UNIVERZITA PARDUBICE

Fakulta elektrotechniky a informatiky

ENKODÉRY V GENETICKÉM PROGRAMOVÁNÍ

Bc. Václav Hrbek

Diplomová práce

2019

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2019/2020

ZADÁNÍ DIPLOMOVÉ PRÁCE
(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Václav Hrbek**
Osobní číslo:	**I17190**
Studijní program:	**N2612 Elektrotechnika a informatika**
Studijní obor:	**Řízení procesů**
Téma práce:	**Enkodéry v genetickém programování**
Zadávající katedra:	**Katedra řízení procesů**

Zásady pro vypracování

Cílem práce bude zmapovat a otestovat možnosti enkodérů v genetickém programování. V teoretické části student popíše existující (případně navrhne vlastní) enkodéry v genetickém algoritmů, v části praktické provede experimenty a vyhodnotí jejich výsledky.

Rozsah pracovní zprávy: **40-50**
Rozsah grafických prací:
Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

HYNEK, Josef. Genetické algoritmy a genetické programování. Praha: Grada, 2008. Průvodce (Grada). ISBN 978-80-247-2695-3.
MITCHELL, Melanie. An introduction to genetic algorithms. Cambridge, Mass.: MIT Press, c1996. ISBN 978-0262133166.
MCDERMOTT, James. Why Is Auto-Encoding Difficult for Genetic Programming.
SEKANINA, Lukas, Ting HU, Nuno LOURENÇO, Hendrik RICHTER a Pablo GARCÍA-SÁNCHEZ, ed. Genetic Programming [online]. Cham: Springer International Publishing, 2019, 2019-03-27, s. 131-145 [cit. 2019-10-06]. Lecture Notes in Computer Science. DOI: 10.1007/978-3-030-16670-0_9. ISBN 978-3-030-16669-4. Dostupné z: http://link.springer.com/10.1007/978-3-030-16670-0_9

Vedoucí diplomové práce: **Ing. Jan Merta**
Katedra řízení procesů

Datum zadání diplomové práce: **7. listopadu 2019**
Termín odevzdání diplomové práce: **15. května 2020**

Univerzita Pardubice Česká republika

L.S.

Ing. Zdeněk Němec, Ph.D.
děkan

Ing. Daniel Honc, Ph.D.
vedoucí katedry

V Pardubicích dne 7. listopadu 2019

Prohlášení

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 22. 5. 2020

Bc. Václav Hrbek

Poděkování

Rád bych poděkoval Ing. Janu Mertovi za cenné rady, věcné připomínky a vstřícnost při konzultacích a vypracování diplomové práce. Rád bych také poděkoval své rodině a přátelům, kteří mě při vytváření této práce podpořili.

V Pardubicích dne 22. 5. 2020

Václav Hrbek

ANOTACE

Práce je věnována popisu kódování informace prohledávaného prostoru. S využitím genetického programování a hyperheuristických postupů bylo navrhnuto řešení pro obchodního cestujícího. Softwarová část je implementována v programovacím jazyce Python.

KLÍČOVÁ SLOVA

genetické programování, problém obchodního cestujícího, hyperheuristika.

TITLE

ENCODERS IN GENETIC PROGRAMMING

ANNOTATION

This thesis is oriented at encoding the information of the search space. By using the genetic programming and hyperheuristic methods the author proposed a solution for the traveling salesman problem. The software part is implemented in the Python programming language

KEYWORDS

Genetic programming, Traveling salesman problem, Hyperheuristic.

OBSAH

SEZNAM ZKRATEK A ZNAČEK

RRC	Random Respectful křížení
MX	Masked křížení
LGP	Lineární genetický program
CPU	Centrální procesorová jednotka
CGP	Kartézské genetické programování
TPS	Problém obchodního cestujícího
JSS	Job Shop Scheduling
VRP	Vehicle Routing Problem
TSPTW	Problém obchodního cestujícího s časovými okny

SEZNAM ILUSTRACÍ

SEZNAM TABULEK

ÚVOD

Genetické programování v posledních letech získalo na oblibě zejména z důvodu, že jde o automatickou tvorbu programu, při které není nutné znát dopředu jakékoliv informace o výsledném řešení. Jejich využití se ukázalo velice výhodné zejména při řešení kombinatorických úloh. Nicméně časová náročnost evoluce genetických programů představuje nepříjemnost, která omezuje jejich použití pro práci s velkými datovými soubory. Místo genetického programování se používají heuristické postupy, které jsou schopné najít optimální řešení pouze pro určité instance typových úloh.

Práce navrhuje využití genetického programování, ale místo hledání přímého řešení úlohy, genetický program hledá obecný heuristický postup, který povede k optimálnímu řešení nebo alespoň k dostatečně přesnému řešení v přijatelném čase. Výsledný heuristický postup by měl být snadno aplikovatelný i na jiné datové soubory. Přenositelnost heuristického postupu by značně snížila časovou náročnost podobných kombinatorických úloh, protože by se při hledání optimálního řešení mohlo vycházet z již předpřipraveného heuristického postupu. Aby přenesený heuristický postup byl efektivní, měl by být jeho výpočetní čas s rostoucím počtem dat v lineárním poměru.

Cílem této práce je navržení a implementace algoritmu genetického programování, jehož účelem bude nalezení heuristického postupu na kombinatorický problém obchodního cestujícího s časovými okny. Za použití hyperheuristických postupů bude interpretace výsledných řešení zakódována do podoby, s níž může genetický program pracovat. Genetický program bude využívat jednoduché matematické operace, goniometrické funkce a několik konstant vztahujících se k řešené úloze.

1 GENETICKÉ ALGORITMY

Genetický algoritmus je optimalizační nástroj používaný pro hledání řešení složitých problémů. Inspirací pro vznik byly evoluční procesy, které jsou běžně k vidění v přírodě u různých druhů živočichů. Jako evoluce v přírodě pracuje genetický algoritmus na postupném vylepšování množiny všech možných řešení problému. Tato množina je nazývána populací a každý jednotlivý člen této populace je nazýván jedincem. Dále je žádoucí určit vhodnost jedince napříč populací. To se provádí pomocí fitness funkce, přidělené každému jedinci v populaci. Ta uvádí kvalitu nalezeného řešení reprezentovaného jedincem napříč populací. Fitness funkce má dále rozhodující vliv na možnosti reprodukce jedince. V následujících třech krocích je uveden typický genetický algoritmus tak, jak ho popsal (Koza, 1992).

1) Vytvoření nové populace náhodných jedinců.
2) Iterativní opakování kroků *a* a *b*, dokud nejsou splněna ukončující kritéria.
   a. Ohodnocení fitness funkcí každého jedince v populaci.
   b. Vytvoření nové populace pomocí následujících genetických operátorů.
      i. Zkopírováním několika nejlepších jedinců do nové populace.
      ii. Křížením vybraných jedinců.
      iii. Zmutováním vybraných jedinců.
3) Zhodnocení jedince s nejlepší fitness funkcí.

1.1 BIOLOGICKÁ TERMINOLOGIE

Genetické algoritmy pro řešení problémů jsou inspirovány Darwinovým evolučním procesem, a tak i názvosloví při tvorbě programů je odvozeno od genetických termínů. Biologická evoluce by mohla být popsána jako metoda pro hledání optimálního jedince, s vysokou pravděpodobností pro přežití, napříč velkému množství možných vlastností jedince. V biologii jsou tyto vlastnosti určovány pomocí sekvence genů, které zajišťují, že jedinec přežije, a navíc se rozmnoží v prostředí, ve kterém žije. Evoluce může být i metoda pro hledání inovativního řešení na vyskytující se složitý problém, kterému musí generace čelit. Například dorůstající části těl některých plazů jako obrana proti predátorům a jiným přírodním nepřátelům namísto větších zubů.

Živé organismy se skládají z buněk, a každá jedna buňka obsahuje jeden nebo více chromozomů, které jsou tvořeny z řetězců DNA. Tyto řetězce jsou nositeli genetické informace biologického jedince a dají se dál dělit na jednotlivé geny. Ty mají svoji pevně danou pozici v rámci chromozomu, jenž je rozhodujícím kritériem pro vlastnosti přidělené jedinci. Například jakou bude mít barvu očí nebo tělesnou stavbu. Pozice genu v chromozomu, jenž je počítána od jeho počátku, se nazývána lokus. Konkrétní forma genu, která zaujímá určitý lokus, se nazývá alela.

Genetický materiál neboli genom, je u většiny organizmů v přírodě složen z buněk, které obsahují více než jeden chromozom. Termín genotyp odpovídá přesnému uspořádání jednotlivých genů a jejich typu. O dvou organizmech, které mají identické geny a stejnou posloupnost genů, lze říci, že mají shodný genotyp. Organizmy lze dále dělit podle uspořádání chromozomů na dva druhy. Organizmy diploidní, které mají chromozomy uspořádané po párech, nebo haploidní, jejichž chromozomy nejsou spárované. Lidský jedinec má 23 párů diploidních chromozomů v každé buňce ve svém těle. Během sexuální reprodukce dochází k výměně jednotlivých genů mezi těmito páry. To přispívá k vytvoření nového chromozomu, který je pak spojen s jiným chromozomem a dojde k vytvoření diploidního uspořádání chromozomů. Potomek je pak výsledkem této reprodukce, jenž je nazývána křížení, a anomálií vznikajících na jednotlivých genech. Tyto anomálie způsobují změnu jednotlivých genů a tím pozměňují i potomka. Z rodičů s nízkou tělesnou výškou může vzniknout vysoký potomek a naopak. Anomálie je nazývána mutací a často se vyskytne při přenášení chromozomů na potomka. Úspěšnost reprodukce se u organizmů vyčísluje jako pravděpodobnost toho, že potomek přežije a je schopen přenést genetický materiál vhodný pro přežití na další generace. Tato úspěšnost je v genetických algoritmech označována fitness funkcí.

V genetických algoritmech je termínem chromozom označován jedinec, který je jedním z kandidátů na řešení stanoveného problému. Typicky bývá chromozom zakódován pomocí sérií booleovských proměnných uložených v poli. Jednotlivé alely tohoto chromozomu jsou buď samotné proměnné, tedy jedna nebo nula, nebo i více proměnných za sebou představujících určitý prvek, například písmeno. Křížení se v genetických algoritmech sestává z výměny genetického materiálu mezi dvěma jedinci (rodiči). Mutace je nahrazována náhodným zvolením lokusu a změnou formy alely, jenž se nachází na vybrané pozici.

Většina aplikací, které využívají genetické algoritmy, pracuje s haploidními jedinci obsahujícími jeden chromozom. Jako genotyp jedince je použito pole nebo řetězec znaků s bitovými nebo jinými proměnnými. Často se v kontextu s genetickými algoritmy vyskytuje i výraz fenotyp, jenž je výsledkem působení okolního prostředí na genotyp a představuje tedy pozorovatelné vlastnosti jedince.

1.2 KÓDOVÁNÍ JEDINCE

Základním předpokladem pro úspěšné řešení problémů pomocí genetických algoritmů je zakódování ovlivnitelných vlastností do podoby chromozomu, který interpretuje jedno z možných řešení úlohy. Většina aplikací používá řetězec složený z bitových znaků, jenž je náhodně poskládán do pole o pevné délce. Nicméně v průběhu zkoumání genetických algoritmů byly objeveny úlohy, které nelze (nebo je lze jen obtížně) popsat pomocí bitových znaků. Zvolení správného kódování chromozomu se tedy ukázalo být stejně důležité jako samotné řešení problému.

Dalším důvodem pro použití různých druhů kódování je předem stanovená délka chromozomu. Omezená nebo přesně daná velikost ohraničuje prostor, ve kterém se dané řešení může nacházet. Nemožnost prohledávat i okolí, které není v definované oblasti, značně omezuje schopnost nalézat optimální řešení. Kupříkladu pro zakódování čísla 255, je potřeba 8 znaků, konkrétně 11111111. Pokud se řešení nachází v intervalu 0 – 255 je osmi–bitový chromozom postačující, ale pro hodnoty ležící nad hodnotou 255 už postačující není. Pokud prohledávaný prostor nebude omezen vůbec, nalezení preferovaného optima bude časově velice náročné a neefektivní.

1.2.1 Binární kódování

Binární kódování je nejběžnější způsob tvorby reprezentace možného řešení. Z historického hlediska ho navrhl (Holland, 1975) ve své průkopnické práci o genetických algoritmech. Dále většina doporučené teorie je uvedena pro binární kódování o pevně stanovené délce chromozomu. Výhodou tohoto kódování je, že většina heuristických doporučení při nastavování parametrů algoritmu (křížení, selekce, mutace, …) je obecně aplikována a teoreticky vysvětlena v binárních chromozomech. (Holland, 1975) obhajuje použití binárního kódování při aplikaci genetického algoritmu na dva různé zápisy

chromozomů. Oba nesly přibližně stejnou informaci, ale jeden byl zakódován pomocí malého počtu možných alel (1, 0) na dlouhém chromozomu a druhý obsahoval větší možný počet alel (čísla 1 – 10) a kratší chromozom. Dospěl k závěru, že delší chromozom má lepší šanci pro vznik více možných kombinací na adepta optima úlohy, a tím i větší pravděpodobnost na nalezení optimálního řešení.

Navzdory všem výše popsaným výhodám se binární kódování zřídka objevuje v reálných aplikacích. Pro mnoho problémů řešených pomocí genetických algoritmů je nutné pracovat s reálnými daty a binární kódování je viděno spíše sporadicky.

1.2.2 Kódování pomocí reálných hodnot

Použití reálných hodnot pro vytvoření chromozomu je výhodné z mnoha důvodů. Data získaná pomocí měření ze senzorů mohou být přímo aplikována v genetickém algoritmu. Ačkoliv z kapitoly 1.1.1 vyplývá, že genetický algoritmus s reálnými daty bude zaostávat ve výpočetním čase, tak při vhodně zvolených parametrech nemusí být rozdíl tak velký, aby na něm ve větší míře záleželo. Výkon genetického algoritmu závisí na mnoha detailech zvolených při tvorbě genetického algoritmu, a především závisí na řešeném problému.

1.3 OPERACE GENETICKÝCH ALGORITMŮ

Pro hledání nových možných řešení za pomoci genetických algoritmů je nutné provádět různé operace s chromozomy, které přispívají k různorodosti vytvořených potomků. Mezi základní operátory patří selekce, křížení a mutace, které jsou použity v řešení této práce. Vyladění těchto tří operátorů je nutné k zachování diverzity nové populace. Za zmínku stojí i další operátor, jako je například *crowding*, který byl zkoumán v (De Jong, 1975). Tento operátor vezme nově vytvořeného potomka a nahradí ho nejvíce podobným jedincem z předcházející generace. Výhoda *crowdingu* je, že zabraňuje vzniku více podobných jedinců v jedné populaci. Podobný efekt jako *crowding* má také operátor *sharing fitness function*. Po ohodnocení všech jedinců fitness funkcí, je fitness funkce každého jedince penalizována nebo odměněna na základě podobnosti zvoleného chromozomu oproti ostatním jedincům v populaci. Následkem toho jsou trestáni více podobní jedinci a odměňováni jedinci s rozdílnými geny (Goldberg a Ridchason, 1987) ukázali, že operátor *sharing fitness function* vede k diverzifikaci populace a zajištění postupného konvergování k optimu.

1.3.1 Selekce

Úkolem selekce je výběr rodičů z populace, kteří přistoupí k možnosti zformovat ze svých chromozomů potomky do následující generace. Na první pohled může být žádoucím faktorem výběr jedinců s nejlepšími fitness funkcemi, a tak stvořit potomka, který je novým optimem. Ale jak ve své práci uvádí (Mitchel, 2002), selekce musí být v rovnováze s ostatními použitými operátory. Příliš časté volení silných jedinců vede ke zredukování rozdílností mezi jedinci v populaci a příliš rychlé konvergenci k lokálnímu optimu. Pokud je populace příliš jednotvárná, má to za následek uvíznutí nalezeného řešení v lokálním optimu a následnou neschopnost genetického algoritmu se z něho dostat. Naopak výběr slabých jedinců vede k příliš pomalé konvergenci, což při rozsáhlých populacích můžou být hodiny až dny výpočetního času navíc.

Jedním ze základních druhů selekce je využití elitismu. Elitismus spočívá v zachování určitého množství nejlepších jedinců z předchozí populace. Tím je zajištěna konvergence k optimu, nicméně pokud má jedinec dobrou fitness, neznamená to, že vždy přežije do další generace. K pozměně jedince může dojít při jeho výběru k reprodukci a následném nenavrácení jedince do původní populace, nebo změně jeho chromozomu pomocí mutace. Celkově elitismus pomáhá k vytváření dostatečně silné populace a přispívá k výkonu genetického algoritmu.

Přístup turnajové selekce má oproti elitismu tu výhodu, že je možný postup jedince, který není nejsilnější. Při turnaji se spolu utkají dva náhodně zvolení jedinci z jedné populace. Poté se vygeneruje náhodné číslo x, určující pravděpodobnost výběru prvního jedince a y určující pravděpodobnost výběru druhého jedince. Čísla x a y se obvykle volí mezi nulou a jedničkou. O vítězi turnaje rozhodne podmínka $x < y$. Výsledek rovnající se hodnotě TRUE znamená, že je zvolen k reprodukci ten jedinec s lepší fitness funkcí. Pokud podmínka splněná není, zvolí se k reprodukci horší jedinec. Oba jedinci se pak vrátí do populace, kde mohou být znovu vybráni.

Ruletové kolo je jednou z nejstarších selekčních metod v genetických algoritmech. Selekce probíhá tak, že každý jedinec dostane přidělenou část na ruletovém kole, která odpovídá proporcionálně velikosti jeho fitness funkce. Poté se náhodně vygeneruje číslo na ruletovém kole a jedinec, do jehož sektoru číslo zapadne, je zvolen jako rodič do křížení (reprodukce). Pravděpodobnost toho, že bude jedinec vybrán, je dána vztahem

$$p(i) = \frac{f(i)}{x}, \tag{1.1}$$

kde $$x = \sum_{j=1}^{n} f(j). \tag{1.2}$$

Selekce pomocí ruletového kola je jednoduchá na implementaci a relativně rychlá i při velkém množství jedinců v populaci. Jejím problémem je rychlá konvergence k lokálnímu optimu a tím i malá pravděpodobnost nalezení globálního optima. Řešením je změna pravděpodobnosti výběru za pomoci oznámkování jedinců, kdy místo přidělené části na ruletovém kole dle fitness funkce je každý jedinec ohodnocen známkou. Ten s nejlepší fitness funkcí dostane známku N a nejhorší jedinec dostane 1. Jedinci nejsou tedy ohodnoceni proporcionálně, a tak se snižuje pravděpodobnost ovládnutí populace výrazně lepším jedincem. Poté je postup stejný jako v klasickém ruletovém kole. Pravděpodobnost toho, že bude jedinec vybrán, je dána vztahem 1.3. Oznámkování jedinců je robustní, zachovává různorodou populaci a konvergence k optimu je pomalejší než u přidělení prostoru na základě fitness.

