



**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**  
BRNO UNIVERSITY OF TECHNOLOGY

**FAKULTA INFORMAČNÍCH TECHNOLOGIÍ**  
FACULTY OF INFORMATION TECHNOLOGY

**ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ**  
DEPARTMENT OF COMPUTER SYSTEMS

**EVOLUČNÍ NÁVRH POHYBUJÍCÍCH SE OBJEKTŮ**  
EVOLUTIONARY DESIGN OF MOVING OBJECTS

**BAKALÁŘSKÁ PRÁCE**  
BACHELOR'S THESIS

**AUTOR PRÁCE**  
AUTHOR

**JAKUB FAJKUS**

**VEDOUCÍ PRÁCE**  
SUPERVISOR

**Ing. MICHAL BIDLO, Ph.D.**

**BRNO 2018**

## **Abstrakt**

Cílem této práce je návrh a implementace nástroje pro automatizované hledání vhodného kontroléru robotického modelu pro průchod prostorem po dané trajektorii. Pro návrh vhodného programu je použita technika zjednodušeného Lineárního Genetického Programování. Pro simulaci pohybu robotického modelu byl použit simulátor MuJoCo. Účinnost softwarového řízení je experimentálně doložena s použitím dvou robotických modelů na dvou trajektoriích.

## **Abstract**

The aim of this work is to implement a framework which will be used to find a computer program that will control a robotic model in a simulation. The model is supposed to move along specified trajectory. For this purpose a technique of simplified Linear Genetic Programming is used. The MuJoCo simulator is used to simulate the robotic model. The used approach is evaluated by performing experiments with two robotic models and two trajectories.

## **Klíčová slova**

Evolutionary computation, Lineární Genetické Programování, Robotika

## **Keywords**

Evolutionary computation, Linear Genetic Programming, Robotics

## **Citace**

FAJKUS, Jakub. *Evoluční návrh pohybujících se objektů*. Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Michal Bidlo, Ph.D.

# **Evoluční návrh pohybujících se objektů**

## **Prohlášení**

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

.....  
Jakub Fajkus  
2018-05-09

## **Poděkování**

V této sekci je možno uvést poděkování vedoucímu práce a těm, kteří poskytli odbornou pomoc (externí zadavatel, konzultant, apod.).

# Obsah

<b>1</b>	<b>Úvod</b>	<b>3</b>
<b>2</b>	<b>Evoluční algoritmy</b>	<b>4</b>
2.1	Koncept evolučních algoritmů . . . . .	4
2.1.1	Reprezentace . . . . .	5
2.1.2	Fitness funkce . . . . .	5
2.1.3	Populace . . . . .	5
2.1.4	Selekce . . . . .	5
2.1.5	Křížení . . . . .	6
2.1.6	Mutace . . . . .	6
2.1.7	Ukončovací podmínka . . . . .	7
2.2	Genetické programování . . . . .	7
2.3	Lineární genetické programování . . . . .	8
<b>3</b>	<b>Zkoumání pohyblivých objektů</b>	<b>10</b>
3.1	Návrh pohyblivých struktur pomocí L-systémů . . . . .	10
3.2	Evoluce bipedie s použitím LGP . . . . .	11
3.3	Vlastní roboti v prostředí simulátoru MuJoCo . . . . .	11
<b>4</b>	<b>Evoluční návrh pohyblivých objektů</b>	<b>14</b>
4.1	Relizace řízení modelu . . . . .	14
4.1.1	Interpret . . . . .	14
4.1.2	Podprogramy . . . . .	15
4.1.3	Hodnoty vstupních registrů . . . . .	15
4.1.4	Výstupní registry . . . . .	16
4.1.5	Simulátor . . . . .	16
4.2	Instance evolučního algoritmu . . . . .	17
4.2.1	Reprezentace . . . . .	17
4.2.2	Fitness funkce . . . . .	18
4.2.3	Populace a selekce . . . . .	18
4.2.4	Křížení . . . . .	18
4.2.5	Mutace . . . . .	19
4.2.6	Ukončovací podmínka . . . . .	19
<b>5</b>	<b>Experimenty</b>	<b>20</b>
5.1	Přímka . . . . .	20
5.1.1	Model trojnožky . . . . .	20
5.1.2	Model mravence . . . . .	21

5.2	Spirála . . . . .	21
5.2.1	Model trojnožky . . . . .	21
5.2.2	Model mravence . . . . .	22
<b>6</b>	<b>Výsledky experimentů</b>	<b>24</b>
6.1	Přímka . . . . .	24
6.1.1	Model trojnožky na přímce . . . . .	24
6.1.2	Model mravence na přímce . . . . .	27
6.2	Spirála . . . . .	29
6.2.1	Model trojnožky na spirále . . . . .	29
6.2.2	Model mravence na spirále . . . . .	31
<b>7</b>	<b>Závěr</b>	<b>32</b>
<b>Literatura</b>		<b>33</b>
<b>A</b>	<b>Jak pracovat s touto šablonou</b>	<b>34</b>
[[shrnuti kazde kapitoly!]] [[ukazka trajektorie spatneho genomu]] [[vlna!]] [[dat obrazky vedle sebe, asy se usetrilo misto!]] [[zvyraznit terminy, jako napr. main, event, init]]		

# Kapitola 1

## Úvod

Vygenerovano: 2018-05-09 14:12:32+02:00

Robotika představuje v současné době velmi aktuální oblast aplikace výpočetní techniky, jejímž cílem je řízení nejrůznějších elektromechanických systémů. Uplatnění nachází roboti zejména v průmyslu, kde usnadňují a zefektivňují namáhavou a opakující se práci člověku, ale také je populární oblast autonomních robotů používaných např. pro operace v těžko přístupných nebo nebezpečných místech (např. ponorky, podpora záchrannářům apod.). Jedním ze zásadních problémů konstrukce robotů obecně je návrh jejich efektivního řízení ve vztahu ke konkrétnímu typu robota a podmínkám jeho nasazení [1].

Pro tyto potřeby je vhodné mít k dispozici prostředky, které nám umožní rychle a levně prototypovat různé koncepty řízení robotů. Jednou z možností, jak řadiče robotů navrhovat, je použití evolučních algoritmů. Využití evolučních algoritmů osvobozuje konstruktéry od řešení malých detailů při návrhu kontroléru robota a v mnoha případech může evoluce nalézt řešení, které by lidského konstruktéra nenapadlo.

Řízení robota je ovšem možné realizovat mnoha způsoby, např. umělými neuronovými sítěmi [7], konenčními automaty [4] nebo i programy v imperativním jazyce [9]. V závislosti na vybrané metodě řízení robota jsou k dispozici různé algoritmy a automatizované nástroje pro nalezení konfigurace zvoleného způsobu řízení, které bude mít za následek požadované chování robota. Touto konfigurací může být např. váhy neuronové sítě, definice konečného automatu nebo sekvence příkazů imperativního jazyka. Právě řízením robota programy napsanými v imperativním jazyce (byť jednoduchém) se zabývá tato práce, ve které jsou použity evoluční algoritmy pro automatizované hledání vhodného řízení robota.

Cílem této práce je návrh a implementace nástroje pro automatizované hledání vhodného kontroléru robotického modelu pro průchod prostorem po dané trajektorii. Pro návrh vhodného programu je použita technika zjednodušeného Lineárního Genetického Programování [2]. Pro simulaci pohybu robotického modelu byl použit simulátor Mujoco [8]. Účinnost softwarového řízení je experimentálně doložena s použitím dvou robotických modelů na dvou trajektoriích.

# Kapitola 2

## Evoluční algoritmy

V této kapitole si nejpre v sekci 2.1 představíme základní koncept evolučních algoritmů. Poté následuje v sekci 2.2 stručný úvod ke Genetickému Programování (GP), na který naváže sekce 2.3 o Lineárním Genetickém Programování (LGP).

### 2.1 Koncept evolučních algoritmů

Informace v této sekci vycházejí z [3].

Evoluční algoritmy (EA) jsou inspirovány přírodními evolučními procesy a Darwinovou teorií evoluce. EA se využívají ke stochastickému prohledávání stavového prostoru. EA, na rozdíl od jiných metod, pracují s celou populací kandidátních řešení (jedinců), které se vyvíjí paralelně. Každý jedinec v populaci v sobě nese zakódovanou informaci o konkretním řešení, kterou nazýváme genotyp. Genotyp se poté dekóduje na fenotyp, který už reprezentuje řešení daného problému.

Myšlenku EA můžeme popsat následovně. Máme populaci jedinců, kteří jsou všichni umístěni ve společném prostředí, ve kterém soutěží o zdroje. To má za následek přirozený výběr jedinců, který se projevuje tak, že horší jedinci mají menší pravděpodobnost reprodukce, než-li ti lepší. Tímto se přirozeně zvyšuje kvalita populace. Reprodukce probíhá dvojím způsobem, a to křížením a mutací. Křížení pracuje se dvěma rodiči a má za následek vytvoření dvou potomků, kteří vznikají kombinací genotypu obou rodičů. Mutace pracuje nad jedním rodičem a produkuje jednoho potomka, který má narozdíl od svého ročice lehce pozměněný genotyp.

Činnost obecného EA můžeme vidět na algoritmu 1:

#### Algoritmus 1: Obecný evoluční algoritmus

```
inicializuj populaci náhodně vygenerovanými jedinci;  
vyhodnoť všechny jedince;  
while (není splněna ukončující podmínka) do  
    vyber rodiče;  
    aplikuj křížení na dvojice rodičů;  
    mutuj potomky;  
    vyhodnoť kvalitu potomků;  
    vyber jedince do další generace;  
end
```

### 2.1.1 Reprezentace

Při návrhu řešení problému je často nutné abstrahovat reálný svět tak, aby jsme vytvořili prostředí, ve kterém budou existovat kandidátní řešení a ve kterém budou tato řešení vyhodnocována. Kandidátní řešení problému nazýváme fenotypy. Zakódované fenotypy nazýváme genotypy. Termín reprezentace se používá ve dvou kontextech. V prvním kontextu specifikuje mapování z fenotypu na genotyp a je synonymem pro kódování. V druhém kontextu označuje spíše strukturu prostoru genotypů.

### 2.1.2 Fitness funkce

Úlohou fitnes funkce je reprezentování požadavků, které by populace měla splňovat. Tvoří základ pro funkci selekčních operátorů. Můžeme mluvit o zobrazení z jedince v prostoru genotypů na reálné, nebo celé, číslo. Z technického hlediska se jedná o funkci, která měří míru kvality genotypu a přiřazuje mu tzv. fitness. Výpočet fitness funkce zahrnuje dekódování genotypu na fenotyp a následné vyhodnocení fenotypu, jako kandidátního řešení pro daný problém.

### 2.1.3 Populace

Úlohou populace je obsahovat množinu kandidátních řešení — jedná se o multimnožinu jedinců. Populace je struktura, která podléhá evoluci, zatímco jedinci jsou statické struktury. Definice populace se může omezit pouze na její velikost, ale existují i specializované evoluční algoritmy, které pracují i s rozložením jedinců v prostoru.

### 2.1.4 Selekce

Operátor selekce slouží k výběru rodičů pro křížení na základě jejich kvality. Tento operátor je nejčastěji založen na náhodě a to tak, že kvalitnější řešení jsou vybárána s větší pravděpodobností. Změnou parametrů selekce nastavujeme tzv. selekční tlak, který ovlivňuje, do jaké míry jsou upřednostňováni lepsi jedinci.

Jako představitele selekce si můžeme uvést ruletovou selekci a selekci turnajem.

Při použití ruletové selekce pravděpodobnost, že jedinec  $j$  bude vybrán závisí na poměru fitness tohoto jedince a fitness celé populace:  $P_r(j) = \frac{f_j}{\sum_{i=1}^N f_i}$

Selekce turnajem, narození od ruletové selekce nevyžaduje znalost celé populace a pře-počítávání fitness hodnot. Selekce turnajem ani nevyžaduje, aby vyhodnocení fitness funkcí bylo kvantifikované, ale pro jeho funkci stačí, aby byla mezi jedinci definována operace uspořádání. Díky toho je tato metoda nenáročná na implementaci a na výpočetní čas. Základní algoritmus selekce turnajem můžeme vidět na algoritmu 2.

