Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

FIIT-5220-56167

Bc. Marek Hlaváč

POUŽITIE GRAMATICKÉHO ROJA PRE VÝVOJ AGENTA V HRE ROBOCODE

Diplomová práca

Študijný program: Softvérové inžinierstvo
Študijný odbor: 9.2.5 Softvérové inžinierstvo
Miesto vypracovania: Ústav aplikovanej informatiky, FIIT STU Bratislava
Vedúci práce: prof. RNDr. Jiří Pospíchal, DrSc.

Máj 2013

Anotácia

Slovenská technická univerzita v Bratislave
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ
Študijný program: SOFTVÉROVÉ INŽINIERSTVO

Autor: Marek Hlaváč
Diplomový projekt: Použitie gramatického roja v riadení robotických tankov pre hru Robocode
Vedenie diplomového projektu: Prof. RNDr. Jiří Pospíchal, DrSc.
Máj, 2013

Diplomový projekt je zameraný na vytvorenie robotického tanku pre hru Robocode, ktorý prispôsobuje svoje správanie v dynamickom prostredí. Cieľ práce spočíva v použití gramatického roja za účelom evolvovať komplexné správanie robota, ktoré by bolo konkurencieschopné voči existujúcim robotom. Gramatický roj kombinuje gramatickú evolúciu a optimalizáciu rojom častíc za účelom nájsť optimálne riešenia pre dynamické problémy, ktoré sú reprezentované vhodne zvolenou gramatikou. Gramatika definuje povolenú množinu operácií, ktorými je správanie robota ohraničené.

Práca obsahuje podrobnú analýzu použitých metód a existujúcich riešení. Detailne sa venuje návrhu gramatiky, možnostiam modifikácie optimalizačných metód, kvantifikácii kvality riešení a architektúre softvérového prototypu. Následne je rozobratá etapa implementácie, ktorá sa venuje postupu práce a komunikácie medzi jednotlivými softvérovými modulmi. Posledné časti sa týkajú výsledkov testovania nájdených robotov v súbojoch s existujúcimi robotmi. Tento spôsob umožňuje určiť úroveň ich konkurencieschopnosti, odhaliť slabiny a opísať možnosti budúceho vylepšenia.

Annotation

Slovak University of Technology Bratislava
FACULTY OF INFORMATICS AND INFORMATION TECHNOLOGIES
Degree Course: SOFTWARE ENGINEERING

Author: Marek Hlaváč
Diploma project: Using grammatical swarm robotic control in tanks for the game Robocode
Supervisor: Prof. RNDr. Jiří Pospíchal, DrSc.
2013, May

Master thesis is focused on creating a robotic tank for Robocode game which adapts its behavior in dynamic environments. The main objective of the work lies in the usage of grammatical swarm to evolve complex robot behavior that is based on improving robots competitiveness against existing robots. Grammatical swarm combines grammatical evolution and particle swarm optimization to find optimal solutions for dynamic problems which are represented by properly chosen grammar. The grammar defines a set of allowed operations that bounds the options of robot`s behavior.

The document contains a detailed analysis of applied methods and existing solutions to the problem. Several chapters discussed the grammar design, possible modification of optimization methods, quality quantification of solutions and software prototype architecture. Implementation phase describes the work process and communication between individual software modules. The last section is focused on the testing results of designed robots based on battles with existing robots. This method allows to determine their level of competitiveness, identify weaknesses and describe opportunities for future improvements.

Poďakovanie

Chcel by som sa poďakovať vedúcemu diplomovej práce Prof. RNDr. Jiřímu Pospíchalovi, DrSc., za jeho odbornú pomoc pri riešení diplomovej práce a taktiež všetkým, ktorí ma pri jej vypracovávaní podporovali.

Čestné prehlásenie

Čestne prehlasujem, že som diplomovú prácu vypracoval samostatne, s použitím vlastných znalostí a štúdiom uvedenej odbornej literatúry.

Bratislava, 10. máj 2013 .....

podpis

Obsah

1 Úvod

Umelej inteligencii sa v softvérovom inžinierstve venuje čoraz viac pozornosti, pretože si nachádza široké uplatnenie v mnohých softvérových systémoch a disponuje obrovským potenciálom nasadzovania v budúcnosti. Jednou z oblastí, ktorou sa odborníci aktívne zaoberajú, sú optimalizačné úlohy. Cieľom úloh je aplikácia vhodnej optimalizácie vzhľadom na vlastnosti hľadaného riešenia.