Boltzmannova selekce je způsob výběru podobný simulovanému žíhání. Boltzmannova selekce kontroluje podíl, jakým jsou vybíráni vhodnější jedinci tak, že je zavedena konstanta T, která v čase klesá. Pokud je konstanta T ve vysokých hodnotách, má každý jedinec rozumnou šanci na výběr. Jakmile konstanta T začne proporcionálně klesat, začnou se volit jedinci s lepší fitness funkcí. Tím genetický algoritmus začne prohledávat relativně malý prostor a blížit se k optimálnímu řešení.

$$p(i) = \frac{N(i)}{n \times (n-1)} \tag{1.3}$$

1.3.2 Křížení

Křížení je spolu s mutací jeden z hlavních genetických operátorů, kterými jsou genetické algoritmy vybaveny. Cílem křížení je stvoření potomka za pomoci dvou jedinců ze stejné generace. V literatuře je popsáno mnoho rozdílných způsobů křížení pro hledání optimálního řešení při použití co nejmenšího počtu generací. Ale žádný způsob není univerzální nebo aplikovatelný na všechny řešené problémy při očekávání stejného výkonu. Podle (Umbark a Sheth, 2015) je křížení možné rozdělit do čtyř kategorií. Normální křížení, binární křížení, křížení reálných hodnot nebo stromové křížení programů. Stromové struktury jsou popsány v kapitole 2, jelikož při tomto druhu křížení vznikají omezení, která je nutno respektovat.

Normální křížení je nejlépe znázorněno na binárním kódování jedinců, ale umožňuje i užití dat v jiné formě. Jedno z jednoduchých a nejstarších typů je jednobodové křížení. Využívá pevné délky chromozomu, kdy se zvolí náhodný lokus v chromozomu a rodiče se rozdělí na dvě části. Lokus je pro oba rodiče stejný. První část chromozomu z jednoho rodiče je spojena s druhou částí chromozomu od druhého rodiče a vytvoří potomka. Následně se spojí i zbylé části a je stvořen druhý potomek. Rozšířením jednobodového křížení je vícebodové křížení (*k-point crossover*). U tohoto typu se nevolí jeden bod, ale k náhodných bodů. Takto rozdělení rodiče následně poskládají své části k vytvoření dvou potomků. Další variací na jednobodové křížení je *shuffle* křížení. Zde se zpřehází pořadí genů chromozomů u obou rodičů. Poté se aplikuje jednobodové křížení, které stvoří dva potomky. Po vzniku potomků je v obou rodičích uvedeno pořadí genů zpět do původních pozic. Zpřeházením genů vznikají potomci, jejichž zvolený bod křížení je náhodný. *Uniform* křížení vytváří potomky pomocí náhodného reálného čísla u, kdy se postupně prochází chromozomy obou rodičů a je vybírán jeden gen na totožné pozici. Vybraný gen je poté přiřazen potomkovi, přičemž o výběru rodiče, ze kterého gen pochází, rozhoduje právě číslo u. To je voleno obvykle mezi nulou a jedničkou. Gen, který není vybrán, je přiřazen druhému potomkovi. Dalším typem je *Average* křížení, jež pracuje s pouze číselnými hodnotami. Ze genů dvou chromozomů na stejném lokusu je spočítán průměr a výsledek je pak přiřazen do potomka. *Average* křížení generuje pouze jednoho potomka, a tudíž je nutné doplnit populaci více opakováními nebo jinými způsoby. *Discrete* křížení je velice podobné *Uniform* křížení, ovšem zde se vytváří pouze jeden potomek tak, že gen nevybraného rodiče je zahozen. *Flat* křížení vychází z *Discrete* křížení, ale na rozdíl od něj je upraveno pro reálná čísla v genech. Je vybrána podmnožina genů w chromozomů od všech rodičů. Pravidlem je, že elementy w pochází

ze stejného lokusu svých chromozomů. Tato množina je dále upravena na dvě hodnoty tak, že je zvolena minimální a maximální hodnota, která je následně přiřazena ke dvěma potomkům. Křížením řešící a rozmanitost zachovávající je *Statstic-based adaptive non-uniform* křížení neboli SAMUX. To zabraňuje převzetí potomka se stejnými hodnotami. SAMUX dále ve své práci rozvijí (Yang, 2003).

Random Respectful křížení (RRC) je binární křížení vytvářející ze dvou rodičů dva potomky. Nejprve je vytvořen vektor, který je roven délce rodičů. Do něj jsou ukládány hodnoty jedna, nula nebo null, a to dle hodnoty genů na stejném lokusu v chromozomu. Jestliže je hodnota obou genů stejná, uloží se tato hodnota do vektoru. Když jsou hodnoty rozdílné, je vložena hodnota null. Potomek je vytvořen z vektoru a pokud je hodnota null, je aplikováno *Uniform* křížení. *Masked* křížení (MX) opět využívá náhodně vygenerované masky, které jsou dlouhé jako rodiče. Po náhodném zvolení jednoho z rodičů je zkopírován první gen od prvního rodiče do prvního potomka a od druhého rodiče do druhého potomka. Potomci se poté porovnají proti masce, a tam, kde má maska hodnotu jedna, následuje výměna genů mezi potomky. *Homologous* křížení vychází z normálního vícebodového *k-point* křížení. Modifikuje ho tak, že ke křížení jsou vybrány jen řetězce určité délky nebo s předem danou podobností mezi rodiči. Zmíněná strategie se zaměřuje na výměnu řetězců specifických parametrů. *Elitist* křížení opravuje nedostatek příliš rychlé konvergence elitismu. Navíc odstraňuje selekci tím, že křížení a selekci integruje do jednoho kroku. V prvním kroku je celá populace náhodně zamíchána. V druhém se postupuje populací po párech, z kterých křížením vznikají vždy dva potomci. Nejlepší z této sady potomků postupuje do další generace.

1.3.3 Mutace

Mutace patří mezi nejpoužívanější genetické operátory. Má za úkol zabránit uvíznutí v lokální optimu tím, že náhodně mění jednotlivé geny. U chromozomu složeného z binárních genů může být mutace provedena jednoduchou záměnou alel chromozomů. Tak lze začít prohledávat i jiné optimum, než je to, ke kterému konverguje. V literatuře je význam mutace předmětem diskuzí, zvláště jeho použití u genetických algoritmů. (Mitchel, 2002) uvádí, že úspěch genetického algoritmu nezáleží na výběru samotných operátorů, ale na pečlivé rovnováze mezi nimi.

2 GENETICKÉ PROGRAMOVÁNÍ

Na rozdíl od klasických genetických algoritmů, kde je chromozom složen z parametrů interpretující řešení a algoritmus se snaží najít kombinaci těchto parametrů vedoucí k nejlepší fitness funkci, v genetickém programování je snaha pomocí evolučních metod vytvořit počítačový algoritmus, který by toto řešení našel bez pomoci člověka. Přesněji vyjádřeno, je snahou nalézt takový algoritmus, který bude nejlépe plnit zadané úkoly z množiny všech možných počítačových algoritmů a jejich kombinací. Z toho vyplývá, že není potřeba přímé účasti programátora při hledání řešení nebo jakákoliv znalost jeho struktury dopředu. Základ pro vylepšování nalezeného řešení poskytují stochastické metody spolu s výběrem vhodných jedinců na základě fitness funkce. Je třeba mít na paměti, že genetické programování je náhodný proces, a tudíž lze předpokládat výsledky pouze blížící se globálnímu optimu.

2.1 TYPY GENETICKÝCH PROGRAMŮ

Dnešní počítačové softwarové vybavení nabízí širokou škálu možností při volbě programovacího jazyka. Ačkoliv je možné vytvořit jakýkoliv typ genetického programu v téměř kterémkoliv jazyce, některé jazyky jsou vhodnější pro zvolený typ genetického programu.

2.1.1 Lineární genetický program

Termín lineární odkazuje na strukturu programu, kdy je chromozom reprezentován instrukcemi poskládanými za sebou, viz obrázek 2.1. Lineární genetický program (LGP) vykonává instrukce v takovém pořadí, v jakém jsou mu předkládány. Sekvenční řazení instrukcí má za následek absenci cyklů a jejich vykonávání je určeno pouze pozicí

Obrázek 2.1 – Lineární genetický program (Poli, 2008)

v programu. Množství instrukcí může být omezeno, takže každý jedinec v populaci má stejně danou délku, nebo je populace reprezentována jedinci s rozdílným množstvím instrukcí.

Instrukce v LGP lze reprezentovat dvěma způsoby. Jednak jako strojový kód, nebo jako kód zvoleného programovacího jazyka, jenž je typicky nízkoúrovňový programovací jazyk s vysokou efektivitou vykonávaného kódu, kdy se využívá překladač interpretující instrukce jazyka do strojového kódu. U strojového kódu jsou instrukce spustitelné přímo na CPU a odpadá potřeba převedení programovacího jazyka na kód strojový, přičemž pro práci s proměnnými je využito registrů zvolené počítačové architektury. Pokud je cílem rychlost LGP, je použití strojového kódu na reálném počítači nesporně výhodnější než za použití interpretujícího překladače.

2.1.2 Syntaktický strom

Ve své knize navrhl (Koza, 1998) reprezentovat jedince pomocí hierarchistických stromových struktur, tzv. syntaktických stromů. Syntaktický strom si lze představit jako graf s vrcholy a hranami. Stromy jsou tvořeny z funkcí, které představují vrcholy neukončující rozvoj stromu a terminálů, jež jsou zpravidla posledními vrcholy završující jednotlivé větve. Mezi výhody stromových struktur patří snadná reprezentace nalezeného řešení a evoluce pomocí genetických operátorů, kdy lze jakýkoliv vrchol nahradit jiným vrcholem nebo celou novou větví. Dále je možné, aby byl celý strom vložen do jiné stromové struktury jako samostatná větev.

Množina funkcí může obsahovat funkce typu:

- matematické funkce (*sin, cos, +, /, ...*),
- booleovské operace (*AND, OR, ...*),
- iterační funkce (*For, Do, ...*),
- podmíněné funkce (*If, While, ...*),
- speciálně definované funkce (*rand, ...*).

Z předcházejícího výčtu je patrná potřeba definovat počet terminálů vázajících se na určitou funkci. Například funkce cosinus vyžaduje pouze jeden argument, kdežto funkce AND už argumenty dva. Zde se zavádí pojem *arita funkce*, který je uveden v (Koza, 1998), kdy každá funkce má takovou aritu, kolik je třeba na ni navázat vstupních argumentů. Množina terminálů je tvořena vstupními proměnnými, konstantami nebo funkcemi, které nevyžadují žádný vstupní argument. Speciálním případem terminálu je generování náhodného čísla. Náhodné číslo může způsobit odlišné chování programu pro každý jeho běh.

Předpoklad uzavřenosti (closure)

Pro bezproblémový chod genetického programu je nezbytné, aby množina funkcí splňovala určité předpoklady. Oba jsou detailně popsány v (Koza, 1998). Prvním je předpoklad uzavřenosti (*Closure*), kdy při formování jedinců a následných genetických operacích je nutné, aby se jakákoliv větev dala zaměnit za libovolný vrchol vybraného stromu. Uzavřeností se rozumí, že všechny funkce mohou přijmout jako vstupní argument výstup z jakékoliv jiné funkce nebo jakýkoliv terminál. Příkladem může být funkce *OR*, na jejíž vstup jsou přivedena reálná čísla. Takováto kombinace způsobí předčasné ukončení genetického algoritmu generujícího chybu. Řešením je automatické přetypování výstupu na požadovaný vstup, nicméně konverze výstupních argumentů může způsobit tendenci ke změně prohledávaného prostoru, jak uvádí (Poli, 2008).

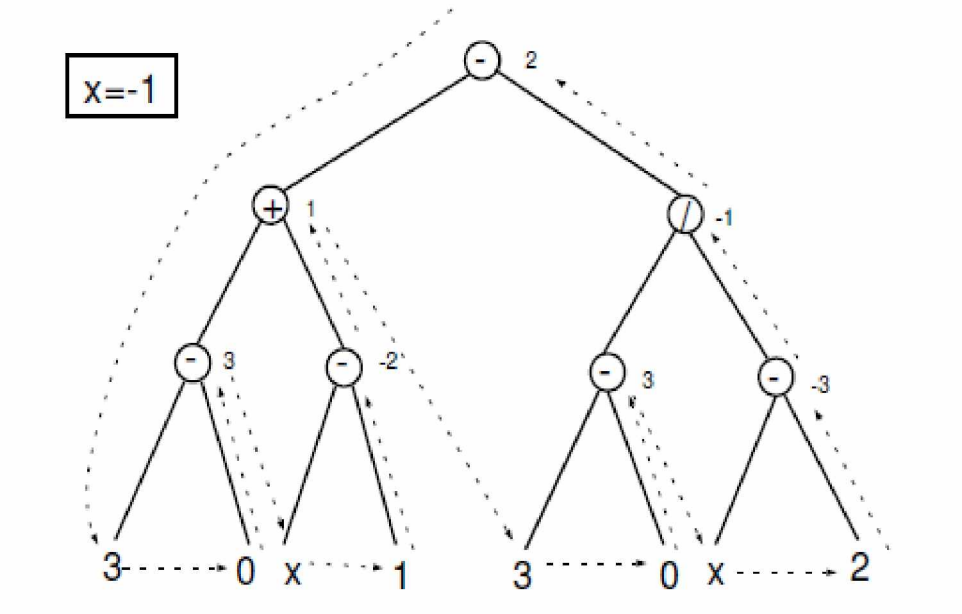
Předpoklad uzavřenosti není omezen pouze na rozdílné datové typy, ale i na nedovolené vstupní argumenty. Například dělení nulou nebo příkaz zaslaný robotovi pro postup do zakázaného směru jsou taktéž operace generující chybová hlášení. Zábranou pro vyvolání nežádoucího stavu je ověřování vstupních argumentů a vyvolání výjimky při dostavení se nežádoucí situace. Při nemožnosti kompletního uzavření genetického programu je penalizován jedinec, u kterého nastane nežádoucí stav. Pokud bude v generaci velké procento jedinců obsahujících chyby, je pro selekci obtížné nalézt vhodné jedince pro křížení.

Předpoklad postačitelnosti

Druhým předpokladem navrhovaným (Koza, 1998) je postačitelnost. Postačitelností se rozumí možnost vyjádřit požadované řešení pomocí předdefinovaných množin funkcí a terminálů. Garantovat postačitelnost lze jen za podmínek, že řešení je možné vytvořit kombinací elementů obou množin. Příkladem nesplnění předpokladu postačitelnosti je pokus o nalezení transcendentní funkce pomocí množiny $\{+, -, \times, \div, x, 0, 1, 2\}$. Například funkci $\exp(x)$ nelze vyjádřit podílem ani žádnou jinou kombinací zmíněných elementů.

Interpretace stromu

Zvolená interpretace programu syntaktického stromu rozhoduje o pořadí vykonávání funkcí. Interpretace musí garantovat, že jednotlivé funkce nejsou vykonávány dříve, než jsou známy jejich vstupní argumenty. Výpočet je obvykle konán od kořenového vrcholu stromu, kdy se postupuje přes funkce, dokud není nalezen element z množiny terminálů. Následně je



Obrázek 2.2 – Interpretace syntaktického stromu (Poli, 2008)

vypočítána nejspodnější funkce, která slouží jako vstupní argument pro další funkci. Výše popsané prohledávání se nazývá *depth-first*. Viz obrázek 2.2.

2.1.3 Kartézské GP

Kartézské genetické programování (CGP) bylo v (Miller, 1999) použito k návrhu digitálního obvodu. Reprezentace genů je ve formě pevně daného grafu, kdy geny jsou reálná čísla reprezentující umístění jednotlivých funkcí v grafu. Genotyp je pevné délky a pořadí výkonu jednotlivých funkcí interpretuje samotné řešení a je označeno jako fenotyp. Při interpretaci řešení mohou být určité funkce vynechány. Takové funkce se v literatuře označují jako *non-coding*. Fenotyp CGP má proměnlivou délku v intervalu 0 až maximální počet funkcí definovaných genotypem. V případě, že má fenotyp délku 0, jsou všechny vstupy programu převedeny přímo na výstup. Naopak, pokud se délka fenotypu rovná počtu všech

definovaných funkcí v genotypu, v řešení jsou zapojeny všechny funkce. Mapování řešení pomocí genotypu a fenotypu je charakteristickým znakem pro CGP.

2.2 GENETICKÉ OPERÁTORY

Genetické operátory jsou základem evolučních algoritmů a od jejich úspěšné implementace se odvíjí výsledek nalezeného řešení. V této kapitole jsou popsány metody pro genetické programování odlišující se od genetických algoritmů, jež jsou popsány v kapitole 1. Budou použity jen ty metody, zaměřující se na stromovou strukturu genetického programování, která je nejvíce rozšířená a používaná.

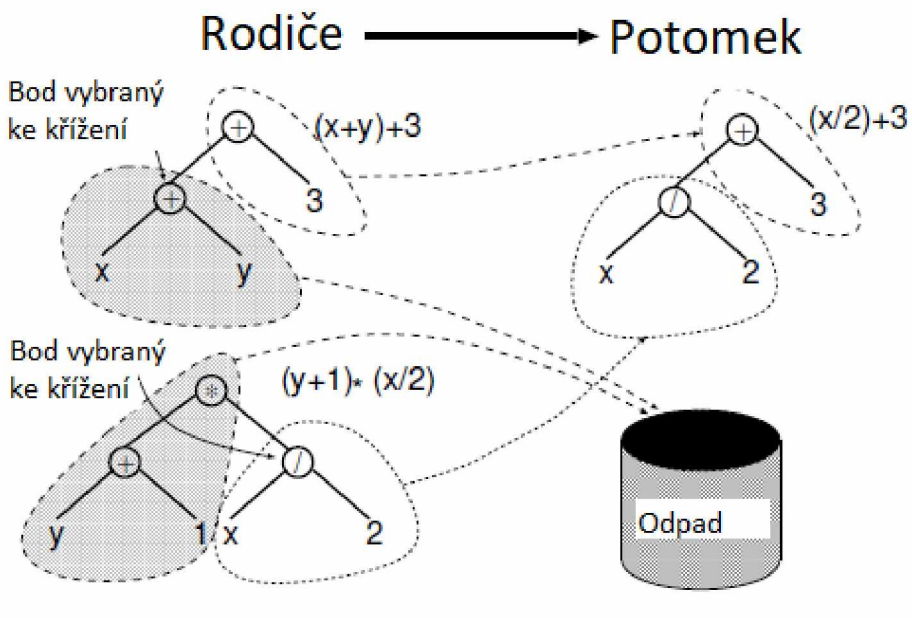
2.2.1 Inicializace počáteční populace

Historicky nejužívanější přístup pro konstrukci první populace je její náhodné generování. Základní jsou dvě náhodné metody *full* a *grow*. V obou těchto metodách jsou úvodní jedinci generováni takovým způsobem, aby nepřekročili maximální hloubku stromu definovanou uživatelem. Výraz hloubka vrcholu je definována jako počet hran mezi kořenovým vrcholem a zvoleným vrcholem v grafu. Hloubkou stromu se rozumí nejvzdálenější vrchol od kořenového vrcholu. Metoda *full* tvoří úplný strom tak, že všechny větve mají totožnou hloubku posledního vrcholu. Jednotlivé vrcholy jsou vybírány z množiny funkcí až do stavu dosažení hloubky $n-1$, kde n je hloubka stromu. Poslední vrcholy se osadí elementem z množiny terminálů. Ačkoliv metoda *full* při inicializaci populace generuje stromy, kde větve mají stejnou hloubku, není samozřejmostí shodný počet vrcholů nebo shodný tvar stromu. Takové případy nastávají pouze ve stromech, kde všechny funkce mají stejnou aritu. Naopak metoda *grow* dovoluje vznik stromu o náhodné hloubce nebo tvaru. Zde jsou prvky stromu vybírány náhodně z obou dostupných množin. Při dosažení maximální hloubky je postup stejný jako u metody *full*.