#### Algoritmus 2: Selekce turnajem

```
// Chceme vytvořit mating pool velikosti  $\lambda$  jedinců;
while (dokud mating pool neobsahuje  $\lambda$  jedinců) do
    Náhodně vyber  $k$  jedinců;
    Porovnej mezi sebou těchto  $k$  jedinců a vyber z nich nejlepšího jedince  $i$ ;
    Vlož jedince  $i$  do mating pool
end
```

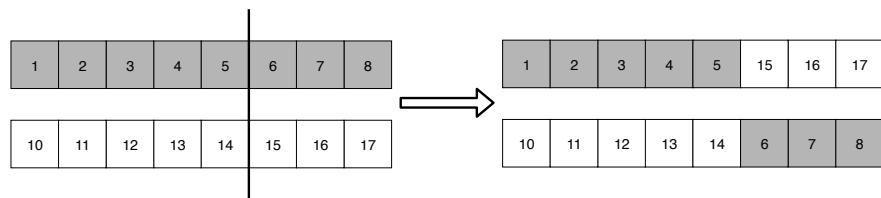
Úpravou parametru  $k$  se nastavuje selekční tlak — čím větší je parametr, tím více jedinců je vybráno pro porovnání a to má za následek vyšší selekční tlak.

### 2.1.5 Křížení

Operátor křížení slouží ke spojení částí genotypů rodičů a tím dochází k vytvoření nových jedinců. Operátor je většinou aplikován pouze s určitou pravděpodobností  $p_r$ . Nejčastěji se používá křížení pracující se dvěma rodiči. Výběr, jaké části ze kterého rodiče budou vybrány, je založen na náhodě. Myšlenka za použitím křížení je následující. Zkombinováním rodičů, kteří mají odlišné, ale vhodné vlastnosti, může vzniknout jedinec, který tyto vlastnosti v sobě kombinuje a dosahuje tak vyšší kvality.

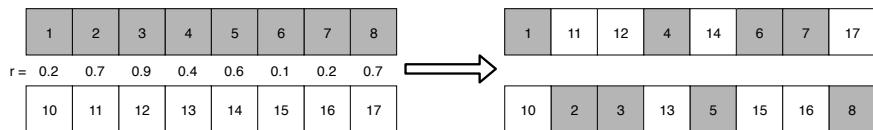
Jako představitele si můžeme uvést jednobodové a uniformní křížení.

Jednobodové křížení bylo původně představeno J. H. Hollandem v [[220]] a jeho funkce je následující. Vygeneruje se náhodné číslo  $r$  z intervalu  $\langle 1, L - 1 \rangle$ , kde  $L$  je délka genomu. Poté se genomy rozdělí v tomto bodě  $r$  na počáteční a koncovou část a noví jedinci vznikají záměnou těchto koncových částí, viz obrázek 2.1.



Obrázek 2.1: Jednobodové křížení

Uniformní křížení bylo představeno v [[422]] a pracuje následovně. Pro každý gen se vygeneruje náhodné číslo  $r$  z rovnoměrného rozložení na intervalu  $\langle 0, 1 \rangle$ . Číslo  $r$  se poté porovná s parametrem  $p$ , který bývá obvykle roven 0.5, a pokud je  $r < p$ , tak se použije gen z prvního rodiče, jinak ze druhého. Druhý potomek je vytvořen z genů, které nebyly vybrány do prvního potomka, viz obrázek 2.2.



Obrázek 2.2: Uniformní křížení

### 2.1.6 Mutace

Operátor mutace se aplikuje na jendoho rodiče a jeho výsledkem je jeden potomek. Cílem mutace je provést malou změnu genotypu. Operátor je většinou aplikován pouze s určitou pravděpodobností  $p_m$  a jeho implementace je závislá na použité reprezentaci.

Např. pro reprezentace využívající celá čísla, existují 2 základní přístupy, které mutují každý gen s pravděpodobností  $p_m$ . Prvním přístupem je úplné nahrazení aktuální hodnoty genu novou hodnotou. Tato varianta je vhodná, pokud mezi hodnotami neexistuje relace uspořádání. Druhým způsobem je malá změna aktuální hodnoty, při které se k hodnotě genu přičte nebo odečte náhodně vygenerované malé číslo. Tato varianta je vhodná, jsou-li hodnoty genomu např. parametry systému nebo funkce.

### 2.1.7 Ukončovací podmínka

Ukončovací podmínka určuje, kdy se zastaví evoluce aktuální populace. K tomuto je možné využít: dosažení předem dané fitness hodnoty, spotřebování určeného strojového času, provedení určeného počtu vyhodnocení fitness funkce, ukončení v případě, že se populace dostatečně rychle nezlepšuje nebo že pokud dojde ke snížení diverzity populace po daný limit.

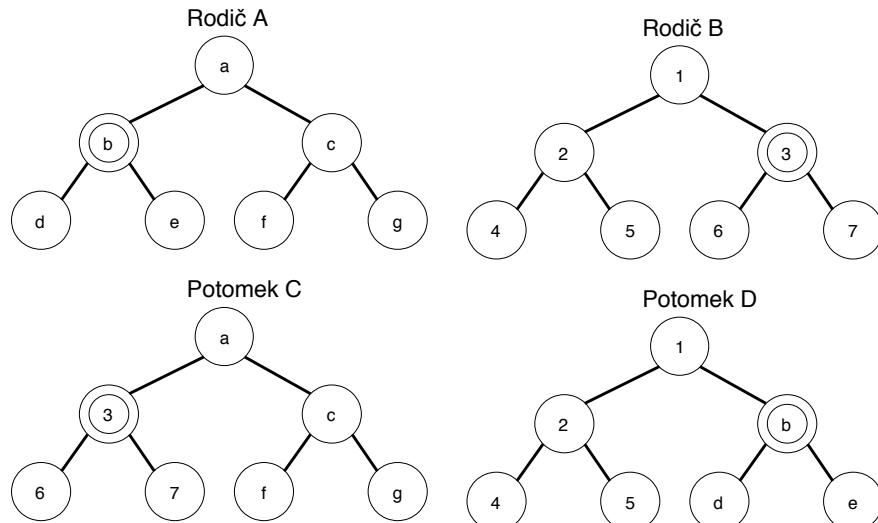
## 2.2 Genetické programování

Informace v této sekci vycházejí z [6].

Genetické programování (GP) se zabývá evolucí programu, který je reprezentován stromovou strukturou. Programy v GP jsou složeny z funkcí, které ve stromové struktuře odpovídají uzlům, a terminálů, které odpovídají listům.

Vyvíjené programy jsou typicky vyhodnocovány nad sadou vstupů, tzv. fitness cases, u kterých jsou známy požadované výstupy programu. Výsledná fitness programu se pak může počítat jako suma nebo průměr výsledků z každého "fitness case".

V GA je nejdůležitějším operátorem křížení. Křížení pracuje na základě výměny náhodně vybraných podstromů z rodičů. Toto křížení se skládá z několika kroků. Prvním krokem je výběr rodičů. Druhý krok je zvolení náhodného uzlu v každém z rodičů. Tento uzel bude kořenový uzel pro podstromy, které se budou později mezi rodiči vyměňovat. Třetím krokem je vyjmutí podstromu z obou rodičů, jejichž kořenovým uzlem je uzel, který byl vybrán v předchozím kroku. Následuje vytvoření obou potomků. První potomek vznikne tak, že se do stromu prvního rodiče, do místa dříve vybraného uzlu, vloží podstrom z druhého rodiče. Druhý jedinec vzniká obdobným způsobem, viz obrázek 2.3.



Obrázek 2.3: Křížení v GP. Na tomto obrázku je zachycena situace, kdy je v každém z rodičů A a B náhodně vybrán uzel(vyznačen dvojitě). Tyto uzly a jejich podstromy jsou poté zaměněny a vznikají tak potomci C a D.

Velikost programu, definována jako výška stromu, je omezena aby se předešlo velmi velkým programům. Pokud by potomek po křížení přesáhl tuto velikost, nebude vložen do nové generace - místo něj se do nové generace zkopíruje jeden z jeho rodičů.

Mutace v GA hraje menší roli a provádí drobné změny struktur v populaci. Mutace se stává z několika kroků: První krok je zvolení náhodného uzlu ve struktuře. Tento uzel může být vnitřní (funkce) i vnější (terminál). Druhým krokem je odstranění tohoto uzlu is celým podstromem, který je k němu připojený. Posledním krokem je vygenerování náhodného podstromu, který se poté připojí na místo odstraněného uzlu. Tato operace je řízena parametrem který říká, jakou výšku bude mít vygenerovaný podstrom. Speciálním případem mutace je operace, která vloží jeden terminál do náhodně vybraného uzlu ve stromu.

## 2.3 Lineární genetické programování

V této práci se používá přístup inspirovaný Lineárním genetickým programováním (LGP), které si stručně popíšeme v této sekci, krerá vychází z [2].

LGP je varianta GP, které býlo stručně popsáno v sekci 2.2, ve které jsou programy reprezentovány jako posloupnost instrukcí strojového kódu nebo příkazů vhodného imperativního jazyka. Obecně jsou data, zpracovávaná pomocí programu LGP, uchovávána v registrech, které jsou součástí interpretu LGP.

Interpret má k dispozici sadu registrů, jejichž počet je definován uživatelem. Tyto registry se dělí na vstupní, výstupní a pracovní registry. Vstupní registry obsahují vstupní data, se kterými je program spuštěn. Pracovní registry jsou určeny pro uložení mezinásledků výpočtů a jsou před spuštěním programu inicializovány vhodnou konstantou, např. 1. Jeden nebo více vstupních nebo pracovních registrů může být označen jako výstupní registr, ze kterého se poté po ukončení programu čte výsledek.

Programy v LGP mohou být také vykonávány bez interpretu, a to tak, že se převedou do programu v jazyce C. Ukázkový program v jazyce C můžeme vidět na obrázku 2.1.

Instrukční sada definuje programovací jazyk, ve kterém jsou vyvýjené programy napsány. LGP systém je založen na dvou základních typech instrukcí: operace a podmíněné větvení. Příklady operací mohou být: různé aritmetické operace, exponenciální funkce, trigonometrické funkce, nebo booleovské operace. Imperativní instrukce obsahuje operaci, která pracuje nad zdrojovými registry, a přiřazení výsledku do cílového registru.

V LGP je nutné zajistit, aby byly vytvářeny pouze programy, které jsou validní. Gentické operátory mutace a křížení proto musí zachovat syntaktickou správnost nově vytvořených programů. To lze zajistit například tak, že bod křížení nemůže být určen urvnitř instrukce, nebo při aplikování operátoru mutace zakážeme záměnu operátoru instrukce za registr. Pro zajištění sémantické správnosti je nutné ošetřit všechny operátory a funkce, které nejsou definovány pro všechny možné hodnoty, tak aby pro nedefinovaný výstup vraceły konstantu, např. velké celé číslo.

Schopnost genetického programování nalézt řešení velmi závisí na zvolené instrukční sadě. Kompletní instrukční sada je taková, která obsahuje všechny prvky, které jsou potřebné k vytvoření optimálního řešení za předpokladu, že je počet registrů a rozsah použitých konstant dostatečný. Na druhou stranu, rozsah stavového prostoru, který obsahuje všechny programy, které je možné sestavit z dostupných instrukcí, roste exponenciálně s počtem instrukcí a registrů.

Na programu 2.1 v jazyce C si ukážeme průběh vykonávání programu pro výpočet průměru 2 čísel. Tento program má k dispozici výstupní registr  $r[0]$ , ve kterém bude uložen výsledek, dva vstupní registry  $r[1]$  a  $r[2]$ , které obsahují vstupy programu a jeden pracovní registr  $r[3]$ . Před spuštěním programu jsou do vstupních registrů nakopírovány hodnoty a pracovní registr je inicializován na hodnotu 1. První instrukcí programu je sečtení dvou

vstupních registrů a uložení výsledku do pracovního registru. Tento pracovní registr je poté využit jako operand pro instrukci dělení, která uloží výsledek do výstupního registru r[0].

```
double r[4];  
  
void lgp(r)  
{  
    r[3] = r[1] + r[2];  
    r[0] = r[3] / 2;  
}
```

Program 2.1: LGP program v C pro sečtení 3 čísel

## Kapitola 3

# Zkoumání pohyblivých objektů

Tato práce se zabývá evolučním návrhem řízení vybraných typů pohyblivých struktur (robotů) pomocí evolučních algoritmů. V této kapitole shrneme myšlenky vycházející z relevantní literatury a představíme vlastní koncept evolučního návrhu řízení robotů.