Rozsiahly stavový priestor je hlavným problémom mnohých úloh, ktorých cieľom je vytváranie inteligentných umelých agentov. Agenty sú špecifické tým, že prispôsobujú svoju činnosť okolitému prostrediu za účelom splnenia zadaných cieľov. Pri riešení špecifických úloh musí agent získať znalosť, ktorá predstavuje vhodnú postupnosť krokov smerujúcich k riešeniu. Pri komplexnejších úlohách môže byť postupné prehľadávanie stavového priestoru nemožné z hľadiska výpočtových prostriedkov. Ďalšou komplikáciou je dynamický charakter úloh, ktorý spôsobuje zmenu vlastností problému v čase.

V optimalizačných úlohách tohto druhu sa veľmi často pristupuje k použitiu evolučných výpočtových techník s využitím adaptívnych prístupov riešenia. Z hľadiska adaptívnych umelých agentov patria medzi najrozšírenejšie oblasti aplikácie evolučných techník, robotika a herný priemysel. Dôvodom sú pozitívne výsledky nasadenia, flexibilné možnosti aplikácie riešení a jednoduché testovanie vyvinutých riešení.

Primárnym cieľom práce je evolučný návrh gramatiky pretransformovanej do vytvorenia návrhu softvérového prototypu umelého agenta prispôsobujúceho svoje správanie vonkajším vplyvom pomocou špecifikovanej evolučnej techniky. Správanie agenta je opísané zadefinovanou gramatikou, ktorá sa prostredníctvom gramatickej evolúcie v kombinácii s optimalizáciou rojom častíc vyvíja a umožňuje tak vytvárať lepšie a komplexnejšie riešenia agentov.

Druhá časť práce obsahuje postup implementácie robotického tanku pre programovaciu hru Robocode. V hre súťažia proti sebe umelé agenty, pričom ich cieľom je získať čo najviac víťazstiev v súbojoch s inými agentmi. Týmto spôsobom je možné vo fáze testovania kvantifikovať kvalitu riešenia na základe porovnania výsledkov s existujúcimi riešeniami a tým pádom overiť jeho efektívnosť a použiteľnosť.

2 Analýza problému

2.1 Opis problémovej oblasti

Pri tvorbe inteligentného agenta je nutné vykonať dôkladnú analýzu vlastností cieľového správania agenta a pravidiel, ktoré musia byť pri hľadaní dodržané. Následne vyvstáva otázka ako vhodne navrhnúť postup hľadania cieľového správania s tým zámerom, aby realizoval zadané ciele správne a efektívne. V prípade, že môžeme niektorý z aspektov v hľadaní optimálneho riešenia vylepšiť, tak je možné transformovať úlohu na optimalizačný problém a metódu vylepšenia nazývať optimalizačnou metódou. Optimalizačný problém je následne možné zadefinovať prostredníctvom:

- cieľovej funkcie opisujúcej problém, ktorý sa snažíme minimalizovať alebo maximalizovať,
- množiny parametrov vstupujúcich do problému,
- množiny ohraničení problému.

Pomocou týchto vlastností sme schopní vytvoriť optimalizačnú metódu, ktorá prehľadáva stavový priestor so zámerom nájsť optimálne riešenie z kandidátskych (aktuálne nájdených) riešení na základe priradenia hodnôt z definičného oboru cieľovej funkcie do oboru hodnôt tak, aby boli všetky ohraničenia problému splnené a funkcia minimalizovaná. Optimalizačné problémy je možné triediť na základe:

- počtu premenných ovplyvňujúcich cieľovú funkciu,
- typu hodnôt premenných (diskrétne, kombinatoriálne...),
- priebehom cieľovej funkcie (lineárny, kvadratický, nelineárny...),
- typu ohraničení problému,
- počtu lokálnych a globálnych optím,
- počtu optimalizačných kritérií (počtu a typu cieľových funkcií).

Pri vytváraní inteligentného agenta musíme zvoliť vhodný návrh optimalizačnej metódy, vzhľadom na optimalizáciu hľadania vhodného správania agenta v komplexnom stavovom priestore kandidátskych riešení, pričom treba brať do úvahy dynamický charakter úlohy, ktorý predurčuje agenta k tomu, aby bol schopný prispôsobovať svoje správanie aktuálnemu stavu prostredia.