Jelikož ani jedna výše popsaná metoda neposkytuje široké množství možných tvarů, navrhl (Koza, 1998) kombinaci nazvanou *ramped half-and-half*. Polovina populace je tvořena metodou *full* a druhá polovina pomocí metody *grow*. Navíc přidal různé rozsahy pro hloubky stromu, aby byla zajištěna dostatečná rozmanitost populace.

Počáteční populace nemusí být složena pouze z náhodných jedinců. Je možné pomoci genetickému algoritmu při konvergenci k optimu dosazením jedince, který není nejlepším

řešením, nicméně má k tomuto řešení blízko. Informace v podobě dosazeného jedince je vhodný startovní bod pro začátek evoluce. Tento jedinec může vzniknout dřívějším během algoritmu, nebo konstrukcí řešitelem. Při nevhodném použití dosazení mohou nastat komplikace, kdy je jedinec, který je výrazně lepší než náhodné generované stromy, schopen



Obrázek 2.3 – Příklad křížení syntaktického stromu (Poli, 2008)

v prvních několika generacích ovládnout celou populaci a snížit její rozmanitost.

Při generování první populace musí být zásadně dodržen předpoklad uzavřenosti. Nesmí nastat nežádoucí stav, kdy není k funkci přiveden dostatek vstupních argumentů na její správné vykonání. A naopak při inicializaci nesmí vzniknout žádný samostatný strom, který by logicky nenavazoval na kořenový vrchol. Vzniku těchto stavů se dá předejít užitím vztahu

$$T = (a - F) + 1, \tag{2.1}$$

kde a – sumace arit funkcí,

F – počet funkcí,

T – počet terminálů,

udávající počet terminálů figurujících ve výsledném jedinci.

2.2.2 Křížení

Křížení se v genetickém programování u stromových struktur používá ve formě výměny větví mezi stromy (podstromové křížení). Jsou vybráni dva rodiče a v každém rodiči je vybrán vrchol ke křížení. Poté se z jednoho rodiče odstraní všechny vrcholy mezi bodem křížení a kořenovým vrcholem, včetně kořenového vrcholu. Zbylá větev je určena ke křížení. Z druhého rodiče se naopak odstraní křížící bod a všechny následující vrcholy. Potomek je zformován tak, že na místo odstraněné větve je přiřazen zbylý strom z prvního rodiče. Postup jednoduchého křížení je znázorněn na obrázku 2.3. Žádoucí funkcí je, aby rodiče mohli být vybráni vícekrát při tvorbě jedné generace, a to v nezměněném stavu. Různé části nejlepšího jedince lze tak rozdat mezi méně vhodné jedince, a tím získat populaci s celkově lepší funkcí. Ze dvou rodičů lze získat dva možné potomky pomocí výměny větve v křížících bodech. (Koza, 1998) navrhuje pro výběr křížícího bodu v 90 % vrchol z množiny funkcí a z 10 % vrchol z množiny terminálů, a tak optimalizovat výměnu celých větví namísto pouhých terminálů.

Podobně jako v genetických algoritmech existuje celá řada různých způsobů pro křížení dvou jedinců stromové struktury. V *Uniform* křížení, uvedeno v (Langdon, 1998), je potomek tvořen výběrem dvou podobných rodičů, kde se porovnají stejné části a se zvolenou pravděpodobností je vybrán vrchol z jednoho nebo druhého rodiče. Dalším příkladem je *Size fair* křížení uvedené v (Langdon, 2000). Zde je první křížící bod vybrán náhodně jako při klasickém křížení a poté je spočtena hloubka odstraněné větve, která omezuje výběr křížícího bodu v druhém rodiči a zamezuje dosazení nepřiměřeně velké větve.

2.2.3 Mutace

Nejběžnější používaná forma mutace je vytvoření náhodného stromu a tím nahrazení zvolené větve v mutovaném jedinci (podstromová mutace). Nebezpečí spočívá v nahrazení majoritní části vybraného stromu náhodným řešením. Lze tak přijít o vhodné řešení, pokud je vybrán jedinec s dobrou fitness funkcí. Jako obměna této formy je vhodné generování náhodného stromu s předem stanovenou hloubkou.

Vhodnou formou je také bodová mutace (*point mutation*), která je hrubým ekvivalentem bitové mutace z kapitoly 1.1.3. Postupem mutování je záměna vybraného vrcholu za jiný ze stejné množiny a s totožnou aritou. Pokud neexistuje v množině žádná

vhodná funkce za nahrazovanou funkci, vrchol zůstane nezměněn. Nicméně lze zmutovat jiné vrcholy v totožném stromu, které už vhodné ekvivalenty mají.

2.3 OMEZUJÍCÍ PODMÍNKY

Úlohy v reálném světě jsou většinou zatíženy omezujícími podmínkami, které zamezují nalezení optimálního a použitelného řešení. (Hynek, 2008) ve své práci popisuje několik metod pro práci s nimi. Dále úlohy s omezujícími podmínkami klasifikuje dle prostoru, v němž se snaží najít řešení. Prostor je rozdělen na podmnožinu přístupných řešení a podmnožinu nepřístupných řešení.

Jedním ze způsobů, jak pracovat s omezujícími podmínkami, je podle (Hynek, 2008) penalizační metoda. Tato metoda pracuje nad celým prohledávaným prostorem a nepřístupná řešení jsou penalizována předem definovanou funkcí. (Hynek, 2008) poznamenává, že úspěch penalizační metody je ovlivněn volbou penalizační funkce vzhledem k řešenému problému. Příliš mírná penalizace nevede ke konvergenci na množině přístupných řešení, protože některá individua mohou profitovat z porušení podmínek. Naopak příliš vysoká míra penalizace vede k uvíznutí v lokálních extrémech, jelikož nebude možné prozkoumávat oblast ležící za podmnožinou nepřístupných řešení.

Dalším nástrojem pro práci s omezujícími podmínkami, o nichž pojednává (Hynek, 2008), jsou opravné algoritmy a speciální operátory. Ty zajišťují, že genetický algoritmus pracuje pouze s podmnožinou přístupných řešení. Opravný algoritmus je metoda, která upravuje jednotlivce patřící do podmnožiny nepřístupných řešení tak, aby jedinec byl překlasifikován do podmnožiny přístupných řešení. Při aplikaci speciálních operátorů vzniknou z nepřístupných řešení přístupná. Speciální jsou proto, že při jejich konstrukci se využívá znalostí řešeného problému. Znalosti pomáhají jak při tvorbě potomků, tak se dají použít i při tvorbě inicializační populace.

3 ROZŠIŘUJÍCÍ TEORIE

3.1 ENKODÉR

Enkodéry nejsou v této práci myšleny jako fyzická zařízení poskytující informaci o poloze sledovaného tělesa, ale jako softwarový program na konverzi informace o řešeném problému do žádaného formátu. Informací je rozuměna reprezentace objektu či činnosti v počítačově zpracovatelném formátu. Příkladem může být obrázek složený z pouze šedých pixelů, kdy každý stupeň šedé dosahuje hodnot 0 – 255. 0 pro bílou a 255 pro černou barvu. Reprezentaci takového obrázku lze převést na řadu čísel obsahujících tolik hodnot, kolik je pixelů v obrázku. Elementy v řadě lze vyjádřit jako jasovou funkci $F(x, y)$, kde x a y jsou hodnoty umístění jednotlivých pixelů v kartézské soustavě obrázku a výsledek funkce je intenzita šedé barvy v pixelu. Žádaný formát je jakákoliv interpretace kódované informace v jiné podobě, než je ta originální.

Jedním z hlavních použití enkodéru je komprese velikosti počítačových souborů. (Ebin, 2016) použil genetické programování pro kompresi a dekompresi černobílé ilustrace. Algoritmus začíná načtením obrázku a jeho rozdělením na části dle bitové velikosti. Genetické programování je aplikováno na jednotlivé části, a výsledek je ukládán do externího souboru. Výsledkem programu po aplikaci na každou zónu je symbolická regrese popisující jasovou funkci šedé barvy pixelů. Fitness funkce nejlepších jedinců je reprezentována jako průměrná odchylka čtverců regresní křivky od pravdivých hodnot. Výsledkem je změna velikosti originálního souboru před kompresí, po kompresi a porovnání kvality po následné dekompresi.

(McDermont, 2019) ve své práci zmiňuje pojem autoenkodér. To je informace, která projde „hrdlem lahve“ a navrátí se do původní podoby. Takto definovaný autoenkodér lze sestrojit pomocí neuronové sítě. Hrdlo lahve zde tvoří vrstva neuronů mezi vstupní a výstupní vrstvou, přičemž tato prostřední vrstva má méně neuronů než ostatní vrstvy. Při zmíněném použití neuronových sítí figurují genetické algoritmy spíše jako pomůcka pro nastavení optimálních parametrů neuronové sítě. (McDermont, 2019) navrhl pro hrdlo lahve použít dva lineární genetické programy, které spolu komunikují pouze prostřednictvím výstupu jednoho, který je následně směřován na vstup druhého. První lineární program použitý v (McDermont, 2019) lze klasifikovat jako enkodér a druhý jako dekodér. Enkodér provádí kompresi vstupní informace a může se použít jako automaticky naučený kompresní program,

který lze přenášet mezi zařízeními nezávisle dekodéru. Analogicky je možné použít dekodér pro dekompresi.

3.2 PROBLÉM OBCHODNÍHO CESTUJÍCÍHO

Problém obchodního cestujícího (TSP) je jednou z nejznámějších optimalizačních otázek, na kterou neexistuje žádná deterministická metoda pro obecné řešení. Motivací za vznikem TSP je nalezení řešení na úlohu, které čelí podomní prodejce navštěvující množství zákazníků nacházejících se v odlišných městech tak, aby prodejce urazil co nejkratší cestu, a tak dosáhl zisku v co nejkratším čase. Více obecná a abstraktní definice TSP může být jako zadání grafu s cestami ohodnocenými váhami, kde úkolem najít cestu s nejmenší součtem vah, při čemž je každý uzel v grafu navštíven pouze jednou. Někdy je v literatuře referováno o TSP a jeho alternativách, jako o Job Shop Sheduling (JSS) nebo Vehicle Routing Problem (VRP).

Nejpřirozenější využití má TSP v logistickém prostředí, nicméně jeho aplikovatelnost na jiné úlohy podobného typu z něj dělají ideální příklad pro vysvětlení teorie v různých průmyslových oblastech. Příkladem je plánování trasy CNC stroje s úkolem vyvrtat požadované množství otvorů v obráběném materiálu. Zde se procestovaná vzdálenost přepočítává na čas potřebný k obrábění a tím i na náklady potřebné na výrobu obrobku. Města v tomto případě představují vrtané otvory.

3.2.1 TSPTW

Při cestování obchodního cestujícího se v ideálním případě obchodník nemusí zabývat žádnými omezeními. V reálné aplikaci této úlohy se vyskytují omezující podmínky, které mají dopad na celkovou charakteristiku cesty a tím i na uraženou vzdálenost. Jedním takovým omezením může být definování časových intervalů, ve kterých lze dorazit k zákazníkovi. Takto definovaná úloha se nazývá *Traveling Salesman Problem with Time Windows* (TSPTW).

Zavedení časových oken sebou následně přináší omezení prohledávaného prostoru, a jak uvádí (Hynek, 2008), jeho rozdělení na podmnožinu přípustných řešení a podmnožinu nepřípustných řešení. Aby nedocházelo při existenci více omezujících podmínek k rozdělení prohledávaného prostoru na více podmnožin, jsou obvykle tyto podmínky interpretovány

v konjunktivním vztahu. (Hynek, 2008) dále poukazuje na to, že i když zavedením podmnožiny přípustných řešení se omezí prostor, ve kterém se nachází nejlepší řešení, je tento prostor rozdělen mnohem méně vhodným způsobem. Následkem je, že i když u obchodního cestujícího je řešením hledání volného extrému, u TSPTW je to už extrém vázaný.

V literatuře je značné množství omezujících podmínek, které (Nguyen, 2019) ve své práci navrhuje klasifikovat do několika kategorií dle množství podmínek a vlastností k nim vztažených. Pro účel této práce postačí rozdělení omezujících podmínek na statické a dynamické. Dynamické podmínky jsou závislé v čase a statické jsou naopak časově nezávislé. V návaznosti na TSPWT to znamená, že rozhodnutí dle dynamického pravidla by záviselo na nejkratším cestovním čase a rozhodnutí podle statického pravidla by mělo za úkol snížit počet nedodržených časových oken.

3.3 ASYMPTOTICKÁ SLOŽITOST PROBLÉMU

Každému algoritmu trvá specifický čas, než nalezne požadované řešení. Ovšem čím je problém náročnější, zvyšuje se i složitost užitého algoritmu, Složitost se dá rozdělit do tříd. Platí pravidlo, že algoritmy z vyšších tříd jsou prostorově nebo časově náročnější na výpočet. Asymptotická složitost je značena $O(N)$, kde N je počet dat vstupujících do problému. Jednotlivé třídy jsou uvedeny v tabulce 3.1, kde k je konstanta. (Adamchik, 2009)

Tabulka 3.1 – Asymptotická složitost

Třída	Matematický zápis	Popis složitosti
1	$O(1)$	Konstantní
2	$O(logN)$	Logaritmická
3	$O(N)$	Lineární
4	$O(N \cdot logN)$	Lineárně logaritmická
5	$O(N^2)$	Kvadratická
6	$O(N^3)$	Kubická
7	$O(N^k)$	Polynomiální
8	$O(k^N)$	Exponenciální
9	$O(N!)$	Faktoriálová

3.3 HEURISTIKY, METAHEURISTIKY A HYPERHEURISTIKY

3.3.1 Heuristiky

Heuristiky jsou přibližné optimalizační metody, které negarantují nalezení správného optimálního výsledku, ale pouze řešení, které se k tomu optimálnímu blíží. Heuristika je založena na přibližném odhadu či na intuici a zkušenosti řešitele, přičemž může být uplatněn iterativní přístup pro nalezení řešení, kdy je na základě změn u předchozích výsledků postupně vylepšováno konečné řešení. Tyto metody se uplatňují v mnoha vědních disciplínách, jako jsou umělá inteligence, logistika či bioinformatika. Úlohy, které heuristiky řeší, jsou často kombinatorického charakteru, a může to být hledání nejkratší cesty v grafu, plánování rozdělení přidělených zdrojů nebo složení 3D struktury proteinu.

Na příkladu obchodního cestujícího, kdy s rostoucí velikostí prohledávaného prostoru roste i výpočetní čas, lze jen těžce nalézt všechny možné kombinace cest a vybrat tu nejlepší. Heuristikou aplikovatelnou na obchodního cestujícího může být hladový algoritmus. Hladový algoritmus nejdříve vypočítá vzdálenosti z aktuálního města do všech ostatních měst a vybere to město, které je nejblíže aktuálnímu. Poté cestuje do vybraného města a znovu hledá nejbližší neprocestované město. Výsledná trasa je lepší než většina možných ostatních tras, ale nemusí to být trasa nejkratší.

3.3.2 Metaheuristiky

Podle (Stühle, 1999) lze metaheuristiky definovat jako optimalizační strategie, které vedou a modifikují podřízené heuristické postupy tak, aby bylo dosaženo lepšího řešení. Hlavním cílem metaheuristik je odstranění nevýhod iterativních postupů a především zajištění možnosti uniknutí z lokálního optima. Zmíněné cíle jsou dosaženy za pomoci metod, které dovolí provádět operace vedoucí k dočasnému zhoršení nalezeného řešení nebo vytvořením nového počátečního řešení užitím vhodnějších metod, než je náhodné generování. Mezi metaheuristiky se dá zařadit několik optimalizačních algoritmů, kterými jsou například Optimalizace kolonií mravenců, Simulované žíhání, Evoluční algoritmy nebo metoda *Tabu search*.

3.3.3 Hyperheuristiky

Hyperheuristiky jsou metody, které při optimalizaci nehledají řešení v prostoru, kde se nachází, ale prohledávají prostor s heuristikami nebo metaheuristikami, které vedou k nejlepšímu řešení. (Maashi, 2017) uvádí jako motivaci pro zavedení hyperheuristiky zobecnění optimalizačních metod a automatické přizpůsobení optimalizačních algoritmů podle jejich silných a slabých stránek. Hyperheuristiky vznikají složením několika heuristických nebo metaheuristických přístupů. Příkladem může být problém obchodního cestujícího, kdy je nejlepší výsledek nalezen nejdříve po optimalizaci pomocí evolučních algoritmů a následném doladění výsledků užitím simulovaného žíhání.

(Hynek, 2008) uvádí pojem dekodér, při jehož použití dochází k zakódování řešení do podoby informace, na jejímž základě je možné vhodné řešení sestrojit. Dekodér je uveden jako alternativa k práci s omezujícími podmínkami. Prostor, který je rozdělen na podmnožinu přístupných řešení a podmnožinu nepřístupných řešení, je zakódován do podoby obsahující pouze množinu přístupných řešení, na kterých lze aplikovat genetický program. Dále (Hynek, 2008) zdůrazňuje, že i když je použití dekodéru relativně jednoduché, tak při jeho návrhu je „nutné využít podrobné znalosti o konkrétním problému a příslušných omezujících podmínkách".

Dle (Burke, 2010) se hyperheuristický přístup dá klasifikovat do dvou tříd: *selection heuristic* a *heuristic generation*. Metody *Selection heuristic* vybírají a skládají heuristiku už z množiny existujících heuristik nazývaných *Low-Level heuristic*. Do takové množiny patří například matematické operátory. Třída *Heuristic generation* generuje novou heuristickou funkci z komponent už existujících heuristických metod. Obě výše zmíněné třídy se dají dělit dále do dvou subkategorií: *Construction heuristic* a *Perturbation heuristic*.

Construction heuristic je přístup, který inkrementálně buduje požadovanou heuristiku. Začíná s prázdnou množinou řešení a cílem je vybrat a použít takové konstrukční heuristiky, které postupně vybudují řešení. Složení množiny konstrukčních heuristik obecně záleží na specifikacích řešeného problému. Konstrukční proces končí samovolně, a to za podmínky, že je nalezena taková heuristika, která řeší zadaný problém.

