jedinců z populace, které se bude snažit jednou změnou co nejvíce zlepšit. Hill Climbing mění pouze jednotky koevoluce (tj. jednotlivé rozvrhy pro centra nebo lidi) a ne celý rozvrh jako takový.

Pokud má rozvrh ohodnocení fitness rovné optimální hodnotě ještě před koncem celé evoluce, znamená to, že algoritmus našel optimální rozvrh. Cyklus se předčasně ukončí a výsledný rozvrh je zapsán do souboru *result.txt*, který je umístěn ve složce se zdrojovými kódy.

Pokud algoritmus nedokáže najít řešení během stanoveného počtu generací, vypíše nejlepší nalezené řešení do souboru *result.txt*.

4.4 Fitness

Ohodnocovací funkce je implementována v souboru *fitness.py* a je rozdělena do menších funkcí, kde každá funkce kontroluje pouze jednotlivou část rozvrhu. Funkce je nastavena tak, že se algoritmus snaží minimalizovat chybu od nejlepší hodnoty funkce, které je rovno 1 000 000. Pokud daný rozvrh nesplňuje nějaká omezení pak jsou mu strhávány body. Fitness funkce hodnotí rozvrhy podle tvrdých a měkkých omezení popsaných v sekci 1.1. Za porušení tvrdých omezení se strhává 100-150 bodů a porušení měkkých 10-25 bodů.

Kapitola 5

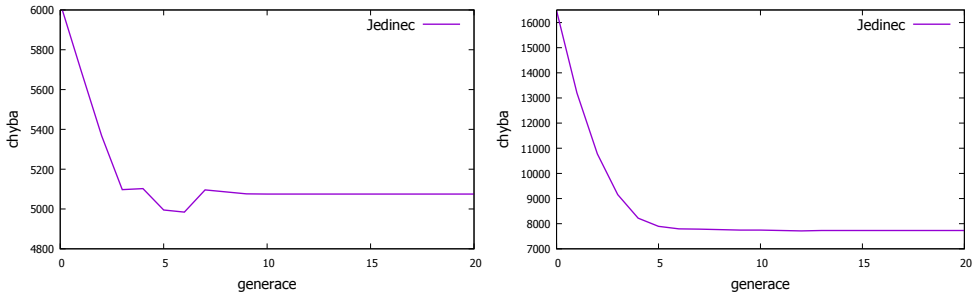
Výsledky

Pro všechny populační přístupy byla zvolena populace o velikosti 600 jedinců, maximální počet generací 1500, šance na křížení 85%, šance na mutaci 4%, velikost mutace na 10% z velikosti jedince a jako selekce byl použit turnaj o velikosti 5. Rozvrhy byly generovány na 31 dnů pro 17 zaměstnanců a 3 datová centra. Všechny konfigurace byly spuštěny 10x a z jejich běhů spočítány průměry, které jsou zobrazeny v následujících grafech.

5.1 Nepopulační přístupy

5.1.1 Hill Climbing

Algoritmus byl spuštěn pouze pro jednoho jedince, na kterého byl Hill Climbing aplikován v každém kroku. Jak je vidět v grafech 5.1a a 5.1b algoritmus je zastaven po 20 krocích, protože již do přibližně 10. kroku dosáhne lokálního optima, ze kterého se už není schopen dostat.

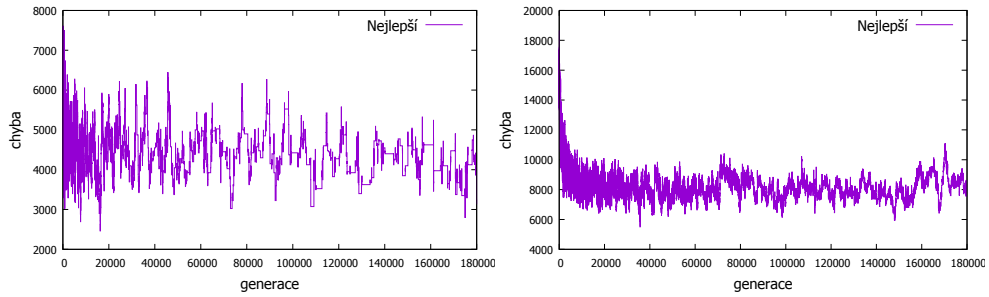


(a) center (b) person

Obrázek 5.1: Hill Climbing

5.1.2 Simulované žíhání

Algoritmus simulovaného žíhání byl implementován s využitím stejné fitness funkce jako evoluční algoritmy. Při testování se ukázalo, že je algoritmus velmi náchylný na nastavení parametrů. Z grafů 5.2a a 5.2b může být vidět, že algoritmus není schopen nalézt globální optimum.