2.2 Evolučné výpočtové techniky

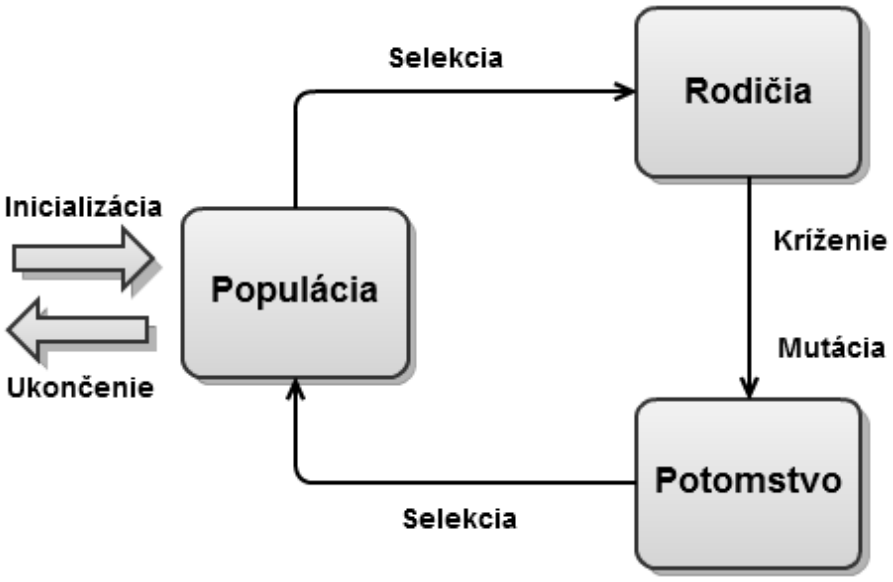
Evolučné výpočtové techniky (EC - Evolutionary Computation) sú techniky, ktoré sú založené na iteratívnom opakovaní metódy hľadania optimálneho riešenia, ktorej cieľom je zlepšovanie aktuálneho riešenia prostredníctvom výberu z populácie kandidátskych riešení. Populácia je vytváraná pomocou stochastických výberových metód so zámerom nájdenia optimálneho riešenia. Procesy hľadania sú často inšpirované biologickými procesmi evolúcie, pretože evolúcia dokáže produkovať vysoko optimálne výsledky.

2.2.1 Evolučné algoritmy

Evolučné algoritmy (EA - Evolutionary Algorithms) zahrňujú veľký počet techník, ktoré majú rovnaký základ v biologickej evolúcii, ale odlišujú sa implementáciou. Patria medzi ne genetické algoritmy, genetické programovanie, evolučné programovanie, evolučné stratégie a ďalšie.

Princíp činnosti [1] evolučných algoritmov (Obr. 1) spočíva v práci s populáciou kandidátskych riešení, ktoré sa evolvovaním vylepšujú za účelom nájdenia optimálneho riešenia. Algoritmus začína so vstupnou populáciou jedincov, ktorí predstavujú kandidátske riešenia. Kvalita riešení je vypočítaná pomocou zvolenej fitnes funkcie, ktorá kvantifikuje úroveň jednotlivých riešení. Na jej základe je následne realizovaný výber najlepších riešení. Tieto riešenia budú slúžiť ako rodičovské jedince pre ďalšiu generáciu prostredníctvom zvolenej metódy selekcie. Medzi najpoužívanejšie metódy selekcie patrí:

- náhodná selekcia - jedinci sú vyberaní bez ohľadu na fitnes,
- proporcionálna selekcia (ruleta) - pravdepodobnosť výberu jedinca je priamoúmerná vzhľadom na jeho fitnes,
- selekcia turnajom - jedinci sú náhodne zoskupený do turnajov určitej veľkosti, v ktorých sú vybrané najlepšie riešenia,
- selekcia na základe poradia – jedinci sú vyberaní na základe poradia určeného veľkosťou fitnes,
- elitizmus – najlepších jedincov ponechávame v populácii nepozmenených.



Obr. 1. Model princípu funkčnosti evolučných algoritmov.

Pri tvorbe nového potomstva sa v jednotlivých generáciách používajú dva postupy odvodené z biologických mechanizmov:

1. kríženie - krížením dvoch a viacerých rodičov vznikne nový potomok,
2. mutácia - zmena aplikovaná na nových potomkoch.

Kombinovanie metódy selekcie a operátorov kríženia a mutácie vedie k postupnému zlepšovaniu fitnes v jednotlivých generáciách. Evolúcia môže byť v tomto prípade označená ako proces adaptácie vzhľadom na fitnes funkciu. Nutné je spomenúť aj stochastický charakter evolučných procesov, ktorý spočíva najmä pri vytváraní nových riešení pomocou kríženia a mutácie, keďže ich princíp spočíva na náhodných procesoch. Pri selekcii je to podobné, avšak popri stochastických metódach selekcie je možné použiť aj deterministické metódy.

2.2.2 Gramatická evolúcia

Gramatická evolúcia (GE – Grammatical Evolution) je jeden z najmladších prístupov evolučných výpočtových techník, ktorý je inšpirovaný molekulárnou biológiou a formálnymi gramatikami. Cieľ GE spočíva vo vylepšení gramatík, ktoré sa používajú v genetickom programovaní za účelom riešiť problémy v dynamických prostrediach.