### 3.1 Návrh pohyblivých struktur pomocí L-systémů

G. Hornby ve své disertaci [5] pomocí generativních reprezentací (L-systémů) mimo jiné vyvíjel fyzickou strukturu i řízení robotických modelů. Pro vývoj řízení byly použity 2 přístupy. První přístup byl založen na oscilujících poháněných kloubech, které měly 2 parametry – rychlosť oscilace a fázový posun. Druhý přístup byl založen na neuronových sítích, které se budovaly použitím instrukcí a které svým výstupem ovládaly jedotlivé poháněné klouby.

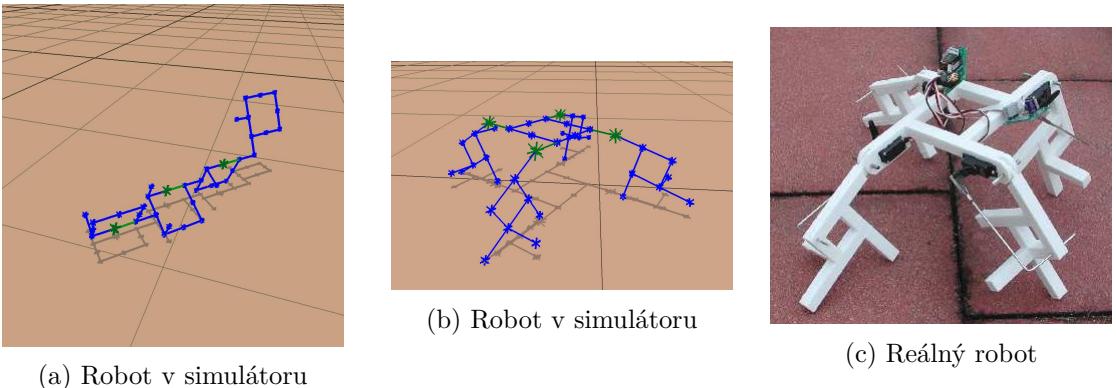
V této práci byli roboti složeni ze 3 prvků – tyče fixní délky, bloky pro spojení těchto tyčí a poháněné klouby, viz obrázek 3.1.



Obrázek 3.1: Stavební prvky robotů v práci G. Hornbyho: tyče fixní délky, bloky pro spojení těchto tyčí a poháněné klouby.

V programu, který konstruoval robota, se vyskytovaly instrukce pro budování fyzické struktury a také instukce pro konfiguraci kloubů a neuronové sítě. Tímto se struktura robota vytváří současně s vytvářením jeho řízení.

G. Hornby touto metodou vyvinul množství robotů, z nichž některé i skutečně vyrobil, viz obrázek 3.2.



Obrázek 3.2: Robot vyvinutý G. Hornbym

### 3.2 Evoluce bipedie s použitím LGP

Wolff a Wahde ve své práci [9] využili koncept LGP pro vývoj kontroléru pro humanoidního robota, který měl za úkol chodit v simulovaném prostředí. Použili zde 7 typů instrukcí: sčítání, odčítání, násobení, dělení, funkce sinus, a dvě varianty větvení. Programy měly k dispozici řadu hodnot ze senzorů, jako např. natočení jednotlivých kloubů modelu, aktuální zrychlení jednotlivých částí robota nebo relativní natočení částí těla k některé z os.

Jejich řízení bylo navrženo tak, že se v definovaných časových okamžicích simulace spustil celý program, který na základě vstupních hodnot vypočetl výstupní hodnoty, které se použily pro řízení kloubů modelu. Pro evoluci programů zde použili steady state algoritmus, selekci turnajem velikosti 4, klasické 2 bodové křížení a mutaci, která náhodně vybranou instrukci nahradí novou, náhodně vygenerovanou.

V této práci se podařilo vyvinout kontroléry, které byly schopny řídit robota tak, aby se pohyboval. Tento pohyb byl ovšem velmi pomalý (0.054 m/s), ačkoli trvalý (až 20 minut).

### 3.3 Vlastní roboti v prostředí simulátoru MuJoCo

Podobně jako v práci G. Hornbyho [5] jsou roboti v této práci složeni z pevných tyčí, které mají různou délku, bloků pro spojení těchto tyčí a poháněných kloubů. Pro účely této práce byly vytvořeny dva počítačové modely robotů, pro které bylo cílem evolučně navrhnut řídící program tak, aby robot vykazoval požadované chování. Pro tyto potřeby byl použit zjednodušený koncept LGP, který bude blíže představen v sekci 4.1, a simulátor MuJoCo<sup>1</sup>.

MuJoCo je fyzikální simulátor určený k vývoji v oblasti robotiky, biomechaniky, grafiky a simulace, strojového učení a všech dalších oblastí, ve kterých je potřeba provádět rychlé a přesné simulace komplexních dynamických systémů. Tento simulátor byl od začátku postaven tak, aby byl co nejvýkonnější a bylo jej tedy možné využít v situacích, kdy je potřeba provádět simulace mnohem rychleji, než-li v reálném čase. Simulátor je napsán jako dynamická knihovna s rozhraním v jazyce C a má zabudovanou interaktivní vizualizaci, založenou na OpenGL.

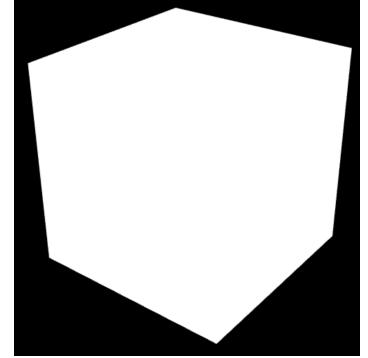
Modely pro tento simulátor se specifikují v XML souboru ve formátu MJCF, který je lidsky čitelný a je relativně jednoduché s ním pracovat. Tento soubor také obsahuje

<sup>1</sup><http://www.mujoco.org/index.html>

nastavení všech parametrů simulace. Všechny parametry mají své výchozí hodnoty, takže je není nutné pro jednoduché pokusy nastavovat. V programu 3.1 můžeme vidět ukázku minimalistického MJCF souboru definující objekt, který je vidět na obrázku 3.3. Tento MJCF soubor je při spuštění simulace přeložen na nízkoúrovňové struktury, se kterými se poté pracuje v samotné simulaci.

```
<mujoco>
  <worldbody>
    <geom type="box" pos="1 1 1" size="1 1 1"
          rgba="255 255 255 1"/>
  </worldbody>
</mujoco>
```

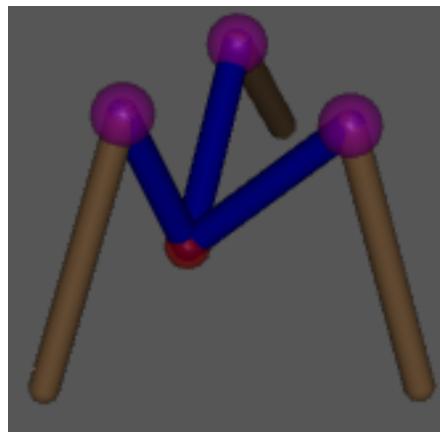
Program 3.1: Minimalistický MJCF soubor. Tento soubor definuje bílou krychli (viz obrázek 3.3) jednotkové velikosti, která je umístěna v prostoru na souřadnicích 1,1,1.



Obrázek 3.3: Výsledek MJCF souboru 3.1

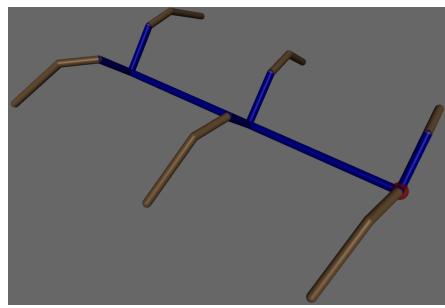
Níže si popíšeme dva modely robotů, se kterými byly provedeny 2 experimenty. Tyto experimenty zahrnovaly pohyb po přímce a spirále.

První model, zvaný trojnožka, je vidět na obrázku 3.4. Robot má 3 nohy, každá z nich je spojena s jádrem robota kloubem, který se otáčí v jedné ose. Rozsah pohybu těchto kloubů je omezen na 50 stupňů. Model robota má v horní části v místě kloubů umístěny kontaktní body. Tyto body slouží k detekci převrácení robota (kolizi ze země).



Obrázek 3.4: Model robota zvaný trojnožka. Tento model je složen z jádra robota (modře) a 3 nohou (hnědě). Každá z nohou je s jádrem spojena kloubem, který se otáčí pouze v jedné ose a je umístěn pod fialovou koulí. Tyto fialové koule slouží k detekci převrácení robota. Ve středu robota je bod zvaný hlava (červeně).

Druhý robot, zvaný mravenec, je vidět na obrázku 3.5. Robot má 3 páry nohou, které jsou všechny připojeny k tělu robota. Rozsah pohybu kloubů, které rotují kolem horizontální osy a spojují tělo robota s jeho nohou, je omezen na 100 stupňů. Rozsah kloubů, které rotují kolem svislé osy a spojují dvě části nohy, je omezen na 65 stupňů.



Obrázek 3.5: Model robota zvaný mravenec. Tento model je složen z těla robota (modře) a 6-ti nohou (hnědě). V přední části robota je bod zvaný hlava (červeně). Každá noha je složena ze dvou pevných částí a jednoho kloubu. Tento kloub se pohybuje pouze ve svislé ose, tj. může zvedat a snižovat robota. Celá noha je připojena k tělu robota kloubem, který se pohybuje horizontálně, tj. slouží k odrážení.

## Kapitola 4

# Evoluční návrh pohyblivých objektů

V této kapitole si nejdříve v sekci 4.1 popíšeme způsob, jakým je realizováno řízení robota a poté si v sekci 4.2 nstanci EA, která je v této práci použita.

### 4.1 Relizace řízení modelu

V simulátoru Mujoco je vytvořena scéna, která obsahuje model a množinu referenčních bodů, které svým pořadím a umístěním ve scéně definují trajektorii, kterou má modelu následovat. Tato scéna je poté využita v simulaci, ve které probíhá vyhodnocování daného robotického modelu s vyvýjeným řízením.

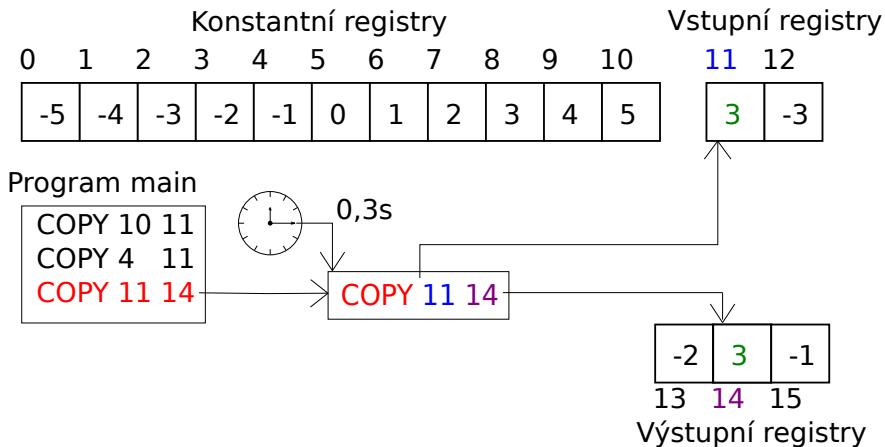
#### 4.1.1 Interpret

Program, který řídí model robota, je vykonáván v interpretu. Interpret obsahuje množinu registrů, kde každý registr je identifikovaný unikátním číslem, které se navýzvá index. Tyto registry mohou obsahovat celočíselné hodnoty v rozsahu od -5 do 5.

Pracuje se zde se třemi typy registrů. Prvním typem jsou vstupní registry, které obsahují vstupní data programu, jsou chráněné proti přepisu a v interpretu jsou 2. Dalším typem jsou konstantní registry, kterých je v interpretu 11. Tyto registry obsahují všechna celá čísla (v rozsahu od -5 do 5), které mohou být do registrů uložena, a tyto registry jsou také chráněny proti přepisu. Posledním typem jsou výstupní registry, které řídí jednotlivé klouby modelu a jejich obsah je modifikován instrukcemi. Počet těchto registrů závisí na počtu poháněných klubů modelu, pro který je program určen – pro modely, které se v této práci používají, jsou počty výstupních registrů 3 a 12.

