VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

APLIKACE GENETICKÉHO PROGRAMOVÁNÍ

GENETIC PROGRAMMING APPLICATIONS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **TADEÁŠ KACHYŇA**
AUTHOR

VEDOUCÍ PRÁCE **Ing. MICHAL BIDLO, Ph.D.**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

143371

Ústav:	Ústav počítačových systémů (UPSY)
Student:	**Kachyňa Tadeáš**
Program:	Informační technologie
Specializace:	Informační technologie
Název:	**Aplikace genetického programování**
Kategorie:	Umělá inteligence
Akademický rok:	2022/23

Zadání:

1. Seznamte se s variantami genetického programování (GP) a s možnostmi použití GP pro řešení různých úloh.
2. Po dohodě s vedoucím práce zvolte vhodnou variantu GP a problém, který bude předmětem řešení.
3. Ve vhodném prostředí implementujte evoluční systém pro řešení problému z bodu 2 pomocí vybrané varianty GP. Zaměřte se na různé instance problému a různá nastavení evolučního systému.
4. Proveďte sadu experimentů za účelem identifikace vhodných nastavení GP pro dosažení co nejlepších výsledků. Experimenty statisticky vyhodnoťte.
5. Dosažené výsledky srovnejte s těmi dostupnými v literatuře, diskutujte jejich vlastnosti a možnosti pokračování projektu.

Literatura:

- Dle pokynů vedoucího projektu.

Při obhajobě semestrální části projektu je požadováno:
Splnění bodů 1 a 2 zadání, demonstrace prototypu evolučního systému z bodu 3.

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

Vedoucí práce:	**Bidlo Michal, Ing., Ph.D.**
Vedoucí ústavu:	Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání:	1.11.2022
Termín pro odevzdání:	10.5.2023
Datum schválení:	31.10.2022

Abstrakt

Tato práce se věnuje problematice úlohy Langtonova mravence a využití stromového a lineárního genetického programování jako metod pro řešení tohoto problému. Langtonův mravenec je abstraktní matematický model, jenž využívá jednoduchých pravidel k pohybu mravence po dvoudimenzionální mřížce. Úkolem je sesbírat všechnu potravu v co nejmenším počtu kroků. Cílem práce je tedy aplikace výše dvou zmíněných technik na vybrané instance tohoto problému, následné provedení experimentů a statistické vyhodnocení včetně porovnání výsledků s těmi dostupnými v literatuře. Mimo to bude cílem dané metody optimalizovat různými vylepšeními pro dosáhnutí co nejlepších výsledků.

Abstract

This thesis deals with the problem of the Langton's artificial ant task and the use of tree-based and linear genetic programming as methods for solving this problem. Langton's ant is an abstract mathematical model that uses simple rules to move an ant along a two-dimensional grid. The task is to collect all the food in as few steps as possible. The goal of the thesis is the application of the two techniques mentioned above to selected instances of this problem, subsequent experimentation and statistical evaluation, including a comparison of the results with those available in the literature. In addition, the aim will be to optimize the given methods with various improvements to achieve the best possible results.

Klíčová slova

stromové genetické programování, lineární genetické programování, evoluční algoritmy, langtonův mravenec

Keywords

tree-based genetic programming, linear genetic programming, evolutionary algorithms, langton's ant

Citace

KACHYŇA, Tadeáš. *Aplikace genetického programování*. Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Michal Bidlo, Ph.D.

Aplikace genetického programování

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Michala Bidla, PhD. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Tadeáš Kachyňa
8. května 2023

Poděkování

Tímto bych rád poděkoval Ing. Michalu Bidlovi, PhD. za odborné vedení práce, nápomocnost a věnovaný čas při konzultacích.

Obsah

Seznam některých použitých zkratek

GP Genetické programování

TGP Stromové genetické programování

LGP Lineární genetické programování

GA Genetické algoritmy

ES Evoluční strategie

EP Evoluční programování

Kapitola 1

Úvod

V této práci se budu zabývat genetickým programováním a konkrétněji jeho dvěma variantami – stromovým a lineárním. Tyto varianty budu následně aplikovat na vybrané instance zvoleného problému a porovnávat jejich efektivitu a schopnost najít co nejlepší řešení.

Genetické programování (GP) je jednou z několika programovacích technik spadajících pod evoluční algoritmy zpopularizovanou zejména americkým vědcem a matematikem J. Kozou, který o ní sepsal nemálo úspěšných publikací. Jedná se o metodu strojového učení, kdy se každou iterací tohoto algoritmu přibližujeme více k požadovanému řešení. Na počátku je inicializována populace náhodných jedinců, která je v průběhu generací modifikována za pomocí genetických operátorů. Po každé generaci je populace ohodnocena tzv. fitness funkcí, která nám udává kvalitu jednotlivých řešení. Takto běží algoritmus do té doby, dokud není přerušen ukončovací podmínkou, nebo není nalezeno nejlepšího řešení. Díky své popularitě nabylo GP několika podob, aby bylo lépe aplikovatelné na různé problémy. Často se tyto varianty liší právě ve své reprezentaci.

Tato technika má v reálném světě mnoho aplikací počínaje od návrhu inženýrských systémů, klasifikace a rozpoznávání obrázků, navrhování elektrických obvodů až třeba po návrh a řízení robotů. Mezi největší výhody GP patří objevení nových a nečekaných řešení, která by nemusela být tradičními metodami nalezena, flexibilita – tedy schopnost adaptovat se na široké spektrum různorodých problémů, škálovatelnost na velmi komplexní problémy s mnoha proměnnými nebo schopnost vypořádat se stochasticitou.

Cílem této práce je porovnání dvou různých variant GP na vybraných instancích Langtonova mravence. Tato úloha představuje abstraktní matematický model, kdy se umělý mravenec snaží nalézt nejlepší cestu na dvojdimenzionální mřížce k sesbírání potravy v ideálně co nejmenším počtu kroků. Mimo to se v práci budu zabývat zefektivněním a modifikací již zavedených metod, např. genetických operátorů, k nalezení lepších řešení.

Obsah práce je rozdělen do kapitol následovně:

- V 2. kapitole je představeno stromové GP, vysvětlena základní terminologie pojící se s tímto tématem, obecné koncepce aplikovatelné na všechny varianty GP a v závěru kapitoly se nachází krátké srovnání s ostatními evolučními algoritmy.
- V 3. kapitole je popsána varianta lineárního GP, její charakteristika a reprezentace, variační operátory včetně grafického znázornění a v závěru srovnání se stromovým GP.
- V 4. kapitole jsou prezentovány výhody použití metody GP k řešení reálných problémů, následně je detailně představeno několik aplikací.

- 5. kapitola objasňuje úlohu Langtonova mravence a vlastní přístup k jejímu řešení.
- 6. kapitola se odkazuje na popis implementace obou variant GP na problému Langtonova mravence.
- V 7. kapitole je představen způsob experimentování, následné experimenty, jejich vyhodnocení, souhrnné zhodnocení a pojednání o dalších vylepšeních.

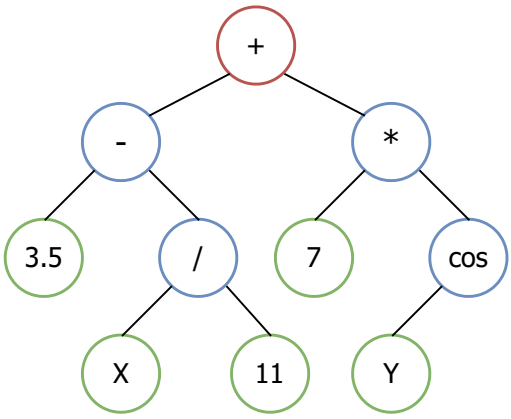
Kapitola 2

Genetické programování

Tato kapitola se zabývá popisem stromového genetického programování a objasněním obecných konceptů aplikovatelných pro všechny varianty GP. Taktéž tu je zmíněna a vysvětlena základní terminologie, která s touto technikou programování souvisí a bude potřebná pro porozumění dalších částí této práce. Ke konci kapitoly je provedeno krátké srovnání s dalšími známými evolučními algoritmy. Tato kapitola vychází především z následujících zdrojů [11, 6, 3, 13].

Genetické programování je jedna z nejnovějších technik evolučních algoritmů založená na strojovém učení sloužící k řešení netriviálních úloh za pomocí biologické evoluce reprezentující jednotlivá řešení jako stromové struktury, viz obr. 2.1. Jiným způsobem by se tato technika dala popsat následujícím citátem.

How can computer learns to solve problems without being explicitly programmed? In other words, how can computers be made to do what is needed to be done, without being told exactly how to do it. – Arthur Samuel (1959)



$$(3.5 - \frac{x}{11}) + (7 + cos(y))$$

Obrázek 2.1: Reprezentace programu pomocí stromového GP a jeho ekvivalentního matematického zápisu. Barevně jsou zde zvýrazněny kořenový (červený), vnitřní (modré) a vnější (zelené) uzly.

Od ostatních evolučních algoritmů se liší nejen svou reprezentací, ale taky svou aplikací na velkou škálu různých problémů včetně optimalizace, predikce, kontroly řízení a datové analýzy. Má schopnost najít řešení, která jsou jednoduchá a zároveň efektivní a také kompetenci řešit problémy s komplexními datovými strukturami a vícero objekty. V průběhu dalších let vznikly další varianty GP. Vyjma stromového je rozlišováno např. mezi lineárním [4], kartézským [15], zásobníkovým [18] nebo gramatickou evolucí [21].

Oblast GP byla poprvé představena v 80. letech americkým vědcem a matematikem Johnem Kozou. Byl to právě on, kdo aplikoval genetické algoritmy na jazyku LISP k řešení velkého rozsahu různých problémů. Ačkoliv myšlenka zkombinovat genetické algoritmy a počítačové programy tu už byla mnohem dříve [21]. Koza zároveň konstatuje, že je GP nejobecnějším paradigmatem strojového učení [14]. Následně sepsal několik publikací o této technice, ale byla to až série 4 knih s doprovodnými videi[1], díky níž se začala tato technika v průběhu 90. let postupně více upevňovat, dostávat se do podvědomí ostatním vědcům, a to i díky každoročním konferencím, jež Koza v r. 1996 započal [21]. V r. 2010 představil Koza 76 výsledků ve kterých bylo GP konkurence schopné lidem [12] v mnoha různých oblastech.

2.1 Terminologie

Ke správnému pochopení následujících kapitol a podkapitol je potřeba zmínit a vysvětlit několik klíčových pojmů, které jsou pro GP zcela typické a jsou důležité pro porozumění evolučního vývoje programů.

- **Populace** – Jedná se o soubor kandidátních řešení v podobě programů, které se postupem generací vyvíjejí.
- **Jedinec** – Je objekt reálného světa představující potenciální řešení daného problému, jinak jej lze taktéž nazvat jako **kandidátní řešení** nebo **fenotyp**.
- **Chromozom** – Odkazuje na reprezentaci jedince v populaci, jinak jej lze taktéž nazvat jako **genotyp**.
- **Fitness funkce** – Je funkce hodnotící kvalitu kandidátních řešení ke vztahu k danému problému na základě určitých kritérií. Je základem mechanismu selekce a slouží taktéž jako ukončovací podmínka algoritmu. Jinak též nazývána jako **účelová funkce**.
- **Genetický operátor** – Algoritmus sloužící k tvorbě jedince nové populace z jednoho či dvou rodičů. Mezi takovéto operátory se například řadí mutace, rekombinace nebo permutace.

Mezi další méně důležitější lze zařadit **gen** – značí element v chromozomu, **alela** – hodnota genu a **lokus**, který značí pozici genu v chromozomu.

2.2 Charakteristika GP a vývoj programu

Při použití GP pro řešení daného problému je nutné určit několik klíčových částí. Těmi jsou reprezentace, fitness funkce, populace, selekční mechanismus rodičů a na závěr va-

[1] Odkaz na první knihu ze série „Genetic Programming: On the Programming of Computers by Means of Natural Selection“ včetně doprovodných videí `http://gpbib.cs.ucl.ac.uk/gp-html/koza_book.html`

riační operátory. Mimo ty je taktéž nutné definovat inicializační proceduru a ukončovací podmínku.

Nejprve je potřeba namapovat řešený problém z domény reálného světa do domény evolučních algoritmů, zde GP. Často tento krok zahrnuje zjednodušení a zabstraktnění daného problému. Následně se rozhodne, jakým způsobem budou řešení v rámci počítačového problému reprezentována. Tato potenciální kandidátní řešení jsou označována jako fenotypy, způsobem kterým jsou reprezentována a kódována v rámci GP, tak jsou označována jako genotypy. Mapování problému na fenotypy a následně na genotypy se nazývá reprezentace. Tu v této technice zastávají syntaktické stromy, jež jsou tvořeny z kořenového, vnitřních a vnějších uzlů. Dále se stanoví zbylé kroky o nichž bude hlouběji pojednáno v dalších podkapitolách.

Nejprve si je třeba stanovit množiny terminálních a neterminálních symbolů, které budou programy tvořit. Musí být zvoleny pečlivě tak, aby vyhovovaly zvolenému problému. Program poté vygeneruje počáteční populaci jedinců, jejíž velikost je dána inicializačním parametrem. Následně dojde k ohodnocení všech jedinců v populaci fitness funkcí, která určuje schopnost a kvalitu těchto jednotlivých kandidátních řešení. Selektivní funkcí se vybere jedinec či jedinci, kteří budou dále použití pro tvorbu potomstva. Pro tvorbu je používána různá množství genetických operátorů, jimiž můžou jsou typicky mutace, křížení, permutace nebo i řada pokročilejších operátorů. Tito potomci jsou poté přidáni do nové populace. Následně je tato populace ohodnocena. Tento cyklus se opakuje až do té doby, dokud není dosaženo alespoň jedné ukončovací podmínky. Jednu z nich vždy tvoří nalezení co nejlepšího řešení pro danou úlohu. Jednotlivé kroky GP jsou ilustrovány v pseudokódu Algoritmu 1.

Algorithm 1 Pseudokód genetického programování

```
vytvoř počáteční populaci p
ohodnoť fitness funkcí každého jedince v p
if některý jedinec dosáhl nejlepší hodnoty fitness then
    vrať tohoto jedince
end if
while není-li dosaženo max. počtu generací, nebo není splněna ukončovací podmínka do
    for proveď tolikrát, jaká je velikost populace do
        vyber dva rodiče
        if splněna pravděpodobnost křížení then
            proveď křížení a vrať potomka
        end if
        if splněna pravděpodobnost mutace then
            proveď mutaci a vrať potomka
        end if
        přidej potomka do nové populace p
    end for
    ohodnoť fitness funkcí každého jedince v p
    if některý jedinec dosáhl nejlepší hodnoty fitness then
        vrať tohoto jedince
    end if
end while
```

2.3 Fitness funkce

Fitness funkce, jinak též zvaná účelová funkce, je matematická funkce, pomocí níž jsou ohodnocována jednotlivá kandidátní řešení v populaci. Vstupem je kandidátní řešení a výstupem je numerická hodnota udávající jeho kvalitu, jinak též zdatnost. Je především důležitá při výběru vhodných jednotlivců pro evoluci. Ve spojitosti s přírodou lze říci, že jedinec s větší (lepší) fitness hodnotou má větší šanci na přežití a na reprodukci. Fitness funkce je pro každý problém zcela individuální a závisí na jednom či více kritériích. Musí se volit pečlivě tak, aby dokázala vhodně reprezentovat kvalitu daného jedince. Lze klasifikovat několik různých fitness funkcí.

- **Hrubá fitness** – Je vždy udávána v hodnotách přirozených v doméně řešeného problému.
- **Normalizovaná fitness** – Udává se jako podíl fitness hodnoty jedince vůči sumě fitness hodnot celé populace. Její rozsah leží na intervalu od 0 do 1 a její součet je vždy 1. Používá se například při proporcionálním výběru, viz kap. 2.5.
- **Přizpůsobená fitness** – Kalkuluje se ze standardizované $s(i)$ fitness následovně $a(i) = \frac{1}{1+s(i)}$, její výhoda oproti standardizované fitness spočívá v tom, že zvýrazňuje důležitost malých rozdílů v populaci v případě, že se standardizovaná fitness blíží hodnotě 0.
- **Standardizovaná fitness** – Přepočítává hrubou fitness tím způsobem, že menší hodnota znamená lepší řešení, tzn. že fitness 0 je nejlepší řešení.

2.4 Inicializační procedury

Inicializace populace probíhá od kořene stromu směrem k listovým uzlům. Před inicializací musí být nejprve stanoveny konečné množiny terminálních a neterminálních symbolů, které následně reprezentují jednotlivé vnitřní, resp. listové uzly. Existuje několik metod, které lze ke generování použít. Každá z nich může mít teoreticky vliv na rychlost a kvalitu nalezeného řešení. Mezi ty úplně nejzákladnější metody patří *full* a *grow*, dále metoda kombinující předchozí *ramped half-and-half*.

- **Grow** – Tato metoda inicializace vybírá náhodně uzly z množin terminálů a neterminálů a ukončí větev stromu, pokud je vybrán terminál, ačkoliv daný program nemusel dosáhnout maximální hloubky. Díky této metodě vznikají stromy s nepravidelnými tvary.
- **Full** – Dokud není dosaženo maximální hloubky, tak jsou uzly vybírány pouze z množiny neterminálů. Jakmile je dosaženo maximální hloubky, tak se větve zakončí pomocí terminálních symbolů. Stromy tímto dostávají pravidelné tvary.
- **Ramped half-and-half** – J. Koza navrhl novou metodu inicializace, která kombinuje metody *full* a *grow* pro dosažení větší rozmanitosti generovaných stromů v počáteční populaci, protože podle něj tyto metody samy o sobě tuto rozmanitost dostatečně neposkytují.

2.5 Selekční operátor

Selekce je proces výběru jedinců z populace, kteří budou následně využiti k tvorbě potomstva a nové generace. Používá se zároveň s variačními operátory. Hlavním účelem selekčního operátoru je upřednostňovat pro reprodukci zdatnější jedince. Následující výčet uvádí několik algoritmů, jež se k výběru aplikují [6].

- **Proporcionální výběr** – Tento technika vybírá jedince z populace na základě jejich ohodnocení. Čím vyšší hodnoty nabývá fitness funkce, tím větší je šance na výběr. Pravděpodobnost výběru jedince je vypočítána jako podíl fitness daného jedince a suma fitness celé populace. Součet všech ohodnocení se musí rovnat 1. Často zde dochází k předčasné konvergenci, kdy jsou obzvláště na počátku upřednostňováni vhodnější jedinci. Pokud je rozptyl fitness v populaci velmi malý, tak se zde nenachází skoro žádný selekční tlak a zejména v pozdějších generacích, když se v populaci nenachází už nejhorší řešení, tak průměrná fitness stoupá velmi pomalu. Existuje několik přístupů pro řešení výše zmíněných problémů, např. *windowing* nebo *sigma scaling* [6, str. 81]. Jeden ze způsobů jak se tenhle algoritmus implementuje se nazývá *vážená ruleta*.

- **Ranking** – Po ohodnocení je populace seřazena podle fitness od nejzdatnějšího jedince po nejhoršího. Každému jedinci je přiřazen rank k, což je hodnota vyjadřující relativní kvalitu jedince vůči ostatním jedincům v populaci. Čím vyšší ohodnocení, tím nižší je pravděpodobnost výběru. To znamená, že jedinci s lepšími hodnotami fitness funkce mají vyšší šanci, že budou vybráni.

- **Turnaj** – Je vybrán určitý vzorek populace v závislosti na zvolené hodnotě a z tohoto vzorku je pomocí turnaje zvolen ten nejvhodnější jedinec. Selekční tlak je nastavitelný velikostí vybraného vzorku jedinců. Velmi často používaná technika.

- **Elitismus** – Tento operátor je od těch výše zmíněných mírně odlišný a lze jej tedy používat v kombinaci. Elitismus v populaci zajišťuje, že na počátku každé generace je do nové populace vložen aktuálně nejschopnější jedinec. Tímto způsobem je možno zabezpečit, že se nejlepší fitness funkce nemůže náhle zhoršovat.