GE je modulárna technika, ktorá má flexibilné možnosti použitia pretože umožňuje aplikovať rôzne vyhľadávacie stratégie a meniť správanie pri zmene zadefinovanej gramatiky. Explicitne zadefinovaná gramatika umožňuje pomocou GE ľahko generovať

riešenia v iných jazykoch. GE bola úspešne použitá pri generovaní riešení pre viaceré jazyky, ako napríklad Java, C/C++, Lisp a iné.

2.2.2.1 Dynamické problémy

Dynamické problémy môžu byť zadefinované [2] prostredníctvom porovnania rozdielov medzi postupom hľadania riešenia v statických a dynamických problémoch. V statických problémoch evolvujeme populáciu riešení, až kým nie sú uspokojené všetky ukončujúce kritéria hľadania. Na druhej strane, v dynamických problémoch sa môžu meniť vlastnosti prostredia alebo fitnes funkcie v prebiehajúcom čase, čo vedie k dynamickým zmenám kritérií hľadania. Práve variabilný časový charakter je pozorovaný v mnohých reálnych problémoch. Pre GE sa v tomto prípade otvárajú rozsiahle možnosti použitia.

Pri riešení dynamických problémov je nutné si vybrať jeden z dvoch prístupov ako riešiť problematiku súvisiacu so zmenou kritérií hľadania. Prvou je úplná výmena populácie po zaznamenanej zmene a druhou je prispôsobenie aktuálnej populácie tejto zmene. Ak nie je zmena kritická, tak je vhodnejšie prispôsobovať populáciu, pretože sa jedná o efektívnejší spôsob. Vytvorené riešenia sú takýmto spôsobom oveľa robustnejšie ako v prípade statických problémov.

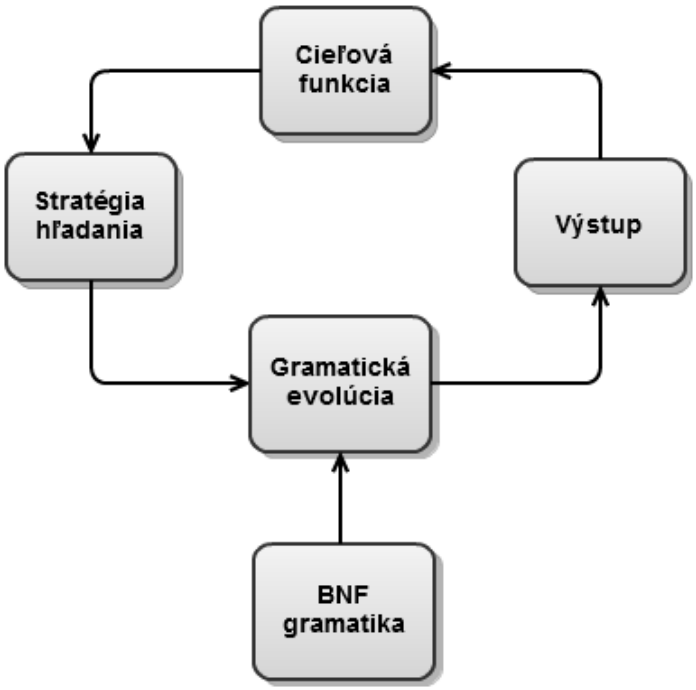
Ďalším problémom je rozsiahle množstvo druhov dynamických problémov, ktoré môžu mať rozličné ohraničenia a zmena v čase môže byť vykonávaná akýmkoľvek spôsobom. V prípade viacerých dimenzií, ktoré ovplyvňujú hľadané riešenie, sa tak cieľom použitého algoritmu stáva sledovanie optimálneho stavu na úkor hľadania najlepších riešení problému v každom stave. Spôsobuje to, že hľadáme riešenie, ktoré musí mať stabilný charakter pri zmene podmienok. Finálna fitnes v tomto prípade nie je najvyššie zaznamenaná počas priebehu vykonávania algoritmu, ale na jej kvantifikáciu sa musia zvoliť dômyselnejšie techniky.

2.2.2.2 Princíp činnosti

Gramatická evolúcia je podobná genetickému programovaniu v tom, že používa evolučný proces za účelom automatického generovania počítačových programov variabilnej dĺžky. Na rozdiel od genetického programovania používa populáciu genotypov, ktoré sú reprezentované binárnymi alebo číselnými reťazcami [3]. Tie sú následne transformované do fenotypu funkčného programu cez proces transformácie genotypu na fenotyp. Transformácia je realizovaná prostredníctvom použitia gramatiky založenej na Backus Naurovej Forme (BNF), ktorá špecifikuje jazyk vyprodukovaného riešenia. Zámerom procesu transformácie je

oddelenie priestoru hľadania a priestoru riešení. Genotypy sú vyvíjané bez znalosti ekvivalentného fenotypu.