Interpret vykonává program, který je složen z instrukcí. Je zde použit jen jeden druh instrukce, a to instrukce s názvem COPY. Tato instrukce má 2 parametry: zdrojový registr a cílový registr. Výsledkem této instrukce je zkopirování hodnoty ze vstupního, nebo konstantního, registru do výstupního registru. Interpret je schématicky znázorněn na obrázku 4.1.

V průběhu simulace je vykonáván program, který čte hodnoty ze vstupních registrů, které obsahují informaci o směru k dalšímu referenčnímu bodu, nebo konstantních registrů a zapisuje hodnoty do výstupních registrů, které se převádí na sílu, která je aplikována v jednotlivých klubech.



Obrázek 4.1: Schéma interpretu. Je zde znázorněna situace, ve které interpret vykonává instrukci z podprogramu main. Tato instrukce, s parametry 11 a 14 způsobí zkopírování hodnoty 3 ze vstupního registru s indexem 11 do výstupního registru s indexem 14.

#### 4.1.2 Podprogramy

Pro účely experimentů se spirálovou trajektorií byl navržen koncept podprogramů, který je popsán dále. V experimentech s přímkovou trajektorií je z těchto podprogramů použit jen podprogram main.

Každý program, který reprezentuje kandidátní řešení, je pro vykonávání v průběhu simulace rozdělen na 3 podprogramy. Tyto podprogramy se nazývají init, main a event.

Podprogram init se vykoná pouze na začátku simulace a všechny instrukce jsou provedeny v nulovém čase a poté je po 1 sekundě spuštěn podprogram main. Účelem podprogramu init je nastavení počátečního natočení kloubů modelu.

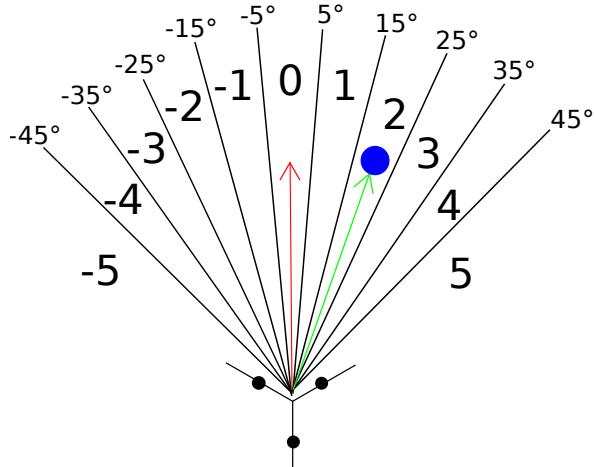
Podprogram main je v průběhu simulace vykonáván v nekonečné smyčce. Instrukce v tomto podprogramu se nevykonají všechny v nulovém čase, ale vykonávají se s časovým rozestupem 0.3 sekundy mezi každou z instrukcí. Jedná se o nejdelší a nejdůležitější podprogram.

Podprogram event se vykoná v situaci, kdy se model přiblíží do určené vzdálenosti od referenčního bodu, avšak pro každý referenční bod pouze jednou. Stejně jako u podprogramu init jsou instrukce provedeny v nulovém čase a poté je po 1 sekundě spuštěn podprogram main. Účelem podprogramu event je změna natočení kloubů modelu jako příprava k pohybu k následujícímu referenčnímu bodu.

#### 4.1.3 Hodnoty vstupních registrů

Hodnoty, které se ukládají do vstupních registrů, vychází z informace o směru k následujícímu referenčnímu bodu. Informace o směru je opět vyjádřena číslem od -5 do 5 a vypočítává se následujícím mechanismem.

Prostor kolem modelu je rozdělen na 11 kruhových výsečí, kde každá výseč je ohodnocena číslem od -5 do 5, viz obrázek 4.2. Informace o směru je rovna ohodnocení výseče, ve které se nachází další referenční bod. Tato informace je v nezměněné podobě vložena do prvního vstupního registru. Do druhého registru je vložena hodnota s převráceným znaménkem. Tyto vstupní hodnoty mohou sloužit programům ke korekci směru, kterým se robot pohybuje, v závislosti na poloze dalšího bodu, ke kterému se má tento robot přiblížit.



Obrázek 4.2: Způsob výpočtu hodnot vstupních registrů. Model (na obrázku dole) směruje směrem nahoru a další referenční bod (modře) se nachází ve výšce ohodnocené číslem 2. Hodnoty vstupních registrů budou tedy čísla 2 a -2.

#### 4.1.4 Výstupní registry

Každý z výstupních registrů interpretu odpovídá jednomu kloubu modelu. Hodnoty z výstupního registru jsou z interpretu čteny a převádí se na ovládací signály, které se předávají simulátoru. Hodnota (ovládací signál) v sobě obsahuje dvě informace. První z nich je dána znaménkem a určuje, kterým směrem bude aplikována síla v kloubu. Druhá informace je dána velikostí hodnoty a určuje velikost této síly.

#### 4.1.5 Simulátor

Simulátor poskytuje řadu funkcí, které umožňují řídit běh simulace. Tyto funkce se poté používají v uživatelském programu, ve kterém nutné z těchto funkcí sestavit algoritmus spojité simulace (ukázkové uživatelské programy jsou součástí distribuce simulátoru). Uživatel má plnou kontrolu nad během simulace, a to tím, že volá knihovní funkci `mj_step()`, která v simulaci pokročí definovaným krokem. Ve smyčce algoritmu spojité simulace jsou vloženy funkce zajišťující běh interpretu, výpočet fitness funkce, mapování výstupních hodnot z

interpretu na ovládací signály a získávání pozic objektů z dat simulace. Ve zjednodušené formě je program znázorněn na algoritmu 3.

**Algoritmus 3:** Algoritmus simulátoru

```

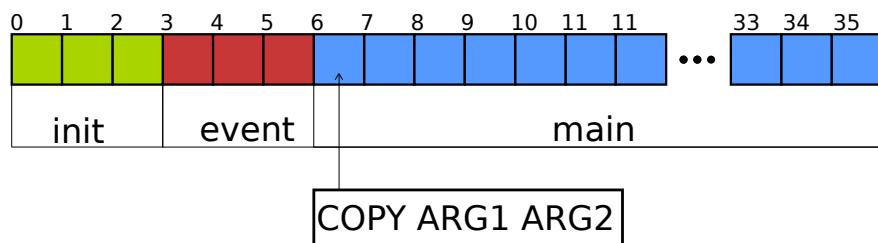
načti model ze souboru;
nastav čas simulace  $t = 0$ ;
vykonej celý program init;
while ( $t < \text{délka simulace}$ ) do
    proved další kroku simulace - mj_step();
    if (došlo k převrácení modelu) then
        | ukonči simulaci;
    end
    vypočti vstupní hodnoty interpretu;
    vykonej instrukci z programu main;
    použij výstupní hodnoty interpretu pro řízení modelu;
    zaznamenej vzdálenosti od jednotlivých referenčních bodů;
    if (došlo k přiblížení k následujícímu referenčnímu bodu) then
        | vykonej celý program event;
    end
    přičti k času  $t$  časový krok simulace;
end
vypočti fitness hodnotu;
```

## 4.2 Instance evolučního algoritmu

### 4.2.1 Reprezentace

Použitá reprezentace pracuje s genomem fixní délky, která se liší pro každý model a experiment. Tato délka bude uvedena u každého z experimentů v následující kapitole. Tato reprezentace kóduje každou instrukci jako n-tici, ve které je prvním prvkem název instrukce a poté následují její argumenty. Aktuálně jediná použitá instrukce je složena ze svého názvu (COPY, nebo i starší název SRE) a dvou celých čísel, které označují zdrojový a cílový registr.

Tato n-tice je použita jako hodnota jednoho genu. Pro experimenty se spirálovou trajektorií je celý genotyp pro účely křížení rozdělen na 3 části, které korespondují s podprogramy, viz obrázek 4.3. Pro experimenty s přímkovou trajektorií je použit jen podprogram main.



Obrázek 4.3: Rodzdělení genotypu na části pro podprogramy. Je zde znázorněn genotyp délky 36 genů, který je rozdělen po 3 genech pro podprogram init a event a 30 genů pro podprogram main. Je zde také znázorněna n-tice, která reprezentuje instrukci.

### 4.2.2 Fitness funkce

Fitness hodnota programu se počítá následující způsobem. V průběhu simulace se pro každý referenční bod  $i$  zaznamenává nejmenší vzdálenost  $D_i$  mezi tímto bodem a modelem.  $D_i = \min F(R, P_i)$ , kde  $F$  je funkce vypočítávající vzdálenost,  $R$  je pozice modelu v prostoru,  $P_i$  je pozice referenčního bodu v prostoru a  $i = 1, \dots, N$ , kde  $N$  je pocet referenčních bodů.

Po dokončení simulace je pro každý referenční bod je vypočteno skóre následujícím způsobem:

$$S_i = \begin{cases} t - D_i & \text{pokud } D_i \leq t \\ 0 & \text{jinak} \end{cases}$$

Parametr  $t$  je minimální vzdálenost, ve které musí být model od referenčního bodu, aby se skóre pro tento bod počítalo. Výsledná fitness hodnota je poté vypočtena jako suma skóre pro všechny body:

$$f = \sum_{i=1}^N S_i$$

Na základě rozměru modelu a celé scény v simulátoru byla zvolena hodnota parametru  $t = 40$ .

### 4.2.3 Populace a selekce

Pro evoluci programů byla vybrána varianta steady-state algoritmu (algoritmus 4) s velikostí populace 1000 nebo 400 jedinců. Operátor selekce je implementován jako turnaj velikosti  $k = 2$ . Nejlepší jedinec v aktuální populaci a jeho mutant jsou zkopirováni do následující populace.

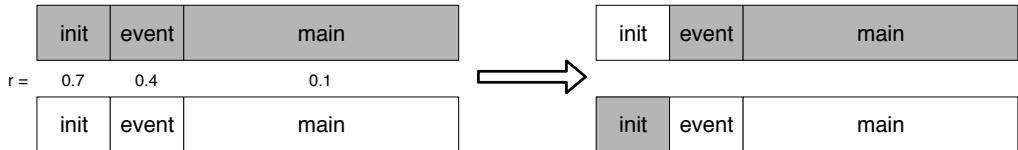
#### Algoritmus 4: Steady state algoritmus

```

nastav čas t = 0;
 inicializuj všechny chromozomy v P(t) náhodnými alelami;
 do
    vypočítj fitness hodnoty v P(t);
    do
        vyber dva jedince z rodičovské populace P(t);
        vytvoř dva potomky použitím operátoru křížení;
        aplikuj operátor mutace na oba potomky;
        vlož tyto dva potomky do dočasně populace T;
    while (v populaci T je méně než k jedinců);
    vlož jedince z T do P(t);
    odstraň k nejhorším jedincům z P(t);
    t = t + 1;
 while (není splněna ukončující podmínka);
```

### 4.2.4 Křížení

Pro účely této práce byl implementován vlastní operátor křížení, který je založen na uniformním křížení. Narození od uniformního křížení, které pracuje na úrovni genů, křížení použité zde pracuje na úrovni podprogramů. V genomu jsou tedy 3 části, které se mohou mezi rodiči vyměňovat, viz obrázek. 4.4.



Obrázek 4.4: Vlastní uniformní křížení. Zde, při křížení rodičů došlo k výměně podprogramu init.

Tento operátor se používá v experimentech se spirálovou trajektorií. Pro experimenty s přímkovou trajektorí se používá jednobodové křížení, které pracuje na úrovni jednotlivých instrukcí.

#### 4.2.5 Mutace

Operátor mutace je nastaven s  $p_m = 1$  a vždy mutuje jen jeden gen. Tento operátor je implementován následovně. Náhodně se vybere jedna instrukce z genotypu. S 20 % pravděpodobností je tato instrukce vymazána a je nahrazena novou, náhodně vygenerovanou. S 80 % pravděpodobnosti je tato instrukce upravena a to tak, že je zde 50 % pravděpodobnost na mutaci prvního, nebo druhého argumentu. Mutace argumentu je implementována zahodením aktuální hodnoty a vygenerováním hodnoty nové.