Perturbation heuristic je přístup, kde začínající množina heuristik je už vytvořena, a to buď z náhodně poskládaných heuristik nebo z jednoduchých fungujících heuristik. Postupným iterativním vylepšováním zkouší vylepšit nalezené řešení. Tento proces se opakuje, dokud

nejsou splněny ukončující podmínky. Na rozdíl od *Construction heuristic* zde proces nekončí samovolně, ale podmínky ukončení je nutné definovat před začátkem programu.

4 ŘEŠENÍ

Cílem praktické části této práce je vytvoření programu řešící úlohu obchodního cestujícího s časovými okny. Hlavním problémem je zakódování jedince a specifikace prohledávaného prostoru. Po prvotních úvahách byly zváženy vlastnosti, jakými je obchodní cestující vybaven. Pohyb mezi městy je vždy rovnoměrný a přímočarý, takže dynamické vlastnosti vzniklé přesunem nejsou brány v potaz. Pohyb je vykonáván ve dvoudimenzionálním prostoru a začíná v prvním městě a končí v posledním městě setříděného seznamu. Jsou tedy známy informace o všech městech, obchodním cestujícím a prostoru, v němž se pohybuje.

Určení pořadí měst je možné na základě skóre, které se přiřadí každému městu. Zavedení skóre použili (Aparnaa a Kousalya, 2014) k ohodnocení úkolů zpracovávaných procesorem. Skóre rozhoduje o tom, jaké zdroje budou danému úkolu přiděleny, a tedy v jakém pořadí se budou úkoly vykonávat. Zmíněná práce navrhuje použití algoritmu, který minimalizoval čas nečinnosti procesoru a optimalizoval využití paměti. Protože jsou úlohy TSP a (Aparnaa a Kousalya, 2014) použitý *Job Shop Scheduling* podobné, je navrženo podobné řešení i v této práci.

Města jsou dále vybavena ve zvoleném intervalu náhodně rozmístěnými časovými okny, což má za následek, vedle uražené vzdálenosti, i vliv omezení při ohodnocování jedinců fitness funkcí. Časová okna u problému *Vehicle Routing Problem* uvádí (Barke, 2018). Na rozdíl od této práce Barke použil vícekriteriální genetické programování a pevně daná časová okna. To znamená, že pokud vozidlo přijelo na místo před začátkem časového okna, čekalo na jeho začátek.

Navrhované cíle a omezení této práce si vzájemně odporují a je nutné vhodným nastavením vybalancovat výsledek tak, aby byly dle zadání uspokojeny oba parametry. Je třeba mít na paměti, že algoritmy genetického programování mají tendenci konvergovat pouze k jednomu cíli. Pro dosažení více výsledků z genetického programu je v literatuře popsáno několik metod. (Nguyen, 2019) uvádí například koevoluční algoritmus, ve kterém jsou paralelně rozvíjeny dvě populace s různými cíli a jejich fitness funkce se vzájemně ovlivňují. Dalším příkladem je vícestupňová optimalizace, kde je inicializační populace vytvořena složením dvou již vygenerovaných populací, z nichž se každá zaměřuje na odlišný cíl. Dále je možné použít vícekriteriální (*multi-objective)* genetické programování, které hledá řešení uspokojující všechny stanovené cíle a omezující podmínky. Celá řada *vícekriteriálních*

genetických algoritmů je podrobněji popsána v (Konak, 2006). V této práci je zvolena metoda skalárního součinu fitness funkce a penalizační složky, jež je vyjádřena následujícím vztahem

$$f = d + \sum_{i=1}^{n} p_i, \tag{4.1}$$

kde f – fitness funkce,

n – počet měst,

p – penalizační složka při nedodržení časového okna,

d – procestovaná cena.

Úloha obchodního cestujícího by se dala řešit pomocí obyčejných genetických algoritmů, ve kterých by byly chromozomy seznamem setříděných měst a genetické operátory by se staraly o kombinování takovým způsobem, aby byly uspokojeny zvolené cíle. Nicméně se stoupajícím množstvím vstupních dat se zvedají časové a prostorové nároky polynomiálně.

Ze zmíněných vlastností, kterými jsou například časová náročnost nebo omezující podmínky rozhodující o dalším postupu, bylo usouzeno, že klasický genetický algoritmus nevyhovuje požadavkům na zpracování. Známé heuristické postupy také nesplňují požadavky a byl tedy navrhnut hyperheuristický postup s cílem dekódovat požadované řešení do takové podoby tak, aby byly splněny následující požadavky:

1. Výsledná heuristika je přenositelná.
2. Heuristika je schopna splnit požadované cíle.
3. Časová a prostorová výpočetní náročnost bude se stoupající obtížností řešeného problému v maximálně lineárním vztahu.

Jak uvádí (Nguyen, 2019), hyperheuristický postup v kombinaci s genetickým programováním se v posledních letech stal velice populární z důvodu schopnosti vyprodukovat značný objem programových struktur, jež souvisí s různými typy pravidel a heuristik. Další výhodou, která dělá z hyperheuristického postupu a genetického programování ideální kombinaci, je schopnost jednoduché interpretace výsledného algoritmu, což je vlastnost, která umožňuje aplikaci heuristiky v reálných aplikacích.

Základ fitness funkce je vypočítáván z uražené vzdálenosti obchodního cestujícího. O pořadí, v jakém budou města navštívena, rozhoduje skóre, které je přiřazeno každému městu a je odvozeno z heuristické funkce. Zavedením skóre umožňuje dekódovat

prohledávaný prostor pomocí *Low-level* heuristik a následně vytvořit jejich kombinaci, která je schopná nalézt optimální řešení.

Aby bylo možné provádět penalizaci jedinců, kteří dostatečně nesplňují omezení v podobě časových oken, je uražená vzdálenost přepočítávána na čas za pomoci rychlosti dopravního leteckého prostředku Boeing 737-800, jenž má průměrnou cestovní rychlost 842 $km \cdot h^{-1}$. Jelikož se jedná o problém obchodního cestujícího, je každá hodina ohodnocena sumou 11,6 USD. Zvolená suma je odvozena dle ceny provozu Boeingu 737-800, která je rozpočítána mezi všechny pasažéry.

Zásadním problémem je velikost penalizační složky (pokuty), šíře časového okna a interval, ve kterém může být časové okno generováno. Význam velikosti penalizační složky je probrán v kapitole 2.3. Šíře časového okna a interval, ve kterém se časová okna nachází, musí být experimentálně odvozeny pomocí vzorce

$$(t_l, t_h) \in \{0, \bar{x}_m \cdot n\}, \tag{4.2}$$

kde t_l – dolní hranice časového okna,

t_h – horní hranice časového okna,

$$t_h = t_l + k \cdot \bar{x}_m, \tag{4.3}$$

$\bar{x}_m$ – průměrná doba cesty mezi dvěma náhodnými městy,

n – počet měst,

k – nastavovaný parametr.

Řešitel nastavuje parametr *k*, díky kterému určuje šíři časových oken.

4.1 REPREZENTACE JEDINCE

Za výsledného jedince je považován vzorec pro ohodnocení měst skórem. Vzorec je složen z matematických funkcí a proměnných vybraných na začátku programu. Matematické funkce představují ve stromové struktuře funkce neboli uzly a proměnné jsou představiteli terminálů.

Množina terminálů je zvolena na základě vlastností města, které by mohly ovlivňovat výsledný vztah tak, aby byl zároveň splněn předpoklad postačitelnosti. Do množiny terminálů byly zvoleny funkce:

- x – hodnota osy x vybraného města,
- y – hodnota osy y vybraného města,
- s – je suma vzdáleností do x nejbližších měst,
- f – je vypočítaný polární úhel φ,
- *th* – je horní hranice časového okna.

Hodnoty x a y jsou pevně dané a jsou odvozeny z polohy vypočítávaného města vzhledem k počátku souřadnic. Hodnota s je suma vzdáleností x nejbližších měst od vybraného města. Úhel svíraný mezi přímkou spojující vypočítávané město a osu x, s vrcholem v počátku souřadnic je označen φ.

Hodnoty x a y jsou jedny z hlavních parametrů určujících polohu měst vyjádřenou v kartézském souřadnicovém systému. Zakódovat tuto polohu a předat ji genetickému programu lze i pomocí jiných souřadnicových systémů než pouze kartézských. Alternativou je zápis pomocí polárního souřadnicového systému, který používá úhel f a hodnoty x, y. Motivací pro zavedení sumy vzdáleností nejbližších měst s je předpoklad, že výslednou cestu může ovlivňovat i znalost nejbližšího okolí obchodního cestujícího. Horní hranice časového okna byla přidána, aby při výpočtu skóre města měl výsledný heuristický postup informaci také o omezujících podmínkách, kterou jinak získá pouze z výsledné fitness funkce.

Množinu funkcí stromové struktury tvoří mix matematických operací a goniometrických funkcí. Do této množiny patří:

- *plus* – vrátí součet dvou vstupních argumentů,
- *minus* – vrátí rozdíl dvou vstupních argumentů,
- *divide* – vrátí podíl dvou vstupních argumentů,
- *add* – vrátí součin dvou vstupních argumentů,
- *sin* – vrátí hodnotu sinus ze vstupního argumentu,
- *cos* – vrátí hodnotu cosinus ze vstupního argumentu,
- *tan* – vrátí hodnotu tangent ze vstupního argumentu.

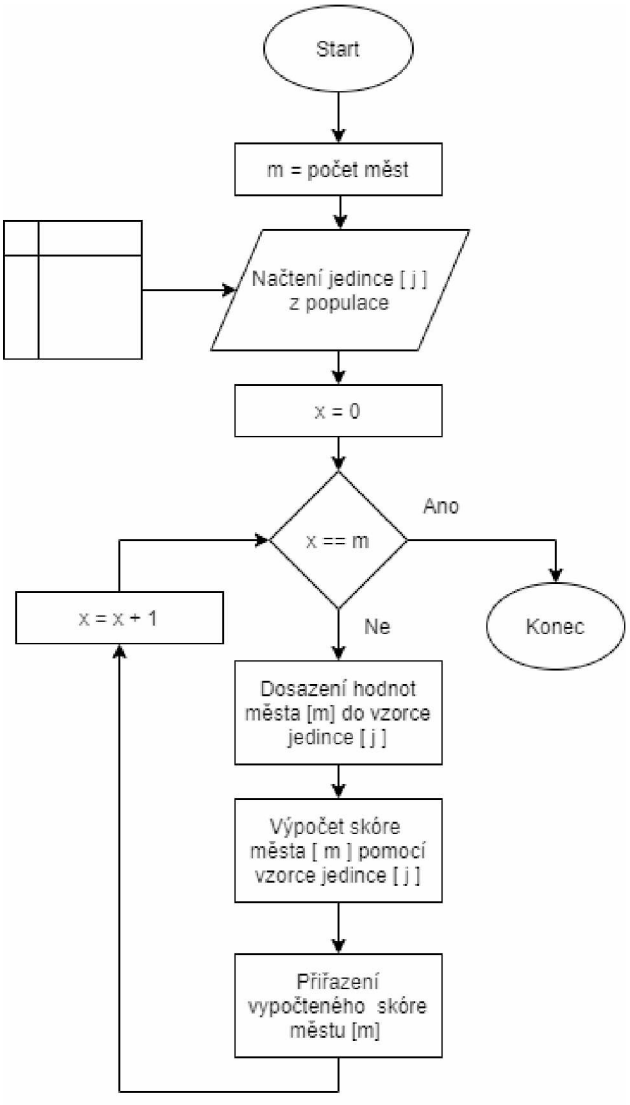
Množinu funkcí je možné dále rozdělit na dvě podmnožiny podle přidělené arity. Všechny funkce vyjadřující matematické operace (*plus, minus, divide, add*) mají aritu rovnou dvěma a goniometrické funkce (*sin, tan, cos*) mají aritu rovnající se jedné.

5 IMPLEMENTACE

5.1 PYTHON

Python je objektově orientovaný programovací jazyk, který lze využít pro vývoj množství různých aplikací. Zahrnuje vysokoúrovňové datové struktury a mnoho dalších užitečných funkcí, které usnadňují práci programátora. Lze ho implementovat na většině operačních systémech jako jsou Linux, Windows, MacOs a další.

Syntaxe v Pythonu je vyzdvihována pro svoji jednoduchost. Jako oddělovač zde slouží odřádkování a k oddělení bloků se používá odsazení, nejčastěji čtyřmi mezerami. Hlavním



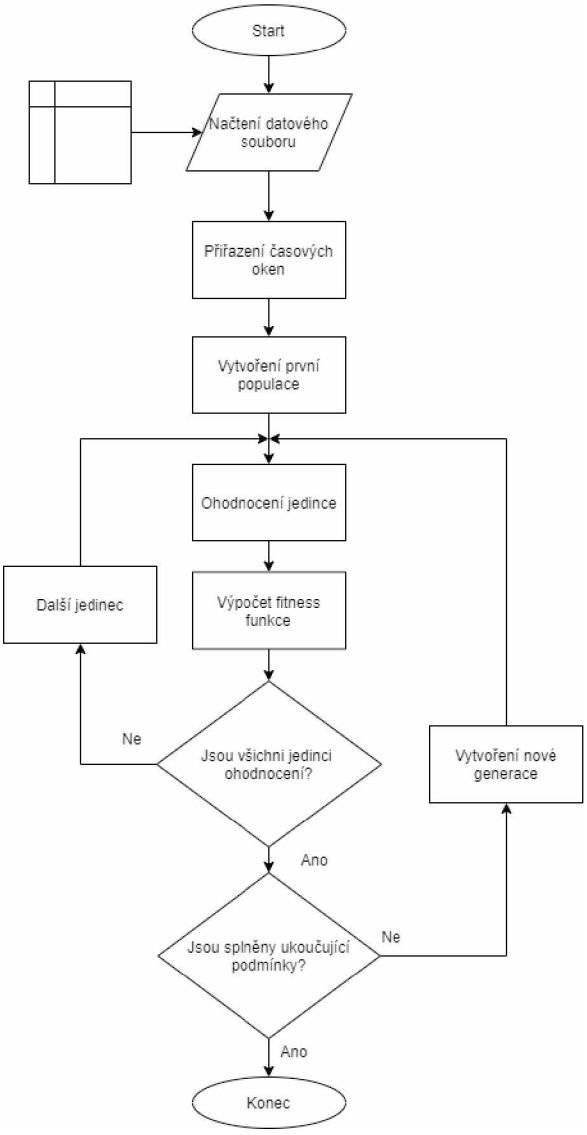
Obrázek 5.1 – Výpočet skóre

znakem tohoto programovacího jazyka je, že všechna data jsou reprezentována jako objekty,

ke kterým se přistupuje výhradně pomocí referencí. Objekty mají svou identitu neboli adresní číslo, přidělené při vzniku a které se nemění až do svého zániku. K objektu se řadí dále jeho hodnota, se kterou pracuje vykonávaný program. Třídy v Pythonu umožňují všechny pokročilé vlastnosti vhodné k objektově orientovanému programování.

5.2 FUNKCE PROGRAMU

Zjednodušený algoritmus implementovaný v práci je znázorněn na obrázku 2.1. V algoritmu jsou na začátku určena nastavení pro genetický program a je načten datový

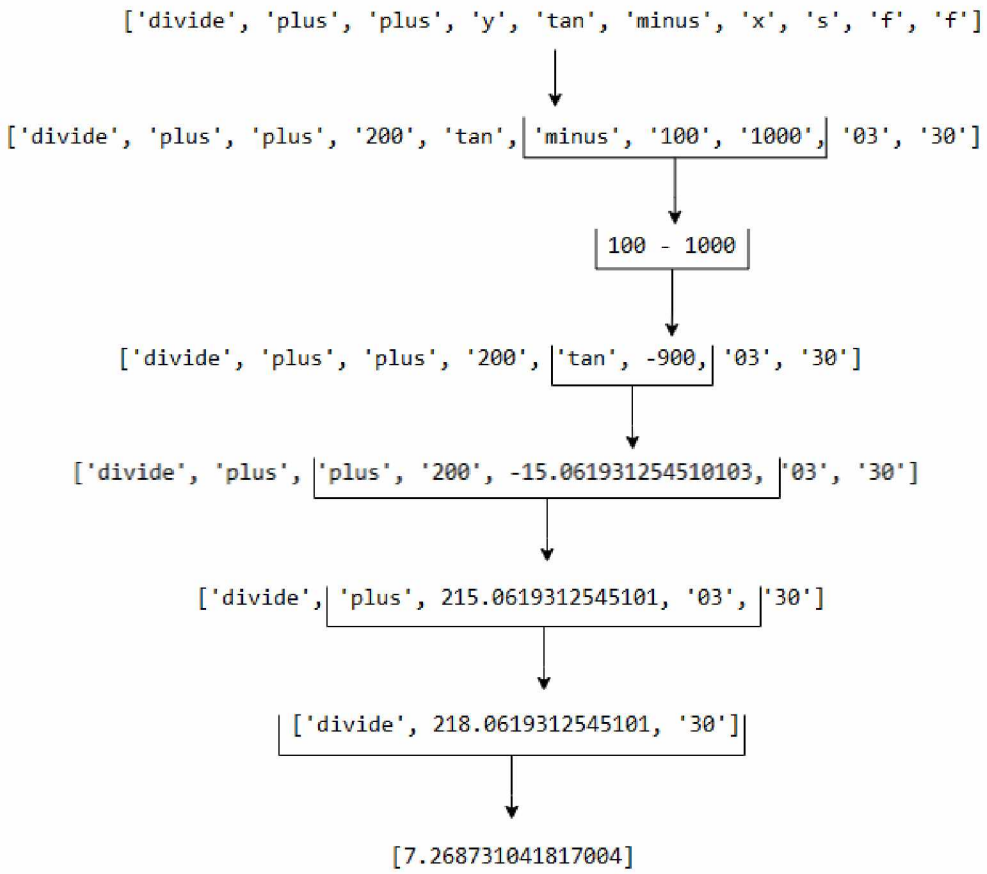


Obrázek 5.2 – Algoritmus programu

soubor obsahující *x* a *y* souřadnice, které určují pozici města na mapě. K jednotlivým městům jsou přidělena časová okna. Poté se vytvoří adresář, kam se ukládají výsledné fitness funkce, nastavení genetického programu a další požadované výstupní informace. Následně je vytvořena inicializační populace, vypočtena fitness funkce u každého jedince a nalezen nejlepší možný jedinec. Genetické operace prováděné na populaci vytvoří novou generaci jedinců nahrazující tu starou. Tento proces probíhá, dokud nejsou splněny ukončující podmínky pro běh programu, které jsou v práci stanoveny maximálním počtem generací a počtem epoch genetického programu s daným nastavením.