(a) center (b) person

Obrázek 5.2: Simulované žíhání

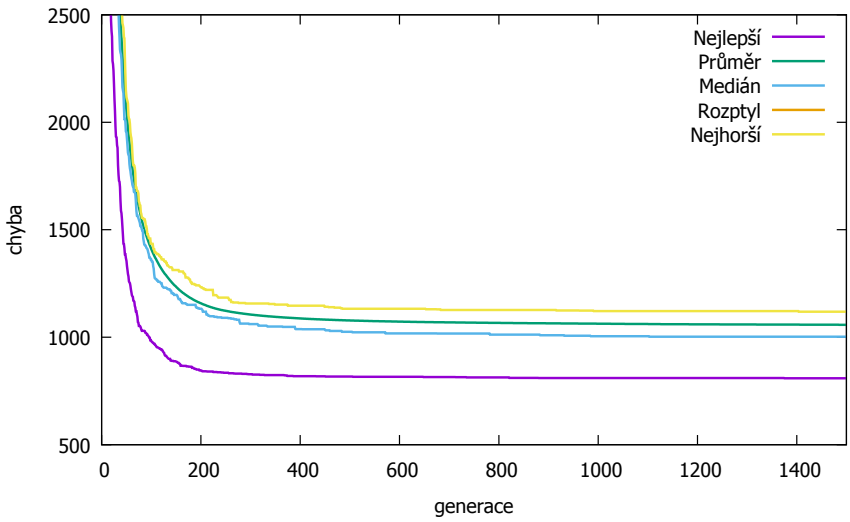
5.2 Populační přístupy

5.2.1 Evoluce

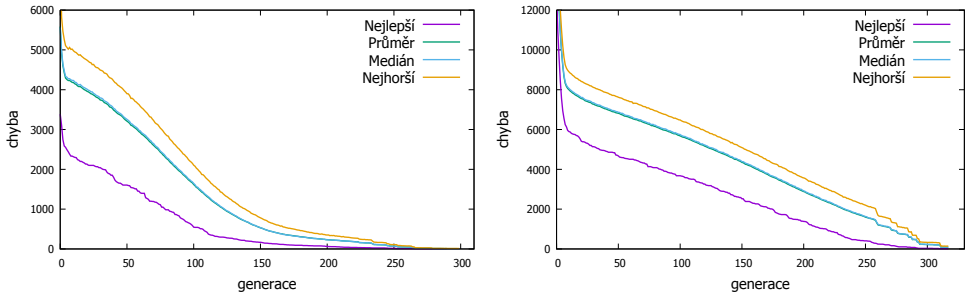
Nejdříve byla implementována pouze samotná evoluce, která ale nedokázala dosáhnout optimálního řešení, jak je vidět v grafu 5.3. Zobrazena je průměrná chyba nejlepšího a nejhoršího jedince spolu s průměrem, mediánem a rozptylem chyb celé generace. Vybráno zde bylo dvoubodové křížení, protože se s ním dosahovalo nejlepších výsledků, spolu s ruletovou selekcí. Z grafu je vidět, že samotná evoluce nedokáže najít optimální řešení, proto byla dále implementována ještě kooperativní koevoluce s memetickým algoritmem.

5.2.2 Kooperativní koevoluce

V kooperativní koevoluci bylo vybráno dvoubodové křížení spolu s turnajovou selekcí o velikosti pět, protože v této konfiguraci dosahoval algoritmus nejlepších výsledků. Jak je vidět v grafech 5.4a a 5.4b, tak koevoluce je schopná dosáhnout optimálního řešení již po přibližně 300 iteracích. Pro osoby je algoritmus pomalejší a horší než pro centra, protože osob je více než center a tak probíhá více evolucí jedinců najednou a je zde větší šance na kolizi ve výsledném rozvrhu.



Obrázek 5.3: Postup evoluce.