Modularita GE umožňuje ľahkú zmenu gramatiky, stratégie prehľadávania a cieľovej funkcie, ktoré môžu byť do GE zasunuté ako externé komponenty (Obr. 2). Modularita GE tak umožňuje použitie vhodnej stratégie prehľadávania na základe vlastností riešeného problému a gramatiky, ktorá sa použije na evolvovanie riešenia pomocou vlastného slovníka.



Obr. 2. Prehľad činnosti modulárneho návrhu GE.

2.2.2.3 Generovanie fenotypu

Riešenie problému prostredníctvom GE vyžaduje zadefinovanie BNF gramatiky, ktorá špecifikuje syntax fenotypov programov, ktoré budú produkované GE [2]. BNF gramatika je tvorená:

- množinou neterminálnych symbolov N,
- množinou terminálnych symbolov T,
- množinou produkčných pravidiel P,
- počiatočným neterminálnym symbolom S.

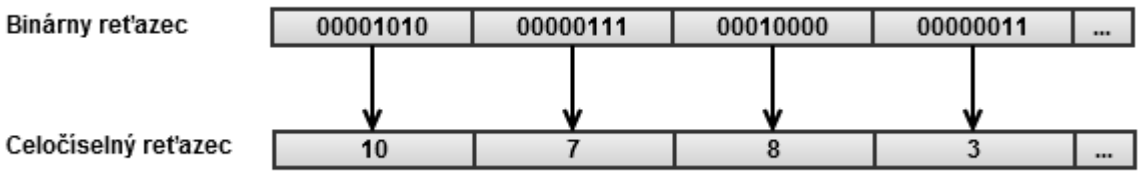
Nasledujúca gramatika je vytvorená ako príklad pre ukážku procesu mapovania a ďalších techník gramatickej evolúcie.

```
N = {<vyraz>, <operator>, <operand>, <premenna>}
T = {1, 2, 3, 4, +, -, /, *, x, y}
S = {<vyraz>}
P:
<vyraz>::=  <vyraz><operator><vyraz>                (0)
          | <operand>                               (1)
<operator>::=  +                                    (0)
             | -                                    (1)
             | /                                    (2)
             | *                                    (3)
<operand>::=   1                                    (0)
             | 2                                    (1)
             | 3                                    (2)
             | 4                                    (3)
             | <premenna>                           (4)
<premenna>::=  x                                    (0)
             | y                                    (1)
```

Gramatická evolúcia aplikuje pre vstupnú gramatiku výraz:

$$Pravidlo = c \ \% \ r,$$

ktorý vyberie jednu z možností produkčných pravidiel P pre konkrétny neterminálny symbol z N. Parameter *c* je genetický kód a *r* je počet dostupných produkčných pravidiel, ktoré sú prístupné pre aktuálny neterminálny symbol. Obr. 3 a Tab. 1 opisujú jeden z možných scenárov procesu transformácie. Prvým krokom je prevod binárneho reťazca na celočíselný reťazec, pričom je použitých 8 bitov pre zakódovanie genetického kódu. Čísla v tomto reťazci postupne označujú výber produkčných pravidiel z BNF gramatiky, ktoré následne transformujú prvý neterminálny symbol v aktuálnej sekvencii podľa vybraného pravidla na základe kódu v reťazci, až kým nie sú nahradené všetky neterminálne symboly a teda vytvorený program pozostávajúci zo sekvencie terminálnych symbolov.



Obr.3. Ukážka reprezentácie genotypu.

Tab.1. Ukážka transformácie genotypu na fenotyp.

Kód	Selekcia	Pravidlo	Stav sekvencie
-	-	-	<vyraz>
10	10 % 2 = 0	<vyraz><operator><vyraz>	<vyraz><operator><vyraz>
7	7 % 2 = 1	<operand>	<operand><operator><vyraz>
8	8 % 5 = 3	4	4<operator><vyraz>
3	3 % 4 = 3	*	4*<vyraz>
4	4 % 2 = 0	<vyraz><operator><vyraz>	4*<vyraz><operator><vyraz>
7	7 % 2 = 1	<operand>	4*<operand><operator><vyraz>
9	9 % 5 = 4	<premenna>	4*<premenna><operator><vyraz>
5	5 % 2 = 1	y	4*y<operator><vyraz>
12	12 % 4 = 0	+	4*y+<vyraz>
1	1 % 2 = 1	<operand>	4*y+<operand>
10	10 % 5 = 0	1	**4*y+1**