5.2.1 Výpočet skóre

Celý výpočetní proces začíná tak, že pro vybrané město jsou vypočítány atributy *s* a φ rozebírané v kapitole 4.2. Poté se dosadí všechny hodnoty do vybraného jedince (vzorce) a



Obrázek 5.3 – Výpočet fitness funkce jedince

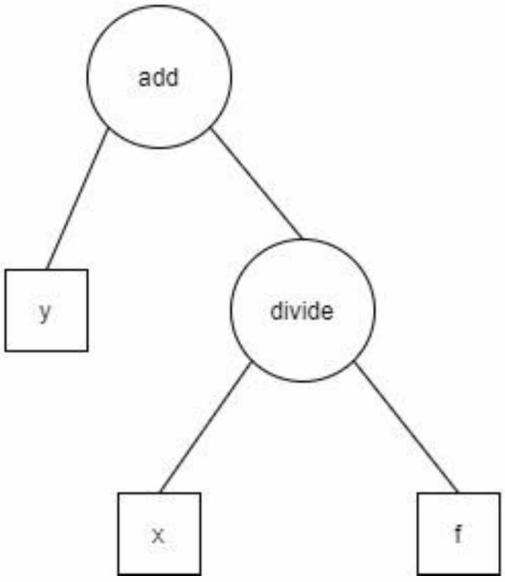
vypočítá se skóre, které se přiřadí městu jako další atribut. Výpočet skóre je znázorněn na obrázku 5.3. Počítané město je přeřazeno na konec listu s městy a celý proces se opakuje,

dokud každé město nemá přiřazeno skóre. Poté se města seřadí vzestupně dle velikosti skóre a suma přímých vzdáleností po sobě jdoucích seřazených měst slouží jako základ pro výpočet fitness funkce. Algoritmus uražené vzdálenosti je k vidění na obrázku 5.1.

5.2.2 Vytvoření jedince

Jedinec je vytvářen metodou *grow*, popsanou v kapitole 2.1.1. Počáteční vlastnosti jsou pouze dvě, a to je minimální a maximální hloubka generovaného stromu. Minimální hloubka je definována z důvodu, aby byl vytvořen graf stromové struktury a nevzniklo pouze pole obsahující osamocený terminál. K zabránění vzniku této situace je implementováno

```
[2761.31605700138, 46, ['add', 'y', 'divide', 'x', 'f']]
```



Obrázek 5.4 – Vzor jedince

vynucení podmínky o zvolení kořenového uzlu pouze z množiny funkcí. Maximální hloubka vygenerovaného stromu je pouze volitelnou vlastností určenou k nastavení genetického programu, která je užitečná k tomu, aby se v počátečních generacích netvořily příliš rozsáhlé struktury. Ukázka jedince je na obrázku 5.4.

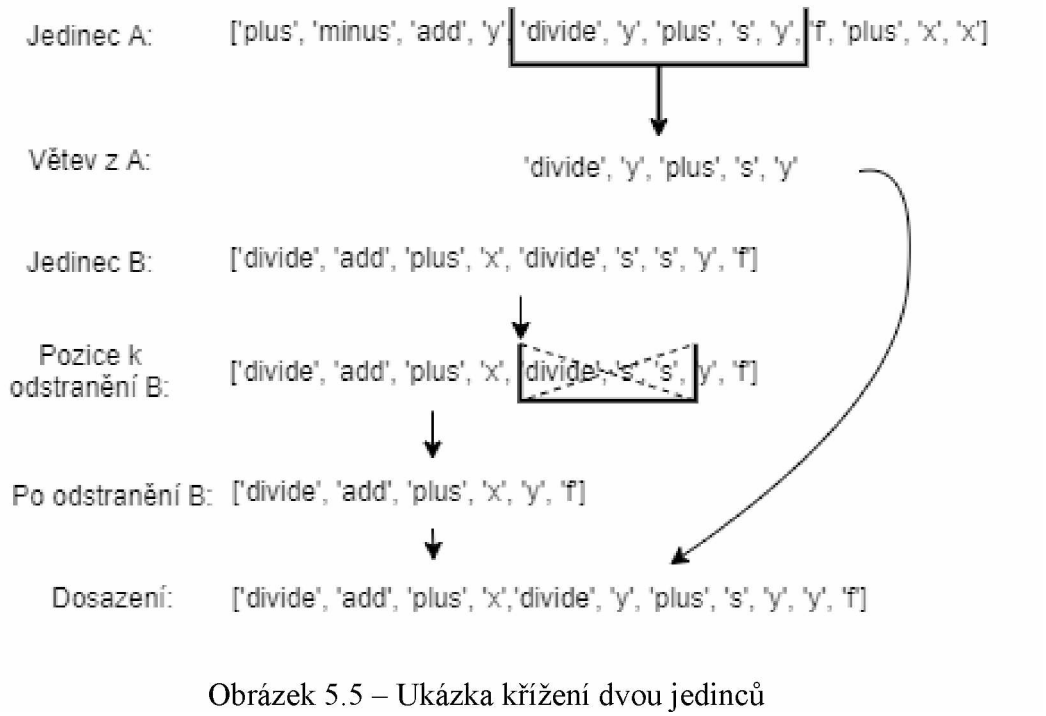
Výsledný jedinec v programu je popsán pomocí tří atributů poskytující informaci o:

- fitness funkci,
- počtu nedodržených časových oken,
- vzorci ve stromové struktuře.

5.2.3 Křížení, mutace a selekce

Výběr jedinců v programu je implementován užitím klasické turnajové selekce popsané v kapitole 2.2.1. Při turnaji vždy zvítězí jedinec s lepší fitness funkcí.

Proces křížení probíhá podobně, tak jak je popsán v kapitole 2.2.2. Na počátku je vybrán jedinec pomocí metod selekce a z něj je oddělena větev vhodná pro křížení. Výběr funkce, ze které je větev určována, je volen pomocí generátoru náhodných čísel a následně je



Obrázek 5.5 – Ukázka křížení dvou jedinců

vypočítán potřebný počet funkcí a terminálů, které na funkci navazují. Vybraná větev musí splňovat předpoklad uzavřenosti. Poté je opět použita metoda selekce a je vybrán druhý jedinec ke křížení. Z druhého jedince, podobně jako při výběru větve pro křížení, je vybrána větev, která bude odstraněna a nahrazena větví z prvního jedince. Před navrácením je zkontrolována hloubka vytvořeného stromu kvůli splnění podmínky o maximální hloubce stromu. Celý proces je znázorněn na obrázku 5.5.

Pro proces mutace byla zvolena jednobodová mutace, při které je nahrazován pouze vrchol z množiny funkcí se stejnou aritou.

5.3 POPIS PROGRAMU

Celý program je rozdělen do čtyř skriptovacích souborů s příponou *.py*. Primární soubor je pojmenována *main.py* a slouží jako výchozí bod pro spuštění programu. Řídí volání dalších funkcí a jsou v něm definovány ukončující podmínky programu. V souboru *TravelingSalesmenVer2_2.py* jsou implementovány třídy a funkce pro práci genetického algoritmu, a to jak s jedinci, tak s celou populací. Soubor *Statistic.py* je určen pro ukládání výsledků do vybraných souborů a adresářů. Dále tam jsou napsány funkce pro práci s výsledky a jejich následné statistické vyhodnocení. Posledním souborem je *main_stat.py* sloužící jako výchozí funkce pro statistické operace.

5.3.1 Main

V souboru *main.py* jsou nejprve připojeny třídy, jejichž funkce se v souboru používají. Jako první jsou to třídy *Individual* a *Population* z *TravelingSalesmenVer2_2.py* a třídy *result* a *Statistic* ze souboru *Statistic.py*. Dále je importována třída *random*, která zajišťuje generování náhodných čísel a jiné operace související s náhodnými operacemi. Nastavení genetického programu má na starosti několik konstant vytvořených po importování požadovaných funkcí. Mezi nastavující atributy patří:

- LIST_OF_NONTERMINAL_NORMAL;
- LIST_OF_NONTERMINAL_SPECIAL;
- LIST_OF_TERMINAL;
- NUMBER_OF_NEAREST_CITIES;
- LENGTH_OF_INITIAL_TREE;
- LENGTH_OF_POPULATION;
- MAX_LENGHT_TREE;
- CROSSOVER_PERCENTIGE;
- MUTATION_PERCENTIGE;
- NUM_OF_GENERATION;
- NUM_OF_ATTEMPT;
- PENALISATION;
- PATH_TO_RESULT;
- TIME_WINDOW.

LIST_OF_NONTERMINAL_NORMAL a LIST_OF_NONTERMINAL_SPECIAL jsou pole názvů funkcí použitých při generování stromu. LIST_OF_TERMINAL je pole terminálů, které je využito taktéž při generování stromu. NUMBER_OF_NEAREST_CITIES je počet měst, jejichž suma je využita při výpočtu terminálu *s*. LENGTH_OF_INITIAL_TREE udává maximální hloubku generovaného stromu při inicializaci první populace. Hodnota LENGTH_OF_POPULATION značí počet jedinců v jedné generaci. Toto číslo se při vykonávání programu nemění a zůstává stejné pro všechny generace. MAX_LENGHT_TREE je důležitým omezením, které zabraňuje vytváření příliš rozsáhlých stromových struktur. CROSSOVER_PERCENTIGE definuje kolik procent jedinců z nově formující se populace bude vytvořeno genetickou operací křížením. Podobně MUTATION_PERCENTIGE udává procento nově vzniklých jedinců v populaci, vytvořených mutací vybraných jedinců ze staré populace. Počet vytvořených generací za jednu epochu je určen pomocí konstanty NUM_OF_GENERATION. Aby řešitel nemusel zasahovat při více opakováních téhož nastavení, byla vytvořena konstanta NUM_OF_ATTEMPT, která udává počet epoch běhu genetického programu. Konstanta PENALISATION nastavuje penalizační složku při nedodržení časového okna obchodním cestujícím. PATH_TO_RESULT je cesta v systému, kam se ukládají výsledky genetického programu. TIME_WINDOW definuje jakým způsobem jsou generována časová okna.

V *main.py* je implementována jediná funkce, a to na generování jedinců dle počátečních podmínek. Funkce má definovaný název *create_individual_form()* a vstupními argumenty jsou omezení hloubky stromu a seznam terminálů a funkcí. Návratovou hodnotou je pak vzorec přiřazený do populace.

Před začátkem běhu genetického programu je vytvořen adresář s výsledky, kam se ukládají provedené výpočty. Jako první je zde uložena konfigurace programu do textového souboru. Vlastní běh algoritmu genetického programování obstarávají dva *for* cykly, z nichž jeden odpočítává počet epoch a druhý počet generací. Při započetí každé epochy jsou vytvořeny soubory typu *.csv*, do kterých je ukládána fitness funkce, počet nedodržených časových oken a výsledný jedinec reprezentující heuristickou funkci. Následně je vytvořena první populace, spočítána fitness funkce každého jedince pomocí třídy *Individual,* nalezen nejlepší jedinec a požadované hodnoty jsou uloženy do předpřipravených souborů. Začátek běhu druhého *for* cyklu, jenž provádí genetické operace na vzniklé generaci, začíná vytvořením nové populace s využitím genetických operací za pomoci třídy *Population.*

Dalším krokem je opětovný výpočet fitness funkce u jedinců v populaci a uložení požadovaných hodnot do souborů.

5.3.2 Implementace genetických operátorů

Třída *Individual* má jako vstupní argument vzorec jedince a seznam měst s časovými okny a vrací jako návratovou hodnotu jedince s vypočtenou fitness funkcí. V této třídě je implementována sada funkcí pro výpočet uzlových bodů vzorce. Dále pomocí funkce *fill_formula()* nahrazuje proměnné ve vzorci za skutečné hodnoty podle algoritmu popsaného v kapitole 5.2.1. Ve třídě je funkce obstarávající výpočet celého vztahu nazývající se *call_form()*. Je zde umístěna i funkce *calculate_route()*, která vypočítává cestovní vzdálenost mezi už seřazenými městy a následně je přepočítává do požadované podoby.

Třída *Population* obstarává genetické operace na celé populaci. Jako vstup vezme populaci ať už nově vytvořenou nebo generaci, na níž už byly aplikovány genetické operátory, a návratová hodnota je nová populace vzniklá dle požadovaných podmínek. Je zde implementována selekční metoda pro výběr jedince v podobě turnajového schématu. Dále funkce obstarávající mutaci a křížení pomocí algoritmu popsaného v kapitole 5.2.3. Je zde aplikován i elitismus jako doplněk genetických operací.

6 DOSAŽENÉ VÝSLEDKY

Nálezu vhodného heuristického řešení předchází zásadní proces vyšetření významnosti jednotlivých elementů v množinách funkcí, terminálů, nastavení šíře časových oken a interval, ve kterém se nacházejí. O jejich významu rozhoduje přispění ke snížení fitness funkce nebo počet chybně navštívených oken. Výkonnost algoritmu genetického programování je možné ovlivnit pomocí parametrů, které se nastavují před začátkem programu. Mezi tyto parametry patří počet generací, velikost populace, procentuální změna populace vzniklá za přispění genetických operátorů a počet nejbližších měst potřebných k výpočtu elementu s z množiny terminálů.

Tabulka 6.1 – Úspěšnost navštívených měst

k	Průměrný počet nedodržených časových oken
1	12,76
2	10,4
3	9,12
4	8,28
5	4
6	6,24
7	12,64
8	4,72
9	3,88
10	2,8

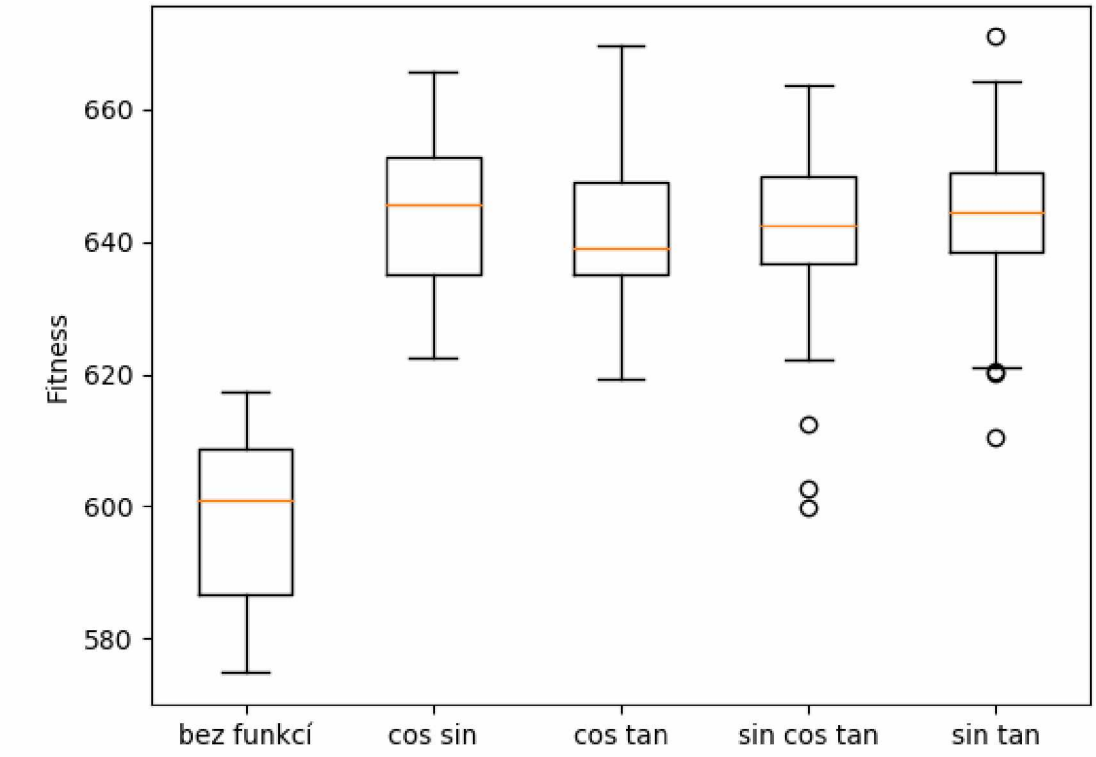
Základní znalostí je, jak ovlivňuje šíře časových oken úspěšnost obchodního cestujícího při jeho cestě. Při hledání úspěšnosti byly použity všechny dostupné terminály a funkce definované v kapitole 4.2. Velikost vstupních dat byla zredukována na počet 20 měst z původního datového souboru 48 měst. Menší datový soubor byl zvolen z důvodu časové úspory při provádění programu. Velikost populace je nastavena na 500 jedinců v každé generaci, počet generací je 300 a počet provedených epoch je roven 25. Konstanta k byla průběžně měněna pro každý běh programu. Do tabulky 6.1 je zaznamenán aritmetický průměr počtu měst nenavštívených v přiděleném časovém okně u nejlepšího jedince vzniklého z běhu algoritmu genetického programování. Průměrná doba cesty mezi dvěma náhodně vybranými městy, potřebná k výpočtu časových oken, je 2,8 a byla odvozena z běhu první generace o 500 jedincích. K jejímu celkovému výpočtu bylo tudíž použito 9500 hodnot. Z výsledných

hodnot vyplývá, že obchodní cestující je schopen navštívit zhruba polovinu měst, pokud je hodnota *k* nastavena na hodnotu dva a šíře časového okna je vypočítávána pomocí vztahu.

6.1 VÝZNAM GONIOMETRICKÝCH FUNKCÍ

V literatuře jsou nejpopulárnější používané funkce při vytváření jedince jednoduché aritmetické operace, které mají předvídatelný význam a dosahují dobrých výsledků. Ovšem nejednoznačný je význam goniometrických funkcí. V práci jsou použité tři různé typy goniometrických funkcí, funkce *sinus,* funkce *cosinus* a funkce *tangent*.

Výsledky vlivu goniometrických funkcí na hodnotu fitness funkce lze vidět na



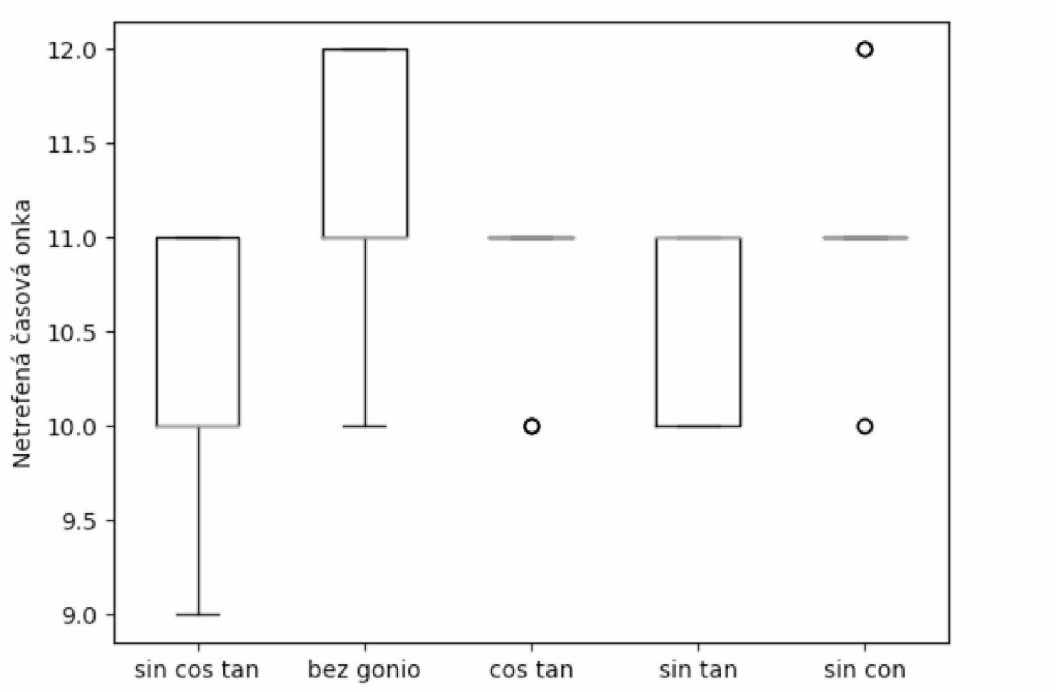
Obrázek 6.1 – Význam goniometrických funkcí při hledání nejkratší trasy

obrázku 6.1, kde jsou porovnávány jejich kombinace vůči běhu genetického programu bez goniometrických funkcí. Kombinace jednotlivých funkcí je zvolena tak, aby případně odhalila méně významné funkce, které mohou ovlivnit výslednou fitness funkci. To znamená, že pokud by funkce *tangent* působila negativně na výslednou hodnotu fitness funkce, její

statistické hodnoty by byly odlišné od ostatních testů, kde zmíněná funkce nefiguruje. V krabicovém grafu jsou vynášeny nejlepší fitness funkce z každé epochy běhu programu.