(a) center (b) person

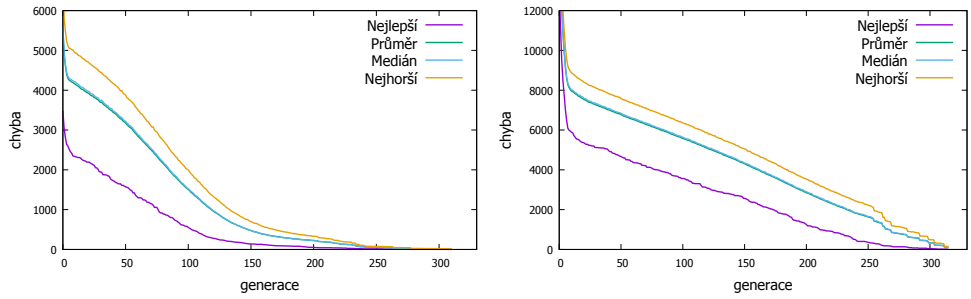
Obrázek 5.4: Kooperativní koevoluce.

5.2.3 Memetický algoritmus

Dále byla vyzkoušena možnost memetického algoritmu, od které se očekávalo urychlení najití optima. Jak je ale možné vidět v grafech 5.5a a 5.5b, tak Hill Climbing vylepšil kooperativní koevoluci skoro zanedbatelně a to hlavně z důvodu vylepšování pouze jednotek rozvrhu a ne celého řešení. Takto mohou i po úspěšném Hill Climbingu vzniknout horší rozvrhy.

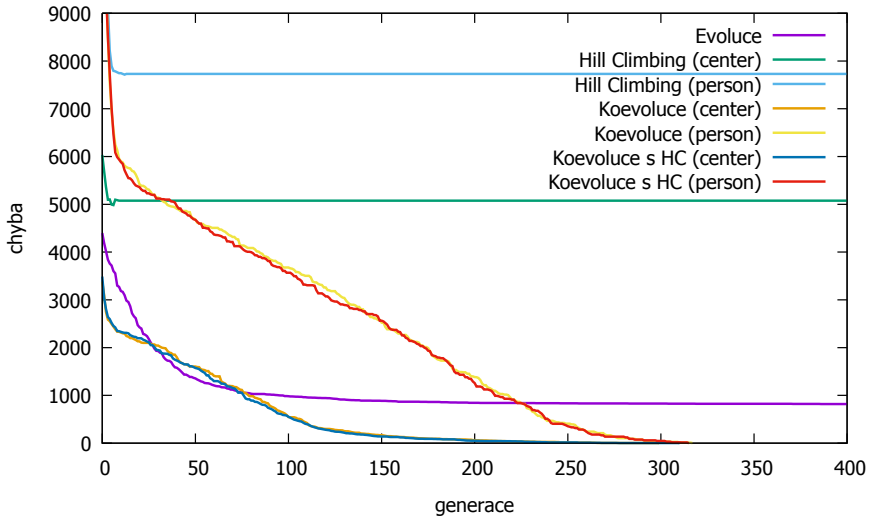
5.3 Shrnutí dosažených výsledků

V grafu 5.6 je vidět porovnání nejlepších jedinců všech výše zmíněných možností. Z toho lze usuzovat, že nejlepší výsledky poskytuje koevoluce, která má za jednotky centra, přičemž Hill Climbing na koevoluci v podstatě nemá vliv. Evoluce a samotný Hill Climbing nejsou schopny ani nalézt optimum.



(a) center (b) person

Obrázek 5.5: Kooperativní koevoluce spojená s Hill Climbingem.



Obrázek 5.6: Graf průměrných ohodnocení nejlepších jedinců.

5.4 Wilcoxon rank-sum test

Na statistická data získaná z běhů programů byl použit Wilcoxon rank-sum test pro porovnání různých přístupů použitých možností. Testován byl předpoklad, že dva porovnávané vzorky jsou ze stejné distribuce, a pokud je hodnota ρ menší než 0.05 tak platí alternativní předpoklad, tedy že nejsou ze stejné distribuce. Pokud platí alternativní předpoklad tak je také možné říci, že jeden algoritmus je lepší než druhý.

Jako varianty programů pro rank-sum test byl zvolen memetický algoritmus pro *center* a *person* a koevoluce pro *center* i *person*.