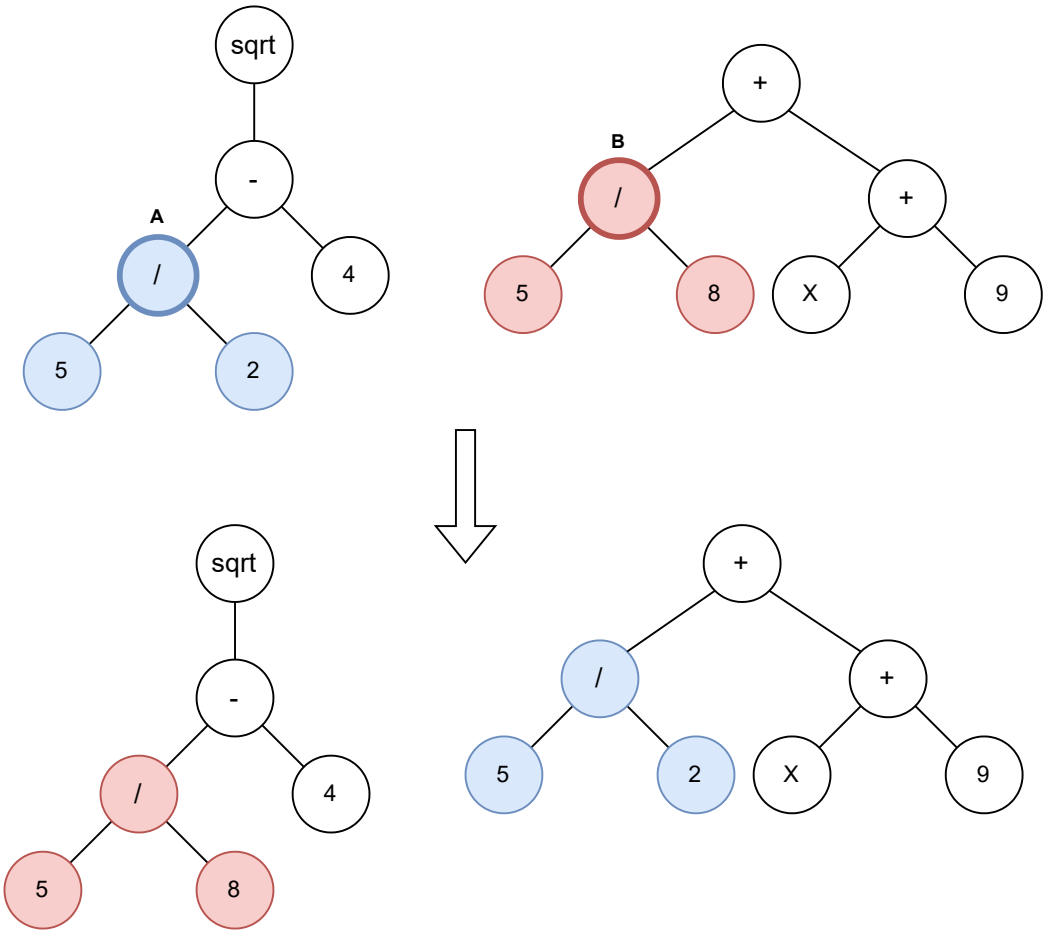
Selekční tlak určuje, do jaké míry jsou upřednostňováni jistí jedinci před jinými a vyjadřuje jak moc velký vliv má selekce na výběr populace. Nízký selekční tlak umožňuje prohledávat širší množinu jedinců ale současně se může prohledávání jevit spíše jako náhodné. Větší selekční tlak může pomoci k rychlejší konvergenci k požadovanému řešení, ale zároveň může způsobit ztrátu diverzity.

2.6 Genetické operátory

Variačními operátory je rozuměno technice, pomocí níž je zajišťována tvorba a diverzita nové populace. Jejich účelem je také zlepšit kvalitu aktuálních řešení. Řešení na nichž jsou operátory aplikovány, jsou vybráni selekční metodou. Jejich počet se liší v závislosti na operátoru. Mezi ty základní je klasifikována mutace a rekombinace (křížení). J. Koza definoval pro komplexnější problémy ještě další operátory permutace a inverze.

2.6.1 Křížení

Je n-ární operátor, ve kterém potomci dědí genetický materiál po rodičích. Nejprve se za pomocí selekce zvolí dva rodiče z populace, ti se následně překříží a vzniknou tak noví potomci, kteří jsou zařazení do nové populace. Používá se klasické jednobodové křížení podstromů, kdy je náhodně vybráno číslo, které vyjadřuje uzel v obou vybraných rodičích, kde se křížení provede. Tato čísla nemusí být stejná a může být vybráno pro každého rodiče rozdílné. V bodu křížení prvního podstromu je připojen podstrom druhého rodiče. Tímto vznikne jeden nový potomek. V závislosti na implementaci můžou vzniknout potomci dva, kdy je analogicky přesný postup proveden i u druhého rodiče. Na obr. 2.2 lze vidět dva programy, jež jsou překříženy v náhodně vybraných bodech A a B.

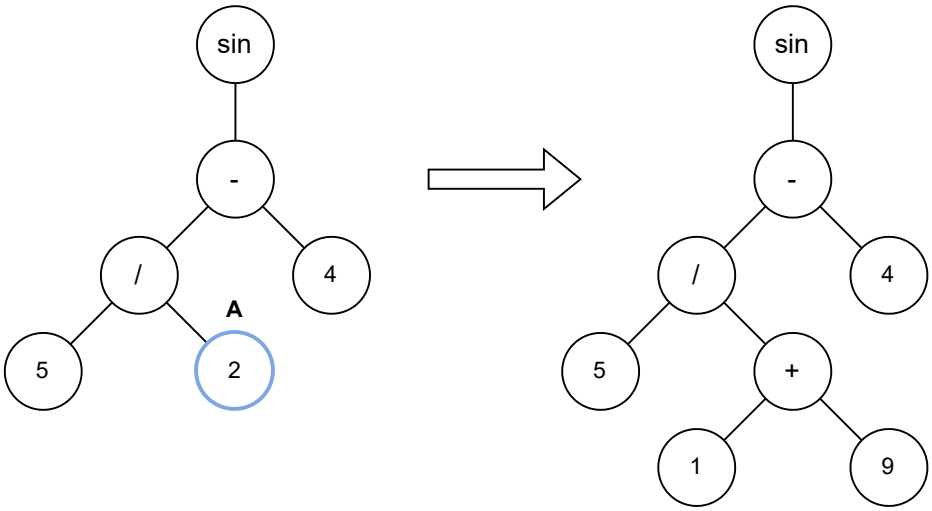


Obrázek 2.2: Genetický operátor křížení

2.6.2 Mutace

Tento unární operátor přináší ve vybraném rodiči náhodné změny a zajišťuje nový genetický materiál, čímž se často předejde konvergenci k lokálnímu optimu. Nejprve se vybere jeden rodič na němž bude operátor mutace aplikován, následně je náhodně vybrán stromový

uzel v němž k mutaci dojde. Na zvoleném místě je poté vygenerovaný nový podstrom o maximální určené výšce. Někdy to může být i pouze jeden uzel, pokud se bod mutace nachází v maximální hloubce. Občas může dojít k mutaci již na úrovni kořenového uzlu, čímž se většinou znehodnotí celý program. Mimo variantu generující podstromy existuje varianta, která prochází systematický celý strom a u každého uzlu dochází s jistou pravděpodobností k mutaci, jinak též zvaná bodová mutace. Na obr. 2.3 lze vidět variantu generující podstrom v vybraném bodě A.

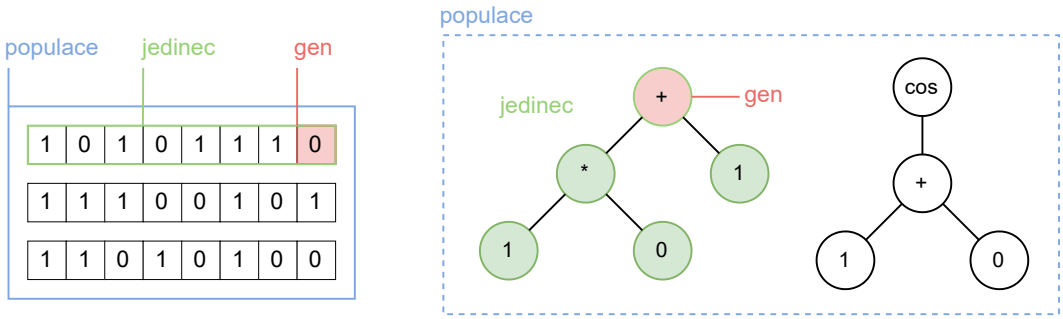


Obrázek 2.3: Genetický operátor mutace

2.7 Srovnání s dalšími evolučními algoritmy

Jak již bylo zmíněno na začátku kapitoly, genetické programování je silně inspirováno **genetickými algoritmy** (GA) a sdílí spolu veliké množství vlastností. Obě techniky využívají podobné genetické operátory, snaží se prohledávat prostor a zároveň neuváznout v lokálním minimu, negarantují nalezení globálního optimálního řešení, ale snaží se konvergovat k vhodnému řešení v rozumném čase. Jeden z rozdílů je možné zaznamenat v reprezentaci jednotlivých řešení, viz obr. 2.4, kde GA reprezentuje populaci jako řetězce čísel neměnné délky, zatímco GP je v jeho základní reprezentaci tvořeno syntaktickými stromy, které pokud nejsou uživatelem explicitně omezeny, tak můžou dosahovat nekonečných velikostí. Rozdíl je taktéž v aplikaci genetického operátoru křížení – v GA se genetický materiál často vyměňuje na stejných pozicích, zatímco v GP se vybrané podstromy mění na různých pozicích. Ačkoliv všechny tyto výše zmíněné rysy se dají modifikovat, ta důležitá rozdílnost spočívá v interpretaci řešení a tedy mapování popisu problému na objekt řešení. To v GA může být *1 to 1* nebo *many to 1*, u GP je to vždy *many to 1* [22].

Evoluční strategie (ES) využívá vektorovou reprezentaci, kde každý prvek vektoru představuje jeden parametr řešení. Mezi genetické operátory se řadí rekombinace a mutace (jakožto hlavní operátor [3]), kde nový jedinec vzniká přidáním náhodného čísla ke každé hodnotě rodičovského vektoru. Dokážou pracovat s reprezentací chromozomů ve vícerozměrném prostoru. Jeden z výrazných rysů ES se nazývá pokročilá samoadaptace parametrů,



Obrázek 2.4: Srovnání reprezentace populace, jedince a genu v genetickém algoritmu (vlevo) a genetickém programování (vpravo).

kdy se tyto parametry mění za běhu specifickým způsobem, například samoadaptace velikosti mutačního kroku. Nedávné formy ES se dokonce řadí mezi špičkové algoritmy sloužící k optimalizaci netriviálních funkcí reálných čísel [6].

Evoluční programování (EP) nemá striktní pravidla pro reprezentaci jedinců. Klasické EP reprezentovalo jedince v podobě konečných automatů, novodobé pomocí vektorů reálných čísel, takže se velmi přiblížilo ES a lze jej tedy někdy chápat jako speciální případ ES. Hlavní rozdíl, kde se tyto techniky odlišují, spočívá v biologické inspiraci, kde každý jedinec je považován za odlišný druh, proto zde neexistuje žádný rekombinační operátor a pouze jediným tu je tedy mutace [6]. Tím pádem každý jedinec populace generuje pouze jednoho potomka na rozdíl od GP, kde genetické operátory můžou generovat až dva potomky.

2.8 Nedostatky stromového GP

Ačkoliv je stromová technika GP tou nejznámější, tak skýtá i pár problémů. Protože velikost řešení zde není prakticky omezená, tzn. syntaktické stromy reprezentující jednotlivá řešení můžou nabývat jakékoliv výšky, omezení je většinou dáno jen u generace první populace, tak zde může docházet k tzv. *bloatu*, což znamená nadměrný růst řešení v průběhů generací, které je většinou způsobeno genetickými operátory. Neexistuje univerzální řešení a často závisí na implementaci. Řešení tohoto problému jsou navržena v praktické části této práce.

Vyhodnocení stromové reprezentace programu je často velmi náročné, obzvláště pokud jsou řešení komplexní nebo dochází k výše zmiňovanému nekontrolovatelnému růstu. To jak bude řešení komplexní závisí i na výběru neterminálních symbolů, které tvoří vnitřní uzly, a jejich aritě. Stromové uzly často nejsou uloženy v paměti kontinuálně. Toto může mít za následek pomalejší přístup do pamětí. Taktéž rekurze využívaná ve stromových strukturách je výpočetně náročná operace (zejména u jazyků jako je C nebo Python, které implementují rekurzi jako volání nové funkce včetně zásobníkových operací), jež si žádá alokaci většího množství paměti [2]. Proto lze v literatuře [11] často vidět GP implementováno pomocí jazyka LISP, který je považován za funkcionální jazyk, a prefixové notace.

Kapitola 3

Lineární genetické programování

Lineární genetické programování je jedna z dalších variant GP. V této kapitole je detailně popsána její charakteristika, reprezentace programů, postupy jakými se tyto programy inicializují, genetické operátory včetně grafické ilustrace a na závěr krátké porovnání se stromovou variantou GP. Zdrojem informací pro tuto kapitolu je literatura [4]. Tato technika reprezentuje jednotlivá kandidátní řešení jako sérii instrukcí, jež se vykonávají v sekvenčním pořadí jako v tradičním imperativním jazyce nebo jazyce symbolických instrukcí. Nejen že se tato varianta GP vyznačuje kompaktností generovaných řešeních, ale také umožněním velmi malých variačních změn v programech [4, str. 8]. Je využívána například v problémech, kde je zapotřebí vykonat určité instrukce ve specifickém pořadí, jako např. kontrolní systémy [16], kde se ve zmíněném problému tato metoda projevila jako nejúčinnější.

3.1 Reprezentace programu

Program se skládá ze sekvenčně se vykonávajících instrukcí. Každá instrukce může být složená z operace a operandů, které mohou být vyjádřeny registrem či konstantou. Na pravé straně se nachází zdrojové operandy a operace a na levé cílový registr pro uložení výsledku. V inicializační fázi je vždy stanovena množina možných konstant, operací a skupina registrů. Na obrázku 3.1 lze vidět příklad matematického programu zapsaného v jazyce C využívající jednoduché aritmetické operace, 7 vstupních registrů a jeden registr výstupní.

Programová řešení nejsou omezena pouze na jednoduché instrukce, ale mohou podporovat i složitější koncepty. Zatímco ve stromovém GP existují různě neterminální uzly s podmínkou, které zaručují různé chování programu na různé vstupní situace, tak obdobný koncept lze realizovat v lineárním GP větvením programů, které může být implementováno podmíněnou instrukcí např. `if (x < 5) then` a následně počtem instrukcí, které se po splnění této podmínky vykonají. Existují zde i vnořené podmínky, kdy je několik podmínek pod sebou spojeno logickými operátory konjunkce (`AND`) nebo disjunkce (`OR`). Druhá varianta je podpora podprogramů, kdy se opět po splnění podmínky odkazuje na místo v paměti označující daný podprogram po jehož ukončení se pokračuje ve vykonávání hlavního programu. Koncepce iteračních bloků kódu není v lineárním GP zas až tak důležitá a hlavně častá. Mimo jiné s sebou přináší řadů problémů, které je potřeba řešit. To zahrnuje možné nekonečné zacyklení nebo třeba vnořené cykly.

```
void gp(r)
  double r[8];
{  ...
   r[0] = r[5] + 71;
// r[7] = r[0] - 59;
   if (r[1] > 0)
   if (r[5] > 2)
     r[4] = r[2] * r[1];
// r[2] = r[5] + r[4];
   r[6] = r[4] * 13;
   r[1] = r[3] / 2;
// if (r[0] > r[1])
//   r[3] = r[5] * r[5];
   r[7] = r[6] - 2;
// r[5] = r[7] + 15;
   if (r[1] <= r[6])
     r[0] = sin(r[7]);
}
```

Obrázek 3.1: Reprezentace lineárního GP v notaci jazyka C. Převzato z lit. [4, str. 14].

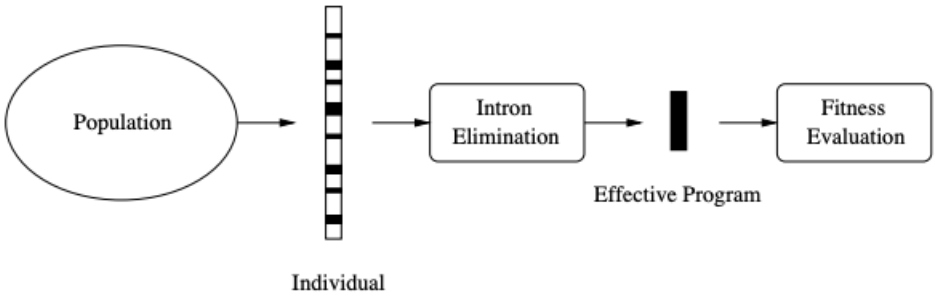
3.2 Charakteristika lineárního programu

Jeden z důvodů vzniku lineárního GP bylo překonat různé nevýhody spjaté s tradičním stromovým GP, které zahrnují nevhodnou reprezentaci genotypu nebo náročné vyhodnocování. Velmi charakteristické pro tuto sekvenční reprezentaci jsou například nevyužité části programového kódu. S lineárním GP se proto pojí následující pojmy.

Pokud se v programu nachází taková instrukce, která ovlivňuje výslednou hodnotu programu, nebo fitness hodnotu alespoň pro jeden možný vstup daného problému, tak je nazývána **efektivní instrukcí**. Naproti tomu **neefektivní instrukcí** nebo **intronem** jsou nazývány ty instrukce, které mají nulový vliv na výsledek pro celou množinu různých vstupů. Další možný výklad tohoto termínu by mohl být, že nemá žádný vliv na konečnou hodnotu funkce fitness. Neefektivní instrukce může být z programu odstraněna, viz obr. 3.2, nebo zachována. Na obr. 3.1 je neefektivní instrukce v rámci programu značená symboly dvou lomítek `//`, čímž tedy není při interpretaci vykonána, ale je nadále jako genetický materiál zachována.

Strukturální intron označuje v programu takovou neefektivní instrukci, jež vznikla z manipulace s neefektivními registry, které nemají vliv na konečný výsledek programu. Ve stromovém GP k strukturálním intronům nedochází. Algoritmem který program zpětně analyzuje, tedy každou instrukci a její cílový registr a závislost tohoto registru na předchozí instrukci, lze eliminovat neefektivní kód, který zefektivní následné vyhodnocování fitness funkce. Velký vliv na vznik těchto neefektivních úseků kódu mají třeba parametry délky programu nebo počtu registrů.

Sémantickým intronem je nazývána ta neefektivní instrukce nebo skupina neefektivních instrukcí, jež jsou strukturálně správně, tedy manipulují s registry, které mají vliv na výsledek programu, ale zjednodušeně řečeno, hodnota efektivních registrů se po provedení instrukce nijak nezmění. Na vznik těchto intronů má z největší části vliv vybraná množina instrukcí a konstant. Nelze se jim v lineárním GP bohužel úplně vyhnout, ale je dobré si stanovit a formulovat pravidla k vyvarování se alespoň těm nejzákladnějším intronům a zároveň zanechání dostatečného prostoru evoluci.



Obrázek 3.2: Obecný proces odstranění neefektivních instrukcí (vyznačeny bílou barvou). Převzato z lit. [4, str. 39].

$$(a) \quad r_0 := r_i/0 \qquad (b) \quad r_1 := r_0 - r_0 \qquad r_0 := r_i/r_1$$

$$(c) \quad if\,(r_i > r_i) \qquad r_0 := r_j + c$$

Příklady sémantických intronů – dělení nulou (a), (b) a nenaplnitelná podmínka (c), tudíž nikdy nedojde k vykonání instrukce, jež je závislá právě na této podmínce.

Detekce a odstranění není sémantických intronů již tak zcela snadné jak tomu bylo u strukturálních. Programy dosahují dostatečného zrychlení již odstraněním strukturálních intronů a je tedy otázkou, zdali se vůbec vyplatí algoritmus pro odstranění těch sémantických intronů implementovat. Možnou motivací by mohla být snadnější interpretace a porozumění jednotlivým programovým řešením [4, str. 44].

3.3 Inicializace populace

U varianty lineárního GP existují několik klíčových faktorů, které ovlivňují to, jak bude generovaná populace vypadat. Těmito faktory, parametry, jsou zde délka programu a počet registrů. U délky programu je třeba stanovit dolní a horní hranici. Oba tyto parametry musí být zvoleny s rozvahou a úměrou vzhledem k velikosti problému. Je neefektivní například generovat dlouhé programy o stovkách řádcích k vyřešení jednoduchého matematického problému. Nejen že by opakované vyhodnocování tohoto programu zabralo velké množství času, ale zřejmé by nikdy nedošlo k nalezení nejlepšího řešení. Zde je uvedeno několik inicializačních technik.

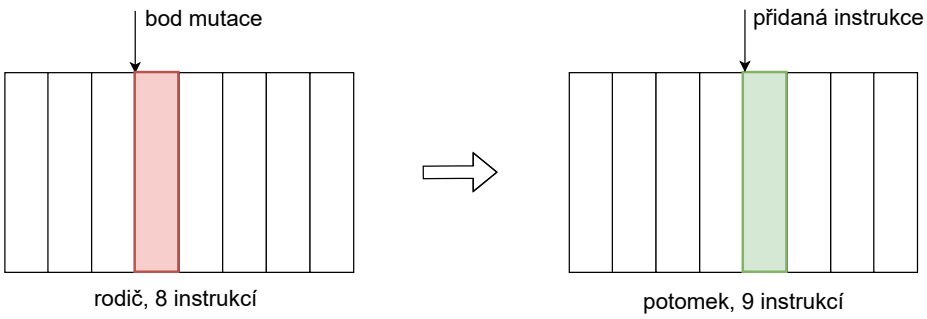
- **Volná inicializace** – Zahrnuje standardní inicializaci bez omezení.
- **Plně efektivní inicializace** – Program nebude obsahovat neefektivní instrukce, tzv. *introny*.
- **Maximální inicializace** – Inicializace programů v jejich maximální možné délce.
- **Konstantní inicializace** – Inicializace všech programů v délce určené konstantou.
- **Proměnlivá inicializace** – Programy budou mít proměnlivou délku na základě definovaného rozsahu.