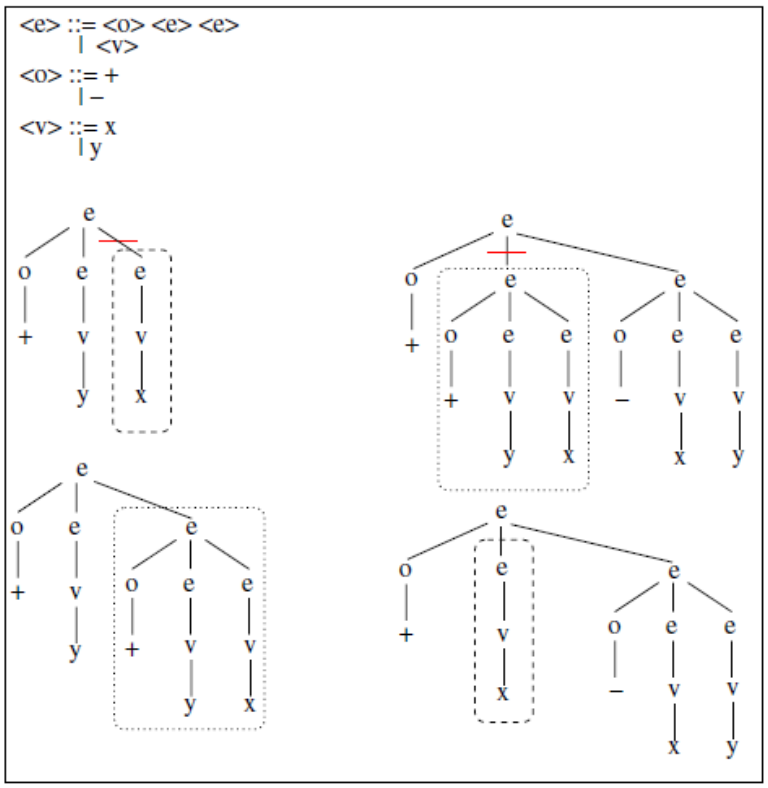
Proces transformácie môže byť ukončený jedným z troch spôsobov:

1. Kompletný program je vytvorený ešte pred ukončením sekvencie genotypu
2. Pri predčasnom konci sekvencie genotypu sa pokračuje znova od začiatku sekvencie genotypu, pričom transformácia musí byť ukončená pod zadefinovaný maximálny prah, ktorý predstavuje najväčší možný počet transformácií
3. Ak sa prekročí povolený limit transformácií pravidiel a v sekvencii sa stále nachádzajú neterminálne symboly, tak je transformácia pozastavená a jedinec je ocenený najhoršou fitnes.

2.2.2.4 Mutácia a kríženie

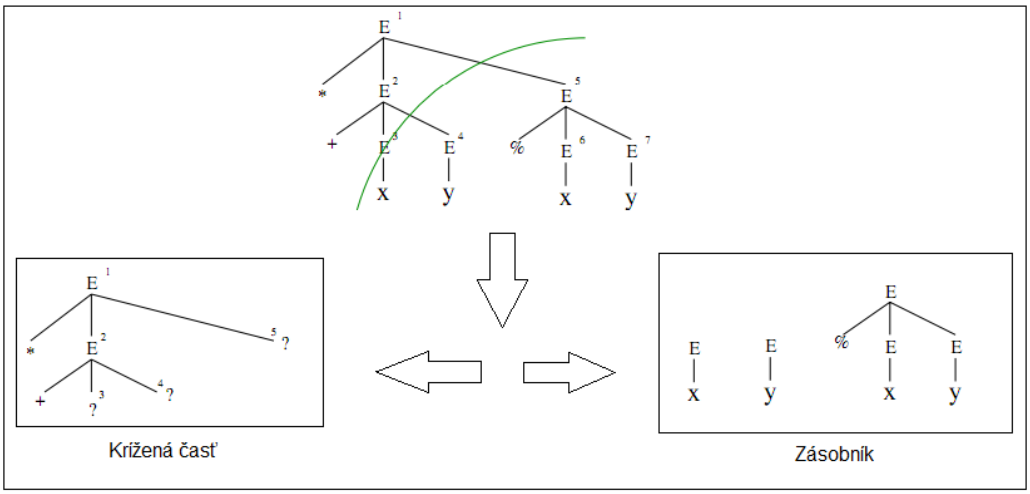
Mutácia je realizovaná prostredníctvom výmeny bitu v bitovom reťazci alebo hodnoty v celočíselnom reťazci na inú náhodnú hodnotu. Keďže mutácia sa vykonáva na genotype, tak môže nastať prípad, že po vykonaní mutácie bude jej efekt neutrálny. Neutrálna mutácia sa prejavuje na strane fenotypu, keď sa dva rôzne genotypy transformujú do rovnakého fenotypu.

Kríženie je vždy jednobodové a realizuje sa medzi dvoma sekciami genetického kódu dvoch rodičov [4], ktorý je zobrazený na Obr. 4. Pre ilustráciu je postup transformácie znázornený ako strom, v ktorom sú prepojené jednotlivé zmeny symbolov.



Obr.4. Aplikácia jednobodového kríženia.