Závěrem odvozeným z grafu je, že jakákoliv goniometrická funkce či jejich kombinace zhoršují výslednou fitness funkci významnou měrou. Při vyhodnocení byly porovnány i klasické statistické veličiny, mezi které patří: průměr, medián, dolní a horní kvartily. Nicméně tyto hodnoty potvrzují jasný výsledek znázorněný na grafu.

Význam přispění goniometrických funkcí lze experimentálně měřit i na druhém cíli genetického programu, což je množství zákazníků, kteří nebyli obslouženi v přiděleném



Obrázek 6.2 – Význam goniometrických funkcí u omezujících podmínek

časovém okně. Pokud by goniometrické funkce snižovaly počet nedodržených oken, jejich zachování by záleželo na volbě prioritního cíle.

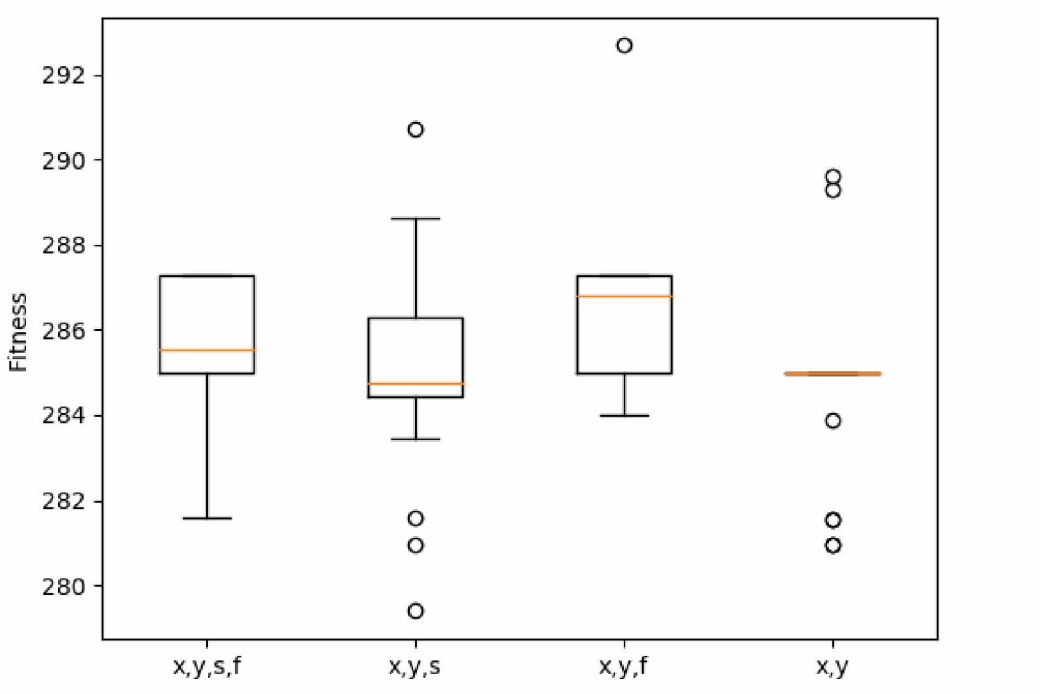
Šíře okna byla odvozena dle vztahu 4.2, přičemž parametr k byl nastaven na hodnotu 2. Z tabulky 6.1 vyplývá, že průměrný počet nedodržených časových oken byl 10,4 a tak hodnoty ležící pod zmíněným průměrem by znamenaly zlepšení.

Z obrázku 6.2 je patrné, že přispění goniometrických funkcí ke snížení počtu nedodržených oken je zanedbatelné. Střední hodnoty u kombinací různých funkcí oscilují okolo změřeného průměru a žádný nepřinesl výrazné zlepšení.

Lze tedy konstatovat, že začlenění goniometrických funkcí zhoršuje efektivnost u nalezené trasy a nepřináší zlepšení v nálezu řešení, které by vyhovovalo omezujícím podmínkám v problému TSPTW. Pro následující běhy bude tedy upuštěno od jejich začlenění do množiny funkcí.

6.2 VÝZNAM TERMINÁLŮ

Porovnání bylo provedeno na programu, který vytvořil 25 epoch a v každé epoše bylo provedeno 500 generací. V jedné generaci bylo 500 jedinců. Z časových úspor neběžel program na celém datovém souboru, ale pouze na 20 prvních městech. Z obrázku 6.3 je patrné, že informace v podobě kartézského souřadnicového systému přispívají k nalezení nejkratší cesty, ale dá se usoudit, že tyto informace nestačí samy o sobě pro dostatečné prohledání prostoru. Jak je vidět na krabicovém grafu (první zprava na obrázku 6.3 pod



Obrázek 6.3 – Význam terminálů při hledání nejkratší trasy

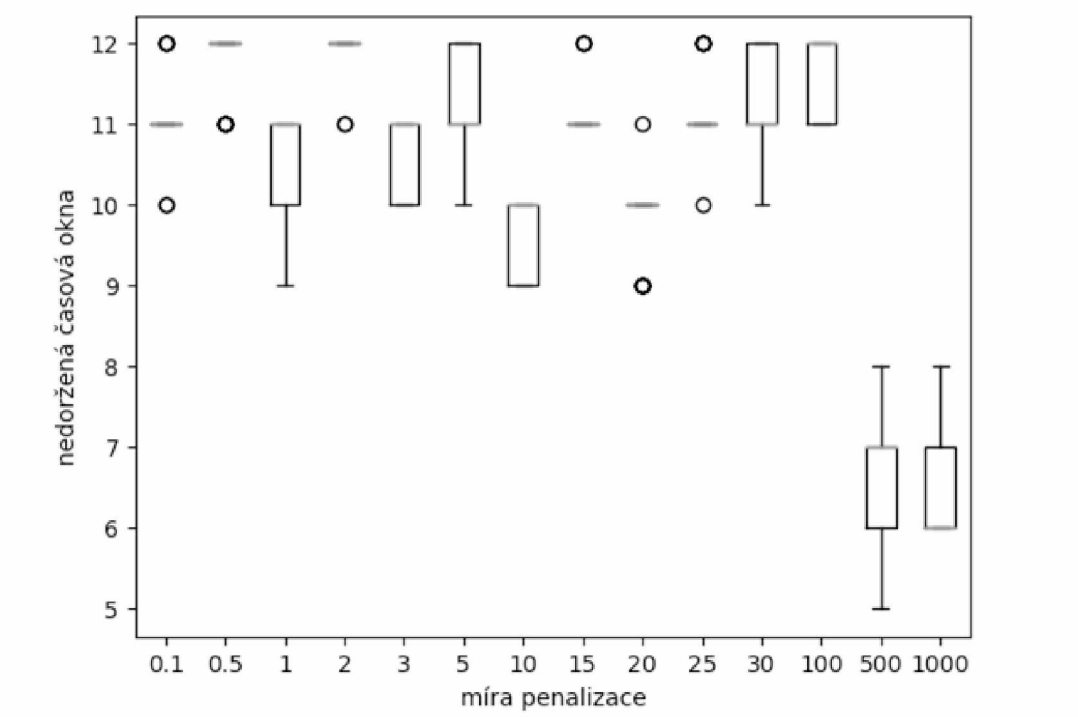
označením *x,y*) tak se genetický algoritmus často zasekl na jedné hodnotě, ze které neunikl. Přidáním informace o okolním prostoru užitím terminálu *s* a úhlu od počátku souřadnic φ, se značně zvýšila rozmanitost nejlepších řešení. Výsledky dále ukazují, že terminál *s* se zdá být pro program nepatrně užitečnější než polární souřadnice. Po vynechání polární souřadnice se

podařilo nalézt nejlepšího jedince ze všech generací a střední hodnota je také nepatrně lepší. Nicméně střední hodnoty nejsou od sebe nikterak zásadně rozdílné, a tak rozdíl ve významu terminálů se dá považovat za zanedbatelný. Hodnota *th* byla z výsledného porovnání vynechána, protože její význam se vztahuje výhradně k omezujícím podmínkám. Pro zachování rozmanitosti řešení budou při dalších pokusech nadále používány všechny navržené terminály.

6.3 VÝZNAM PENALIZACE

Penalizace byla v této práci zvolena jako způsob, jak splnit omezující podmínky při zachování jedné fitness funkce. Bližší informace o penalizaci jsou uvedeny v kapitole 2.3.

Program, který testoval význam penalizace, měl stejné nastavení jako program z kapitoly 6.2. Penalizační složka se zvyšovala vždy po 25 epochách. Byly voleny hodnoty: 0,1; 0,5; 1; 2; 3; 5; 10; 15; 25; 30; 100; 500; 1000. Výsledek je k vidění na obrázku



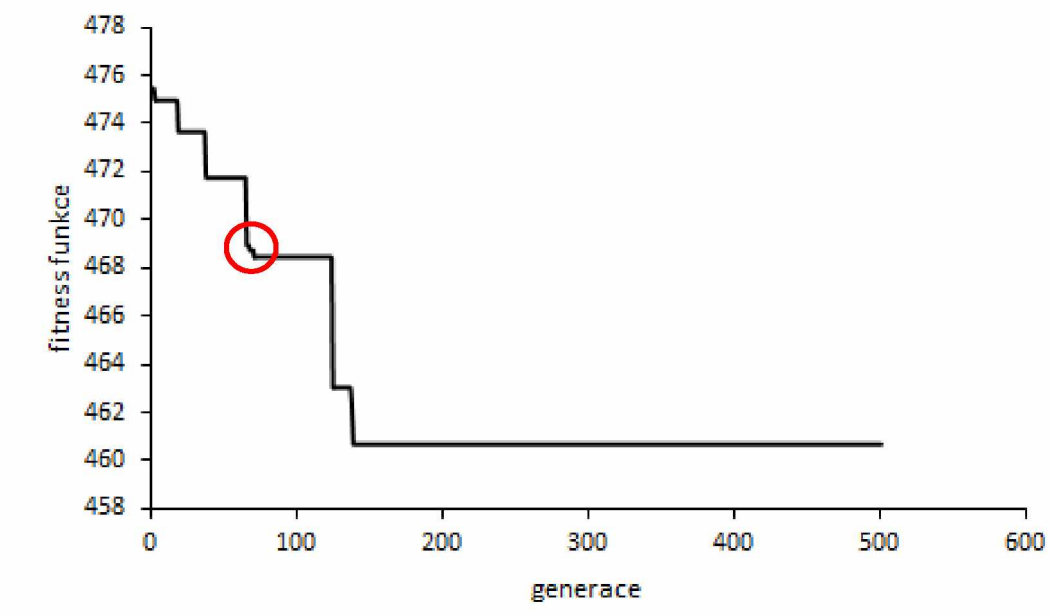
Obrázek 6.4 – Význam penalizace

6.4. U nízké hodnoty penalizace se očekávalo, že dodržování časových oken nebude hlavní cíl genetického programu. Za nízkou hodnotu penalizace lze považovat hodnoty v intervalu 0 až 10. Nicméně při hodnotách 10 až 100 bylo očekáváno postupné snižování nedodržených

oken, protože celková míra penalizace se blížila nebo často překračovala vlastní cenu cesty obchodního cestujícího. Teprve až jedna penalizační složka přesahovala cenu cesty obchodního cestujícího, snížil se počet nedodržených časových oken.

6.4 FITNESS FUNKCE

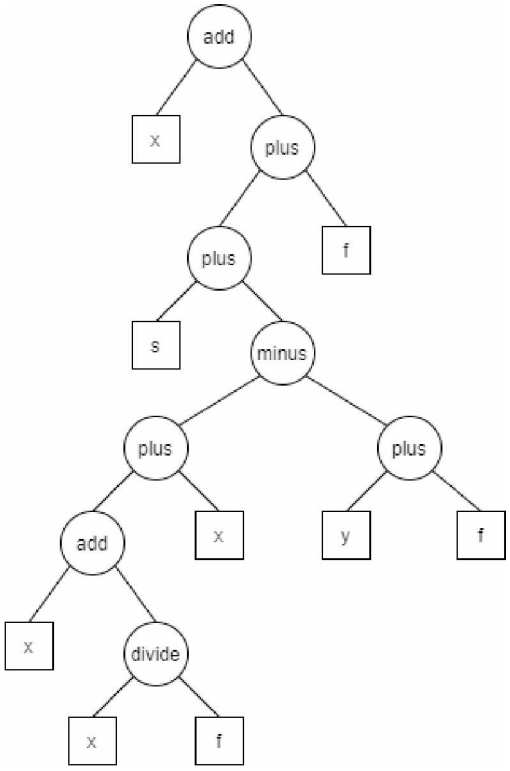
Fitness funkce je základním prvkem genetických algoritmů a v této práci je zmíněna už v kapitole 1. Tato kapitola je věnována vývoji fitness funkce v implementovaném programu. Jeden z možných průběhů fitness funkce je znázorněn na obrázku 6.5. Ten ukazuje její klesání v jednotlivých generacích. Vylepšení fitness funkce se povedlo pouze párkrát za daný běh programu a téměř vždy skokově. Obrázek 6.5 je jednou z mála výjimek, kde je v červeném kolečku označeno zlepšení fitness funkce v každé po sobě jdoucí generaci. Hodnota fitness se měnila hlavně v počátečních generacích a pokud genetický algoritmus nalezl lepší heuristický postup od dvousté generace a výše, byla to spíše malá změna.



Obrázek 6.5 – Ukázka vývoje fitness funkce napříč generacemi

Výjimku tvořily programy s větší hodnotou penalizace, typicky 500 a 1000. Nalezení trasy, která lépe vyhověla omezujícím podmínkám, se projevilo razantní změnou výsledné fitness funkce způsobené penalizací.

6.5 VÝSLEDNÉ HEURISTIKÉ POSTUPY



Obrázek 6.6 – Ukázka výsledné heuristiky

$$c = x \cdot \left(s - y + x + \left(\frac{x^2}{f} \right) \right) \tag{6.1}$$

kde c je skóre města.

Generované heuristické postupy dosahovaly často maximální hloubky stromu. Objevovaly se i syntaktické stromy, které dosahovaly lepších výsledků a nebyly tak rozsáhlé. Příkladem je jedinec na obrázku 6.6, který byl nalezen v běhu uvedeném na obrázku 6.5. Jedinec poskytuje druhou nejlepší nalezenou fitness funkci jejíž hodnota je 463,0522. Heuristiku z jedince lze vyjádřit jako vztah 6.1, z něhož se počítá skóre pro jednotlivá města.

7 ZHODNOCENÍ

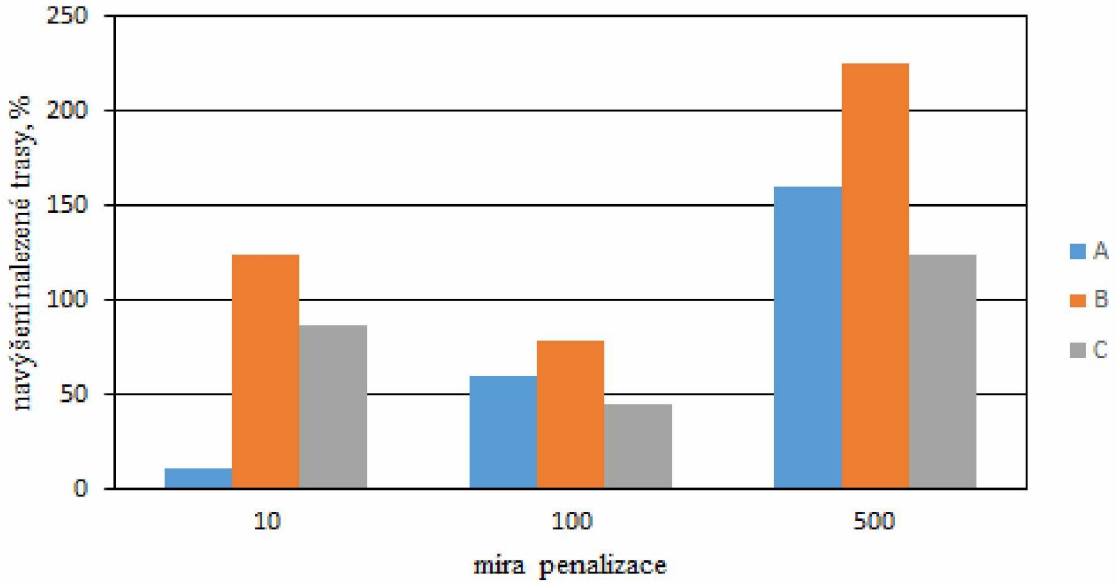
Pro zhodnocení výkonu genetického programu byla porovnána výkonnost výsledných heuristik na třech datových souborech. Poté byly porovnány hodnoty z tabulky 7.1 a tabulky 7.3. První porovnávaná hodnota je počet nedodržených časových oken vůči jejich maximálnímu počtu. Druhou porovnávanou hodnotou je vzdálenost vůči minimální vzdálenosti trasy. V tabulce 7.1 jsou uvedeny hodnoty třech nejlepších heuristických postupů, které genetický algoritmus našel.