3.4 Genetické operátory

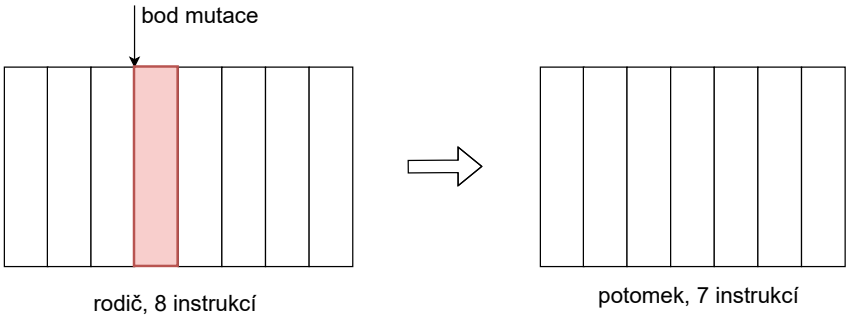
3.4.1 Mutace

Mutační operátor je klasifikován do dvou kategorií – mikro mutace a makro mutace. Makro mutace manipuluje s instrukcí s jakožto atomickým prvkem, takže operací s ní spojené se dějí na úrovní celých instrukcí. Kdežto mikro mutace se zaměřuje na změnu prvků uvnitř instrukce. Ta může zahrnovat změnu konstant, registrů či operací. U makro mutací se rozlišují následující tři operace (definované zkratky – $l(gp)$ vyjadřuje délku programu; $l_{min},\ l_{max}$ minimální a maximální délku programu):

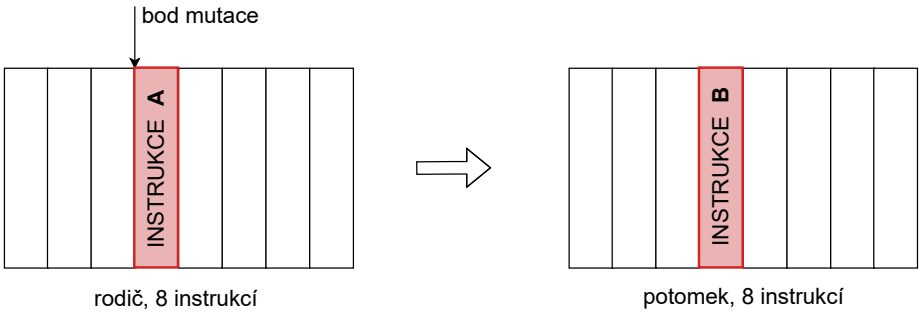
- **Aditivní mutace** – Před vybranou instrukci je vložena nová instrukce za podmínky, že je splněno $l(gp) < l_{max}$, viz obr. 3.3.
- **Destruktivní mutace** – Vybraná instrukce je odstraněna za podmínky, že $l(gp) > l_{min}$, viz obr. 3.4. Je doporučeno vzájemně aplikovat s aditivní mutací k předcházení nadměrné redukci programu.
- **Měnící mutace** – Vybraná instrukce je nahrazena za jinou, viz obr. 3.5, kde dochází k výměně instrukce A za instrukci B.



Obrázek 3.3: Aditivní mutace



Obrázek 3.4: Destruktivní mutace

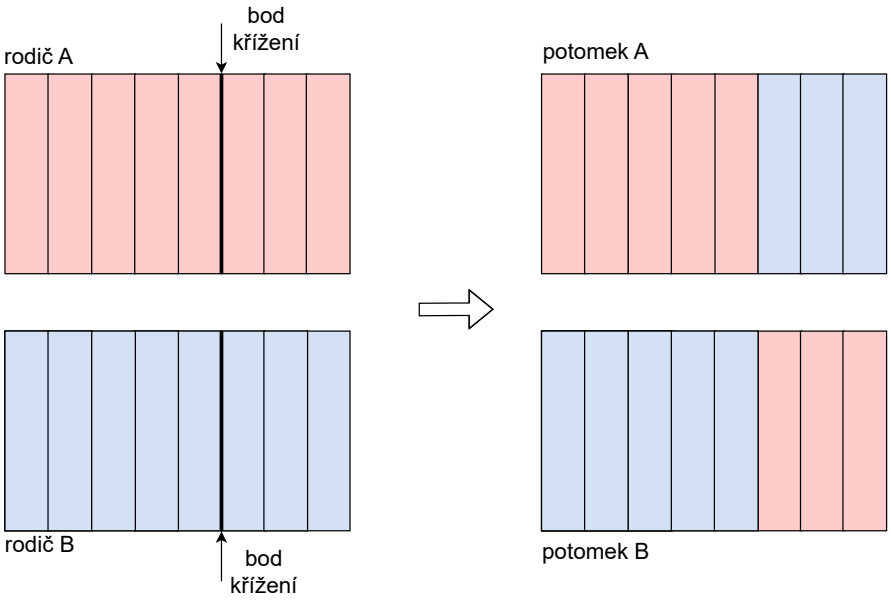


Obrázek 3.5: Měnící mutace

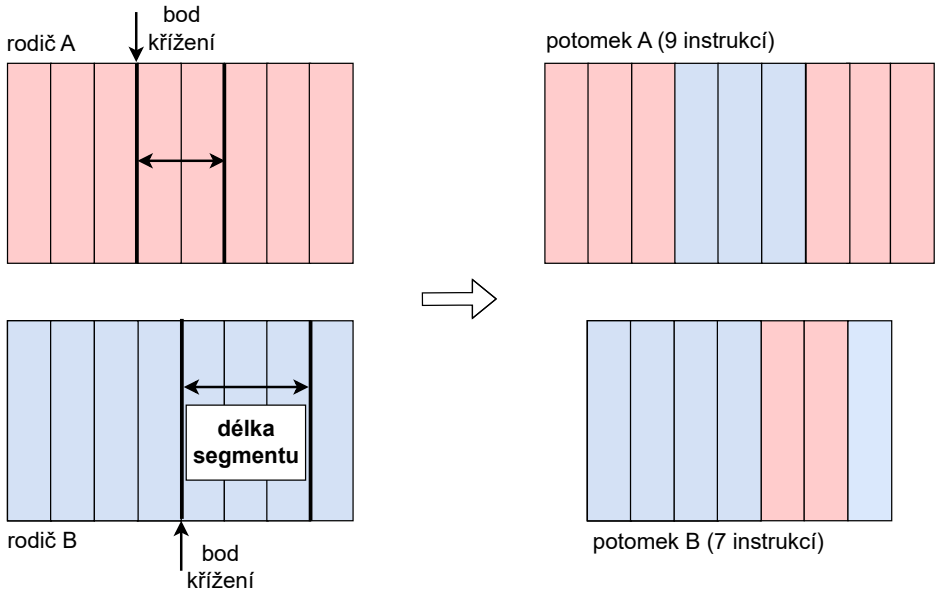
3.4.2 Křížení

Křížení je segmentový variační operátor, který vytváří v závislosti na implementaci jednoho či dva nové potomky z dvou vybraných rodičů. Tato metoda spočívá v tom, že chromozomy rodičů se rozdělí na dva či více segmenty a ty se pak následně mezi sebou vymění. Existují dva základní typy tohoto operátoru.

- **Jednobodové křížení** – V obou rodičích je náhodně zvolen jeden bod, ve kterém následně nastane křížení, kdy se vymění poslední segmenty obou rodičů, viz obr. 3.6. Pokud je jeden z potomků delší než je maximální délka programu, tak proběhne křížení znovu, ale se zvolením stejné pozice křížení v obou rodičích. Často zde dochází k větší absolutní velikosti kroku v porovnání s vícebodovým křížením.
- **Vícebodové křížení** – Jsou zvoleny dva body křížení v obou rodičích, z nichž druhý je dán délkou segmentu od prvního bodu. Prostřední segmenty obou rodičů, jež se mezi těmito dvěma body nachází, se poté vzájemně vymění, viz obr. 3.7.



Obrázek 3.6: Jednobodové křížení



Obrázek 3.7: Dvoubodové křížení

3.5 Srovnání se stromovým GP

Stromové a lineární genetické programování jsou techniky GP, které sdílejí podobné koncepty, ale i kvůli na první pohled odlišné programové reprezentací implementují spoustu věcí odlišně a vykazují i jiné chování. Stromové GP implementuje datový tok omezený stromovou strukturou, zatímco lineární GP využívá mnohem flexibilnějšího datového toku v registrech. Komplexita a náročnost vyhodnocování těchto reprezentací je značně odlišná. U stromového GP je potřeba rekurzivně procházet stromovou strukturu, zatímco u lineárního GP jsou sekvenčně zpracovávány instrukce jako u strojového kódu. Bylo tomu dokázáno na kolekcí dvou benchmarků, jež navzájem tyto dvě techniky porovnávaly [4, kap. 8]. Taktéž se na základě něj prokázalo, že lineární GP je schopno vytvářet menší a efektivnější programy. Nejen že těží z výhody právě své reprezentace, která neskýtá mnohá omezení, ale taktéž znovu využívá obsahy registrů a odstraňuje neúčinný kód, zatímco stromové GP vykoná program bez ohledu na efektivitu. Právě i proto je lineární GP flexibilnější strukturovaným změnám oproti syntaktickému stromu ve stromovém GP, kde je velice obtížné přidávat anebo odstraňovat vnitřní uzly. Vložením či odstraněním instrukce u lineárního GP nejsou ztraceny případné podstruktury programu jak by tomu bylo u stromového GP. Obě techniky implementují odlišným způsobem i genetické operátory, podrobněji viz předchozí kapitoly. Zatímco u stromového dochází ke křížení stromových podstruktur v náhodně vybraném bodu, tak u lineárního je možné aplikovat i vícebodové kříženi s variabilní délkou segmentu. Mutační operátor u stromového GP běžně generuje v bodu mutace náhodný podstrom, zatímco u lineárního je rozlišováno mezi mikro a makro mutacemi.

Kapitola 4

Aplikace a vývoj genetického programování

Genetické programování našlo již mnohé uplatnění v řešení různých problémů. V této kapitole jsou krátce zmíněné aplikace GP, jak na ty nejzákladnější úlohy na kterých je často tato technika představována, tak i složitější. Použití GP má zcela mnoho výhod – dokáže automaticky generovat programy, které mohou řešit komplexní problémy, které jsou jen stěží řešitelné pomocí konvenčních metod, ať již programování nebo jiných metod strojového učení. Je praktické při problémech, kde není jasné, jakým postupem se dostat k požadovanému výsledku, dále taktéž pokud je nutné před výpočtem předzpracovat a klasifikovat velký objem vstupních dat [12]. Mezi další vlastnosti a benefity použití této techniky patří:

- podpora veliké škály programových struktur, znovupoužití kódu,
- optimalizace a paralelní zpracování,
- redukce doby vývoje,
- prohledávaní širokého prostoru možných řešení,
- nezávislost na řešeném problému,
- přínos konkurenceschopných výsledků.

Obzvláště poslední bod připomíná důležitost toho, proč je GP tak stále využívané. Aby produkovalo užitečné výsledky, namísto pouze řešení problémů, které se ve vědeckých disciplínách nazývají *toy problem*, které tedy nejsou pro vědu nikterak užitečné a slouží spíše pro demonstrační ukázky nějaké obecné techniky [1]. Pokud není jinak konkrétněji uvedeno, vychází tato kapitola z literatury [11, 12].

4.1 Aplikace genetického programovaní

- **Symbolická regrese** – Jeden z nejznámějších problémů na kterém je GP prvotně demonstrováno, je právě symbolická regrese. Cílem programu je nalézt takovou matematickou rovnici, která splňuje podmínky pro zadané vstupní a výstupní hodnoty a tedy najít souvislost mezi těmito daty. Na počátku je vygenerována populace programů, kde jsou funkční symboly reprezentovány matematickými funkcemi (např. +, -, *, /, sin, cos, ...) a terminály proměnnými či konstantami. Fitness funkce následně měří přesnost každého kandidátního řešení na daném datasetu. Ta je měřena na základě odchylky mezi očekávanými a aktuálními výsledky. Tento problém může být

aplikován na širokou řadu aplikací včetně funkčních aproximací, identifikaci systému nebo vhodnosti nějakého modelu. Je aplikován v různých oborech, např. v inženýrství, biologii či financích.

- **Analogové obvody** – Zatímco jednoduché obvody lze řešit pomocí tradičních výpočetních metod, u složitějších problémů, jako je třeba návrh komplexní ALU jednotky, se tento proces stává časově náročným. Jednotlivé programy jsou stromově reprezentovány obsahující logické členy (AND, OR, NOT), sčítačky (celá, poloviční, ...) a klopné obvody (JK, RS, D, ...). Terminálními symboly jsou reprezentovány proměnné uvnitř obvodu. Kalkulace fitness hodnoty zde probíhá na základě pravdivostní tabulky, která nastiňuje vztahy mezi vstupy a výstupy obvodu. Pokud je dosáhnuto požadované hodnoty fitness, tak je provedena validace elektrického obvodu, zdali je plně vyhovující [5].
- **Cart centering problem** – V tomto problému se snaží GP nalézt takovou řídicí strategii, která vyvine dostatečnou sílu k posunutí vozíku na přímce na předem určené místo za co nejlepší čas. Tento problém zahrnuje systém, jehož stav je popsán stavovými proměnnými. To znamená aktuální pozici $x(t)$ a rychlost $v(t)$. Nachází se zde taktéž jedna kontrolní proměnná tzv. *big-bang* síla, která dokáže posunout vozíkem jak v pozitivním tak negativním směru. Její hodnotu se snaží program určit tak, aby se systém dostal do cílového stavu za co nejlepších podmínek. Ty můžou být měřeny například v ceně nebo času [11].

Genetické programování se už běžně používá jako metoda řešení různých problémů v mnoha oblastech včetně hydrologie, hydrauliky, prediktivního modelování, elektronice, klasifikace obrázků, diagnostiky nemocí a mnoha dalších. V následujících podkapitolách je podrobněji popsáno pár z nich, např. jak tato metoda dokáže opravovat a vylepšovat software a jak může být užitečná v inženýrství při návrhu stavebních základů.

Geneticky vylepšený software

Genetické programování se již osvědčilo v mnoha případech jako vhodný nástroj k úpravě či generaci programového kódu [9]. První aplikace a případ užití bylo pro hashovaní, caching a garbage collector, kde i velká moderní paměť a její správa může znamenat značnou režii. Další příklady využití jsou například internetové mashupy, což v překladu znamená nějakou internetovou stránku, která v rámci poskytování služeb kombinuje data či funkcionalitu z více zbrojů, nebo také pro generaci kódu. Kompilátory by totiž neměly pouze vytvářet korektní kód, ale zároveň ideálně krátký a dostatečně efektivní.

GP se taktéž osvědčilo v automatické opravě programového kódu. Ačkoliv se podle několika teoretických analýz a studií stále jedná o ne velmi rozšířenou metodu. Na mnoha příkladech se snaží GP využít již napsaný kód, který se snaží přeskupit. A zde to platí ještě víc, že jeden řádek kódu může rozbít celý program, ale taktéž můžou v rámci mutace nebo křížení vzniknout kvalitní programy. Tato technika dokázala vyřešit problém nalezený 31. prosince 2008, kdy uživatelé zaznamenali rozsáhlý bug v přehrávačích Microsoft Zune, který zapříčinil jejich zamrznutí. Tento příklad vychází z článku [7].

Aplikace GP v inženýrských systémech

V posledních letech bylo dokázáno, že GP je úspěšné i v oblasti návrhu geotechnických systémů [9]. Materiály které se s tímto inženýrstvím často pojí, např. zemina anebo kamení,

vykazují ze své podstaty nejisté chování spojené často s fyzikálními procesy spojené s tvorbou těchto materiálů. Modelování chování těchto materiálů je často za hranicemi tradičních metod. Proto se nabízela možnost využití umělé inteligence v této oblasti. Tyto techniky se používají ve strojírenství k řešení problémů učením se z příkladů datových vstupů a výstupů. To pomáhá zachytit funkční vztahy mezi daty, i když základní vztahy jsou neznámé nebo obtížně vysvětlitelné. Podobný koncept je de facto vysvětlen na příkladu symbolické regrese v podkapitole 4.1. Z technik umělé inteligence se v této oblasti velmi dobře uchytila metoda umělých neuronových sítí, ačkoliv se v poslední době velmi dobře osvědčila právě i metoda GP. Přestože neuronové sítě byly úspěšné v analýze a simulaci mnoha aplikací geotechnického inženýrství, tak jsou kritizovány za to, že jsou „černými skříňkami“ kvůli jejich nedostatečné transparentnosti a extrakci znalostí. Nejnovější metodou GP, která je pro toto inženýrství používána, se nazývá evoluční polynomiální regrese (EPR). Je to metoda, která využívá polynomiální funkce a genetické operátory k vývoji matematických výrazů pro regresní analýzu. Z pohledu transparentnosti jsou klasifikovány jako „šedé skřínky“, u nichž jsou známy matematické výrazy, které nejsou těžké na interpretaci. Takže tedy o třídu lepší než neuronové sítě. Ideální by byly samozřejmě „bílé skříňky“ u nichž jsou používány prvotní principy, tedy fyzikální zákony a fungování systému tedy může být jasně vysvětleno.

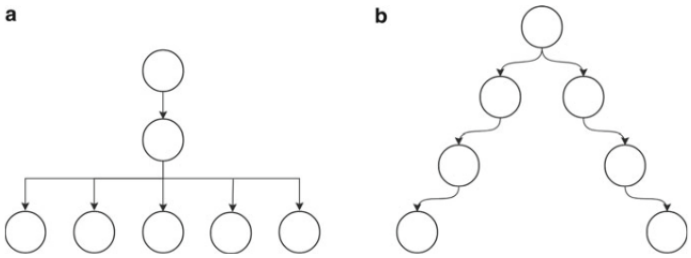
Jedna z konkrétních aplikací v tomto oboru byla provedena při návrhu mělkých základů na nesoudržné půdě. Usazení takovýchto základů je stále velké téma, které je komplexní a stále ještě ne zcela pochopené. Tento fakt podnítil řadu vědců k použití GP, konkrétně výše zmíněné EPR techniky, kdy na základě datasetu dokázali model, jenž dosahoval vynikajících výsledků. Na základě pěti vstupních proměnných, jako je například výška a šířka základu nebo hloubka ukotvení základů, generoval výsledné usazení základů. Výsledky tedy prokázaly, že tento model dosahoval nejen eminentních výsledků, ale byl taktéž úspěšnější než tradiční metody a dokonce než umělé neuronové sítě.

4.2 Aktuální vývoj v oblasti GP

Výzkum v oblasti GP se nezaměřuje pouze na zkoumání nových aplikací, ale taktéž na poznávání vlastností GP jako samotného, jež ještě nejsou známé. Jedna z nich je populační diverzita. Diverzita v populaci je jedna z klíčových vlastností potřebná k tvorbě nové populace, jež pomáhá efektivněji nalézt optimálnější řešení. Se ztrátou diverzity v populaci je riskována rychlá konvergence k sub-optimálnímu řešení. Zápis z poslední konference o GP z r. 2022 [10] se snaží o tomto problému diskutovat a dokázat, že jisté metriky dokážou předpovědět důležité informace a pomoci k dosáhnutí optimálního řešení.

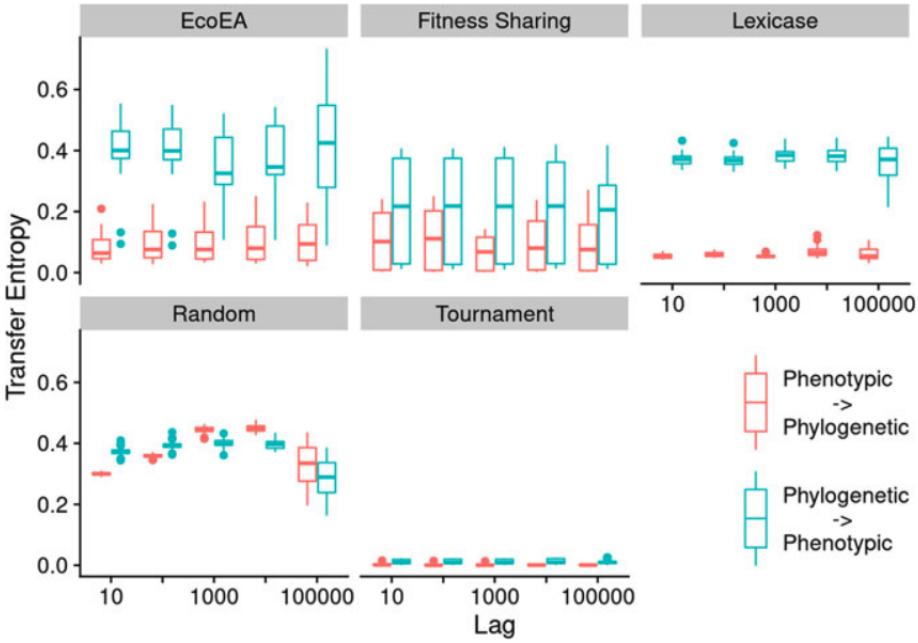
Existuje celá řada technik, která se o tento problém zhroucení populace zajímá, ale stále není dostatečně porozuměno, co ji způsobuje, či k ní přispívá. Diverzita je standardně v populaci vymezena počtem unikátnosti jednotlivých řešení dle stanovených metrik. Ale tyto metriky jsou jen malým vzorkem, které jsou dnes využívány. Jedna z nich, která ještě nebyla dostatečně prozkoumána se nazývá fylogenetická, jež určuje diverzitu na základě předků v populaci. Na obr. 4.1 lze vidět dvě rozdílné diverzity. Šipkami jsou zde znázorněny vztahy mezi rodičem a potomky. Listové uzly zde představují aktuální populaci, kde každý jedinec je unikátní. Zatímco na obrázku (a) je zřejmé, že fenotypová rozmanitost je velká

(5), tak fylogenetická malá (*mean pairwise distance*[1] = 2), u obrázku (b) je tomu zcela obráceně.