Problém kríženia spočíva v efekte nenaplnenia (angl. *ripple effect*), ktorý je možné sledovať na strane fenotypu po označení časti genetického kódu, ktorá sa bude krížiť. V tomto kroku sa môže kontext genetického materiálu jedného rodiča líšiť od kontextu druhého rodiča. Riešením problému je vytvorenie dodatočného zásobníka, ktorý si bude pamätať orezané vetvy stromu [5]. V prípade nutnosti sa použijú vetvy zo zásobníku na doplnenie neurčeného kontextu nového jedinca, ktorý vznikol po krížení (Obr. 5).



Obr. 5. Ukážka stavu kríženia pri predchádzaní efektu nenaplnenia.

2.3 Rojová inteligencia

V kontexte hľadania riešenia v stavovom priestore je možné použiť techniku hľadania na základe rojov. Rojom (angl. swarm) môžeme označiť spolupracujúcu skupinu jedincov, ktorí sa snažia nájsť riešenie na určitý problém. Detailnejšia definícia je označenie roja ako dynamickej skupiny agentov, ktorá medzi sebou komunikuje priamou alebo nepriamou cestou, pričom ich konanie je obmedzené lokálnym prostredím, v ktorom sa aktuálne nachádzajú [6]. Pod lokálnym prostredím rozumieme špecifickú časť stavového priestoru riešenia problému vzhľadom na možný globálny priestor riešenia problému.

Interakcie medzi agentmi vedú k rôznym stratégiám distribuovaných a kolektívnych riešení problémov. Riešenie problémov prostredníctvom konkrétnych stratégií môžeme v kontexte roja označiť ako rojovú inteligenciu (SI - Swarm Intelligence), ktorá je podmienená interakciami medzi jedincami roja. Podstatu inteligencie roja je možné sledovať v situáciách, ktorých cieľom je priniesť úžitok všetkým jeho jedincom na základe ich vzájomnej kooperácie, pričom jedinci nepoznajú globálny stav prostredia, v ktorom sa nachádzajú. Jedinci sa snažia konať v rámci skupinovej interakcie so zámerom splnenia ich globálneho cieľa prostredníctvom výmeny lokálne dostupných informácií, ktoré sú postupne posielané naprieč celou skupinou, čo v konečnom dôsledku znamená, že efektivita hľadania riešenia problému je vyššia pri hľadaní viacerých kooperatívnych jednotlivcov ako keby ho mal riešiť jedinec sám. Na základe tohto princípu je možné SI označiť aj ako kolektívnu inteligenciu.

SI dala podnet vzniku novým výpočtovým metódam a technikám [7], ktoré sú označované pod pojmom výpočtová rojová inteligencia (CSI - Computational Swarm Intelligence). CSI je postavená na základe znalostí z SI a ústi do konkrétnych typov algoritmických riešení pre komplexné problémy. Hlavnou inšpiráciou výpočtových modelov sú rojové systémy nachádzajúce sa v prírode, ako napríklad mraveniská, včelie úle, húfy rýb a kŕdle vtákov. V týchto biologických systémoch majú jedinci relatívne jednoduché správanie, ale z globálneho hľadiska je ich kolektívne správanie veľmi komplexné. Správanie jedincov je definované trojicou pravidiel:

1. Vyhýbanie sa kolízii – cieľom je vyhnúť sa kolízii s blízkymi jedincami v roji.
2. Prispôsobovanie rýchlosti – cieľom je prispôsobiť rýchlosť pohybu susedným jedincom.
3. Zoskupovanie – každý jedinec sa snaží udržať čo najbližšie pri susedných jedincoch.

Komplexné správanie sa vyvíja v čase na základe interakcií medzi jedincami. Výsledné správanie roja nie je vlastníctvom žiadneho jedinca a zvyčajne sa nedá predurčiť alebo vydedukovať zo základného správania jedincov, ale je možné ho označiť ako kolektívne správanie podnietené sociálnym aspektom jedincov, ktoré formuje a ovplyvňuje roj. Na druhej strane, je možné sledovať cyklický vplyv medzi správaním roju a jedincami. Roj ovplyvňuje podmienky, podľa ktorých jedinci konajú a jedinci spätne vykonávajú akcie, ktoré ovplyvňujú roj. Na základe tohto faktu je nutné si uvedomiť, že interakcia a kooperácia je základným činiteľom SI.

Interakcia medzi jedincami udržiava skúsenosť a znalosť získanú z okolitého prostredia, čo zaručuje dynamické prispôsobovanie sa novým podmienkam. Interakciu je v biologických rojových systémoch možné rozdeliť do dvoch typov:

- priama interakcia – fyzický, audiovizuálny alebo iný percepčný kontakt,
- nepriama interakcia – lokálne zmeny prostredia.