#### 4.2.6 Ukončovací podmínka

Jako ukončovací podmínka byla zvoleno ukončení evoluce po provedení daného počtu generací.

**[[zminka o implementaci programu?]]**

# Kapitola 5

# Experimenty

Pro účely experimentálního ověření funkčnosti použitých postupů byly navrženy 2 experimenty, které byly provedeny s každým modelem, tedy celkem 4 experimenty.

## 5.1 Přímka

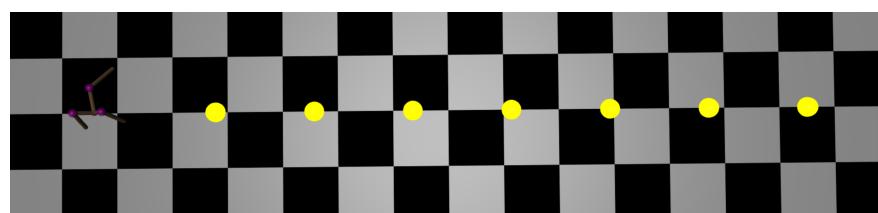
Tento experiment měl za cíl ověřit, zda-li je použitý způsob řízení modelu funkční. Úlohou modelu bylo pohybovat se po přímce, na které leželo 7 referenčních bodů. V průběhu tohoto experimentu byl laděn způsob řízení modelu a také parametry evolučního algoritmu.

### 5.1.1 Model trojnožky

V tomto experimentu byla použita následující nastavení:

- 1000 jedinců v populaci
- jednobodové křížení s pravděpodobností  $p_r = 0.8$
- ukončení běhu evoluce po 200 generacích
- 18 genů v genotypu
- pouze podprogram main
- délka simulace 80s

Snímek scény simulátoru pro tento experiment je na obrázku 5.1.



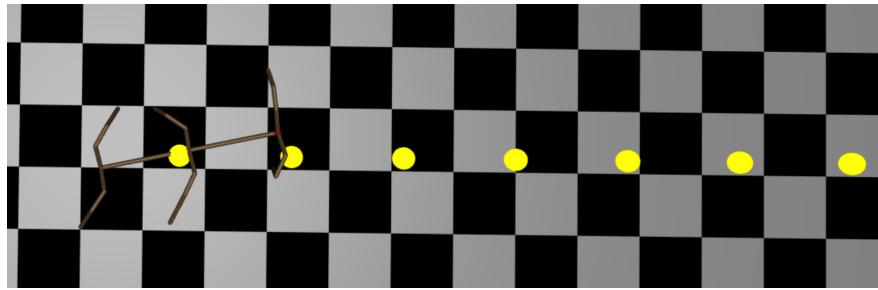
Obrázek 5.1: Scéna pro experiment s trojnožkou na přímce

### 5.1.2 Model mravence

V tomto experimentu byla použita následující nastavení:

- 400 jedinců v populaci
- jednobodové křížení s pravděpodobností  $p_r = 0.8$
- ukončení běhu evoluce po 200 generacích
- 72 genů v genotypu
- pouze podprogram main
- délka simulace 80s

Snímek scény simulátoru pro tento experiment je na obrázku 5.2.



Obrázek 5.2: Scéna pro experiment s mravencem na přímce

## 5.2 Spirála

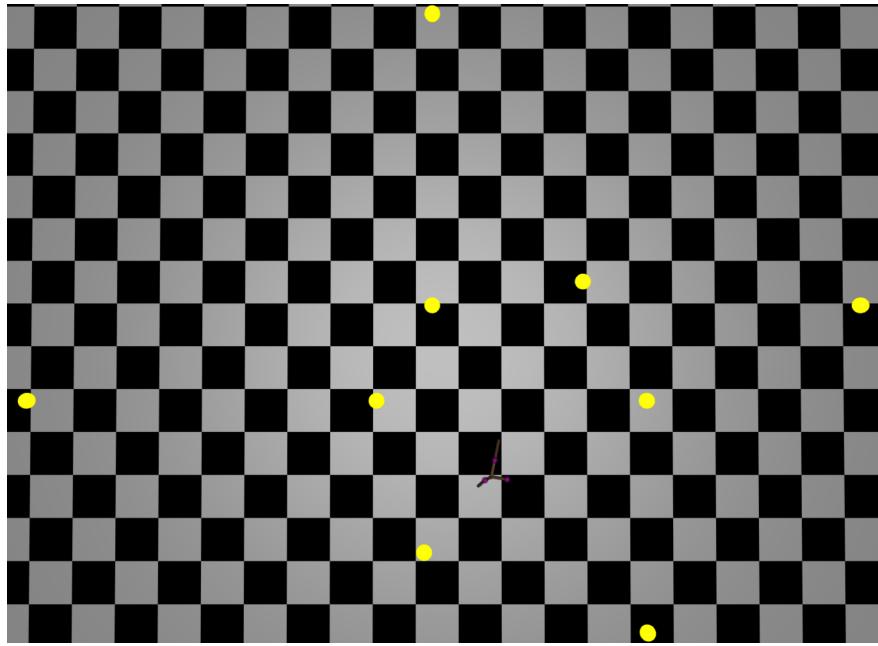
Tento experiment měl za cíl ověřit, zda-li je použitý způsob řízení modelu schopný pohybu i po netriviální trajektorii, která byla definována pomocí 9 referenčních bodů. Úlohou modelu bylo pohybovat se po spirále, na které leželo 9 referenčních bodů. V průběhu tohoto experimentu byl laděn systém podprogramů a vstupních informací pro program.

### 5.2.1 Model trojnožky

V tomto experimentu byla použita následující nastavení:

- 1000 jedinců v populaci
- podprogramové uniformní křížení s pravděpodobností  $p_r = 0.8$
- ukončení běhu evoluce po 300 generacích
- 36 genů v genotypu
- délka simulace 600s

Snímek scény simulátoru pro tento experiment je na obrázku 5.3.



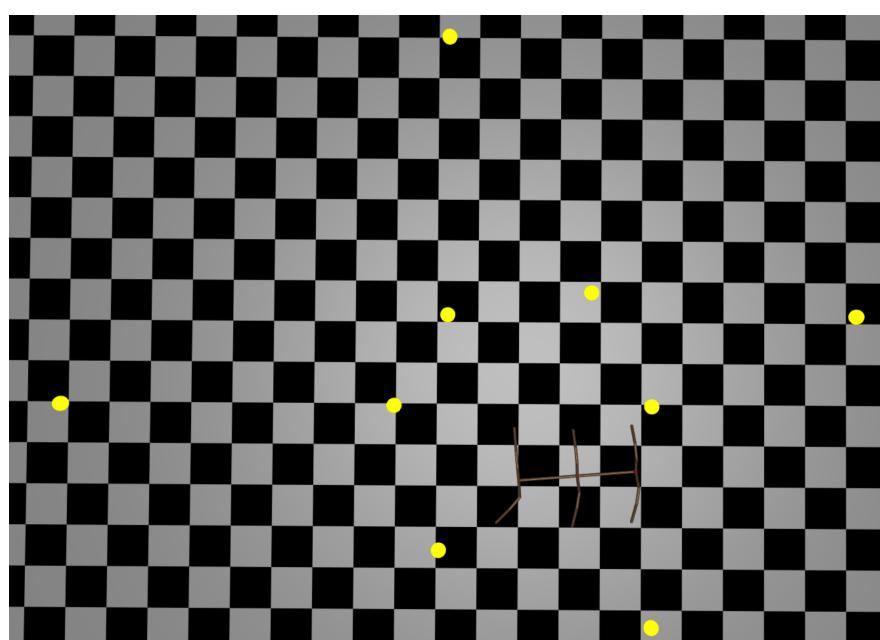
Obrázek 5.3: Scéna pro experiment s trojnožkou na spirále

### 5.2.2 Model mravence

V tomto experimentu byla použita následující nastavení:

- 1000 jedinců v populaci
- podprogramové uniformní křížení s pravděpodobností  $p_r = 0.8$
- ukončení běhu evoluce po 300 generacích
- 100 genů v genotypu
- délka podprogramů init a event je 12 instrukcí
- délka simulace 600s

Snímek scény simulátoru pro tento experiment je na obrázku 5.4.



Obrázek 5.4: Scéna pro experiment s mravencem na spirále

# Kapitola 6

## Výsledky experimentů

**[[umistit obrázky pod sebe na jednu stranu: 3 sloupce, 2 radky]]** Pro každý experiment bylo provedeno 20 evolučních běhů, ze kterých byly zaznamenány fitness hodnoty všech jedinců. Pro každý evoluční běh byly z těchto hodnot vykresleny 2 typy grafů:

- vývoj maximální a průměrné fitness hodnoty jedinců v závislosti na generaci
- box plot fitness hodnot pro vybrané generace

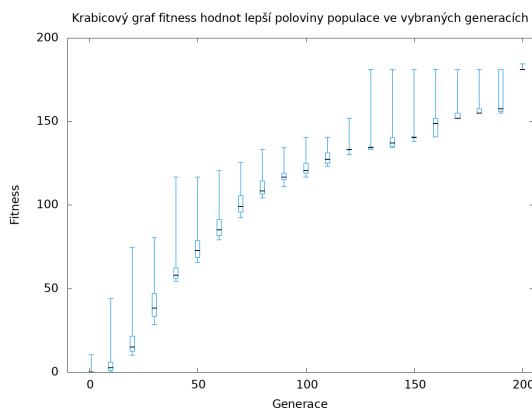
Pro každý experiment byla poté vykreslena sada histogramů, které ukazují rozložení fitness hodnot jedinců napříč všemi evolučními běhy.

V této kapitole je pro každý experiment prezentován graf vývoje průměrné a maximální fitness hodnoty v populaci a také box plot fitness hodnot pro vybrané generace pro nejlepší evoluční běh. Veškeré další grafy, které byly vygenerovány, je možné nalézt v přílohách na přiloženém médiu.

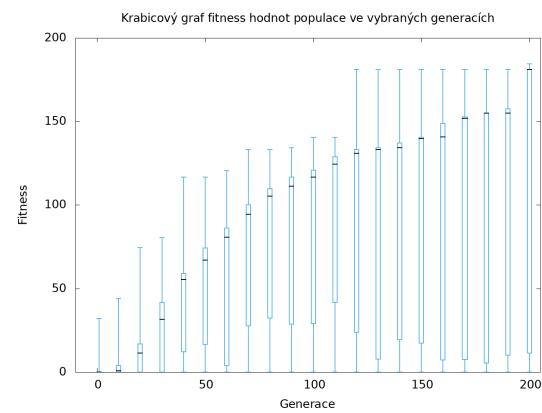
### 6.1 Přímka

#### 6.1.1 Model trojnožky na přímce

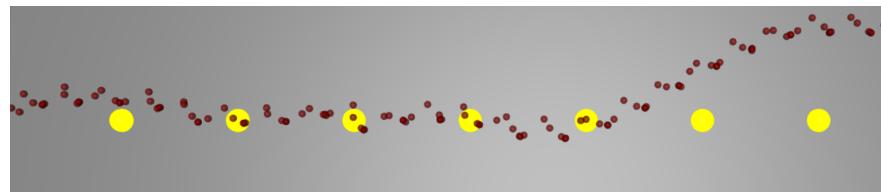
**[[možna vypsat do tabulky pro kazdy beh: nejlepsi, prumernou fitness?]] [[zduvodnit velkou variaci v hodnotach fitness]]**



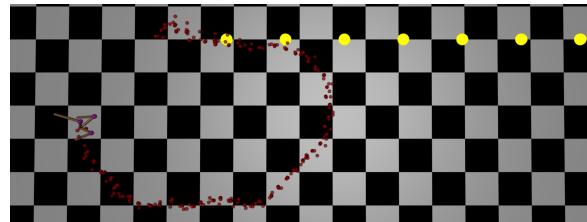
Obrázek 6.1: Box plot fitness hodnot lepší poloviny populace nejlepšího běhu experimentu trojnožky na přímce