Dle výsledků v tabulce 5.1 pro ρ, test dává významný důkaz pro tvrzení, že koevoluce pro *center* je lepší než koevoluce pro *person* a memetický algoritmus pro *person*. A dle tabulky 5.2 pro Z je vidět, že memetický algoritmus pro *center* je v průměru rychlejší než memetický algoritmus pro *person*. Pro

Varianta algoritmu		Hodnota ρ
memetický algoritmus center	memetický algoritmus person	0,007285
koevoluce center	koevoluce person	0,010165
memetický algoritmus center	koevoluce center	0,226476
memetický algoritmus person	koevoluce person	0,939743

Tabulka 5.1: Tabulka ρ hodnot

Varianta algoritmu		Hodnota Z
memetický algoritmus center	memetický algoritmus person	-2.683548
koevoluce center	koevoluce person	-2.570158
memetický algoritmus center	koevoluce center	-1.209486
memetický algoritmus person	koevoluce person	0.075592

Tabulka 5.2: Tabulka Z hodnot

toto tvrzení opět dává test významný důkaz. Dále je možné vidět, že použití memetického algoritmu není prokazatelně lepší než koevoluce.

5.5 Porovnání s běžně užívanými přístupy

Mezi běžně užívané přístupy stále patří ruční vytváření rozvrhu a vyčerpávající prohledávání stavového prostoru, kde prohledávání stavového prostoru má výhodu v tom, že pokud existuje optimální řešení, pak ho určitě najde, ale prohledávání trvá velice dlouho a činí tím toto řešení nepoužitelné. Mezi výhody ručního vytváření patří větší možnost vkládání požadavků a také možnost jednotlivých optimalizací na základě konkrétních žádostí zaměstnanců, ale nevýhodou je opět vysoká časová náročnost. Řešení vyvinuté a implementované v této práci je mnohem efektivnější a také rychlejší než oba přístupy.

Závěr

V této práci byly nejdříve analyzovány požadavky společnosti Seznam.cz na vytváření rozvrhů a rozložení směn, a následně různé heuristické algoritmy k vytvoření algoritmu pro rozvrhování směn zaměstnanců.

Jako první přístup byl zvolen evoluční algoritmus, který se ale neukázal jako dobrá cesta, protože nebyl schopen najít optimální řešení. Proto byl dále upraven na kooperativně koevoluční algoritmus, kde jako jednotky koevoluce mohou být zvolena datová centra nebo zaměstnanci.

Tento přístup je schopný najít optimální výsledek, a pokud jsou jednotkami centra, algoritmus dokáže najít optimální rozvrh velmi rychle jak přes počty generací, tak i ve skutečném čase. Pokud zvolíme jako jednotky zaměstnance, pak tvorba rozvrhu trvá déle jak v generacích, tak i v čase, ale výsledný rozvrh může být výhodnější, protože je v něm obsazeno na některých směnách více zaměstnanců, což poskytuje větší variabilitu a může vést k větší spokojenosti zaměstnanců.

Použití memetických algoritmů (zde využití Hill Climbingu v koevoluci) v tomto problému nevedlo k výraznému zlepšení oproti koevoluci. To může být způsobeno použitím Hill Climbingu na jednotky koevoluce a ne na celý rozvrh, kdy zlepšením rozvrhu jednotlivce mohou vzniknout kolize s ostatními jedinci ve výsledném rozvrhu. Například pokud je zaměstnanci posunuta směna, protože měl dvě směny po sobě, pak ve výsledném rozvrhu může vzniknout za posunutou směnu prázdné místo, což je v rozporu s požadovanými tvrdými omezeními.

Oproti evolučním přístupům podávaly algoritmy založené na lokálním prohledávání výrazně horší výsledky. V žádném z běhů se jim nepodařilo dosáhnout optimálního řešení před vyčerpáním nastaveného maximálního počtu ohodnocení.

Jako možné budoucí vylepšení se naskýtá možnost zlepšení lokální optimalizace memetického algoritmu a to změnou aplikace Hill Climbingu z jednotek rozvrhu na celé rozvrhy, nebo popřípadě výměna Hill Climbingu za jiný heuristický algoritmus.

V další fázi bude třeba zakomponovat program do systému společnosti Seznam.cz, kde se momentálně čeká na rozšíření docházkového systému, tak aby odpovídal parametrům práce v datových centrech.

Literatura

[1] Solos, I. P.; Tassopoulos, I. X.; Beligiannis, G. N.: A generic two-phase stochastic variable neighborhood approach for effectively solving the nurse rostering problem. *Algorithms*, ročník 6, č. 2, 2013: s. 278–308.

[2] Xu, H.; Zhou, Z.: Hill-climbing genetic algorithm optimization in cognitive radio decision engine. In *15th IEEE International Conference on Communication Technology (ICCT)*, Listopad 2013, s. 115–119, doi: 10.1109/ICCT.2013.6820357.