Obrázek 4.1: Zobrazení dvou různých typů diverzit. Převzato z lit. [10, str. 65].

Předběžná data podporují dvě hypotézy, že fylogenetická diverzita zachytává data, jež jinší diverzity nezachytávají a zároveň, že může být lepším prediktorem pro úspěch ostatních metrik. Tato studie se snaží za pomocí vybraných selekčních metod a problémů tyto hypotézy ověřit.



Obrázek 4.2: Graf zobrazující míru přenosové entropie ve dvou směrech na různých selekčních metodách a zpoždění. Převzato z lit. [10, str. 77].

Na počátku bylo porovnáno několik metrik od obou diverzit. Protože se vyskytla vysoká korelace mezi vybranými metrikami u fenotypové diverzity, tak byla pro další účely zvolena pouze jedna. U fylogenetické diverzity byla vybrána střední vzdálenost mezi všemi jedinci aneb *mean pairwise distance*. Ačkoliv při následném porovnávaní výše dvou vybraných metrik byla očekávána vysoká korelace, vzešlo najevo, že je tomu opak a došlo se k závěru, že se v průběhu času chovají velmi odlišeně.

[1]Mean pairwise distance vyjadřuje vzdálenost vyjádřenou v počtu hran (v grafu) v nejkratší cestě mezi každou dvojici jedinců v aktuální populaci.

Schopnost předpovídat jedna diverzita druhou byla otestována za pomocí změření přenosové entropie z fylogenetické diverzity na fenotypovou. Jak lze vidět na obr. 4.2, tak značně vyšší hodnoty byly nalezeny u přenosu z fylogenetické na fenotypovou, což tedy znamená, že fylogenetická diverzita může poskytovat i jiné informace, které ostatní diverzity nemůžou. Menší rozdíly lze pozorovat u turnajové a náhodné metody, kde se obě diverzity predikují navzájem. Dále byla provedena další analýza za účelem zjištění, zdali výsledky získané na obr. 4.2 jsou nějak užitečné, ze kterých následně vyplynulo, že je fylogenetická diverzita skutečně prediktivnější.

Na závěr lze tedy konstatovat, že fylogenetická diverzita obecně více předpovídá úspěch než fenotypová. Stále zde ale záleží na řešeném problému a aplikované selekční metodě.

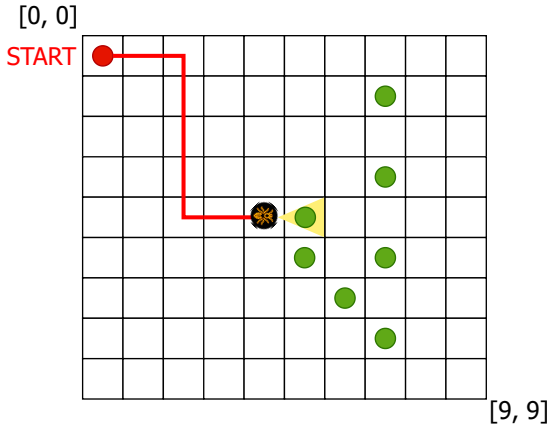
Kapitola 5

Úloha Langtonova mravence

Cílem této kapitoly je popsat úlohu Langtonova mravence, na jejíž instancích jsou aplikovány obě dvě varianty GP. Hlavním zdrojem k této kapitole je literatura [11]. Tato úloha byla poprvé představena Jeffersonem a kol. s využitím genetických algoritmů. Jejich cílem bylo studium několika základních aspektů genetických algoritmů, kde využili reprezentaci konečných automatů. Později byl tento problém zpopularizován J. Kozou, jehož přístup k řešení navrhoval aplikaci GP.

5.1 Princip úlohy

Tato úloha zahrnuje nalezení kontrolní strategie umělého mravence pohybujícího se ve dvojdimenzionálním poli buněk, jež může obsahovat potravu. Cílem je sesbírat v co nejlepším čase, tím je zde rozuměno v co nejmenším počtu kroků mravence, všechnu potravu za pomocí sady pravidel sloužících k navigaci a pohybu mravence. Tato pravidla jsou zakódována v genu chromozomu jednotlivých kandidátních řešeních.



Obrázek 5.1: Langtonův mravenec

Na obr. 5.1 lze vidět mřížku o velikosti 10x10, kde zelená kolečka reprezentují buňky s potravou, které musí mravenec sesbírat a červená čára znázorňuje jeho dosavadní cestu včetně startovní pozice. Mravenec obvykle začíná na pozici [0, 0], tzn. v levém horním rohu. Mravenec má v základu velmi omezené schopnosti pohybu, vidí pouze o jedno políčko vpřed (na obr. znázorněno žlutým trojúhelníkem) a nemá žádné informace o své současné pozici.

Literatura aktuálně přistupuje k tomuto problému pomocí stromového GP a definuje následující sadu pravidel, pomocí níž se může mravenec pohybovat. Protože se jedná o stromové GP, tak se pravidla dělí na terminální, tedy ty co jsou umístěny v listech syntaktických stromů a definují pohyb mravence, tak neterminální, tedy funkce pomocí kterých se rozhoduje, jakým způsobem se budou pravidla vykonávat. V aktuálním řešení dle literatury je definována tato sada pravidel.

`LEFT` – Mravenec se na místě otočí o 90° v protisměru hodinových ručiček.

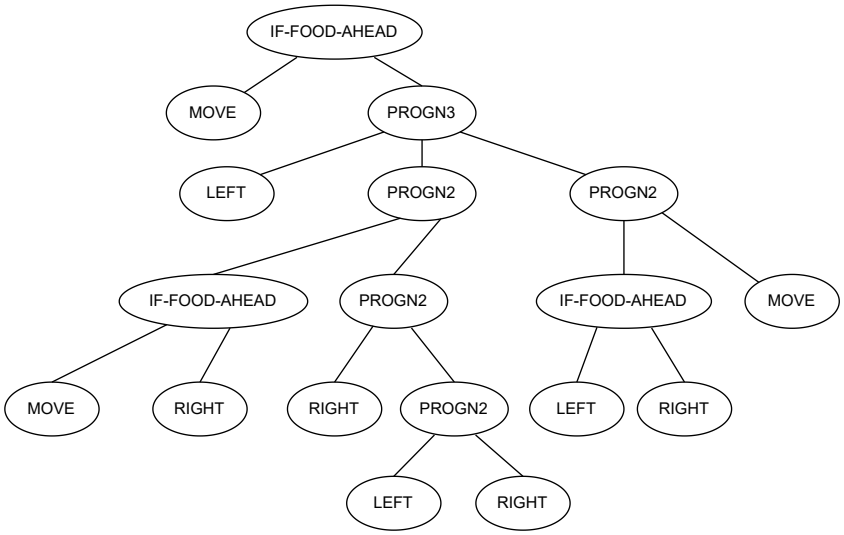
`RIGHT` – Mravenec se na místě otočí o 90° ve směru hodinových ručiček.

`MOVE` – Mravenec provede pohyb o jedno políčko vpřed.

`PROGN2` – Binární funkční symbol pro jehož splnění musí být vykonán pravý i levý podstrom.

`PROGN3` – Ternární funkční symbol pro jehož splnění musí být vykonán levý, střední a pravý podstrom.

`IF-FOOD-AHEAD` – Podmíněný binární funkční symbol, pokud je podmínka splněna, tak je vykonán levý podstrom, v opačném případě podstrom pravý.

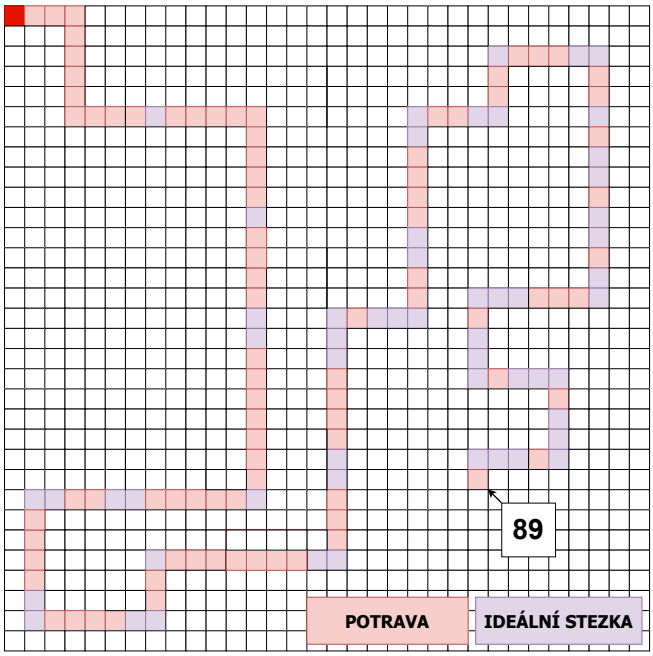


Obrázek 5.2: Reprezentace kandidátního řešení úlohy Langtonova mravence. Převzato z lit. [11, str. 154] a graficky upraveno.

Na obr. 5.2 je vyzobrazeno jedno z nekonečno možných řešení. Při interpretaci tohoto řešení se nejprve mravenec podívá, jestli před sebou vidí potravu, protože kořenový uzel obsahuje funkční symbol `IF-FOOD-AHEAD`. Pokud potravu vidí, tak se vydá levým podstromem k instrukci `MOVE`. V opačném případě dojde k vykonání pravého podstromu a funkčního symbolu `PROGN3`, pro jehož splnění musí být vykonány všechny ostatní podstromy, takže se nejdříve vykoná levý, prostřední a nakonec pravý. Takto analogicky se pokračuje, dokud se nevykoná celý strom.

Úloha je často prezentována na stezkách[1], jež byly navrhnuty jak samotným autorem, tak i osobami, které se o tento problém zajímaly. Původně byla pro tento problém využívána stezka Johna Muira.

- Stezka **Santa Fe** – Často používána v odkazované literatuře, navržena Christopherem Langtonem, viz obr. 5.3. Je složitější než původní stezka J. Muira.
- Stezka **J. Muira** – Původně využívána stezka pro tento problém, obsahuje 89 buněk s potravou, 20 zatáček, 4 mezery, 7 dvojtých mezer a 7 trojtých mezer.
- Stezka **Los Altos Hills** – Složitější stezka o rozloze 100x100 buněk. Pro řešení je ideální přidat nový funkční symbol `PROGN4`, což je kvartální funkční symbol fungující analogicky jako `PROGN2/3`.



Obrázek 5.3: Stezka Santa Fe (výrazněji červené políčko znázorňuje startovní pozici, číslo 89 v pravém dolním rohu znázorňuje poslední buňku s potravou. Převzato z lit. [11, str. 55] a graficky upraveno.

5.2 Vlastní přístup k řešení

K problému přistupuji velmi obdobně, tak jak je tomu popsáno v literatuře, která mi byla předlohou. S odlišnostmi se stavím ke genetickým operátorům či inicializačním parametrům mravence včetně počáteční polohy, která se nemusí vždy nacházet v levém horním rohu. Rozložení potravy po mřížce je generováno stochasticky, nebo již na předem nadefinovaných stezkách. Ačkoliv je lineární GP v základu velice tomu stromovému podobné, tak je zde jiná reprezentace programů. Protože mi implementace této úlohy v této technice nebyla dříve známa, tak jsem se inspiroval stromovým GP a patřičně jsem jej pro tuto techniku upravil.

[1]Stezkou je zde myšlena dvojdimenzionální mřížka obsahující políčka s potravou.

Například jsem musel opustit od funkcí typu `PROGN2/3`, které jsou zcela typické pouze pro stromové GP a v lineárním GP nejsou využitelné. Podrobnější implementace je popsána v následující kapitole.

Kapitola 6

Implementace

V této kapitole je představen a popsán způsob implementace úlohy Langtonova mravence v obou variantách GP. Nemalou část implementace tyto varianty navzájem sdílí. Odlišují se hlavně v návrhu, jak budou jednotlivá programová řešení realizována, následně interpretována a taktéž jak s nimi bude zacházeno například při aplikaci genetických operátorů. Ke konci této kapitole je přestaven způsob, kterým je úloha vizualizována uživateli.

Pro implementaci byl zvolen jazyk Python3[1] a to z několika důvodů. Poskytuje veliké množství knihoven, které byly k vývoji použity. Je to multiplatformní jazyk [17], takže je velice snadné tento program spouštět v různých operačních systémech a distribucích bez větších potíží. Mezi primárně využívané knihovny patří matplotlib[2], realizující grafovou vizualizaci, numpy[3] pro výpočty nejen u statistického vyhodnocovaní a imageio[4] pro výstupní vizualizaci úlohy ve formátu *gif*.

Nejprve jsou shrnuty implementační detaily, jež obě varianty sdílí. Mřížka s potravou a prázdnými políčky je reprezentována pomocí dvojrozměrné matice obsahující tři různé hodnoty. Nula (0) představuje prázdná políčka, jednička (1) políčka s potravou a dvojka (2) dosud nalezenou cestu mravencem. Při každém vyhodnocování fitness je tedy počítán počet políček s hodnotou 1, v nejideálnějším případě je požadováno, aby se jejich počet rovnal nule, v tom případě je nalezeno nejlepší řešení. Proto také byla zvolena standardizovaná fitness, která tomuto problému nejlépe vyhovuje. Pro statistické účely jsou v průběhu běhu sbírány hodnoty nejhorší, průměrné a nejlepší této fitness funkce. Obě dvě techniky sdílí stejné implementace selekčních operátorů – turnaj, ruletu a ranking.

V čem se tyto dvě implementace liší, a to zejména v množině symbolů, interpretaci programu a způsob implementace variačních operátorů, je popsáno v následujících podkapitolách.

6.1 Stromové GP

Varianta tohoto GP je postavena na základech již existujícího algoritmu sloužícího k výpočtu symbolické regrese [20], jež byla následně upravena, aby vyhovovala problému Langtonova mravence. V této variantě reprezentují populaci kolekce jedinců, v nichž každý jedinec představuje stromovou strukturu, viz obr. 5.2, kde si každý uzel uchovává hodnotu genu,

[1] https://www.python.org/
[2] https://matplotlib.org/
[3] https://numpy.org/
[4] https://pypi.org/project/imageio/

ukazatele na další následovníky a informace o aktuální pozici a směru mravence včetně jeho aktuálně uražené cesty. Program využívá v základu množiny terminálních (T) a funkčních (F) symbolů. Jejich význam a funkčnost je popsána v kapitole 5.

```
T = { LEFT, RIGHT, MOVE }
F = { PROGN2, PROGN3, IF_FOOD_AHEAD }
```

Množiny terminálních (T) a funkčních (F) symbolů u stromového GP bez rozšíření.

Každý tento jedinec tedy představuje instanci stromové třídy, jež obsahuje různé metody mezi které patří například zjištění výšky či velkosti stromu, vygenerování náhodného podstromu ve zvoleném bodě, aplikaci genetických operátorů a další. Interpretace takového jedince probíhá vždy vyhodnocením aktuálního uzlu a pokud je to možné, tak následně vyhodnocením levého podstromu (jinak středního, popřípadě pravého). Tento postup je shodný s metodou procházení stromu *pre-order*. Před vykonáním instrukce `MOVE` je vždy zkontrolováno, aby se mravenec nedostal za hranice mřížky. Pokud instrukci nelze provést, tak se pozice mravence nemění.

V rámci řešení operátoru křížení bylo implementováno několik variant, z nichž se každá vypořádává jiným způsobem s problémem *bloatu*. V základu ale tyto varianty fungují téměř podobně, kdy náhodně ze dvou vybraných rodičů vznikne pouze jeden potomek, který je použit pro novou generaci. První varianta křížení se s problémem *bloatu* vypořádává tím nejjednodušším způsobem. Po vytvoření potomka je provedena kontrola, zdali nový strom vyhovuje maximálnímu výškovému omezení a v případě nevyhovění je strom zarovnán terminálními symboly. Druhá varianta operátoru postupuje obdobným způsobem, ale namísto zarovnání stromu se provede křížení s původními rodiči znovu a pokud opět nevyhovuje výšce, tak je s původními rodiči provedena mutace a případně i poté strom zarovnán. Mutační operátor je implementovaný ve variantě generující podstromy se standardně omezenou výškou 2 v náhodně vybraném uzlu mutace. Pokud opět následně potomek nevyhovuje výškovým omezením, tak je zarovnán na maximální hloubku terminálními symboly.

6.2 Lineární GP

Populaci zde tvoří kolekce objektů, jež reprezentují jednotlivá kandidátní řešení, která si udržují svou hodnotu fitness a podrobné informace o aktuálním řešení problému. Každé kandidátní řešení je definováno kolekcí objektů, instrukcemi, které obsahují informace o jejím typu a obsahujících datech. Každý jedinec obsahuje několik metod, např. pro výpočet fitness, vložení zjištění délky celého programu, hlavního programu či získání první instrukce požadovaného podprogramu. Reprezentace programu je lineární, kde se jednotlivé instrukce vykonávají sekvenčně. Pro základní variantu této techniky jsou definovány následující množiny terminálních (T) a funkčních (F) symbolů.

```
T = { LEFT, RIGHT, MOVE }
F = { IF_FOOD_AHEAD ? X : Y }
```

Množiny terminálních (T) a funkčních (F) symbolů u lineárního GP bez rozšíření.

Mimo to program může obsahovat předem definovaný počet podprogramů, které jsou na jejich počátku značeny pomocí návěští ve tvaru `* SR [X]`, kde `X` je písmenné označení podprogramu. Na podprogram se odkazuje v rámci ternárních operátorů. Význam terminálních

symbolů je vysvětlen v kapitole 5. Jediný zde přítomný funkční symbol vyjadřuje známý ternární operátor s podmínkou. Pokud mravenec o políčko vpřed vycítí potravu, tak je vykonán kód `X`, jinak `Y`. Po spuštění programu je nejprve vygenerována počáteční populace, nastavením parametru může být nastaven typ inicializace. Je stanoven dolní a horní limit pro délku hlavního těla programu a podprogramu a také jejich počet. Podprogramy se nemůžou odkazovat v rámci podmínek na další podprogramy.

```
MOVE
LEFT
LEFT
IF_FOOD_AHEAD ? MOVE : A
RIGHT
* SR A:
MOVE
MOVE
LEFT
* SR B:
RIGHT
LEFT
```

Ukázka jednoduchého programu skládající se ho z celkově z 10 instrukcí, z čehož 5 instrukcí je součástí podprogramů A a B (návěští se jako instrukce nezapočítávají).

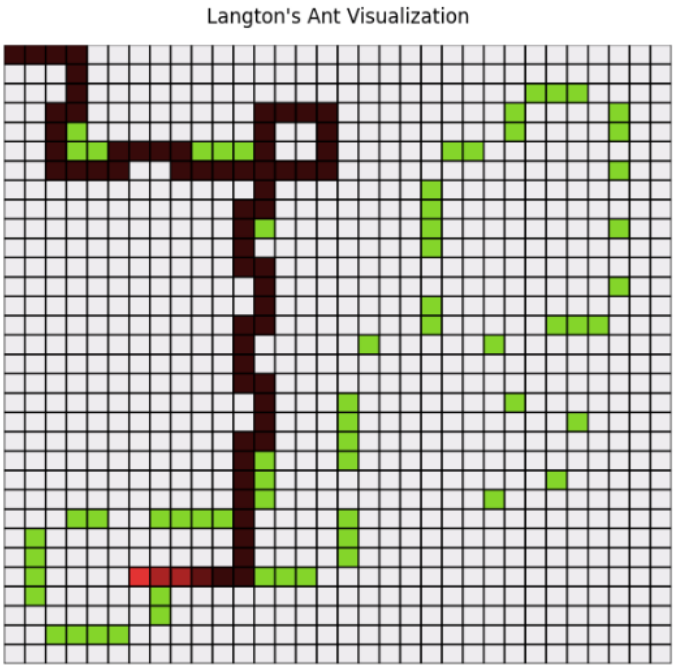
Interpretace probíhá sekvenčně instrukce po instrukci. Pokud by měl být vykonán skok na adresu s podprogramem, tak dojde k uložení aktuální adresy instrukce v hlavním programu a následně se pokračuje na první instrukci daného podprogramu. Pokud podprogram končí, odkazuje se program na uloženou adresu.