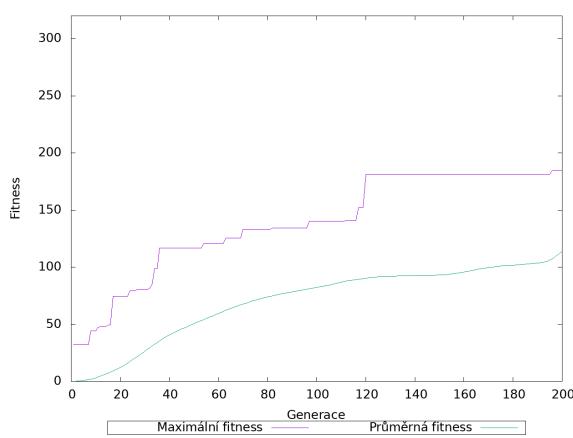
Obrázek 6.2: Box plot fitness hodnot populace nejlepšího běhu experimentu trojnožky na přímce



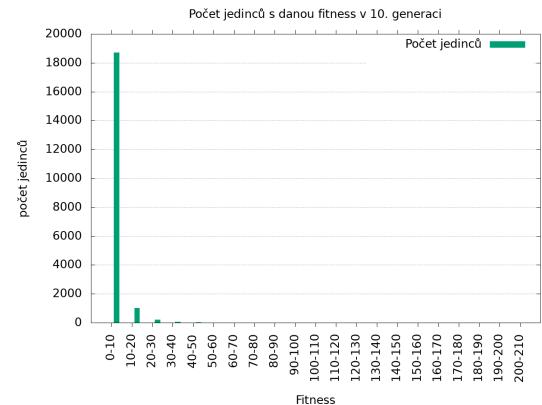
Obrázek 6.3: Trajektorie nejlepšího řešení trojnožky na přímce



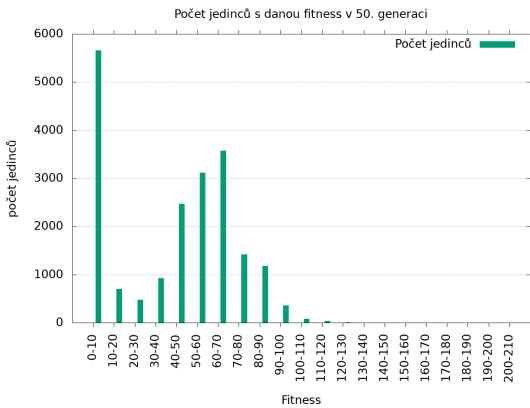
Obrázek 6.4: Trajektorie neoptimálního řešení trojnožky na přímce



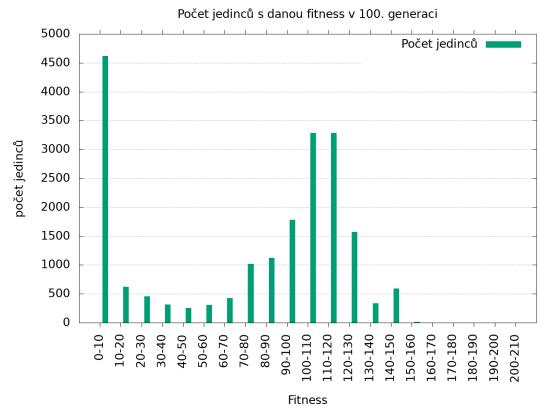
Obrázek 6.5: Nejlepší běh pro experiment s trojnožkou na přímce



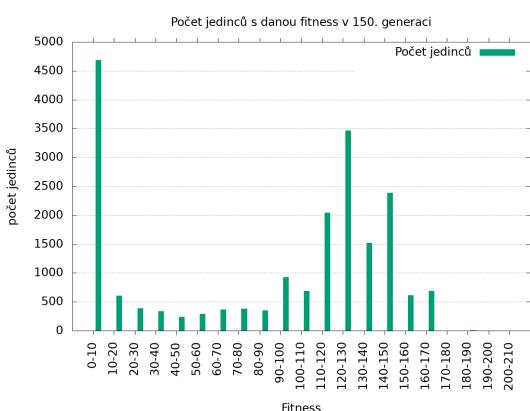
Obrázek 6.6: Histogram celkového počtu jedinců ve všech bězích trojnožky na přímce v 10. generaci



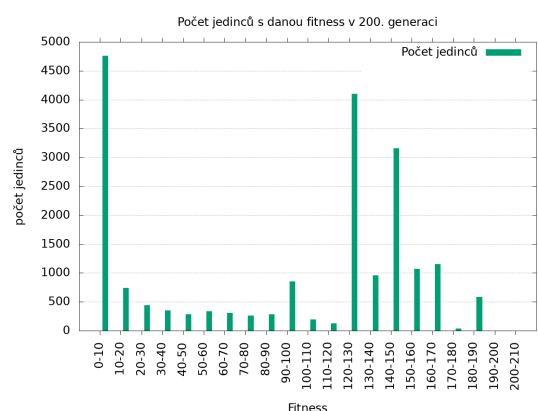
Obrázek 6.7: Histogram celkového počtu jedinců ve všech bězích trojnožky na přímce v 50. generaci



Obrázek 6.8: Histogram celkového počtu jedinců ve všech bězích trojnožky na přímce v 100. generaci

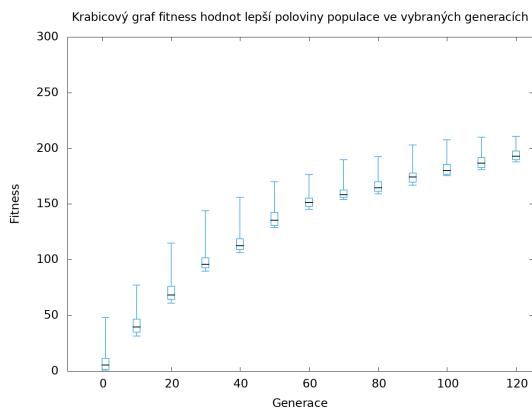


Obrázek 6.9: Histogram celkového počtu jedinců ve všech bězích trojnožky na přímce v 150. generaci

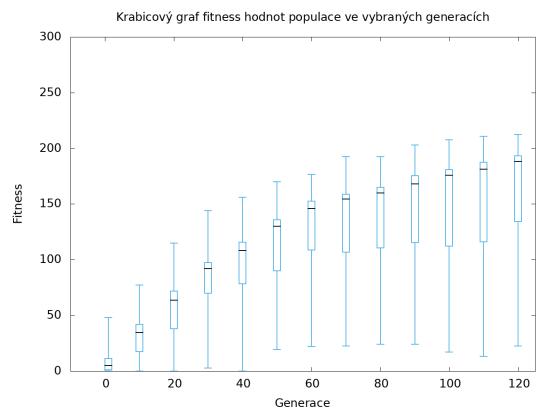


Obrázek 6.10: Histogram celkového počtu jedinců ve všech bězích trojnožky na přímce v 200. generaci

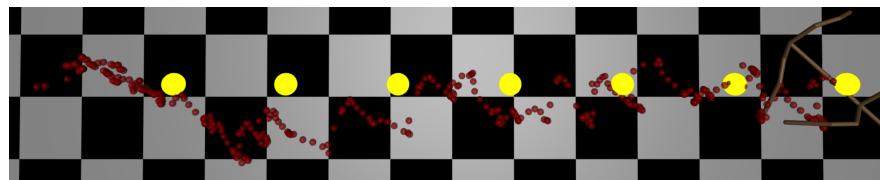
### 6.1.2 Model mravence na přímce



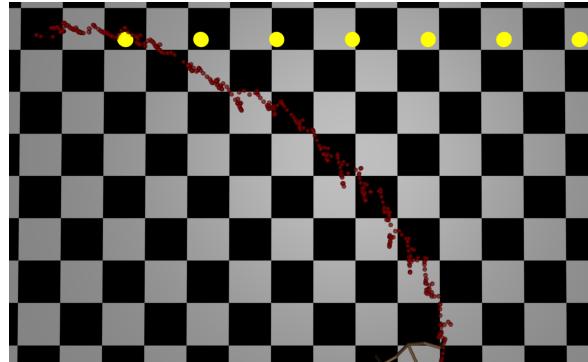
Obrázek 6.11: Box plot fitness hodnot lepší poloviny populace nejlepšího běhu experimentu mravence na přímce



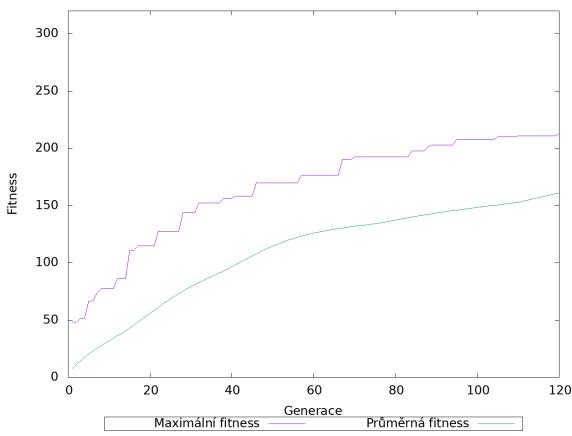
Obrázek 6.12: Box plot fitness hodnot populace nejlepšího běhu experimentu mravence na přímce



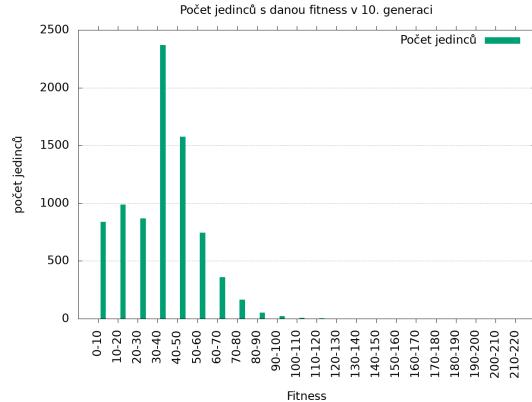
Obrázek 6.13: Trajektorie nejlepšího řešení mravence na přímce



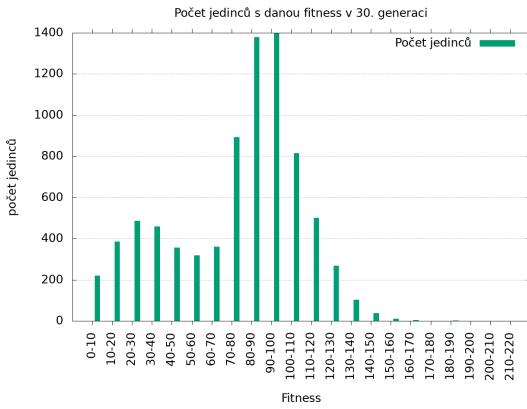
Obrázek 6.14: Trajektorie neoptimálního řešení mravence na přímce



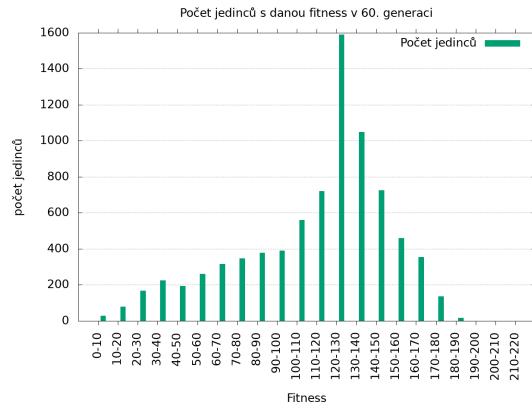
Obrázek 6.15: Nejlepší běh pro experiment s mravencem na přímce



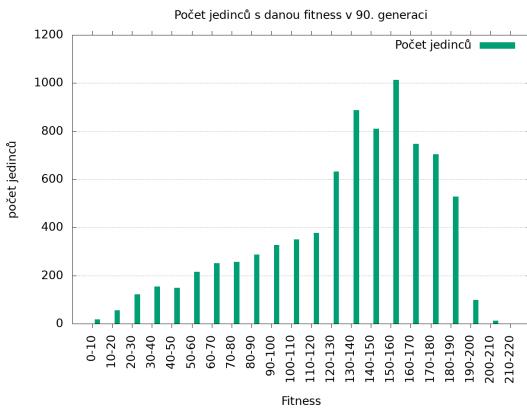
Obrázek 6.16: Histogram celkového počtu jedinců ve všech bězích mravence na přímce v 10. generaci