[3] Mausa, G.; Grbac, T.; Basic, B.; aj.: Hill Climbing and simulated annealing in large scale next release problem. In *EUROCON, 2013 IEEE*, Červenec 2013, s. 452–459, doi:10.1109/EUROCON.2013.6625021.

[4] Coppin, B.: *Artificial Intelligence Illuminated*. Jones and Bartlett illuminated series, Jones and Bartlett Publishers, 2004, ISBN 9780763732301. Dostupné z: `http://books.google.cz/books?id=LcOLqodW28EC`

[5] Kicinger, R.; Arciszewski, T.; Jong, K. D.: Evolutionary Computation and Structural Design: A Survey of the State-of-the-art. *Comput. Struct.*, ročník 83, č. 23-24, Září 2005: s. 1943–1978, ISSN 0045-7949, doi:10.1016/j.compstruc.2005.03.002. Dostupné z: `http://dx.doi.org/10.1016/j.compstruc.2005.03.002`

[6] Hynek, J.: *Genetické algoritmy a genetické programování*. Praha: Grada, první vydání, 2008, ISBN 9788024726953.

[7] Eiben, A. E.; Smith, J. E.: *Introduction to evolutionary computing*. Springer Science & Business Media, 2003.

[8] Back, T.; Hammel, U.; Schwefel, H.-P.: Evolutionary computation: comments on the history and current state. *IEEE Transactions on Evolutionary Computation*, ročník 1, č. 1, Duben 1997: s. 3–17, ISSN 1089-778X, doi:10.1109/4235.585888.

[9] POŠÍK, P.: Paralelní genetické algoritmy. 2001.

[10] Poli, R.: Why the schema theorem is correct also in the presence of stochastic effects. *Proceedings of the 2000 Congress on Evolutionary Computation*, ročník 1, 2000: s. 487–492 vol.1, doi:10.1109/CEC.2000.870336.

[11] Srinivas, M.; Patnaik, L. M.: Genetic algorithms: A survey. *Computer*, ročník 27, č. 6, 1994: s. 17–26.

[12] Koza, J. R.: *Genetic programming: on the programming of computers by means of natural selection*, ročník 1. MIT press, 1992.

[13] Kawanaka, H.; Yamamoto, K.; Yoshikawa, T.; aj.: Genetic algorithm with the constraints for nurse scheduling problem. In *Proceedings of the 2001 Congress on Evolutionary Computation*, ročník 2, 2001, s. 1123–1130 vol. 2, doi:10.1109/CEC.2001.934317.

[14] Wiegand, R. P.: *An analysis of cooperative coevolutionary algorithms*. Dizertační práce, Citeseer, 2003.

[15] Hrbáček, R.: *Koevoluční algoritmus v FPGA*. Dizertační práce, Vysoké Učení Technické v Brně, 2013.

[16] Zhang, K.; Li, B.: Cooperative Coevolution with global search for large scale global optimization. In *IEEE Congress on Evolutionary Computation (CEC)*, Červen 2012, s. 1–7, doi:10.1109/CEC.2012.6252936.

[17] Dawkins, R.: *The selfish gene. New York*. Oxford university press, 2006.

[18] Ong, Y.-S.; Lim, M.-H.; Zhu, N.; aj.: Classification of adaptive memetic algorithms: a comparative study. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, ročník 36, č. 1, Únor 2006: s. 141–152, ISSN 1083-4419, doi:10.1109/TSMCB.2005.856143.

[19] Moscato, P.; Cotta, C.: A Gentle Introduction to Memetic Algorithms. In *Handbook of Metaheuristics*, *International Series in Operations Research and Management Science*, ročník 57, editace F. Glover; G. Kochenberger, Springer US, 2003, ISBN 978-1-4020-7263-5, s. 105–144, doi:10.1007/0-306-48056-5_5.

Příloha **A**

Seznam použitých zkratek

SUS Stochastické univerzální vzorkování

NSP Nurse Scheduling Problem

NRP Nurse Rostering Problem

GA Genetický algoritmus

HC Hill Climbing

CSV soubor s daty oddělenými středníky (Comma Separated Values)

Příloha **B**

Obsah přiloženého USB disku

- `readme.txt` ..... stručný popis obsahu USB disku
- `scheduling.py` ..... spustitelný skript
- `csv` ..... adresář s ukázkovými soubory CSV
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... adresář s textem práce
- `python` ..... adresář s instalačními soubory Pythonu