V základu je implementováno jednobodové křížení, přičemž jsou vybrány náhodně body, kde se hlavní programy rodičů navzájem překříží. Následně je provedena kontrola, zdali je zachována minimální a maximální délka nového potomka. Pokud je delší, tak dojde k ořezu nadbytečných instrukcí. Pokud je kratší, tak je vybrán druhý potomek. Protože podobná manipulace s podprogramy by byla mnohonásobně náročnější, kde by museli být kontrolovány jednotlivé podprogramy a zdali vyhovují všem podmínkám, což se při prvních náznacích implementace této varianty nezdálo ve většině případech vyhovující, tak se v závěru vybere náhodně jeden podprogram v obou nových potomcích a ty se překříží. Analogickým postupem, ale se třemi segmenty je implementováno i dvoubodové křížení. Mutační operátor se vyskytuje jak v jeho makro verzi, kde pracuje s instrukcemi jako atomickými prvky, tak i mikro verzi, kdy může dojít u funkčního symbolu `IF_FOOD_AHEAD ? X : Y` k náhodné obměně proměnných `X` a `Y`, a to buď odkazem na na podprogram nebo instrukcí. U makro mutace může nastat jedna z variant – a to buď mutace aditivní, destruktivní anebo měnící.

Než byla implementována varianta s charakterem jazyka symbolických instrukcí a podprogramy, tak bylo prvotně uvažováno o variantě registrové, která se později pro tento problém ukázala ne zcela ideální. O něco slibněji již vypadal první prototyp využívající větvení `if...else`, ale stále bez lepších výsledků.

6.3 Vizualizace problému

První verze vizualizace problému a jeho řešení byla implementována prostým zobrazením v terminálu za pomocí ASCII Art. Tuhle vizualizaci lze vidět po každém doběhnutí algoritmu ve spuštěném okně. Toto zobrazení bylo doplněno barevnou interpretací pomocí knihovny *colored*. S využitím knihovny *matplotlib* a *imageio* bylo dokázáno tuhle vizualizaci ještě vylepšit, viz obr. 6.1. Pomocí prvně zmíněné knihovny a modulu *cmap* a několika zvolených barev je zaznamenáván každý pohyb mravence. Každý krok se uloží jako obrázek a ty se následně spojí pomocí funkce z knihovny *imageio* a je vytvořena *gif* animace.



Obrázek 6.1: Pokročilá vizualizace řešení

Zelená políčka v obr. 6.1 reprezentují potravu, šedá políčka bez potravy, červená mravence a hnědá jeho cestu. Jednotlivé obrázky se uchovávají ve složce `/images` a výsledný gif lze najít ve složce `/gif`. Aktivaci a deaktivaci je možné nastavit v souboru `init_params.py` u parametru `CREATE_GIF`.

Kapitola 7

Experimentování

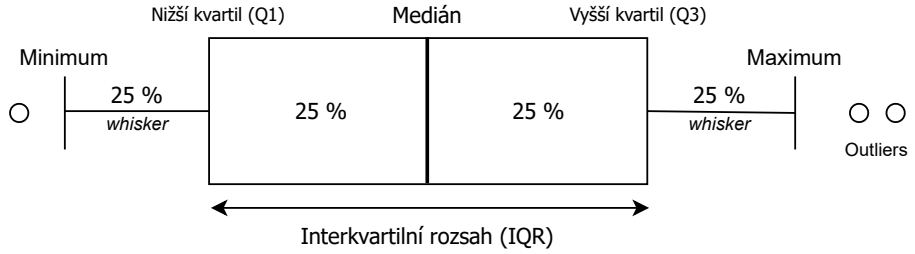
Praktické části práce se věnuje právě tato kapitola, jež nejprve obsahuje obecný popis metod pomocí kterých jsou experimenty statisticky vyhodnocovány a poté i následné experimenty. Ty se prvně zaměřují na lineární GP (LGP), porovnání různých vylepšení, selekčních metod a taktéž hledání ideálních parametrů pro dané nastavení u vybrané a náhodné stezky. Poté analogicky následuje stromové GP (TGP) s podobnými experimenty. Mezitím jsou taktéž srovnány obě varianty s výsledky z literatury. Nejsou zde bohužel zahrnuta všechna vylepšení a také kombinace různých nastavení, která byla v průběhu implementace nalezena a otestována z důvodu již tak pravděpodobně velkého rozsahu praktické části. Na závěr je o těchto dalších vylepšeních, případně těch, které se na první pohled zdály jako vylepšení, ale později se ukázalo, že algoritmus spíše degradují, pojednáno.

7.1 Způsob vyhodnocování experimentů

Statistické vyhodnocování je prováděno a vizualizováno níže vysvětlenými metodami.

Boxploty

Často jsou výsledky vizualizovány pomocí boxplotu, což je grafická reprezentace, která zobrazuje data na základě pětičíselného souhrnu. Ten se skládá z minimální a maximální hodnoty, dolního (Q1) a horního (Q3) kvartilu a mediánu. Medián je reprezentován vertikální čárou uvnitř grafu, viz obr. 7.1. Kvartily vyjadřují střední hodnotu mezi mediánem a nejmenší či největší hodnotou. Dále nám dokáže tato reprezentace říct, zdali jsou data symetrická, jejich rozptyl a variabilitu nebo jestli se v nich nachází nějaké vyčnívající hodnoty (angl. *outliers*) [8].



Obrázek 7.1: Vizualizace a popis boxplotu

Dalším výsledkem experimentů jsou **konvergenční křivky**, jež vyjadřují grafické závislosti konvergence algoritmu na počtu iterací, tedy jak rychle se algoritmus přibližuje k řešení. Posledním prostředkem k vizualizaci je **tepelná mapa**, což je grafické zobrazení dat, ve kterém je každá hodnota reprezentována barvou určitého barevného spektra. Grafy v následujících experimentech jsou zpracovány knihovnou *matplotlib*.

Každá sada experimentů má definovanou skupinu stálých parametrů, které jsou shodné pro všechny testy v této sadě. Tato tabulka obsahuje:

- *(jen LGP)* Uspořádanou pětici P = $(L_{min}, L_{max}, SR_{num}, SR_{min}, SR_{max})$ vyjadřující velikostní omezení programu, kde:

  L_{min} – minimální délka hlavního programu
  L_{max} – maximální délka hlavního programu
  SR_{num} – počet generovaných podprogramů
  SR_{min} – minimální délka podprogramu
  SR_{max} – maximální délka podprogramu

- *(jen TGP)* výškové omezení programu min. a max. hloubkou stromové struktury,
- inicializační metodu,
- selekční, rekombinační, mutační operátory,
- vybranou stezku (nahlédnutí na konkrétní stezku je možné ve zdrojových kódech v souboru `trails_plots.py`).

Pokud je experimentováno s velikostí populace, tak je důležité, aby bylo dodrženo stejné velikosti prohledávaného prostoru k najití nejlepšího řešení každému běhu, tzn. musí se zajistit, aby platila rovnice:

velikost populace × počet generací = konstanta

Pro všechny experimenty bude společná následující tab. parametrů 7.1.

Parametry	
Cíl	sesbírání veškeré potravy na mřížce
Fitness funkce	standardizovaná
Elitismus	ano
Počáteční pozice	[0, 0]

Tabulka 7.1: Společné parametry pro všechny experimenty

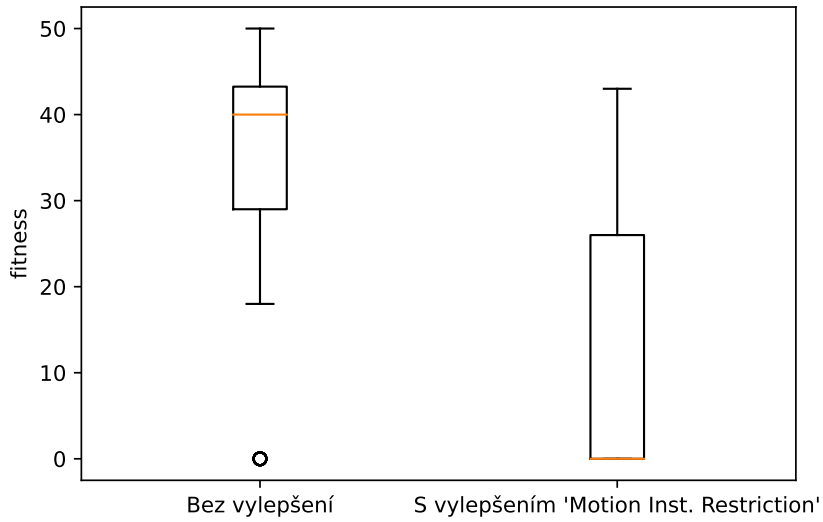
7.2 Experimenty

7.2.1 Vylepšení LGP I – Motion Instruction Restriction

Parametry	
Délka programu	P = (10, 15, 5, 4, 5)
Stezka	Santa Fe (max. fitness 89)
Inicializační metoda	náhodná s vel. omezeními
Selekce	turnajová
Křížení	jednobodové s P(0.5)
Mutace	makro & mikro s P(0.1)
Populace	100
Generace	250

Tabulka 7.2: Parametry

Toto vylepšení spočívá v znemožnění algoritmu v rámci genetického operátoru mutace v podprogramech přidávat nebo nahrazovat jinou instrukci instrukcí `MOVE` (při inicializaci počáteční populace se v podprogramech nadále nacházet může!). V experimentu bylo spuštěno 60 nezávislých testů pro každé nastavení.



Obrázek 7.2: Porovnání aplikovaného vylepšení

Z výsledků testů na obr. 7.2 je patrné, že i taková drobnost jako odstranění instrukce může učinit tak znatelný rozdíl ve výsledcích. Chvíli jsem nad tímto chování programu přemýšlel a dospěl jsem k závěru, že je pro algoritmus mnohem snazší najít řešení, pokud dostane během evoluce více prostoru ke správnému nasměrování se v rámci mřížky, než vykonávání ukvapených pohybů k uražení co největší vzdálenosti, ale neefektivně. Často se tedy v rámci hlavních programů vyskytují pouze jeden či dvě pohybové instrukce, které si s úlohou hravě poradí, než kupříkladu deset pohybových instrukcí a minimum instrukcí k určení směru pohybu. Dále z výsledků vyplynulo, že bez vylepšení se podařilo algoritmu

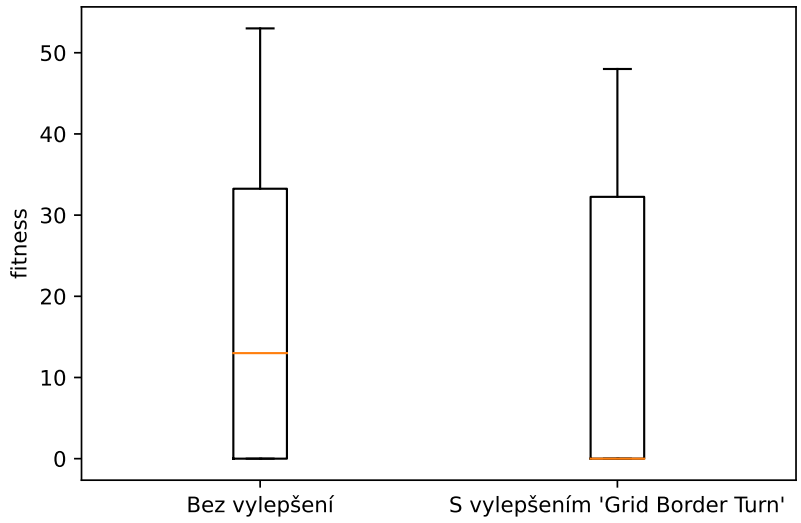
najít řešení v celkem 11 případech z 60, zatímco s vylepšením v 36 případech. Nárůst řešení je tedy více než trojnásobný a pokles mediánu dokonce o 40 čísel!

7.2.2 Vylepšení LGP II – Grid Border Turn

Parametry	
Délka programu	P = (10, 18, 6, 3, 4)
Stezka	Santa Fe (max. fitness 89)
Inicializační metoda	náhodná s vel. omezeními
Selekce	turnajová
Křížení	jednobodové s P(0.4)
Mutace	makro & mikro s P(0.1)
Populace	100
Generace	250

Tabulka 7.3: Paramtery

Toto vylepšení spočívá v tom, že pokud je mravenci znemožněn pohyb instrukcí `MOVE`, protože se nachází na hranici dvojdimenzionální mřížky, tak se pootočí o 90° ve směru hodinových ručiček. Toto opatření vkládá do pravidel určitou další diverzitu, která může znamenat najití lepších řešení. Pro každé z obou variant bylo nezávisle spuštěno 4krát 40 testů, tedy 160 testů. Obě dvě varianty již zahrnují implementované rozšíření I.



Obrázek 7.3: Porovnání aplikovaného vylepšení, obě nastavení již implementují první vylepšení

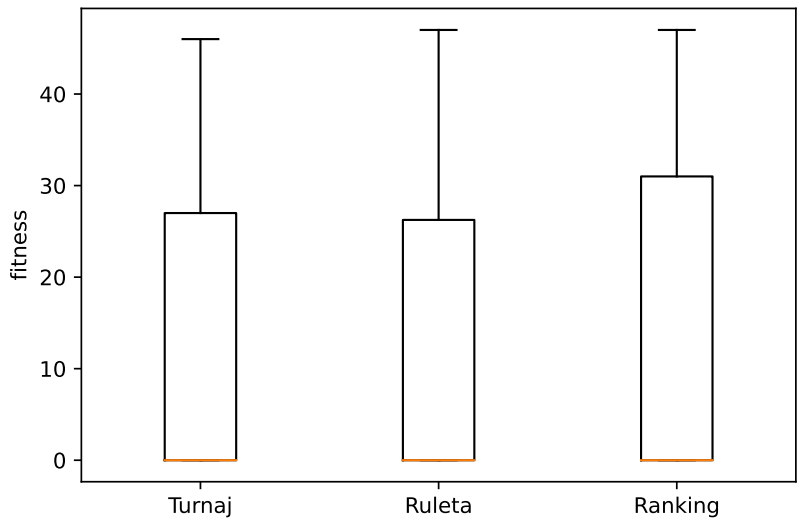
Na první pohled je jasné, že se zdaleka nejedná o tak markantní zlepšení jako v případě prvního vylepšení, viz 7.2.1. I proto bylo provedeno mnohem více testů. Taktéž je dobré připomenout, že počáteční fixní hodnoty pro oba algoritmy se od předchozího, prvního, mírně liší. I tak bylo dosaženo lepších výsledků. Hodnota mediánu se v obou variantách mírně liší, konkrétně o hodnotu 8 ve prospěch varianty s vylepšením.

7.2.3 Srovnání selekčních metod

Parametry	
Délka programu	P = (10, 15, 5, 4, 5)
Stezka	Santa Fe (max. fitness 89)
Inicializační metoda	náhodná s vel. omezeními
Selekce	turnajová / ruleta / ranking
Křížení	jednobodové s P(0.5)
Mutace	makro & mikro s P (0.1)
Populace	100
Generace	250

Tabulka 7.4: Parametry

V tomto experimentu jsem se zaměřil na porovnaní několika různých selekčních operátorů. Často je aplikován pouze operátor turnajový, ale co když existují i takové operátory, které si dokážou právě na této úloze vést lépe? Pro porovnání byla ještě kromě turnaje vybrána selekce ruletová a ranking. Pro každý níže zobrazený box plot bylo spuštěno 120 nezávislých běhů.



Obrázek 7.4: Výsledky experimentů selekčních metod

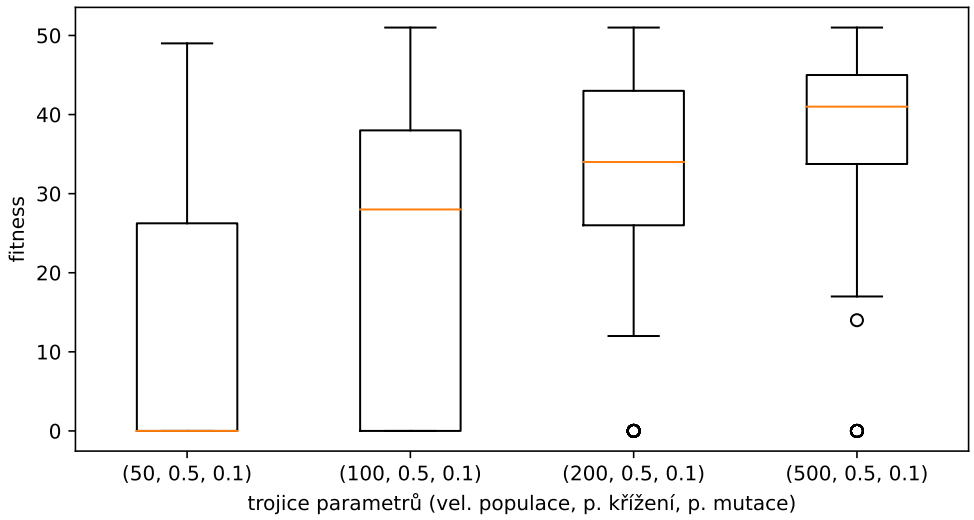
Na obr. 7.4 lze vidět, že všechny operátory si s touto instancí problému úspěšně poradily. Medián všech se drží na té nejlepší hodnotě. Menší výkyvy je možné zaznamenat ve vyšších hodnotách, ačkoliv ani tyto rozdíly nejsou nikterak velké, 75 procent všech hodnot je nejmenších právě u selekce ruletové. Konečné výsledky běhů ale nebyly pouze to jediné co mě zajímalo. Bohužel se u tohoto problému neprojevila skutečnost zmíněná v teoretické části, že některé z těchto operátorů jsou náchylnější k rychlejší konvergenci nebo v nadměrném upřednostňování vhodnějších jedinců. Na všech vzorcích nebyl patrný větší rozdíl. U turnajové selekce lze selekční tlak ještě ovlivnit nastavením velikostí počtu jedinců, jež do turnaje vstupují.

7.2.4 Hledání ideálních parametrů LGP na vybrané stezce

Parametry	
Délka programu	P = (10, 15, 5, 4, 5)
Stezka	Santa Fe (max. fitness 89)
Inicializační metoda	náhodná s vel. omezeními
Selekce	turnajová
Křížení	jednobodové
Mutace	makro & mikro

Tabulka 7.5: Parametry

Tento experiment již zahrnoval komplexnější hledání ideálního nastavení algoritmu. A to konkrétněji u tří parametrů – velikosti populace, pravděpodobnosti křížení a mutace pro nalezení optimálního řešení. Nejprve byly testy spuštěny pro parametr populace, následně byl vybrán nejlepší výsledek, fixace této velikosti a analogickým postupem provedení stejného postupu i u p. křížení a mutace. Bylo provedeno 120 nezávislých testů pro každé nastavení algoritmu. Tato i následující sady experimentů již zahrnují první implementované rozšíření.