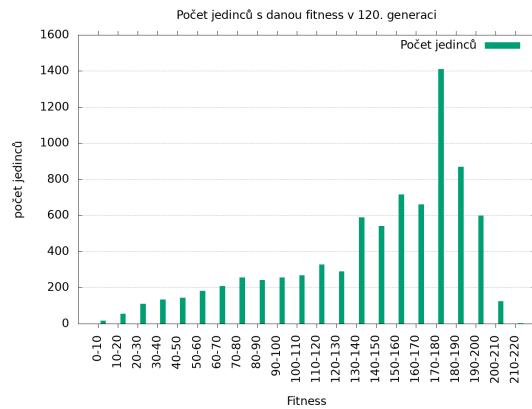
Obrázek 6.17: Histogram celkového počtu jedinců ve všech bězích mravence na přímce ve 30. generaci



Obrázek 6.18: Histogram celkového počtu jedinců ve všech bězích mravence na přímce v 60. generaci



Obrázek 6.19: Histogram celkového počtu jedinců ve všech bězích mravence na přímce v 90. generaci

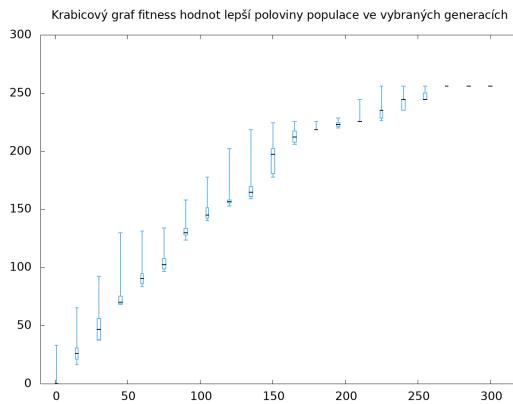


Obrázek 6.20: Histogram celkového počtu jedinců ve všech bězích mravence na přímce v 120. generaci

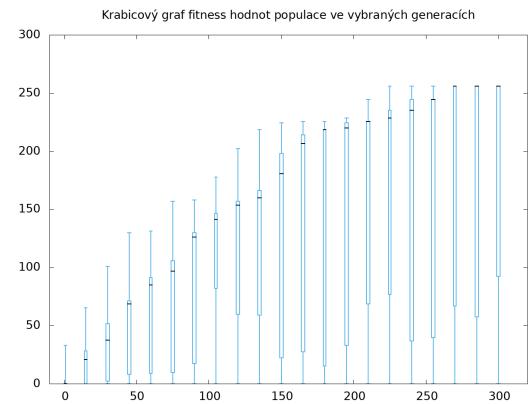
## 6.2 Spirála

[[napsat, ze bez podprogramu a input hodnot to neslo]] [[napsat, ze trojnozka casto upadne a proto je vysoke cislo jedincu, kteri mji nulu!]]

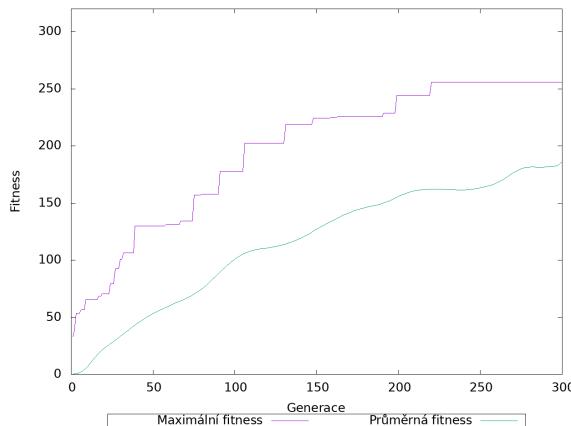
### 6.2.1 Model trojnožky na spirále



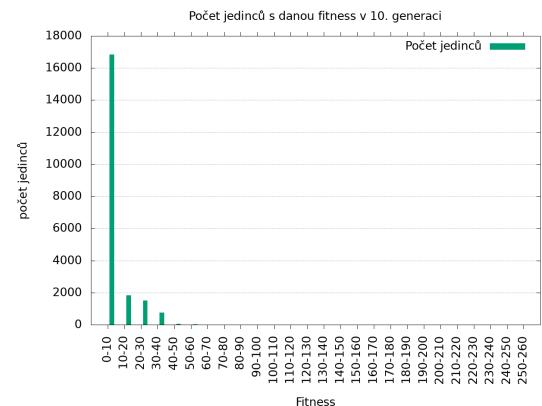
Obrázek 6.21: Box plot fitness hodnot lepší poloviny populace pro vybrané generace



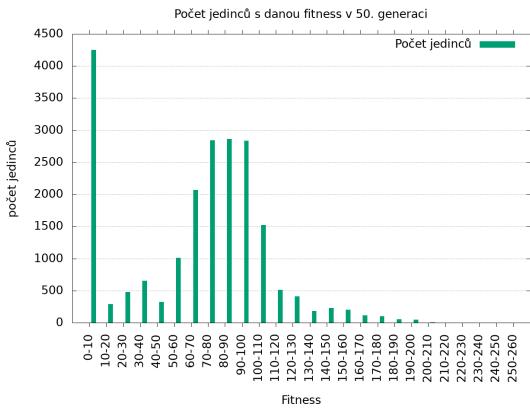
Obrázek 6.22: Box plot fitness hodnot populace pro vybrané generace



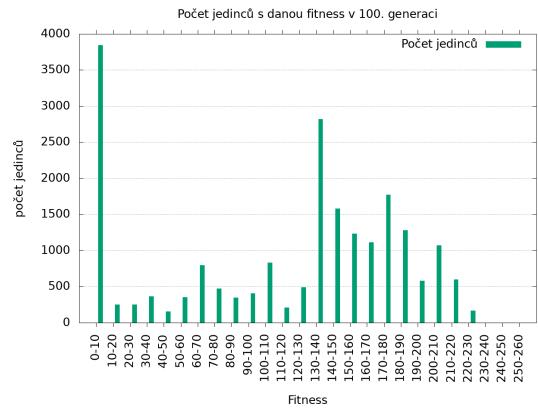
Obrázek 6.23: Nejlepší běh pro experiment s trojnožkou na přímce



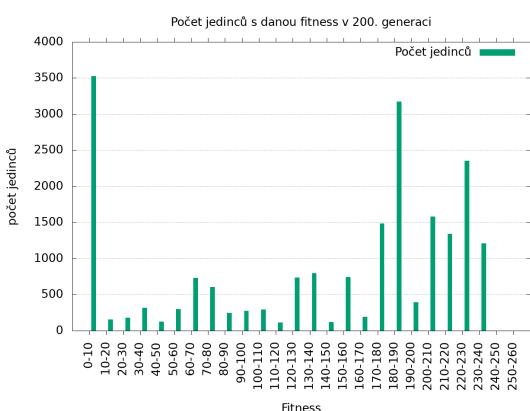
Obrázek 6.24: Histogram celkového počtu jedinců ve všech bězích trojnožky na přímce v 10. generaci



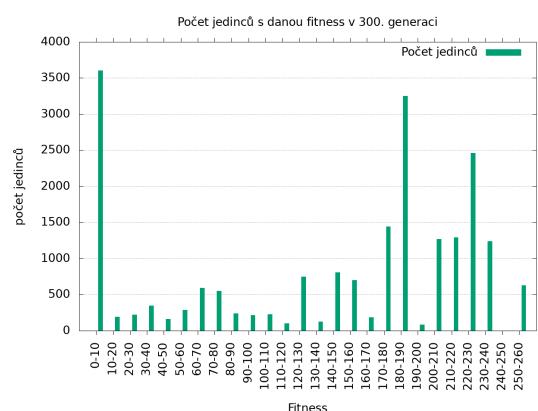
Obrázek 6.25: Histogram celkového počtu jedinců ve všech bězích trojnožky na přímce v 10. generaci



Obrázek 6.26: Histogram celkového počtu jedinců ve všech bězích trojnožky na přímce v 10. generaci

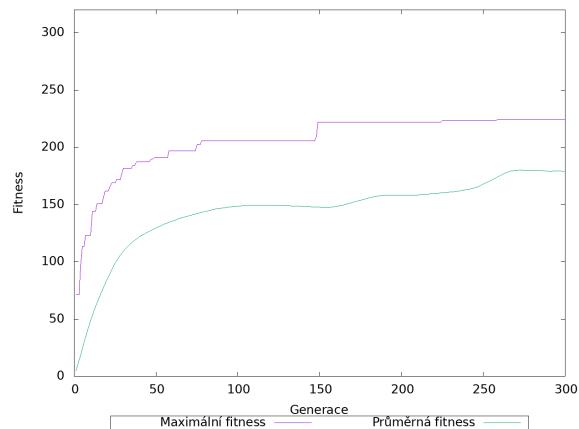


Obrázek 6.27: Histogram celkového počtu jedinců ve všech bězích trojnožky na přímce v 10. generaci



Obrázek 6.28: Histogram celkového počtu jedinců ve všech bězích trojnožky na přímce v 10. generaci

### 6.2.2 Model mravence na spirále



Obrázek 6.29: Nejlepší běh pro experiment s mravencem na spirále

# Kapitola 7

## Závěr

**[[obsah cd]] [[dokumentace nebo manual]]**

Závěrečná kapitola obsahuje zhodnocení dosažených výsledků se zvlášť vyznačeným vlastním přínosem studenta. Povinně se zde objeví i zhodnocení z pohledu dalšího vývoje projektu, student uvede náměty vycházející ze zkušeností s řešeným projektem a uvede rovněž návaznosti na právě dokončené projekty.

# Literatura

- [1] Bongard, J. C.: Evolutionary Robotics. *Commun. ACM*, ročník 56, č. 8, Srpen 2013: s. 74–83, ISSN 0001-0782, doi:10.1145/2493883.  
URL <http://doi.acm.org/10.1145/2493883>
- [2] Brameier, M. F.; Banzhaf, W.: *Linear Genetic Programming*. Springer Publishing Company, Incorporated, první vydání, 2010, ISBN 1441940480, 9781441940483.
- [3] Eiben, A. E.; Smith, J. E.: *Introduction to Evolutionary Computing*. Springer Publishing Company, Incorporated, druhé vydání, 2015, ISBN 3662448734, 9783662448731.
- [4] Hodgins, J. K.: Three-dimensional human running. In *Proceedings of IEEE International Conference on Robotics and Automation*, ročník 4, Apr 1996, ISSN 1050-4729, s. 3271–3276 vol.4, doi:10.1109/ROBOT.1996.509211.
- [5] Hornby, G. S.: *Generative Representations for Evolutionary Design Automation*. Dizertační práce, Waltham, MA, USA, 2003, aAI3073875.
- [6] Koza, J. R.: *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA: MIT Press, 1992, ISBN 0-262-11170-5.
- [7] Reil, T.; Husbands, P.: Evolution of central pattern generators for bipedal walking in a real-time physics environment. *IEEE Transactions on Evolutionary Computation*, ročník 6, č. 2, Apr 2002: s. 159–168, ISSN 1089-778X, doi:10.1109/4235.996015.
- [8] Todorov, E.; Erez, T.; Tassa, Y.: MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct 2012, ISSN 2153-0858, s. 5026–5033, doi:10.1109/IROS.2012.6386109.
- [9] Wolff, K.; Wahde, M.: Evolution of Biped Locomotion Using Linear Genetic Programming. [Online; navštívěno 20.03.2018].  
URL [https://www.intechopen.com/books/climbing\\_and\\_walking\\_robots\\_towards\\_new\\_applications/evolution\\_of\\_biped\\_locomotion\\_using\\_linear\\_genetic\\_programming](https://www.intechopen.com/books/climbing_and_walking_robots_towards_new_applications/evolution_of_biped_locomotion_using_linear_genetic_programming)

## Příloha A

# Jak pracovat s touto šablonou

V této kapitole je uveden popis jednotlivých částí šablony, po kterém následuje stručný návod, jak s touto šablonou pracovat.

Jedná se o přechodnou verzi šablony. Nová verze bude zveřejněna do konce roku 2017 a bude navíc obsahovat nové pokyny ke správnému využití šablony, závazné pokyny k vypracování bakalářských a diplomových prací (rekapitulace pokynů, které jsou dostupné na webu) a nezávazná doporučení od vybraných vedoucích, která již teď najdete na webu (viz odkazy v souboru s literaturou). Jediné soubory, které se v nové verzi změní, budou **projekt-01-kapitoly.tex** a **projekt-30-prilohy-appendices.tex**, jejichž obsah každý student vymaže a nahradí vlastním. Šablonu lze tedy bez problémů využít i v současné verzi.