Algoritmus se lišil pouze nastavením penalizační konstanty, takže buď byl kladen důraz na minimalizaci počtu nedodržených časových oken, nebo na snížení procestované vzdálenosti. Ostatní nastavení programu zůstávalo nezměněné pro všechny běhy:

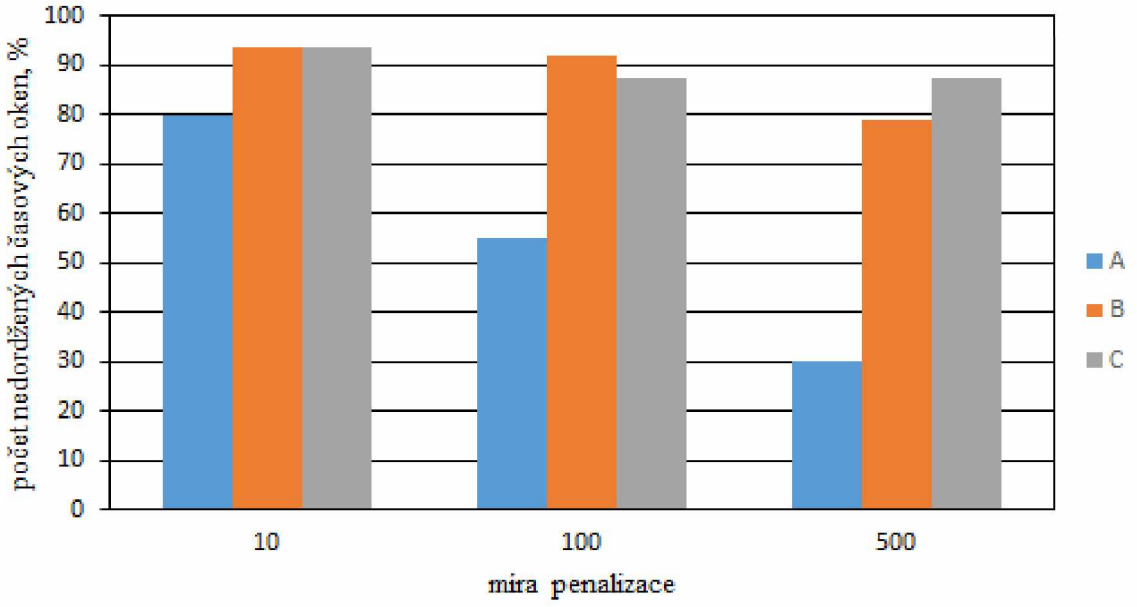
- LIST_OF_NONTERMINAL_NORMAL = ['plus', 'minus', 'add', 'divide'];
- LIST_OF_NONTERMINAL_SPECIAL = [];
- LIST_OF_TERMINAL = ['x', 'y', 's', 'f', 'th'];
- NUMBER_OF_NEAREST_CITIES = 5;
- LENGTH_OF_INITIAL_TREE = 20;
- LENGTH_OF_POPULATION = 500;
- MAX_LENGHT_TREE = 70;
- CROSSOVER_PERCENTIGE = 60;
- MUTATION_PERCENTIGE = 5;
- NUM_OF_GENERATION = 500;
- NUM_OF_ATTEMPT = 25.

- Tabulka 7.1 – Nejlepší výsledné heuristiky

Míra penalizace	Počet nedodržených časových oken			Nalezená vzdálenost s ohledem na časová okna		
	Datový soubor			Datový soubor		
	A	B	C	A	B	C
10	16	45	15	20 138	75 167	543
100	11	44	14	29 043	59 974	422
500	6	38	14	47 326	108 967	651



Obrázek 7.1 – Porovnání procentuálního navýšení tras u různých datových souborů

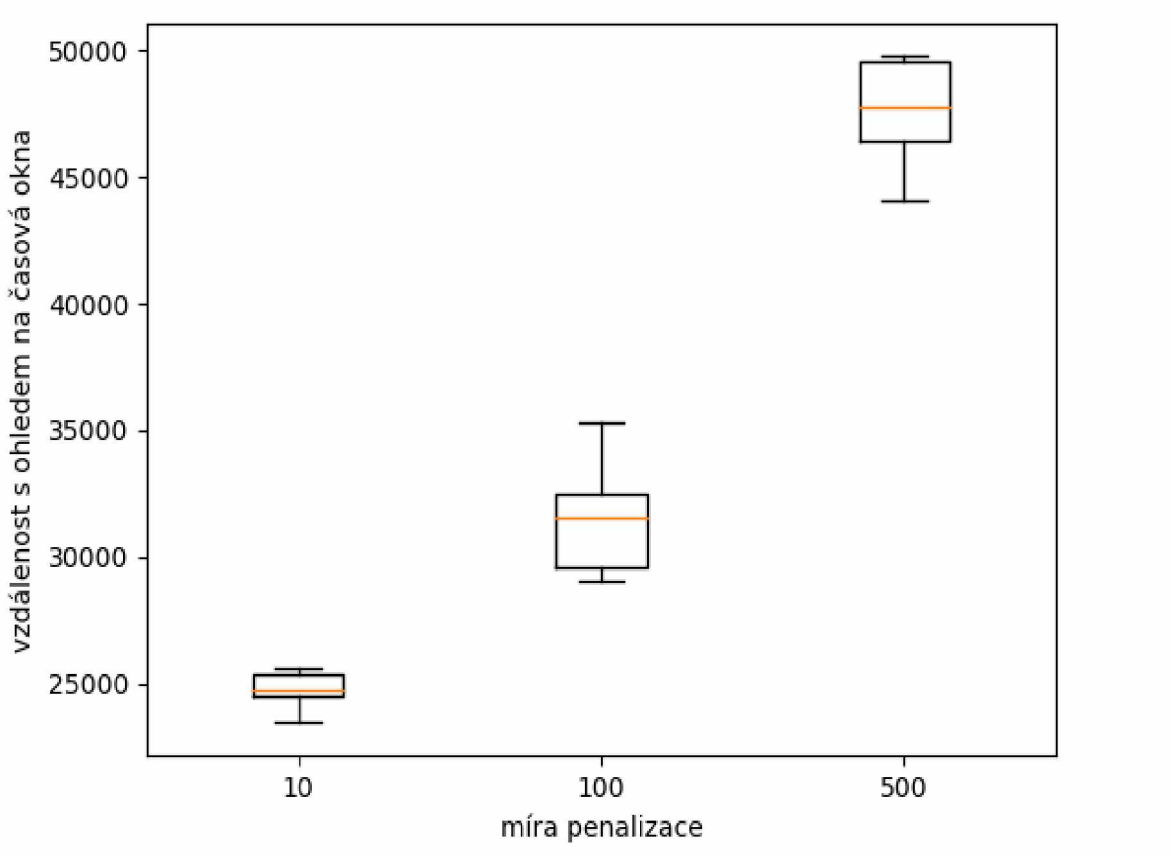


Obrázek 7.2 – Porovnání nedodržených časových oken u různých datových souborů

Genetický program na trénovacím souboru vykonal 25 epoch. Po vykonání všech 25 epoch byla nalezena nejlepší fitness funkce a na jejím základě nalezen nejlepší heuristický postup. Nejlepší heuristické postupy byly poté aplikovány na testovací datové soubory B a C, aby bylo možné porovnat splnění požadavků kladené na heuristiky z kapitoly 4.

Po experimentech z kapitoly 6.1 byly vynechány goniometrické funkce. Algoritmus genetického programování na toto nastavení optimalizoval výslednou heuristiku asi 12 hodin.

Datový soubor A je prvních dvacet hodnot z druhého datového souboru a je stanoven



Obrázek 7.3 – Nejlepší dosažené trasy z běhů u testovacího souboru A

jako trénovací množina dat, na níž byl trénován algoritmus genetického programování pro nalezení nejvhodnějšího heuristického postupu. Druhý datový soubor B obsahuje 48 měst potřebných k navštívení. Třetí soubor C je absolutně odlišný a nemá žádnou vazbu na předchozí dva. U datového souboru C je nutné experimentálně najít hodnotu $\bar{x}_m$ ze vztahu 4.2 a 4.3.

Hodnoty z tabulky 7.1 jsou vyneseny do obrázků 7.1 a 7.2. U obrázků je modrou barvou označena trénovací množina A. Obrázek 7.1 ukazuje závislost mezi zvyšující se trasou a zvolenou penalizační složkou, kde je patrný trend pouze u trénovací množiny dat. Za povšimnutí také stojí výrazně větší procentuální nárůst délky trasy u datového souboru B.

Obrázek 7.2 opět ukazuje trend u trénovací množiny dat, nicméně na rozdíl od obrázku 7.1 je trend opačný. Testovací datové soubory ukazují na obrázku 7.2 pouze malý až

zanedbatelný trend. Lze tedy konstatovat, že u trénovací množiny dat, při zvyšující se penalizační složce, se algoritmus snaží splnit omezující podmínky v podobě časových oken, i za cenu zvýšení uražené vzdálenosti. Bohužel podobná vlastnost se nepodařila prokázat na jiných datových souborech.

Statistické hodnoty nejlepších jedinců z jednotlivých epoch jsou uvedeny v tabulce 7.2. Jelikož algoritmus běžel pouze na trénovací množině dat, vztahují se tyto hodnoty k datovému souboru A. Grafické znázornění tabulky 7.2 je k vidění na obrázcích 6.4 a 7.3. Obrázek 7.3 ukazuje jistou závislost mezi volenou penalizační složkou a výslednou vzdáleností. Zmíněná závislost odpovídá trendu tří heuristických postupů, které jsou znázorněny na obrázku 7.1 a diskutovány o odstavec výše. Znázornění nedodržených časových oken z tabulky 7.2 je na obrázku 6.4 a ukazuje pouze malou podobnost s trendem trénovací množiny na obrázku 7.2. Nepodobnost může být způsobena nízkým počtem časových oken, a tak není trend tak výrazný. Pouze u hodnoty penalizační složky 500 je vidět snaha algoritmu o splnění omezujících podmínek.

Tabulka 7.2 – Statistické hodnoty epoch pro trénovací množinu

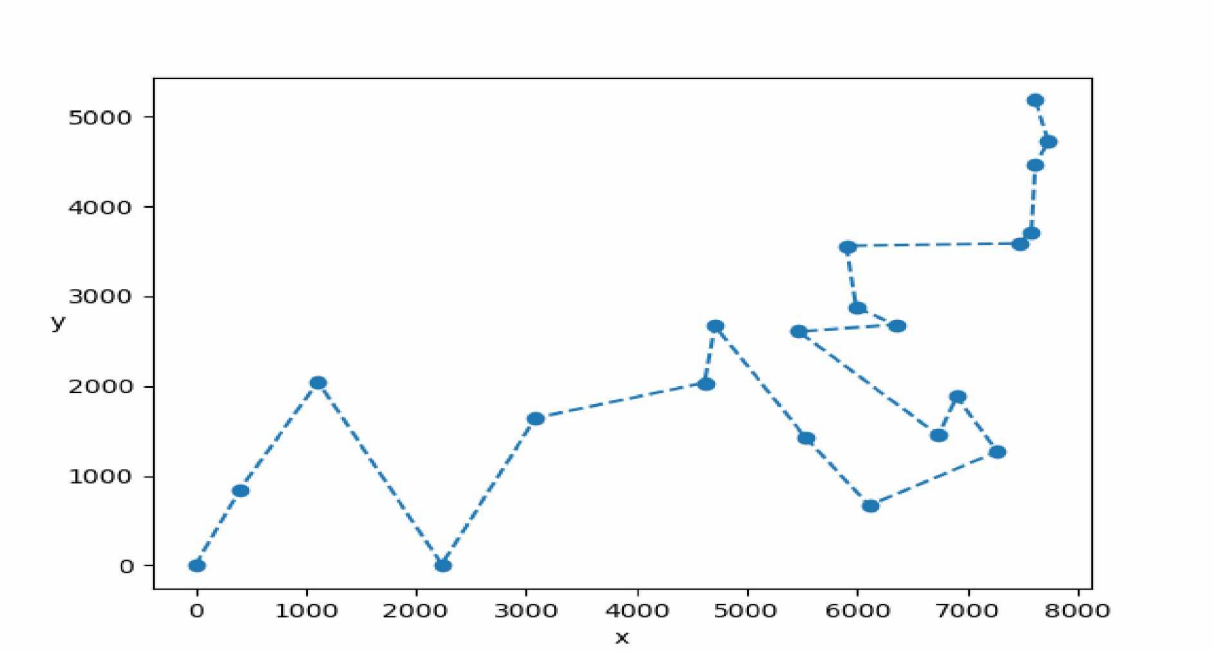
Nedodržená časová okna					
Míra penalizace	Průměr	Medián	Modus	1. Kvartil	3. Kvartil
10	9,56	10	10	9	10
100	9,8	10	10	10	10
500	6,52	6	6	6	7
Vzdálenost s ohledem na časová okna					
Míra penalizace	Průměr	Medián	Modus	1. Kvartil	3. Kvartil
10	25 518	25 600	25 650	25 427	26 121
100	30 971	31 421	31 740	30 460	33 740
500	48 390	47 738	47 396	46 396	49 591

Jak vyplývá z obrázků a tabulek uvedených v této kapitole, přenesení výsledných heuristických postupů na jiné datové soubory neposkytuje očekávané výsledky. Ačkoliv dokáže najít relativně dobrou trasu, není zaručené splnění omezujících podmínek u testovacích souborů. Přenositelnost, která je jedním z požadavků na heuristický postup v kapitole 4, není s omezujícími podmínkami splněna.

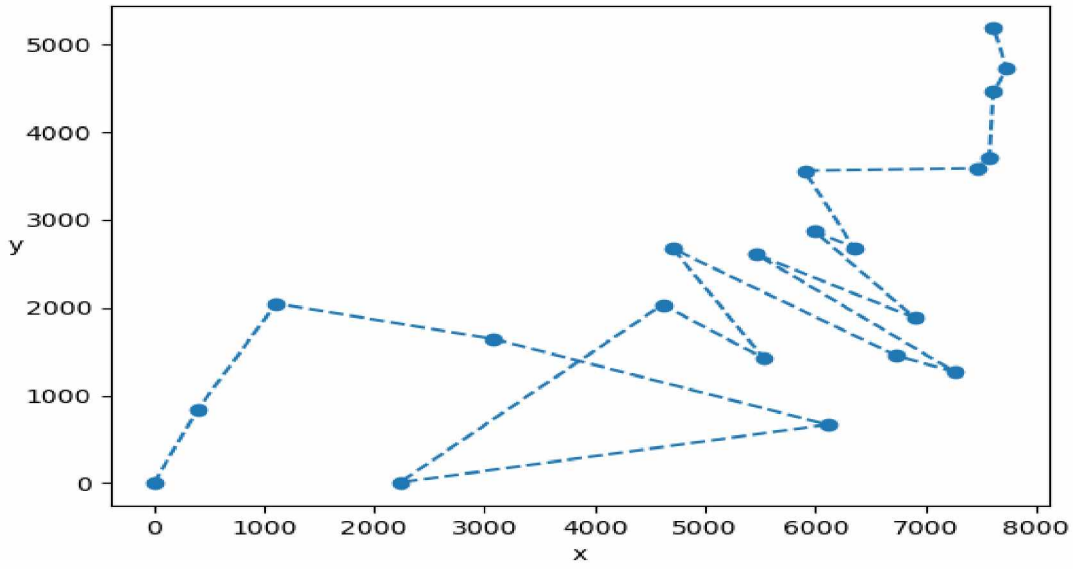
Tabulka 7.3 – Hodnoty datových souborů

Datový soubor	Počet časových oken	Minimální uražená trasa
A	20	18 168
B	48	33 526
C	16	291

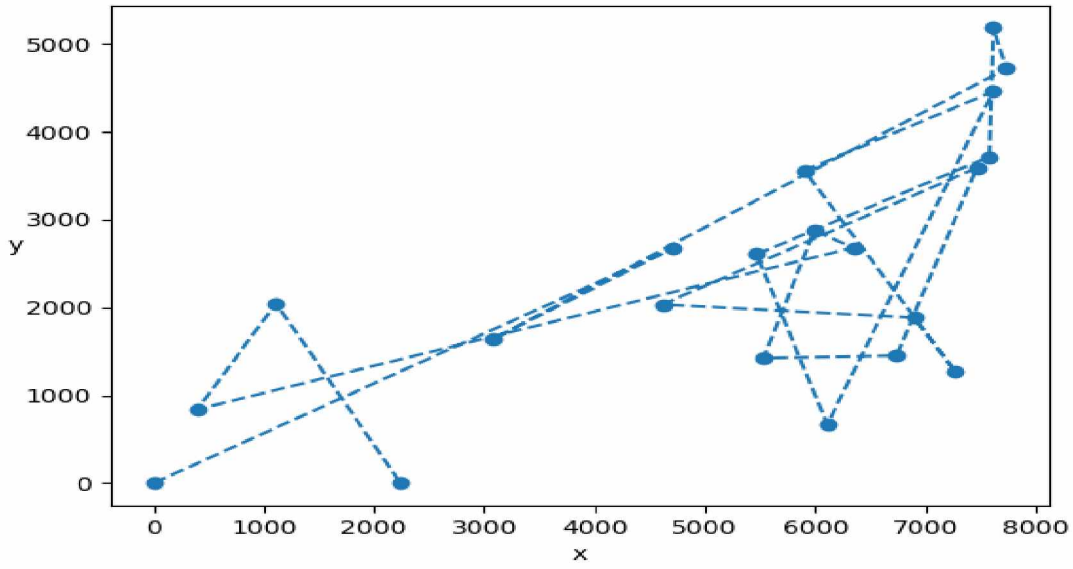
Na obrázcích 7.4, 7.5 a 7.6 jsou graficky znázorněny trasy z tabulky 7.1. Je vidět, že nejkratší trasa byla nalezena při penalizaci 10. U penalizace 500 je to téměř náhodný soubor čar, což je způsobeno tím, že se algoritmus snaží uspokojit omezující podmínky. Na obrázcích 7.7, 7.8 a 7.9 jsou nalezené trasy pro datový soubor B. Na obrázcích 7.10, 7.11 a 7.12 jsou nalezené trasy pro datový soubor C. U datového souboru C jsou dvě hodnoty velice blízké souřadnicím $x = 0$ a $y = 0$. Proto se zdá že obchodní cestující navštívil dané město vícekrát. Pro všechny nalézané trasy byl použit vzorec natrénovaný na datovém souboru A.