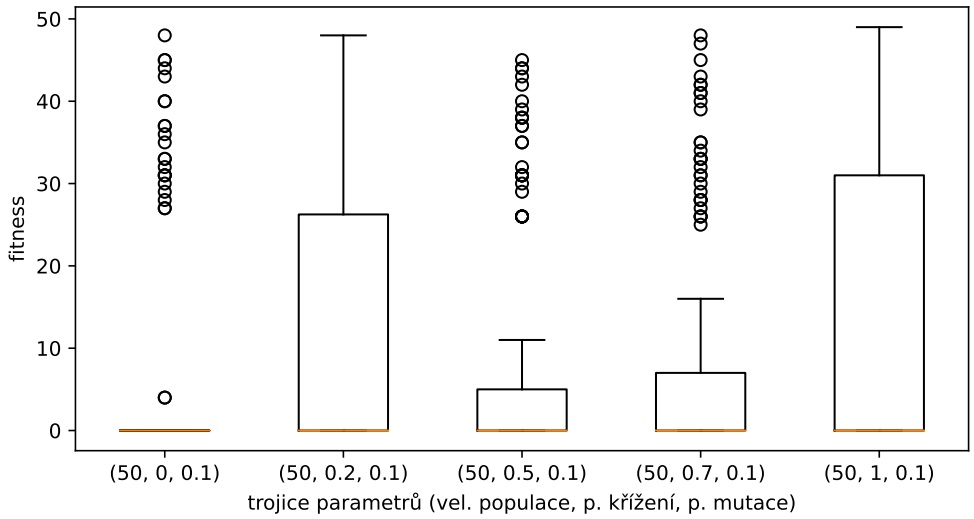


Obrázek 7.5: Zjištění závislosti dosažené fitness na vel. populace

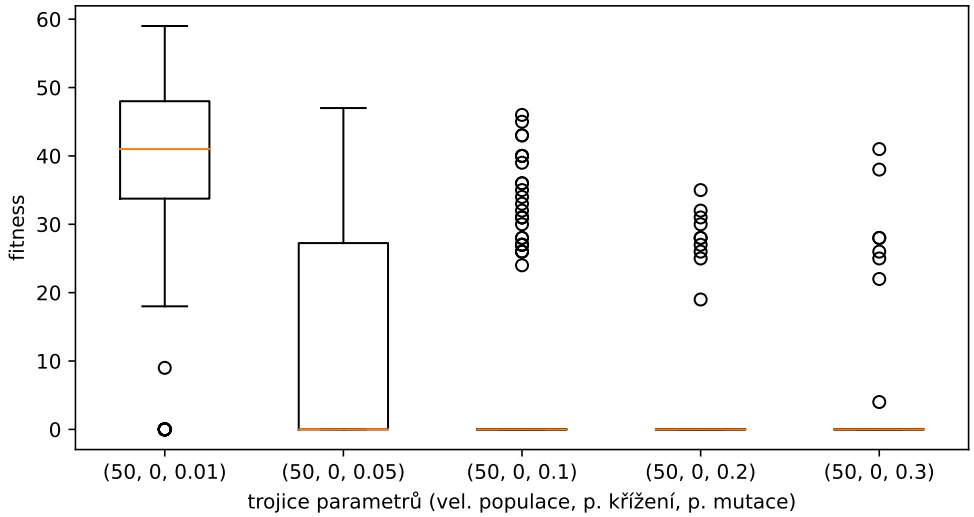
Z grafu 7.5 lze vidět, že řešení byla nalezena u každé velikosti populace. Nejlepší výsledky byly obdrženy při nejmenší populaci. Podobného výsledku dosahuje i boxplot s velikostí populace 100, ale již s vyšším mediánem. Proto do dalšího spuštění programu byla zafixována velikost populace na 50 a zkoumán vliv různých hodnot pravděpodobnosti křížení.

Z obr. 7.6 je patrné, že medián se u všech řešení nachází na hodnotě nula. Parametr pravděpodobnosti křížení byl zafixován pro první nastavení, protože se zde dosáhlo nejlepších výsledků. Ostatní nastavení pravidelněji dosahují vyšších hodnot fitness.

U prvního boxplotu, viz 7.7, je viditelné, že nulová pravděpodobnost křížení a velmi nízká pravděpodobnost mutace způsobila narůst mediánu. Již u druhého nastavení bylo dosaženo viditelného pokroku. Nejlepších výsledků je dosahováno u pravděpodobnosti 0.1, 0.2 a 0.3. Ty se již liší jen v počtu a rozsahu odlehlých hodnot. Nejoptimálnějšímu výsledku



Obrázek 7.6: Zjištění závislosti dosažené fitness na prav. křížení



Obrázek 7.7: Zjištění závislosti dosažené fitness na prav. mutace

proto vyhovují všechna tři poslední nastavení v grafu. Celkově až na některé hodnoty variačních operátorů, které byly například u mutace nastaveny velice nízko, dosahovala ostatní řešení velice dobrých výsledků. Každé nastavení našlo minimálně jedno řešení. Je patrné, že výběr p. křížení zde nehrál tak zásadní roli jako při mutaci či velikosti populace. Je velice obdivuhodné, jak si tento algoritmus dokázal poradit s touto instancí problému.

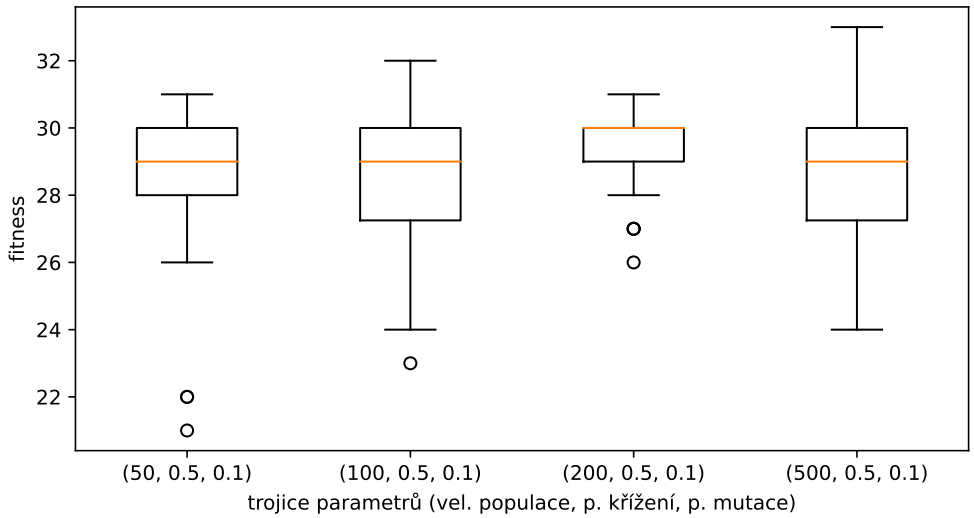
7.2.5 Hledání ideálních parametrů LGP na náhodné stezce

Opět jako u předchozího experimentu byly zjišťovány nejlepší kombinace parametrů – velikosti populace, pravděpodobnost křížení a mutace pro dosažení co nejlepších výsledků.

Parametry	
Délka programu	P = (8, 12, 5, 3, 5)
Stezka	Random Trail L (max. fitness 68)
Inicializační metoda	náhodná s vel. omezeními
Selekce	turnajová
Křížení	jednobodové
Mutace	makro & mikro

Tabulka 7.6: Parametry

Velikost dvojdimenzionální mřížky je opět 32 na 32 buněk, dohromady tedy 1024 buněk, z čehož 68 tvoří náhodně rozmístěnou potravu.

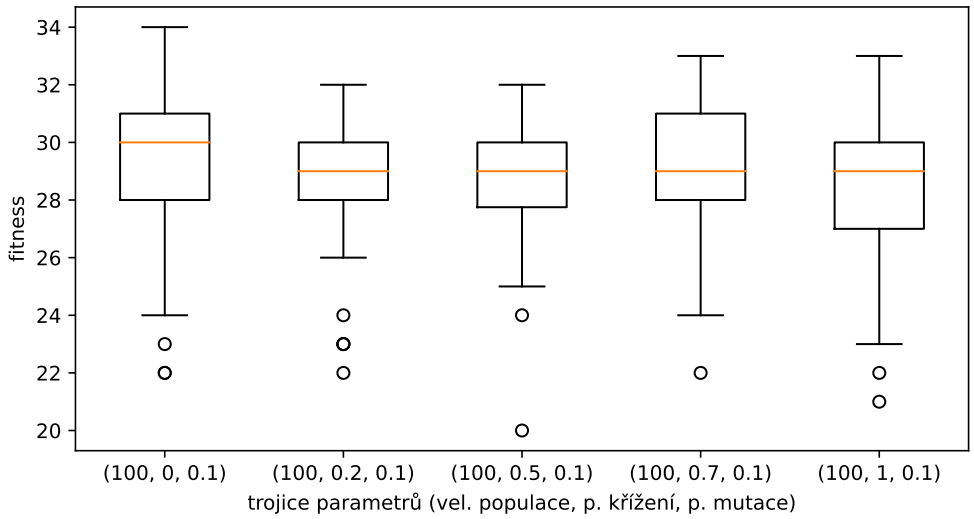


Obrázek 7.8: Zjištění závislosti dosažené fitness na vel. populace

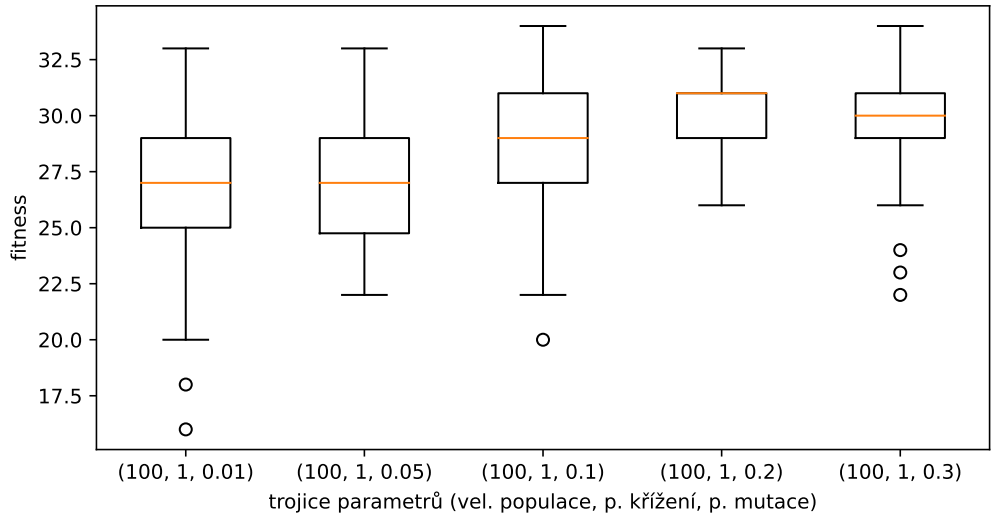
Na obr. 7.8 lze vidět, že rozdíly ve výsledcích testů nejsou nikterak markantní, ale je zde pozorovatelné, že některá nastavení si vedla o něco málo lépe než ostatní. Nejnižší fitness byla nalezena u nastavení s nejmenší populací, ale je to hodnota, která vybočuje. Velice se liší od ostatních hodnot u tohoto nastavení. Proto pro další spuštění algoritmu byla vybrána varianta s populací o velikosti 100, kde minimální hodnoty dosahují častěji menších hodnot než u předchozího nastavení. Varianta o populaci 500 je zde na tom podobně, nejhůř dopadla varianta s velikostí 200, kde se nachází velmi malý rozptyl získaných hodnot, které se ve většině případů pohybují okolo hodnoty fitness 28 až 30.

Křížení na obr. 7.9 opět nevykazuje velký vliv na získaná řešení, ale podařilo se nalézt opět o něco lepší řešení, které dosahuje hodnoty fitness 20. Medián všech box plotů se zde ve většina případů až na malé desetinné odchylky rovná. Nejlepší minima, nepočítaje opět vyčnívající hodnoty, byla nalezena u nastaveni křížení s pravděpodobností 1, a proto bude i toto nastavení použito pro další experimentování.

Kombinace parametrů (100, 1, 0.01) na obr. 7.10 bylo označeno jako nejlepší řešení a to s hodnotou fitness 16. Řešení s vyšší pravděpodobnosti mutace si již nevedou tak dobře, jejich medián je oproti zbylým vyšší a rozptyl hodnot menší.



Obrázek 7.9: Zjištění závislosti dosažené fitness na prav. křížení



Obrázek 7.10: Zjištění závislosti dosažené fitness na prav. mutace

Experimentování s náhodnými stezkami je mnohem náročnější než s těmi ideálními, které jsou především tvořené ze souvislých části obsahující pár menších či větších mezer. Pokud se náhodně na mřížce vyskytuje až příliš mnoho políček s potravou, tak je velice nepravděpodobné nalézt řešení s nejlepší fitness hodnotou, protože by často muselo dojít k systematickému projití celé mřížky než najití nějakého schopného algoritmu s co nejkratší cestou. Na druhou stranu, pokud se políček s náhodně rozmístěnou potravou nachází na mřížce v malém množství, tak sice nalezení takového programu, jenž by instanci tohoto problému zvládnul splnit je možné, ale opět to nebude snadné. Jedna z klíčových vlastností tohoto algoritmu je podmíněnost, kde právě role políček s potravou a podmíněných funkčních symbolů tvoří tu část, že se algoritmus nebude pokaždé chovat stejně. Terminální

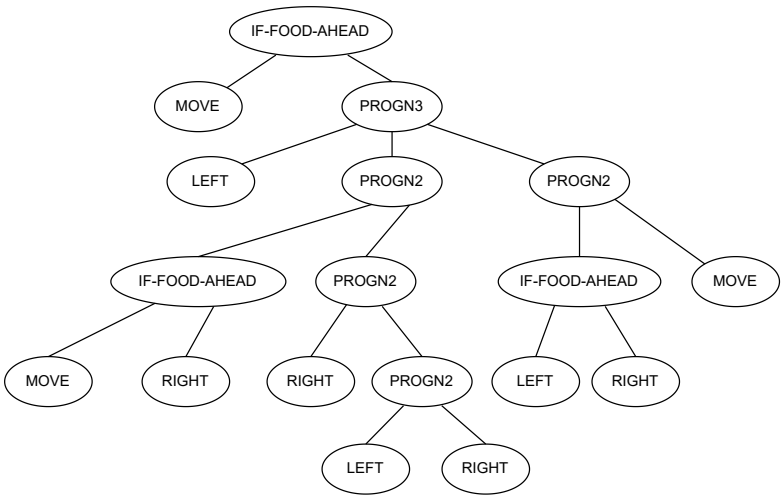
podmínky pro náhodné stezky jsou nadsazenější než u ideálních, většinou na 40 % projitých políček z celé mřížky, což je optimální hodnota pro to, aby řešení nebylo tvořeno jen náhodnou procházkou, ale zároveň bylo poskytnuto dostatečné množství prostoru. Podobné testy byly provedeny i na menší mřížce (Random Trail S) o velikosti 400 políček s náhodně rozmístněnou potravou. Zde si algoritmus vedl o něco lépe, kde nacházel řešení průměrně v 5 bězích ze 40. V bězích u nichž se řešení nenašlo, se průměrná fitness pohybovala kolem 1 až 3.

7.2.6 Porovnání s výsledky dostupnými v literatuře

Srovnání proběhlo na základě řešení dostupného v literatuře [11] (str. 153 – 155) s výsledky získanými s oběma variantami GP, u nichž byly vybrány jedny z nejlepších řešení. Obr. 7.11 znázorňuje řešení, kterého bylo dosaženo za pomocí stromového GP v 21. generaci.

Parametry	Literatura TGP	LGP	TGP
Délka programu	*(nebylo zjištěno)*	P = (6, 12, 5, 4, 5)	min. = 4, max. = 6
Stezka	Santa Fe	Santa Fe	Santa Fe
Populace	500	100	100
Generace	51	120	120

Tabulka 7.7: Parametry

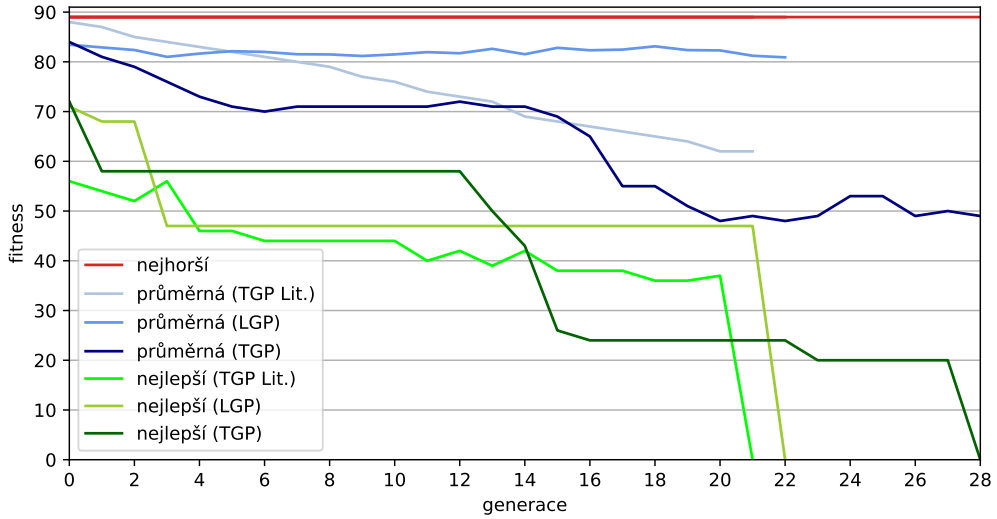


Obrázek 7.11: Řešení uvedené v literatuře

Řešení je velice úsporné, dosahuje hloubky pět a k pohybu mravence zde slouží 11 terminálních symbolů. Podmíněných uzlů se zde nachází tři, obzvláště ten v kořenu stromu má zásadní vliv, jestli vykoná pouze instrukce `MOVE` nebo i zbytek podstromu. Obr. 7.12 značí mnou dosaženého řešení za pomocí lineárního GP v 22. generaci. Program se skládá z hlavního programu obsahujícího 7 instrukcí, z čehož většinu tvoří instrukce mravence a 2 ternární operátory. Šedou barvou jsou vyznačeny části kódu, resp. podprogramy, které se nikdy nevykonají. Je zajímavé, že celý program obsahuje pouze 3 instrukce `MOVE`, z kterých je pouze jedna nepodmíněná a evoluce taktéž upřednostnila podprogramy bez této instrukce, podprogramy A ji totiž obsahuje. Obě zde zmíněná řešení splnila úlohu v co nejkratší možnou dobu, tedy nejmenším počtu kroků.

```
IF FOOD_AHEAD ? MOVE : B    * SR A:
RIGHT                       RIGHT
IF FOOD_AHEAD ? MOVE : C    LEFT
MOVE                        MOVE
LEFT                        RIGHT
LEFT                        * SR B:
RIGHT                       LEFT
* SR E:                     RIGHT
RIGHT                       LEFT
LEFT                        RIGHT
LEFT                        RIGHT
* SR C:                     * SR D:
RIGHT                       LEFT
RIGHT                       LEFT
RIGHT
```

Obrázek 7.12: Řešení nalezené lineárním GP



Obrázek 7.13: Konvergenční křivky s fitness hodnotami

Velikost a strukturu řešení získané vlastním programem stromového GP si je možné prohlédnout na přiloženém médiu ve složce s experimenty.

Ačkoliv mé řešení stromového GP vykazuje o trochu lepší výsledky průměrné fitness, tak trend klesající fitness na mnoha místech kopíruje křivku fitness hodnoty z literatury, viz 7.13. Zajímavé hodnoty vykazuje řešení lineárního GP, kde fitness klesala markantněji pouze na začátku a poté byly zaznamenány pouze menší ojedinělé výkyvy. Řešení byla nalezena vcelku rychle. Lineární GP zde kopíruje křivku nejlepší fitness stromového GP z literatury. Nejdéle trvalo najití řešení mé implementaci stromového GP.