### Popis částí šablony

Po rozbalení šablony naleznete následující soubory a adresáře:

**bib-styles** Styly literatury (viz níže).

**obrazky** Adresář pro Vaše obrázky. Nyní obsahuje placeholder.pdf (tzv. TODO obrázek, který lze použít jako pomůcku při tvorbě technické zprávy), který se s prací neodevzdává. Název adresáře je vhodné zkrátit, aby byl jen ve zvoleném jazyce.

**template-fig** Obrázky šablony (znak VUT).

**fitthesis.cls** Šablona (definice vzhledu).

**Makefile** Makefile pro překlad, počítání normostran, sbalení apod. (viz níže).

**projekt-01-kapitoly.tex** Soubor pro Váš text (obsah nahraďte).

**projekt-20-literatura-bibliography.bib** Seznam literatury (viz níže).

**projekt-30-prilohy-appendices.tex** Soubor pro přílohy (obsah nahraďte).

**projekt.tex** Hlavní soubor práce – definice formálních částí.

Výchozí styl literatury (czechiso) je od Ing. Martínka, přičemž slovenská a anglická verze (slovakiso a englishiso) jsou jeho překlady s drobnými modifikacemi. Oproti normě jsou v něm určité odlišnosti, ale na FIT je dlouhodobě akceptován. Alternativně můžete využít

styl od Ing. Radima Loskota nebo od Ing. Radka Pyšného<sup>1</sup>. Alternativní styly obsahují určitá vylepšení, ale zatím nebyly řádně otestovány větším množstvím uživatelů. Lze je považovat za beta verze pro zájemce, kteří svoji práci chtějí mít dokonalou do detailů a neváhají si nastudovat detaity správného formátování citací, aby si mohli ověřit, že je vysázený výsledek v pořádku.

Makefile kromě překladu do PDF nabízí i další funkce:

- přejmenování souborů (viz níže),
- počítání normostran,
- spuštění vlny pro doplnění nezlomitelných mezer,
- sbalení výsledku pro odeslání vedoucímu ke kontrole (zkontrolujte, zda sbalí všechny Vámi přidané soubory, a případně doplňte).

Nezapomeňte, že vlha neřeší všechny nezlomitelné mezery. Vždy je třeba manuální kontrola, zda na konci řádku nezůstalo něco nevhodného – viz Internetová jazyková příručka<sup>2</sup>.

**Pozor na číslování stránek!** Pokud má obsah 2 strany a na 2. jsou jen „Přílohy“ a „Seznam příloh“ (ale žádná příloha tam není), z nějakého důvodu se posune číslování stránek o 1 (obsah „nesedí“). Stejný efekt má, když je na 2. či 3. stránce obsahu jen „Literatura“ a je možné, že tohoto problému lze dosáhnout i jinak. Řešení je několik (od úpravy obsahu, přes nastavení počítadla až po sofistikovanější metody). **Před odevzdáním proto vždy překontrolujte číslování stran!**

## Doporučený postup práce se šablonou

1. **Zkontrolujte, zda máte aktuální verzi šablony.** Máte-li šablonu z předchozího roku, na stránkách fakulty již může být novější verze šablony s aktualizovanými informacemi, opravenými chybami apod.
2. **Zvolte si jazyk**, ve kterém budete psát svoji technickou zprávu (česky, slovensky nebo anglicky) a svoji volbu konzultujte s vedoucím práce (nebyla-li dohodnuta předem). Pokud Vámi zvoleným jazykem technické zprávy není čeština, nastavte příslušný parametr šablony v souboru projekt.tex (např.: `documentclass[english]{fitthesis}`) a přeložte prohlášení a poděkování do angličtiny či slovenštiny.
3. **Přejmenujte soubory.** Po rozbalení je v šabloně soubor `projekt.tex`. Pokud jej přeložíte, vznikne PDF s technickou zprávou pojmenované `projekt.pdf`. Když vedoucímu více studentů pošle `projekt.pdf` ke kontrole, musí je pracně přejmenovávat. Proto je vždy vhodné tento soubor přejmenovat tak, aby obsahoval Váš login a (případně zkrařené) téma práce. Vyhneťte se však použití mezer, diakritiky a speciálních znaků. Vhodný název může být např.: „`xlogin00-Cistení-a-extrakce-textu.tex`“. K přejmenování můžete využít i přiložený Makefile:

```
make rename NAME=xlogin00-Cistení-a-extrakce-textu
```

<sup>1</sup>BP Ing. Radka Pyšného <http://www.fit.vutbr.cz/study/DP/BP.php?id=7848>

<sup>2</sup>Internetová jazyková příručka <http://prirucka.ujc.cas.cz/?id=880>

4. Vyplňte požadované položky v souboru, který byl původně pojmenován `projekt.tex`, tedy typ, rok (odevzdání), název práce, svoje jméno, ústav (dle zadání), tituly a jméno vedoucího, abstrakt, klíčová slova a další formální náležitosti.
5. Nahraďte obsah souborů s kapitolami práce, literaturou a přílohami obsahem své technické zprávy. Jednotlivé přílohy či kapitoly práce může být výhodné uložit do samostatných souborů – rozhodnete-li se pro toto řešení, je doporučeno zachovat konvenci pro názvy souborů, přičemž za číslem bude následovat název kapitoly.
6. Nepotřebujete-li přílohy, zakomentujte příslušnou část v `projekt.tex` a příslušný soubor vyprázdněte či smažte. Nesnažte se prosím vymyslet nějakou neúčelnou přílohu jen proto, aby daný soubor bylo čím naplnit. Vhodnou přílohou může být obsah přiloženého pamětového média.
7. Nascanované zadání uložte do souboru `zadani.pdf` a povolte jeho vložení do práce parametrem šablony v `projekt.tex` (`documentclass[zadani]{fitthesis}`).
8. Nechcete-li odkazy tisknout barevně (tedy červený obsah – bez konzultace s vedoucím nedoporučuji), budete pro tisk vytvářet druhé PDF s tím, že nastavíte parametr šablony pro tisk: (`documentclass[zadani,print]{fitthesis}`). Barevné logo se nesmí tisknout černobíle!
9. Vzor desek, do kterých bude práce vyvázána, si vygenerujte v informačním systému fakulty u zadání. Pro disertační práci lze zapnout parametrem v šabloně (více naleznete v souboru `fitthesis.cls`).
10. Nezapomeňte, že zdrojové soubory i (obě verze) PDF musíte odevzdat na CD či jiném médiu přiloženém k technické zprávě.

Obsah práce se generuje standardním příkazem `\tableofcontents` (zahrnut v šabloně). Přílohy jsou v něm uvedeny úmyslně.

### **Pokyny pro oboustranný tisk**

- **Oboustranný tisk je doporučeno konzultovat s vedoucím práce.**
- Je-li práce tištěna oboustranně a její tloušťka je menší než tloušťka desek, nevypadá to dobře.
- Zapíná se parametrem šablony: `\documentclass[twoside]{fitthesis}`
- Po vytisknutí oboustranného listu zkонтrolujte, zda je při prosvícení sazební obrazec na obou stranách na stejně pozici. Méně kvalitní tiskárny s duplexní jednotkou mají často posun o 1–3 mm. Toto může být u některých tiskáren řešitelné tak, že vytisknete nejprve liché stránky, pak je dáte do stejněho zásobníku a vytisknete sudé.
- Za titulním listem, obsahem, literaturou, úvodním listem příloh, seznamem příloh a případnými dalšími seznamy je třeba nechat volnou stránku, aby následující část začínala na liché stránce (`\cleardoublepage`).
- Konečný výsledek je nutné pečlivě překontrolovat.

## Styl odstavců

Odstavce se zarovnávají do bloku a pro jejich formátování existuje více metod. U papírové literatury je častá metoda s použitím odstavcové zarážky, kdy se u jednotlivých odstavců textu odsazuje první řádek odstavce asi o jeden až dva čtverčíky (vždy o stejnou, předem zvolenou hodnotu), tedy přibližně o dvě šířky velkého písmene M základního textu. Poslední řádek předchozího odstavce a první řádek následujícího odstavce se v takovém případě neoddělují svislou mezerou. Proklad mezi těmito řádky je stejný jako proklad mezi řádky uvnitř odstavce. [?] Další metodou je odsazení odstavců, které je časté u elektronické sazby textů. První řádek odstavce se při této metodě neodsazuje a mezi odstavce se vkládá vertikální mezera o velikosti 1/2 řádku. Obě metody lze v kvalifikační práci použít, nicméně často je vhodnější druhá z uvedených metod. Metody není vhodné kombinovat.

Jeden z výše uvedených způsobů je v šabloně nastaven jako výchozí, druhý můžete zvolit parametrem šablony „`odsaz`“.

## Užitečné nástroje

Následující seznam není výčtem všech využitelných nástrojů. Máte-li vyzkoušený osvědčený nástroj, neváhejte jej využít. Pokud však nevíte, který nástroj si zvolit, můžete zvážit některý z následujících:

**MikTeX**  $\text{\LaTeX}$  pro Windows – distribuce s jednoduchou instalací a vynikající automatizací stahování balíčků.

**TeXstudio** Přenositelné opensource GUI pro  $\text{\LaTeX}$ . Ctrl+klik umožňuje přepínat mezi zdrojovým textem a PDF. Má integrovanou kontrolu pravopisu, zvýraznění syntaxe apod. Pro jeho využití je nejprve potřeba nainstalovat MikTeX.

**WinEdt** Ve Windows je dobrá kombinace WinEdt + MiKTeX. WinEdt je GUI pro Windows, pro jehož využití je nejprve potřeba nainstalovat **MikTeX** či **TeX Live**.

**Kile** Editor pro desktopové prostředí KDE (Linux). Umožňuje živé zobrazení náhledu. Pro jeho využití je potřeba mít nainstalovaný **TeX Live** a Okular.

**JabRef** Pěkný a jednoduchý program v Java pro správu souborů s bibliografií (literaturou). Není potřeba se nic učit – poskytuje jednoduché okno a formulář pro editaci položek.

**InkScape** Přenositelný opensource editor vektorové grafiky (SVG i PDF). Vynikající nástroj pro tvorbu obrázků do odborného textu. Jeho ovládnutí je obtížnější, ale výsledky stojí za to.

**GIT** Vynikající pro týmovou spolupráci na projektech, ale může výrazně pomoci i jednomu autorovi. Umožňuje jednoduché verzování, zálohování a přenášení mezi více počítači.

**Overleaf** Online nástroj pro  $\text{\LaTeX}$ . Přímo zobrazuje náhled a umožňuje jednoduchou spolupráci (vedoucí může průběžně sledovat psaní práce), vyhledávání ve zdrojovém textu kliknutím do PDF, kontrolu pravopisu apod. Zdarma jej však lze využít pouze s určitými omezeními (někomu stačí na disertaci, jiný na ně může narazit i při psaní bakalářské práce) a pro dlouhé texty je pomalejší.

Pozn.: Overleaf nepoužívá Makefile v šabloně – aby překlad fungoval, je nutné kliknout pravým tlačítkem na `projekt.tex` a zvolit „Set as Main File“.

## Užitečné balíčky pro L<sup>A</sup>T<sub>E</sub>X

Studenti při sazbě textu často řeší stejné problémy. Některé z nich lze vyřešit následujícími balíčky pro L<sup>A</sup>T<sub>E</sub>X:

- **amsmath** – rozšířené možnosti sazby rovnic,
- **float, afterpage, placeins** – úprava umístění obrázků,
- **fancyvrb, alltt** – úpravy vlastností prostředí Verbatim,
- **makecell** – rozšíření možností tabulek,
- **pdflscape, rotating** – natočení stránky o 90 stupňů (pro obrázek či tabulku),
- **hyphenat** – úpravy dělení slov,
- **picture, epic, eepic** – přímé kreslení obrázků.

Některé balíčky jsou využity přímo v šabloně (v dolní části souboru fitthesis.cls). Nakládnutí do jejich dokumentace může být rovněž užitečné.

Sloupec tabulky zarovnaný vlevo s pevnou šířkou je v šabloně definovaný „L“ (používá se jako „p“).