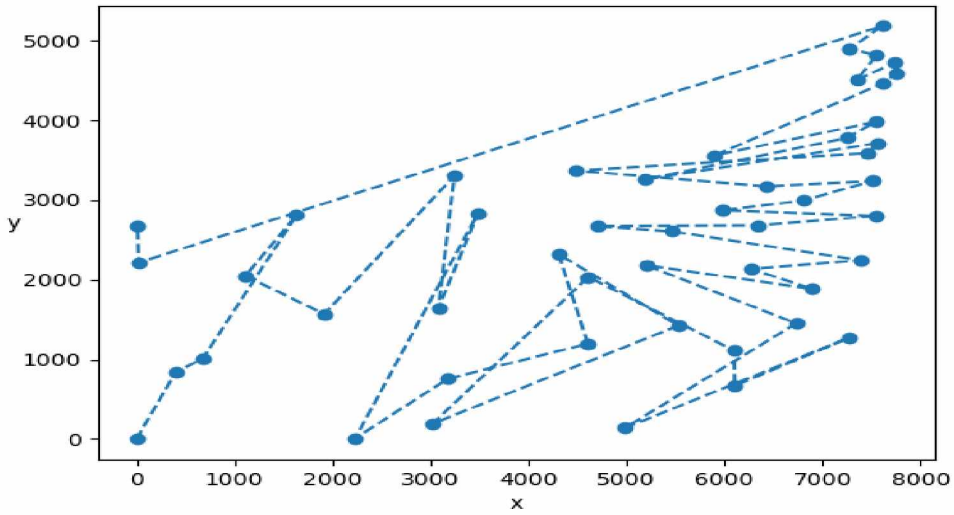
Obrázek 7.4 – Nalezená trasa při penalizaci 10 u datového souboru A



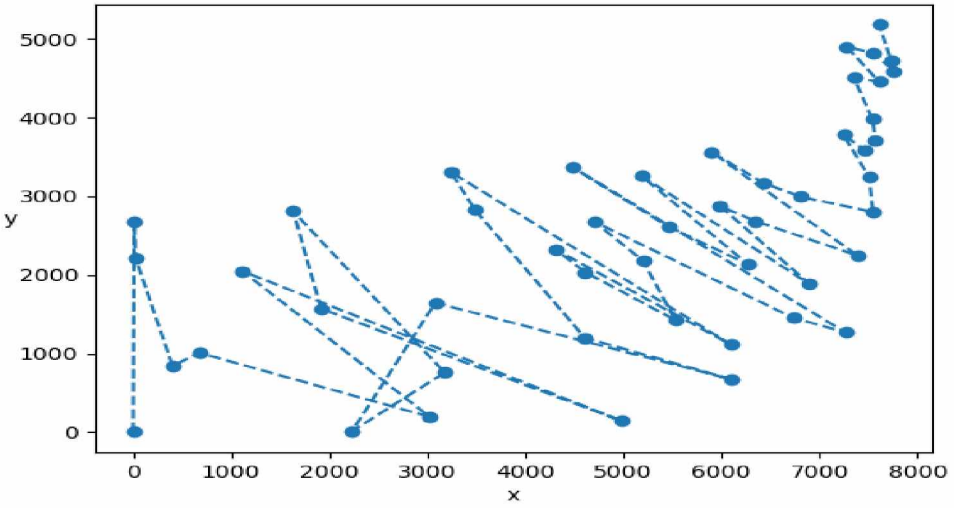
Obrázek 7.5 – Nalezená trasa při penalizaci 100 u datového souboru A



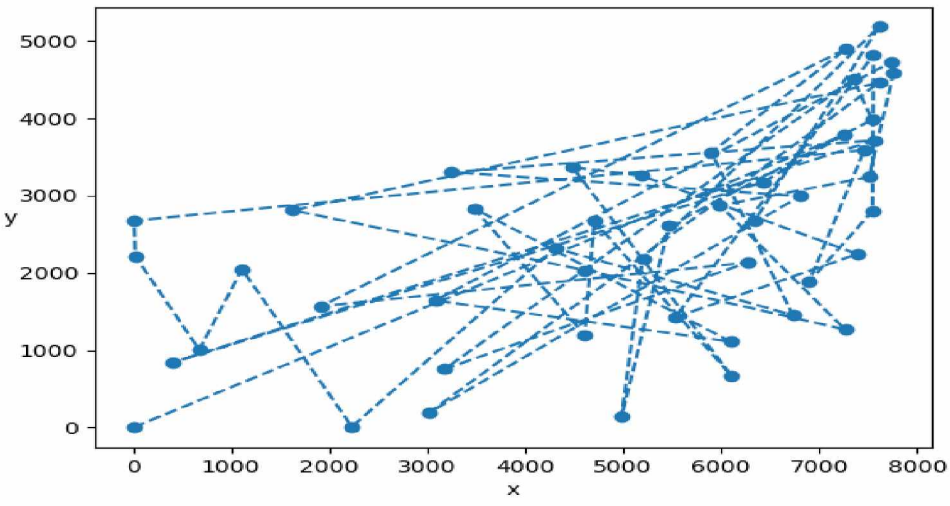
Obrázek 7.6 – Nalezená trasa při penalizaci 500 u datového souboru A



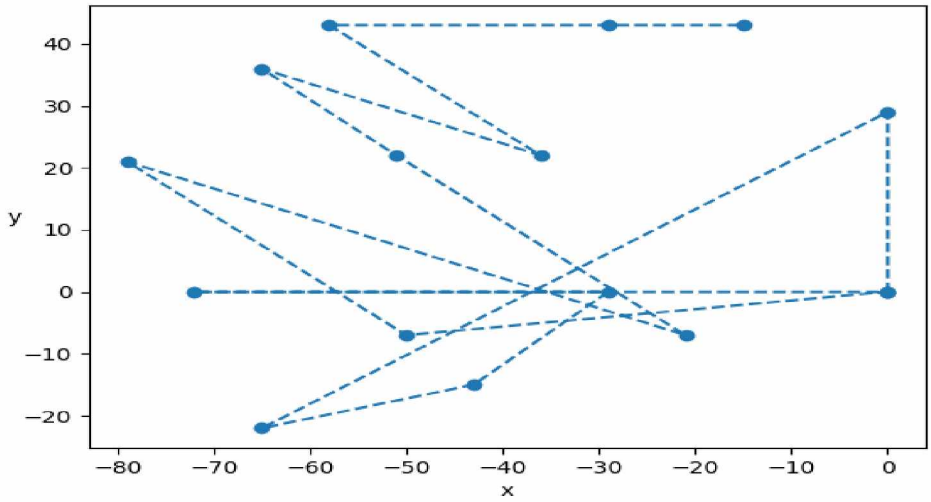
Obrázek 7.7 – Nalezená trasa při penalizaci 10 u datového souboru B



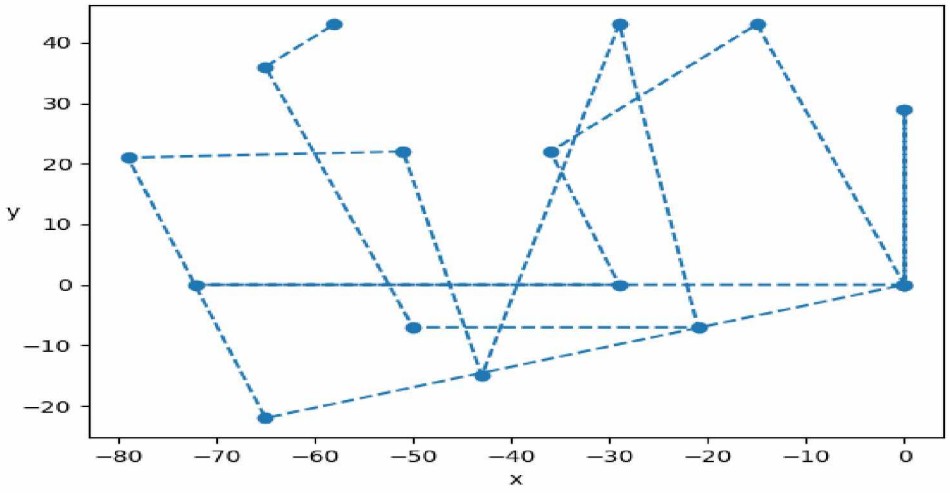
Obrázek 7.8 – Nalezená trasa při penalizaci 100 u datového souboru B



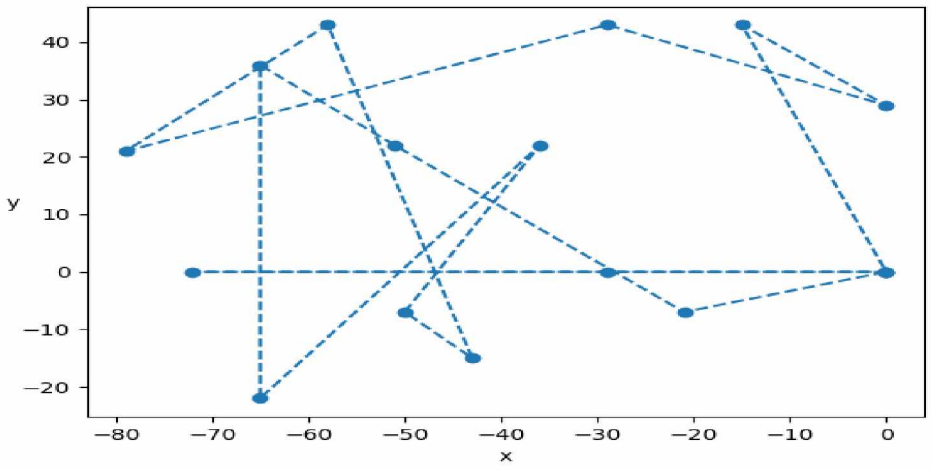
Obrázek 7.9 – Nalezená trasa při penalizaci 500 u datového souboru B



Obrázek 7.10 – Nalezená trasa při penalizaci 10 u datového souboru C



Obrázek 7.11 – Nalezená trasa při penalizaci 100 u datového souboru C



Obrázek 7.12 – Nalezená trasa při penalizaci 500 u datového souboru C

8 ZÁVĚR

Aplikace genetických programů pro hledání řešení kombinatorických úloh je poměrně účinnou metodou, jak nalézt optimální řešení. V technické praxi je jejich využitelnost omezena velikostí prohledávaného prostoru, z důvodu velké časové náročnosti při hledání řešení. Namísto genetických programů se používají heuristické postupy, které dokáží najít optimální řešení pouze na typizované úlohy. Předkládaná práce navrhuje dekódování prohledávaného prostoru, ve kterém se nachází řešení, do prostoru heuristických postupů, které nalézají řešení. Konkrétně popisuje hyperheuristický přístup pro hledání řešení problému obchodního cestujícího s časovými okny.

Práce je členěna do několika oddílů. Nejprve jsou popsány genetické algoritmy a postupy, které genetické programy využívají. Poté je práce rozšířena o genetické programování v rozsahu odpovídajícímu použití v práci. Následují popisy rozšiřující teorie obchodního cestujícího, asymptotické složitosti problému, heuristik, metaheuristik a hyperheuristik.

V praktické části práce je provedena aplikace hyperheuristického přístupu za použití algoritmů genetických programů na úlohu obchodního cestujícího s časovými okny. Je zde vysvětlena problematika jednotlivých částí genetického programu a jeho konkrétní implementace. Prováděné experimenty nejprve umožnily posouzení významu dílčích heuristik k nalézání optimálního řešení, viz kapitola 6. Po optimalizaci algoritmu genetického programu bylo provedeno nalezení optimálního heuristického postupu na trénovací množině a výsledný postup byl aplikován na testovací množiny dat.

Konkrétní závěry výsledného heuristického postupu jsou popsány v kapitole 7, nicméně lze konstatovat, že nalezený heuristický postup dokáže najít cestu blížící se optimu na trénovací množině a relativně dobrou cestu na testovacích množinách dat. Vynucení dodržení omezujících podmínek pomocí penalizace výsledné fitness funkce se také ukázalo úspěšné. Výsledkem práce je navrhnutý enkodér pro dekódování prohledávaného prostoru algoritmem genetického programu za použití hyperheuristických postupů.

Možným rozšířením práce může být penalizace přímo skóre města, které nesplnilo omezující podmínky, nebo volba jiných dílčích heuristik, které hyperheuristický postup používá. Dále může být přidáno více obchodních cestujících a ústředí, ze kterého cestující začínají svou cestu. Za zvážení stojí i změna algoritmu genetického programu, z niž některé jsou popsané v kapitole 4, nebo výměna typu genetického programu, které jsou zmíněné

v kapitole 2.1. Protože program je relativně časově náročný, dal by se rozšířit o implementaci vláken. Vlákna by mohla obstarávat genetické operace a přispívat k zrychlení tvorby populace nebo by mohla paralelně vypočítávat skóre měst. Zajímavým rozšířením by bylo přidat jednotlivým městům atribut ve formě výdělku obchodního cestujícího v daném městě. Zmíněný výdělek by přidal omezující podmínku a rozhodnutí, zda se vyplatí cestovat delší trasu za větším výdělkem a tím zvýšit náklady na dopravu, nebo navštívit více měst s menším jednotlivým výdělkem. Rozšíření by mohlo být i zastavení obchodního cestujícího v daném městě po určitý čas.

POUŽITÁ LITERATURA

ADAMCHIK, V, S. 2009. *Algorithmic Complexity* [online]. CMU, [cit. 27.4.2020]. Dostupné z: https://www.cs.cmu.edu/~adamchik/15-121/lectures/Algorithmic%20Complexity/complexity.html

APARNAA, S. K.; KOUSALYA, K. 2014. An Enhanced Adaptive Scoring Job Scheduling algorithm for minimizing job failure in heterogeneous grid network, In *2014 International Conference on Recent Trends in Information Technology,* [online], Chennai India, 10. – 12. 4. 2014 [online]. ISBN 978-1-4799-4989-2. [cit. 19.3.2020]. Dostupné z: https://www.researchgate.net/publication/286679739_An_Enhanced_Adaptive_Scoring_Job_Scheduling_algorithm_for_minimizing_job_failure_in_heterogeneous_grid_network

BRANKE, J.; et al. 2018. Algorithms for the multi-objective vehicle routing problem with hard time windows and stochastic travel time and service time. In *Applied Soft Computing*. [online], p. 66 – 79. [cit. 19.3.2020]. Dostupné z: https://www.sciencedirect.com/science/article/abs/pii/S1568494618302990

BURKE, E. K.; et al. 2010. A Classification of Hyper-Heuristic Approaches. In *Handbook of Metaheuristics,* [online]. Cham: International Publishing 2019, p. 453 – 477. [cit. 25.3.2020]. Dostupné z: http://link.springer.com/10.1007/978-3-319-91086-4_14

DEB, K.; GOLDNERG, D. E. 1989. An investigation of niche and species formation in genetic function optimization. In *Proceedings of the 3rd International Conference on Genetic Algorithms.* p. 42 – 50. ISBN 978-1-55860-066-9

DE JONG, K. A. 1975. *Analysis of the behavior of a class of genetic adaptive systems.* Dizertační práce. Michigan, 266 p. University of Michigan. Vedoucí práce John H. Holland

EBID, A. M.; 2016. Image compresion using genetic progreming. *International journal of emerging trends and technology in computer science* [online], č. 5. ISSN 2278-6856. Dostupné z: https://www.researchgate.net/publication/308632607_IMAGE_COMPRESSION_USING_GENETIC_PROGRAMMING

GOLDBERG, D, E.; RIDCHARDSON, J. 1987. Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic algorithms and their application*. Erlhaum Associates, 1987, p. 41 – 49. ISBN: 978-0-8058-0158-3

HOLLAND, J, H. 1975. *Adaptation in natural and artificial systems*. Michigan : University of Michigan Press, 182 s. ISBN 0472084607

HYNEK, J. 2008. *Genetické algoritmy a genetické programování*. Praha : Grada, 182 s. ISBN 978-80-247-2695-3

KONAK, A.; et al.; 2006. Multi-objective optimization using genetic algorithms: A tutorial. In *Reliability Engineering & System Safety* [online]. Elsevier, p. 992 – 1007. [cit. 11.5.2020]. Dostupné z: http://www.inf.unioeste.br/~adair/MOA/Artigos/MO/Multiobjective%20Optimization%20Using%20Genetic%20Algorithms%20-%20Konak.pdf

KOZA, J. R. 1998. *Genetic programing: on the programing of computers by means of natural selction.* Cambrige : MIT Press, 609 s. ISBN 0-262-11170-5

LANGDON, W, B.; POLI, R. 1998. *Better trained ants for genetic programing* [online]. University of Birmingham, School of Computer Science [cit. 11.5.2020]. Dostupné z: https://www.researchgate.net/publication/2583111_Better_Trained_Ants_for_Genetic_Programming

LANGDON, W, B. 2000. Size fair and homologous tree genetic programing crossover. In *Programing and Evolvable Machines* [online]. p. 95 – 119. [cit. 11.5.2020]. ISSN 1389-2576 Dostupné z: http://www0.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/WBL_fairxo.pdf

MAASHI, M, S. 2017. *Multi-Objective Hyper-Heuristics, Heuristics and Hyper-Heuristics - Principles and Applications* [online]. IntechOpen, [cit. 27.4.2020]. Dostupné z: https://www.intechopen.com/books/heuristics-and-hyper-heuristics-principles-and-applications/multi-objective-hyper-heuristics

MCDERMOTT, J. 2019. Why Is Auto-Encoding Difficult for Genetic Programming?. In *Sekanina L., Hu T., Lourenço N., Richter H., García-Sánchez P. (eds) Genetic Programming* [online]. Cham: Springer International Publishing, p. 131 – 145 [cit. 19.3.2020]. Dostupné z: http://link.springer.com/10.1007/978-3-030-16670-0_9

MILLER, J, F. 1999. An empirical study of the efficiency of learning {Boolean} functions using a Cartesian Genetic Programming approach. In *Proceedings of the Genetic and Evolutionary Computation Conference*. p. 1135 – 1142. ISBN 978-1558606111

MITCHELL, M. 2002. *An introduction to genetic algorithms.* Cambrige : MIT press, 209 s. ISBN 978-0-262-63185-3

NGUYEN, S.; et al.; 2019. Genetic Programming for Job Shop Scheduling. In *Evolutionary and Swarm Intelligence Algorithms* [online]. Cham: International Publishing 2019, p. 143 – 167 [cit. 16.4.2020] DOI: 10.1007/978-3-319-91341-4_8. ISBN 978-3-319-91339-1. Dostupné z: http://link.springer.com/10.1007/978-3-319-91341-4_8

POLI, R.; LANGDON, W. B.; MCPHEE, N. F. 2008. *A field guide to genetic programing.* (USA): Lulu Press, 236 s. ISBN 978-1-4092-0073-4

STÜHLE, T. 1999. *Local Search Algorithms forCombinatorial Problems—Analysis, Algorithmsand New Applications*. DISKI, 203 s. ISBN 978-15-860-3119-0.

UMBARKAR, A. J.; SHETH, P. D. 2015. Crossover operators on genetic algorithms: a review. *Ictact journal on soft computing* [online]. [cit. 19.3.2020], č. 6, p. 1083 – 1092. ISSN 2229-6956. Dostupné z: https://www.researchgate.net/publication/288749263_CROSSOVER_OPERATORS_IN_GENETIC_ALGORITHMS_A_REVIEW.

YANG, S. 2003. Statistic-based adaptive non-uniform crossover for genetic algorithms. In *Genetic and evolutionary computation--GECCO 2003: Genetic and Evolutionary Computation Conference*, [online]. Chicago (USA), 12.-16.7.2003 [cit. 19.3.2020], ISBN 978-3-540-40603-7. Dostupné z: https://www.dora.dmu.ac.uk/xmlui/bitstream/handle/2086/13826/UKCI02.pdf?sequence=1&isAllowed=y ISSN 2229-6956.

PŘÍLOHY

A – CD

Příloha A

Příloha k diplomové práci

Enkodéry v genetickém programování

Václav Hrbek

CD

Obsah

1. Text diplomové práce ve formátu PDF
2. Úplný zdrojový kód aplikace
3. Datové soubory