7.2.7 Komplexní analýza parametrů TGP na vybrané stezce

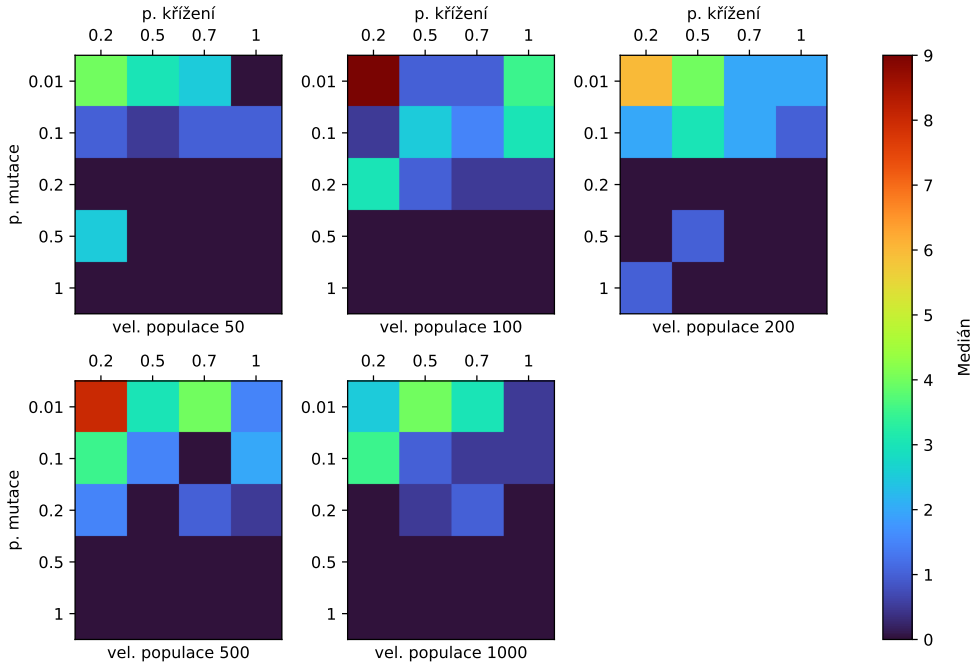
Parametry	
Velikost programů	minimální = 3, maximální = 5
Stezka	Trail C (max. fitness 25)
Inicializační metoda	grow
Selekce	turnajová
Křížení	křížení se zarovnáním
Mutace	generující podstromy

Tabulka 7.8: Parametry

Populace	50	100	200	500	1000
P(Křížení)	0.2	0.5	0.7	1	
P(Mutace)	0.01	0.1	0.2	0.5	1

Tabulka 7.9: Analyzované hodnoty

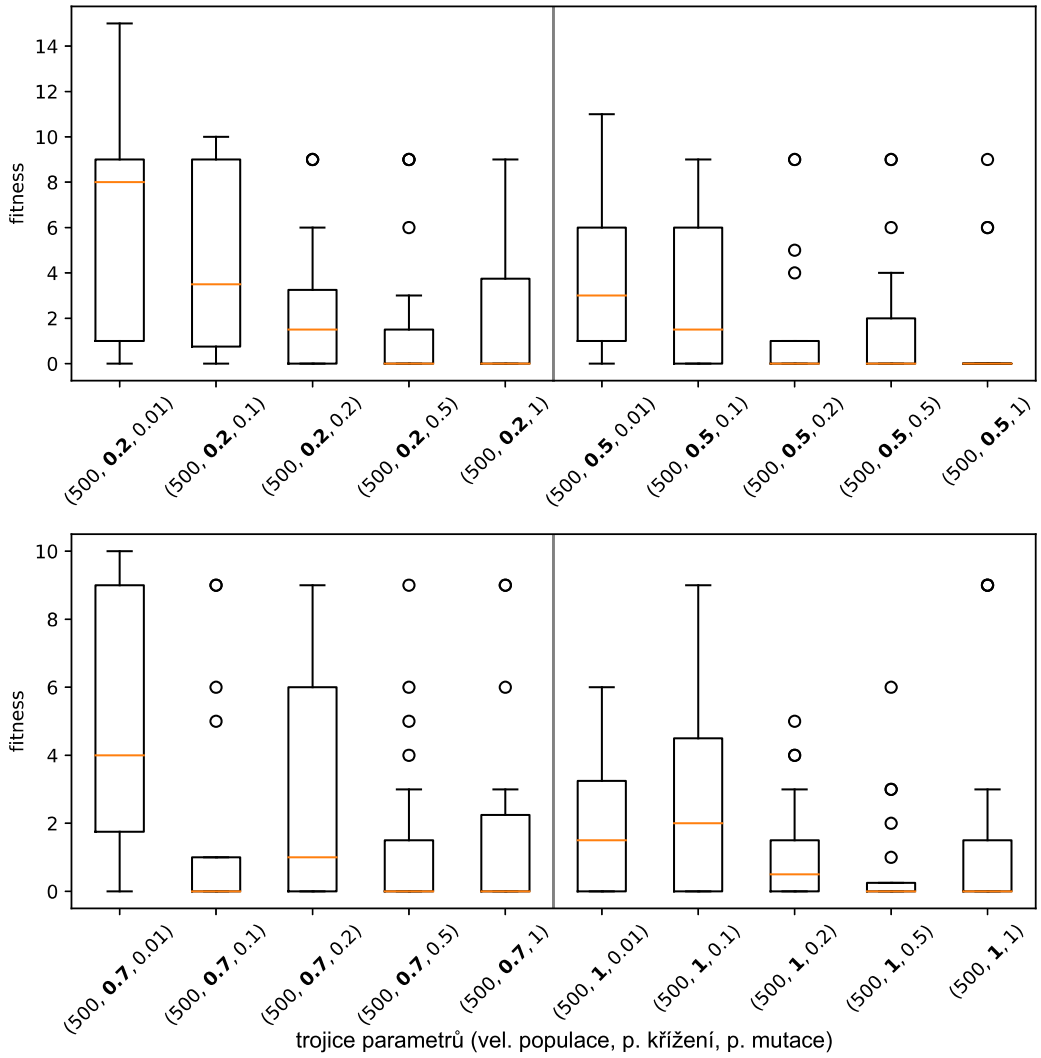
V rámci tohoto experimentu byla provedena komplexní analýza všech třech parametrů a jejich hodnot zmíněných v tab. 7.9, tzn. důkladné testování všech možných trojic, které z této tabulky můžou vzniknout ($5 * 4 * 5 = 100$ trojic). Pro každou trojici bylo provedeno 20 nezávislých běhů.



Obrázek 7.14: Teplotní mapy znázorňující velikost mediánu pro různé populace

V této práci je prezentována podrobněji pouze část experimentů z důvodu rozsáhlosti, která by tato analýza spotřebovala. Hodnota mediánu všech sad je vyobrazena na obr. 7.14. Experimenty jsou rozděleny do 5 skupin (každá o různé populaci), vždy po 20 boxplotech.

Pro podrobnější prezentaci bylo vybráno nastavení s velikostí populace 500, viz obr. 7.15. Osa x je vždy popsána kombinací parametrů pro které byla sada experimentů spuštěna. Tučně je vyznačen parametr shodný pro danou pětici sad, v tomto případě pravděpodobnost křížení. V každé této pětici se následně ještě mění pravděpodobnost mutace. U všech 4 pětic se opakoval obdobný trend, viz obr. 7.15, že řešení s větší pravděpodobností mutace (≥ 0.5), nezávisle na křížení, dosahovala lepších výsledků povětšinou mediánu s hodnotou 0 a malým počtem hodnot s vysokou fitness, než řešení naopak s menší pravděpodobností. Největší medián je k spatření u nejmenší hodnoty křížení. Taktéž nezávisle na populaci, viz obr. 7.14, se zde nikterak neprojevila výhoda větší či menší populace i s tím, že každé z nich byl poskytnut stejný prohledávací prostor řešení. Závěrem lze tedy říct, že mutace se z tohoto experimentu jeví jako silnější operátor než křížení majícího vliv na kvalitu výsledného řešení.



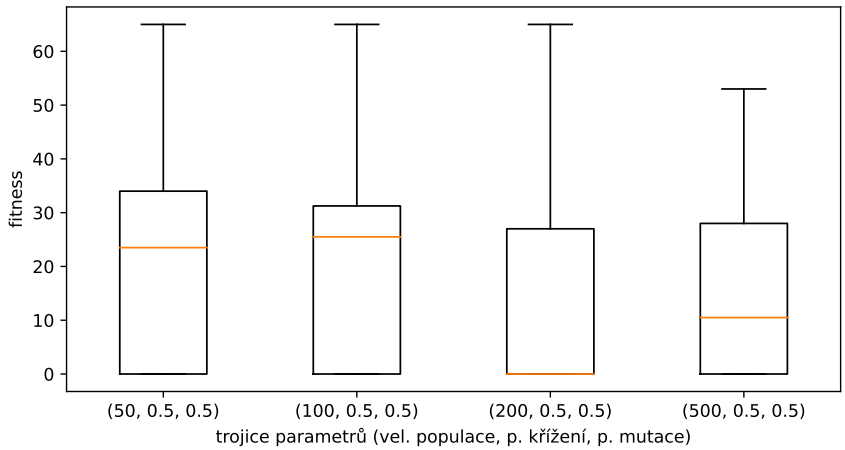
Obrázek 7.15: Vybraná skupina experimentů se stejnou velikostí populace

7.2.8 Hledání ideálních parametrů TGP na vybrané stezce

Parametry	
Velikost programů	minimální = 4, maximální = 6
Stezka	Santa Fe (max. fitness 89)
Inicializační metoda	grow
Selekce	turnajová
Křížení	křížení se zarovnáním
Mutace	generující podstromy

Tabulka 7.10: Parametry

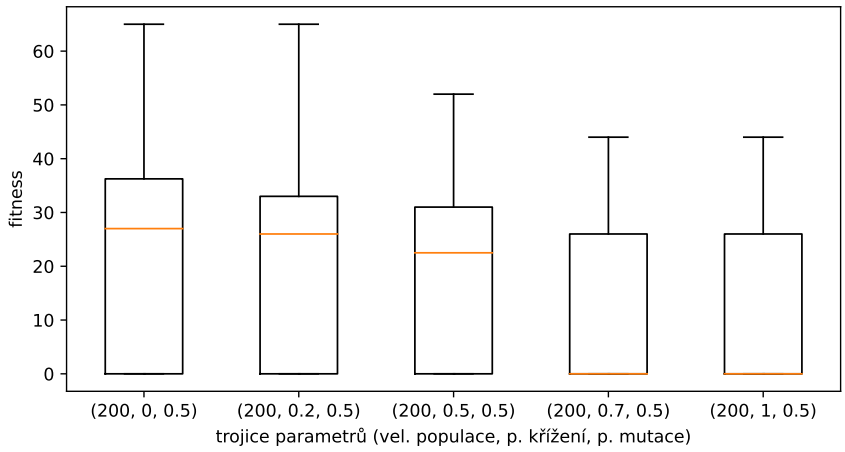
Obdobně jako v experimentu 7.2.4 byly hledány ideální parametry (velikost populace, p. křížení a p. mutace) algoritmu pro hodnoty v tab. 7.10. Ačkoliv velikostní omezení programu je jen o jedničku vyšší ve srovnání s minulým experimentem, což se může zdát pro již takto komplexnější úlohu málo, tak toto omezení nebere v potaz šířku řešení, tzn. stromu, které může být neomezené. Tyto výsledky budou následně použity pro porovnání s lineárním GP. Pro každé nastavení bylo spuštěno 120 nezávislých běhů.



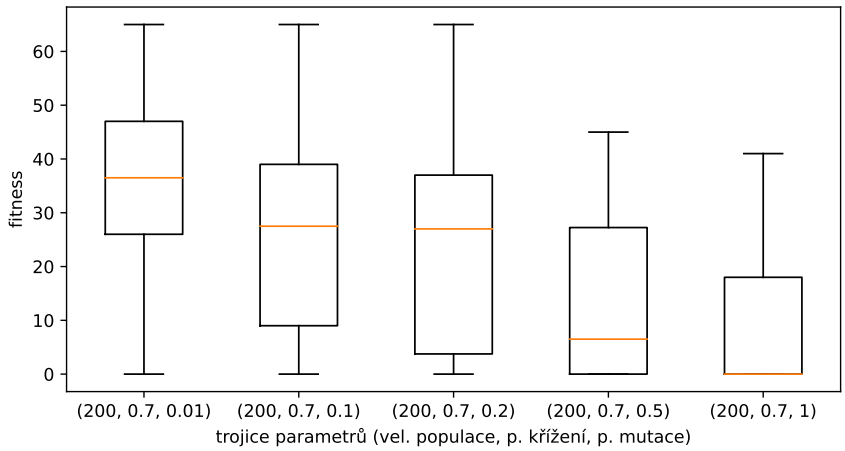
Obrázek 7.16: Zjištění závislosti dosažené fitness na velikosti populace

Všechna nastavení úspěšně našla několik řešení, ale jejich medián se poněkud liší. Za nastavením s populací 200 značně pokulhávají první dvě nastavení. Všechna řešení si taktéž udržují vyšší hodnoty maximální fitness v porovnání s lineárním GP v experimentu 7.2.4. Všechna data jsou ale konzistentní, žádné hodnoty nevybočují. Pro další experimenty bylo vybráno nastavení s populací 200 z důvodu nejnižší hodnoty mediánu.

Na obr. 7.18 lze vidět již tendenci mediánu klesat se zvyšující se pravděpodobností křížení. Poněkud větší skok lze vidět mezi pravděpodobnostmi 0.5 a 0.7. Skoro až identicky se podobají řešení s dvěma nejvyššími pravděpodobnostmi, náhodně jsem proto do dalšího experimentování vybral tu variantu s hodnotou 0.7.



Obrázek 7.17: Zjištění závislosti dosažené fitness na pravděpodobnosti křížení



Obrázek 7.18: Zjištění závislosti dosažené fitness na pravděpodobnosti mutace

Různé pravděpodobnosti mutace přináší do řešení ještě větší diverzitu. Na rozdíl od obr. 7.17, kde se 75 % všech hodnot nacházelo mezi hranicí prvního a třetího kvartilu, zde u prvních třech boxplotech a zejména toho prvního lze pozorovat opět důležitost tohoto operátoru a to jak se fitness zhorší, pokud se operátor vyskytuje s velice malou pravděpodobností. Nejlepším řešením se tedy osvědčila kombinace parametrů (200, 0.7, 0.1), kde jako u jediné kombinace parametrů se medián nachází na hodnotě 0.

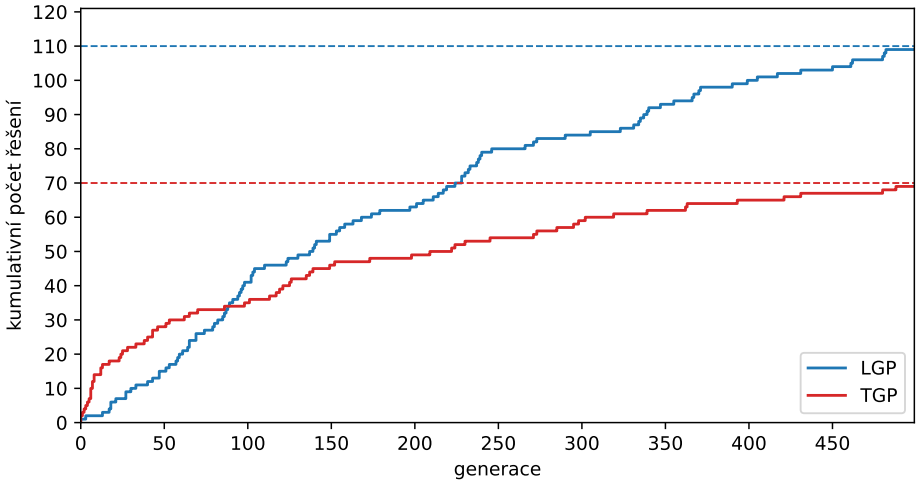
7.2.9 Srovnání stromového a lineárního GP

V rámci tohoto srovnání stromového a lineárního GP bylo primárně vycházeno z experimentů z podkapitol 7.2.4 a 7.2.8, konkrétně z nastavení, které poskytovalo nejlepší výsledky. U lineárního GP kombinace parametrů (50, 0, 0.2) (obr. 7.7), u stromového GP (200, 0.7, 1) (obr. 7.18). Pro každé toto nastavení bylo provedeno 120 nezávislých testů, ze kterých vzejde hlubší statistické vyhodnocení.

	TGP	**LGP**
počet testů	120	120
$\sum$ **fitness**	1067	281
$\varnothing$ **fitness**	8.89	2.34
$\uparrow$ **max. fitness**	41	35
$\downarrow$ **min. fitness**	0	0
modus (počet)	0 (70)	0 (110)

Tabulka 7.11: Statistické porovnání stromového (TGP) a lineárního (LGP) GP

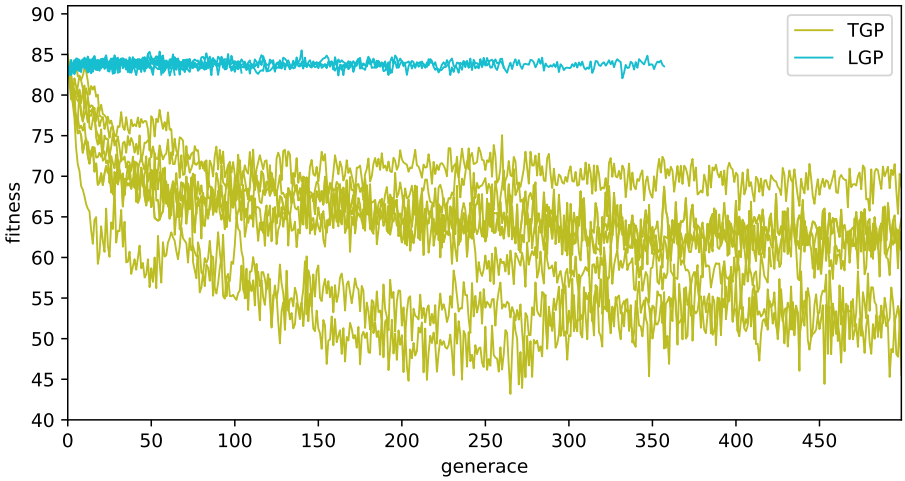
Již na první pohled tab. 7.11 je znatelné, že se varianty v úspěšnosti řešení této úlohy velice liší. Nepatrné odlišnosti lze vidět nejen u součtu celkové hodnoty fitness, ale taktéž u počtu nalezených řešeních. To se v případě lineárního GP podařilo téměř u každého běhu. Maximální fitness hodnota se u obou variant je velice podobá, za to průměrná fitness je u varianty lineárního GP až 4krát nižší.



Obrázek 7.19: Graf zobrazující generace a počet dosud získaných řešení v každé generaci ze všech spuštěných běhů.

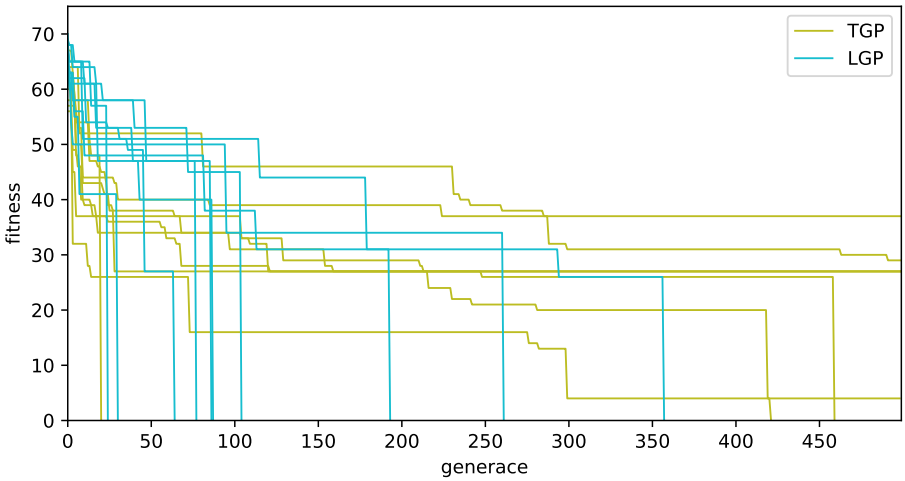
Graf 7.19 znázorňující generace na ose x a počet celkově dosud získaných řešení na ose y v dané generaci z celkově 120 nezávislých běhů. Na počátku lze vidět, že si běhy stromového GP vedou s větší populací jedinců lépe. Téměř 30 běhům se podařilo najít řešení do 50 generací, zatímco u lineárního GP pouze 10 běhům. Zlom nastává po cirka sto generacích, kdy přírůstek již dostává pomalejšího charakteru. Rychlost nalezení řešení lineárního GP se zdá býti až na menší občasné výkyvy konstantní.

Dále byly pro toto srovnání spuštěny sady nezávislých testů o 10 bězích pro každou variantu GP. Následující dva grafy znázorňují konvergenční křivky průměrné a nejlepší fitness.



Obrázek 7.20: Průměrná fitness

Z grafu 7.20 lze vidět, že chování průměrné fitness je u obou variant velmi rozdílné a zároveň i velmi podobné tomu, co bylo prezentováno v podkapitole ve srovnání s literaturou, viz obr. 7.13. Zatímco u stromového GP fitness hodnota v průběhu generací klesá, tak u lineárního GP lze pozorovat velmi podobný model chování u všech běhů, tedy fluktuaci s velmi ojedinělými výkyvy.



Obrázek 7.21: Nejlepší fitness

Graf 7.21 zobrazuje hodnoty nejlepších fitness hodnot v průběhu generací. Zatímco lineárnímu GP se podařilo vyřešit úlohu v každém z deseti běhů, tak u stromového GP bylo nalezeno řešení pouze u čtyř běhů (pozn. první řešení bylo nalezeno již v nulté generaci, které bohužel nelze v grafu vidět). Ke konvergenci zde dochází pomaleji oproti lineárnímu

GP. Většina řešení byla u lineárního GP nalezena poměrně rychle a lze u nich pozorovat v prvních generacích vcelku svižný pokles fitness hodnot.

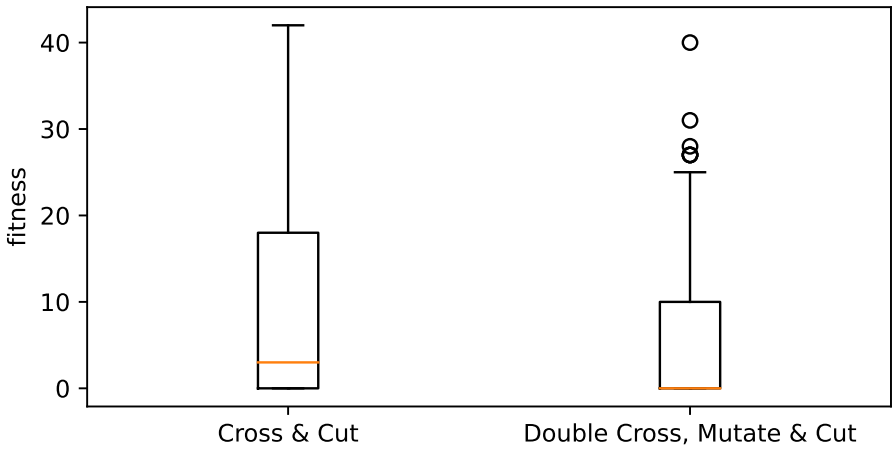
7.2.10 Srovnání variant operátorů křížení u TGP

Parametry	
Velikost programů	minimální = 4, maximální = 6
Stezka	Santa Fe (max. fitness 89)
Inicializační metoda	grow
Selekce	turnajová
Mutace	generující podstromy s P(1)
Populace	200

Tabulka 7.12: Parametry

V tomto posledním experimentu byly porovnány varianty operátoru křížení, jež různě přistupují k řešení problému *bloatu*, které s touto variantou GP souvisí. Bez nich by totiž docházelo k tvorbě řešení s neomezenou velikostí. Pro porovnání byly vybrány následující dvě varianty:

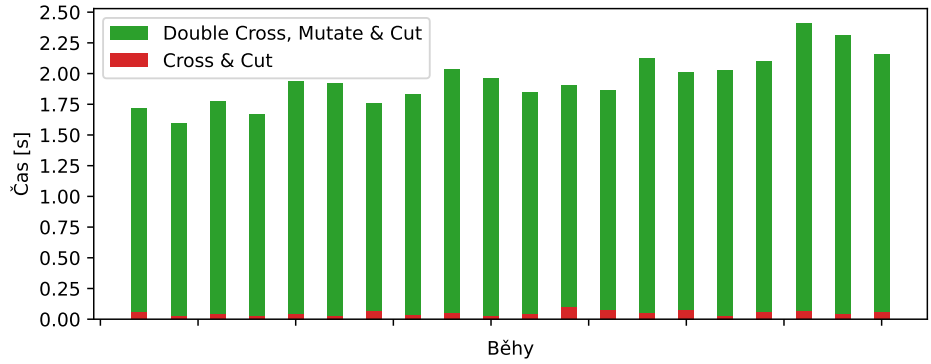
- **Cross & Cut** – Metoda po křížení zkontroluje výšku řešení a případně jej na tuto výšku zarovná terminálními symboly. Využívána v předchozích experimentech u stromového GP.
- **Double Cross, Mutate & Cut** – Provede křížení, zkontroluje výšku, v případě že výška není dodržena provede s původním rodičem opětovné křížení, analogicky opět zkontroluje výšku a pokud stále nevyhovuje velikostním omezením, tak je provedena mutace s případným zarovnáním.



Obrázek 7.22: Porovnání fitness hodnot obou variant křížení (obě s P(0.7))

Z výsledků z obr. 7.22 vyplynulo, že si druhá varianta křížení vede o mnoho lépe. Obě techniky dosahují téměř podobného mediánu, ale liší se v hranicích třetího kvartilu a taktéž

maximálních hodnotách (nepočítaje vyčnívajících hodnot). Druhá varianta je ale výpočetně mnohem složitější. Pro další porovnání bylo spuštěno dalších 20 nezávislých testů s modifikovanou pravděpodobností křížení na hodnotu 1, aby byla zaručena co největší rovnoměrnost oběma variantám. U těchto testů byla změřena výpočetní náročnost operátoru křížení u celé jedné generace, tyto hodnoty následně sečteny a zprůměrovány.



Obrázek 7.23: Porovnání výpočetní náročnosti obou variant křížení

Z grafu 7.23 se jasně potvrdilo, že ačkoliv druhá varianta jasně prokazovala lepší výsledky, tak je vzniká otázka, zdali se i za cenu výpočetní náročnosti obecně vyplatí využívat pokročilejší varianty těchto operátorů. Zrovna křížení u stromových struktur není levná záležitost. Celkový výpočetní čas u první varianty činil **0.659 s**, zatímco u druhé **35.937 s**. Při velkých populacích a generacích tento rozdíl samozřejmě narůstá. Je zde ale stále spousta faktorů, která výsledný čas můžou ovlivnit jako je například počáteční populace. V závěru plyne, že obě varianty si úspěšně poradily s problémem *bloatu*, jedna varianta vykazuje mnohem větší náročnost než druhá, ale také než pouze na těchto číselných hodnotách také závisí na zvolené instanci problému a věcech, které není možné ovlivnit (např. vygenerovanou počáteční populaci).

7.3 Další modifikace

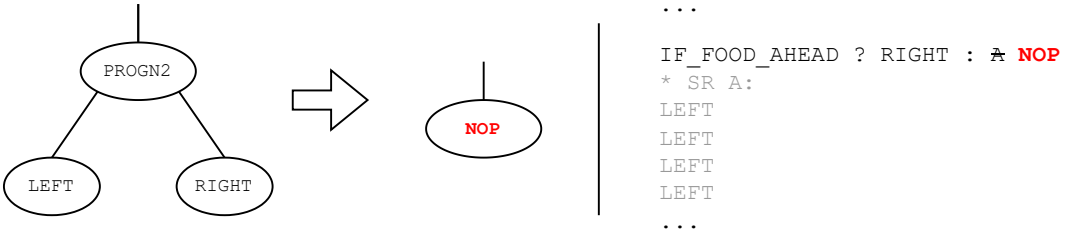
V rámci obou technik GP byla v průběhu vývoje implementována a testována spousta dalších modifikací, které už z důvodu rozumného rozsahu praktické části této práce zde nebyly zahrnuty. Mezi jedno z nich by se například dalo zařadit změna počáteční pozice mravence. Samozřejmě ne všechny modifikace vykazovaly zlepšení, některé algoritmy přímo degradovaly.

Neúspěšná vylepšení
Při experimentování ve snaze dalšího zlepšení obou variant GP byla nalezena i neúspěšná vylepšení, která se na první pohled jevila, že by mohli mít na řešení pozitivní vliv. Nebudu je tu nikterak do detailu porovnávat, ale chci se o nich alespoň krátce zmínit.

Jedno z prvních neúspěšných vylepšení bylo umožnění mravenci pohybovat se namísto čtyř v osmi směrech, tzn. pohyb i v rámci diagonál. Ačkoliv bylo zamýšleno, že by se mravenec mohl více citlivěji pohybovat, lépe se směrovat a reagovat i na potravu, která se může nacházet v diagonálním směru, tak vyústilo v celkový zhoršený pohyb. Modelový příklad, mravenec se chce otočit o 180°, namísto původních dvou instrukcí které k tomuto stačily,

tak by byly teď potřeba čtyři. Byla testována i varianta v přidání nových instrukcí, které by diagonální pohyb umožňovali a zachování těch původních, aby byla omezena četnost pohybu v osmi směrech. Ani toto zlepšení neprokázalo dostatečně dobré výsledky. Mezi další takovéto degradace algoritmu lze zařadit nový terminální symbol `JUMP`, který umožňuje mravenci obskočit jedno políčko. Opět tu je nutná velká regulace, aby se tato instrukce nevyskytovala v řešeních často. Původní zamyšlení bylo možnost prohledávat prostor různých řešeních rychlejším způsobem.

Možná další vylepšení
Mezi další vylepšení lze uvažovat například optimalizaci GP. Často se v programu vyskytuje skupina instrukcí, jež nikterak neovlivňují výslednou pozici mravence. Na obr. 7.24 jsou uvedeny příklady z každé varianty GP. Například skupina čtyř instrukci `LEFT` uvede mravence do stejného stavu jako před jejich vykonáním. Přidáním instrukce `NOP`, jež by nevykonávala žádnou operaci by mohlo dojít až ke čtyřnásobnému ušetření času, obzvláště u komplexních problému my mohlo dojít ke zlepšení a celkové optimalizaci.



Obrázek 7.24: Příklad optimalizace výpočetního času u TGP (vlevo) a LGP

Otázkou zůstává, zdali takové vylepšení aplikovat jen na konečné řešení nebo i v průběhu generací. Zde by mohlo docházet ke ztrátě genetického materiálu.

7.4 Shrnutí dosažených výsledků

V rámci experimentování bylo provedeno několik vyšších stovek a nižších tisíců testů, které měly za úkol ověřit vylepšení a následně porovnat obě varianty GP. U některých vylepšeních byl pokrok velice znatelný, u jiných zase méně. Experimentování započalo nejprve testováním vylepšení lineárního GP, kde byly získány velice příznivé výsledky, jež napomohly k získání většího počtu řešení za lepší čas. V rámci testování selekčních metod nebylo nalezeno žádného většího výkyvu, které by jasně preferoval některou z metod k řešení problému Langtonova mravence oproti jiným. Všechny tři metody – turnajová, ruleta i ranking jsou vhodné pro řešení této úlohy. Následně proběhlo rozsáhlé hledání tří parametrů, které významně ovlivňují chování tohoto programu. Těmi jsou velikost populace a pravděpodobnosti křížení a mutace. Postupným hledáním ideální hodnoty každého parametru, jeho následným zafixováním byly nalezeny ty nejlepší výsledky. Tyto experimenty byly vykonány jak pro stezku Santa Fe, tak i náhodnou stezku, kde již sice nebyly výsledky zas až tak příznivé, ale šel zde alespoň v pozdějších fází experimentování pozorovatelný rozdíl na základě různých parametrů a získaných řešení. S výsledky z literatury proběhlo krátké srovnání s náhodně vybranými výsledky obou implementovaných variant GP. Konvergenční křivky obou variant stromového GP vyznačovaly velkou podobnost, zatímco lineární se skrz svou

povahu a reprezentaci chovalo velice odlišně. Poté proběhla komplexní analýza opět výše třech zmíněných parametrů na vybrané stezce o trochu menší velikosti, kde se na základě vybraných hodnot pro všechny tyto tři parametry zkoumalo chování pro všechny různé kombinace těchto parametrů. Na základě teplotní mapy zobrazující mediány je znatelné, že nezávisle na populaci bylo dosáhnuto velmi podobných výsledků. Poté proběhlo hlubší porovnání těchto dvou variant GP, jež ukázalo, že lineární GP je se osvědčilo jako mnohem lepší nástroj řešení této úlohy. Na závěr se uskutečnilo porovnání dvou vybraných variant křížení u stromového GP, které prokázalo, že ačkoliv komplexnější varianta, která se snaží řešit *bloat*, si s úlohou poradila lépe, tak oproti jednodušší variantě trpí náročností při vyhodnocování těchto operací.

7.5 Možnosti reálné aplikace úlohy

Na základě zveřejněného článku v roce 2022 společností Rohlik.cz [19], zabývající se internetovým prodejem zboží, zejména potravin, a jejich doručením, byl představen nový systém pro vychystávání objednávek v logistickém centru. Tento systém pracuje s dvojdimenzionální mřížkou, ve které jedna buňka reprezentuje objednávku zákazníka. Po této mřížce se pohybují roboti, kteří skládají zboží do objednávek. Analogicky by se dal tento problém namapovat s menšími modifikacemi na úlohu Langtonova mravence.

Jistě se ale najde i spousta dalších aplikací o kterých ale zatím ještě není známo. Například se změnou dvojdimenzionální mřížky, jež by se nahradila uzly, které by mohly například představovat silniční křižovatky, a ty byly navzájem spojené cestami, čímž by se například dokázal problém ještě více adaptovat na logistické odvětví. Tyto cesty spojující uzly by mohly obsahovat nějaké parametry, mít nějakou prioritu, čímž by se opět zvětšil prohledávací prostor možných řešení a šlo by tak ověřit adaptace tohoto algoritmu na složitější podmínky.

Kapitola 8

Závěr

Na počátku této bakalářské práce byla nejprve představena a vysvětlena varianta stromového GP, včetně konceptů použitelných i pro ostatní varianty a v neposlední řadě taktéž stručný úvod do její historie. Následně ke konci první kapitoly proběhlo krátké srovnání s dalšími evolučními algoritmy. Rovněž analogicky bylo popsáno v následující druhé kapitole i lineární GP. Poté byly prezentovány různé aplikace GP, od těch nejzákladnějších, jež jsou lehce pochopitelné a jsou často používány pro demonstraci této techniky, tak v neposlední řadě bylo taktéž zmíněno pár komplexnějších a zajímavějších aplikací. Mimoto bylo rovněž pojednáno o výzkumu a pokroku v problematice GP jako takové. V druhé polovině bakalářské práce byla prezentována úloha Langtonova umělého mravence, jež představuje abstraktní matematický model pohybujícího se mravence po dvojdimenzionální mřížce pomocí definované sady pravidel, který následně sloužil k porovnání výše zmíněných variant GP.

Samotná implementace byla realizována téměř od základu v jazyce Python. Pouze u první varianty GP s využitím již existujícího jednoduchého programu. První grafická reprezentace úlohy byla implementována za pomocí ASCII Artu. Později s využitím knihoven matplotlib a imageio byla vytvořena pokročilejší vizualizace generující řešení ve formátu *gif*, který je snadněji prezentovatelnější. V praktické části proběhla komplexní analýza a následné statistické vyhodnocení několika různých instancí problému s odlišným nastavením. To zahrnovalo experimentování s různými parametry za dosáhnutím co nejlepších výsledků, experimenty s vylepšeními, porovnání různých variačních operátorů v rámci jednotlivých variant GP, srovnání s výsledky dostupnými v literatuře a taktéž srovnání obou variant GP.

Získané výsledky z experimentů vyústily v závěr, že lineární GP se prokázalo jako mnohem efektivnější nástroj k řešení této úlohy, než tradiční stromové GP, které je na tuto úlohu aplikováno v literatuře. Dále bylo zjištěno, že operátor mutace se jeví jako mnohem důležitější a mající pozitivnější vliv na získaná řešení než operátor křížení. Ke konci práce byly zmíněny další rozšíření či optimalizace a taktéž reálné využití na nichž bylo možné tento problém následně aplikovat.

Literatura

[1] Banzhaf, W., Koza, J., Ryan, C., Spector, L. a Jacob, C. Genetic Programming. *IEEE Intelligent Systems* [online]. 1. vyd. Los Alamitos, CA, USA: IEEE Computer Society. Květen 2000, sv. 15, č. 03, s. 74–84, [cit. 2023-04-20]. DOI: 10.1109/MIS.2000.10011. ISSN 1941-1294. Dostupné z: http://www.cs.nott.ac.uk/~pszgxk/courses/g5baim/papers/gp-001.pdf.

[2] Bhargav, N. Recursion and looping. *Baeldung* [online], 1. srpna 2022 [cit. 2023-04-05]. Dostupné z: https://www.baeldung.com/cs/recursion-looping.

[3] Bidlo, M. *Aplikované evoluční algoritmy - přednáškové materiály* [online]. 2023 [cit. 2023-01-05]. Dostupné z: https://www.vut.cz/studis/student.phtml?gm=gm_detail_predmetu&apid=244801.

[4] Brameier, M. *Linear genetic programming*. 1. vyd. New York: Springer, 2007. Genetic and evolutionary computation series. ISBN 978-0387-31029-9.

[5] Eftakhar, S. M. A., Habib, S. M. a Hashem, M. M. A. *Evolutionary Design of Digital Circuits Using Genetic Programming* [online]. Khulna, Bangladesh: [b.n.], duben 2013 [cit. 2023-04-05]. Dostupné z: https://arxiv.org/abs/1304.2467.

[6] Eiben, A. E. a Smith, J. E. *Introduction to Evolutionary Computing*. 1. vyd. Berlin, Heidelberg: Springer Berlin / Heidelberg, 2015. Natural Computing Series. ISBN 3662448734.

[7] Forrest, S., Nguyen, T., Weimer, W. a Goues, C. L. *A Genetic Programming Approach to Automated Software Repair* [online]. Montréal: [b.n.], srpen 2009 [cit. 2023-02-17]. Dostupné z: https://www.cs.cmu.edu/~clegoues/docs/legoues-gecco09.pdf.

[8] Galarnyk, M. *Understanding Boxplots* [online]. Srpen 2022 [cit. 2023-04-26]. Dostupné z: https://builtin.com/data-science/boxplot.

[9] Gandomi, A. H., Alavi, A. H. a Ryan, C. *Handbook of Genetic Programming Applications* [online]. 1st ed. 2015. Cham: Springer International Publishing AG, 2015 [cit. 2023-03-10]. ISBN 3319208829. Dostupné z: https://doi.org/10.1007/978-3-319-20883-1.

[10] Hernandez, J. G., Lalejini, A. a Dolson, E. What Can Phylogenetic Metrics Tell us About Useful Diversity in Evolutionary Algorithms? In: Banzhaf, W., Trujillo, L., Winkler, S. a Worzel, B., ed. *Genetic Programming Theory and Practice XVIII* [online]. Singapore: Springer Nature Singapore, 2022, s. 63–82 [cit.

2023-03-20]. DOI: 10.1007/978-981-16-8113-4_4. ISBN 978-981-16-8113-4. Dostupné z: https://doi.org/10.1007/978-981-16-8113-4_4.

[11] Koza, J. R. *Genetic programming: on the programming of computers by means of natural selection* [online]. 6. vyd. Boston, MA: Massachusetts Institute of Technology, 1998 [cit. 2023-05-01]. 17–190 s. ISBN 0-262-11170-5. Dostupné z: https://doc.lagout.org/science/Artificial%20Intelligence/Evolutionary%20computation/Genetic%20programming%20Complex%20adaptive%20systems%20-%20Koza%20J.R..pdf.

[12] Koza, J. R. Human-competitive results produced by genetic programming. *Genetic Programming and Evolvable Machines* [online]. Květen 2010, sv. 11, s. 251–284, revidováno 17. 4. 2010, [cit. 2023-05-01]. DOI: 10.1.1.297.6227. ISSN 1389-2576. Dostupné z: http://www.genetic-programming.com/GPEM2010article.pdf.

[13] Langdon, W. B., Poli, R., McPhee, N. F. a Koza, J. R. Genetic Programming: An Introduction and Tutorial, with a Survey of Techniques and Applications. In: Fulcher, J. a Jain, L. C., ed. *Computational Intelligence: A Compendium* [online]. 1. vyd. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, s. 927–1028 [cit. 2023-04-28]. DOI: 10.1007/978-3-540-78293-3_22. ISBN 978-3-540-78293-3. Dostupné z: https://doi.org/10.1007/978-3-540-78293-3_22.

[14] Langdon, W. B. a Qureshi, A. Genetic Programming — Computers using "Natural Selection" to generate programs. In: Dept of Computer Science, University College London. [online]. Springer US, 1998, s. 9–42 [cit. 2023-05-05]. Dostupné z: http://www0.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/surveyRN76.pdf.

[15] Miller, J. F. Cartesian Genetic Programming. In: Miller, J. F., ed. *Cartesian Genetic Programming* [online]. 1. vyd. Berlin, Heidelberg: Springer Berlin Heidelberg, Září 2011, s. 17–34 [cit. 2023-04-28]. DOI: 10.1007/978-3-642-17310-3_2. ISBN 978-3-642-17310-3. Dostupné z: https://doi.org/10.1007/978-3-642-17310-3_2.

[16] Passaggia, P.-Y., Quansah, A., Mazellier, N., Maceda, G. Y. C. a Kourta, A. Real-time feedback stall control of an airfoil at large Reynolds numbers using linear genetic programming. *Physics of Fluids* [online]. Duben 2022, sv. 34, č. 4, [cit. 2023-04-27]. DOI: 10.1063/5.0087874. ISSN 1070-6631. Dostupné z: https://doi.org/10.1063/5.0087874.

[17] Payne, J. *Benefits of python programming language* [online]. Developer, srpen 2022 [cit. 2023-05-01]. Dostupné z: https://www.developer.com/languages/python/python-benefits/.

[18] Perkis, T. Stack-Based Genetic Programming. In: *Proceedings of the 1994 IEEE World Congress on Computational Intelligence* [online]. Orlando, Florida, USA: IEEE Press, červenec 1994, sv. 1, s. 148–153 vol.1 [cit. 2023-04-28]. ISBN 0-7803-1899-4. Dostupné z: https://www.perkis.com/_site/writings/trp_stackGP.pdf.

[19] Rohlik Group. *Rohlik group speeds up delivery of purchases through Warehouse Automation* [online]. 2022 [cit. 2023-04-05]. Dostupné z: https://www.rohlik.group/rohlik-group-speeds-delivery-purchases-through-warehouse-automation.

[20] Sipper, M. *Tiny Genetic Programming in Python* [online]. GitHub, 2019 [cit. 2023-05-03]. Dostupné z: https://github.com/moshesipper/tiny_gp.

[21] WIKIPEDIA. *Genetic programming — Wikipedia, The Free Encyclopedia* [online]. 2023 [cit. 2023-04-05]. Dostupné z: http://en.wikipedia.org/w/index.php?title=Genetic%20programming&oldid=1145771317.

[22] WOODWARD, J. GA or GP? That is not the question. In: *The 2003 Congress on Evolutionary Computation, 2003. CEC '03.* [online]. IEEE, 2003, sv. 2, s. 1056–1063 Vol.2 [cit. 2023-04-10]. DOI: 10.1109/CEC.2003.1299785. ISBN 0-7803-7804-0. Dostupné z: https://ieeexplore.ieee.org/document/1299785.

Příloha A

Obsah přiloženého média

- `src/` – Zdrojové soubory s TGP a LGP.
- `exp/` – Získaná data z experimentování následně použitá pro grafové zobrazení.
- `latex/` – Zdrojové soubory k bakalářské práci.
- `README.md` – Návod ke spuštění programu.
- `gp_visualization.png` – Ukázka vizualizace řešení.
- `requirements.txt` – Knihovny a balíčky Pythonu, které je potřeba ke správnému fungování programu instalovat.
- `thesis.pdf` – Bakalářská práce v souboru PDF.
- `thesis-print.pdf` – Bakalářská práce určená k tisku.