Základnými princípmi SI sú [8]:

1. Princíp blízkosti (proximity) – skupina jedincov by mala byť schopná vykonávať jednoduché priestorové a časové výpočty, ktoré predstavujú priame vykonanie aktivity vychádzajúce zo správania, pričom zámer je maximalizácia úžitku pre skupinu.
2. Kvalitatívny princíp – skupina jedincov by mala byť schopná reagovať na faktory v prostredí.
3. Princíp rôznorodých reakcií – skupina jedincov by nemala odpovedať na aktivity vždy rovnakými spôsobmi.
4. Princíp stability – skupina jedincov by nemala meniť svoje správanie za každým, keď sa zmení prostredie.
5. Princíp adaptability – skupina jedincov musí byť schopná zmeniť svoje správanie, keď je výsledok zmeny pozitívny.

Nasledujúca kapitola sa venuje jednej z najrozšírenejších techník, a to optimalizácii rojom častíc, ktorá vychádza z rojovej inteligencie. Obsah kapitoly je spracovaný podľa zdroja [7], ktorý pokrýva všetky aspekty tejto techniky.

2.3.1 Optimalizácia rojom častíc

Optimalizácia rojom častíc (PSO - Particle Swarm Optimization) je postavená na základe sociálno-psychologického modelu vplyvu a učenia sa. Jedinci v roji častíc majú zadefinované

jednoduché správanie – opakovať správanie najúspešnejšieho jedinca spomedzi susedných jedincov.

Algoritmus PSO pracuje s rojom častíc, z ktorých každá častica predstavuje možné riešenie. V analógii s evolučnými algoritmami je možné povedať, že roj predstavuje populáciu a častice jedincov. Častice prechádzajú viacrozmerným priestorom hľadania, kde pozícia každej častice je vypočítaná na základe vlastnej skúsenosti a skúseností susedných častíc.

PSO sa drží princípov rojovej inteligencie, ktoré sú zadefinované v kapitole 2.2.2. Princíp blízkosti je adresovaný viacrozmernými priestorovými výpočtami, ktoré sú vykonávané v niekoľkých krokoch. Roj reaguje na faktor kvality prostredia najlepšou osobnou pozíciou jedinca a najlepšou pozíciou susedov. Prostredníctvom výmeny odpovedí medzi najlepšími pozíciami s inými susedmi je zachovaný princíp rôznorodých reakcií. Princíp stability je tvorený zmenou stavu v prípadoch vylepšenia najlepšej lokálnej alebo globálnej pozície. Adaptívny princíp sa preukazuje pri podmienenej zmene stavu na základe najlepšej lokálnej a globálnej pozície.

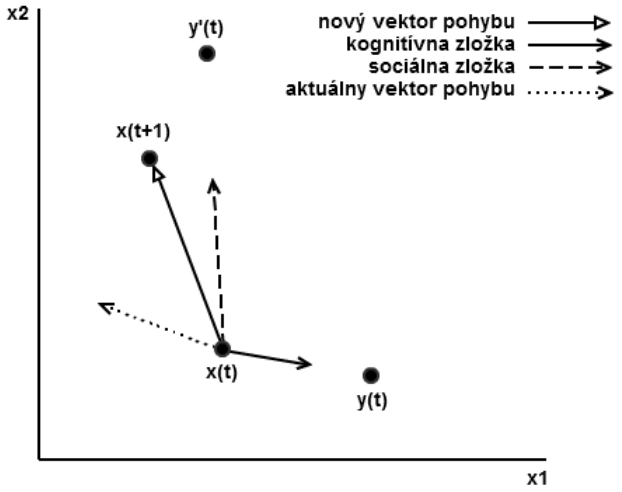
2.3.1.1 Model PSO

Pozícia častice je pre každú iteráciu výpočtu podmienená jej rýchlosťou $v_i(t)$ a aktuálnou pozíciou $x_i(t)$. Výpočet novej pozície $x_i(t+1)$ je realizovaný nasledovne:

$$x_i(t+1) = x_i(t) + v_i(t+1) \tag{2.1}$$

kde vektor rýchlosti zabezpečuje proces optimalizácie a reflektuje znalosti získané na základe vlastnej a cudzej skúsenosti (Obr. 6). Vlastná znalosť častice sa označuje ako kognitívna zložka a cudzia znalosť získaná od inej častice ako sociálna zložka rovnice rýchlosti. Stav a správanie častice môžeme definovať prostredníctvom:

1. vektoru rýchlosti $v_i(t)$ – ktorý predstavuje vnútorný stav častice v aktuálnom čase,
2. kognitívnej zložky – ktorá tvorí základ tendencie častice vracať sa k miestam, ktoré ich uspokojovali najviac,
3. sociálnej zložky – ktorá predstavuje spoločenskú normu a pravidlá, ktoré sa snažia častice udržať.

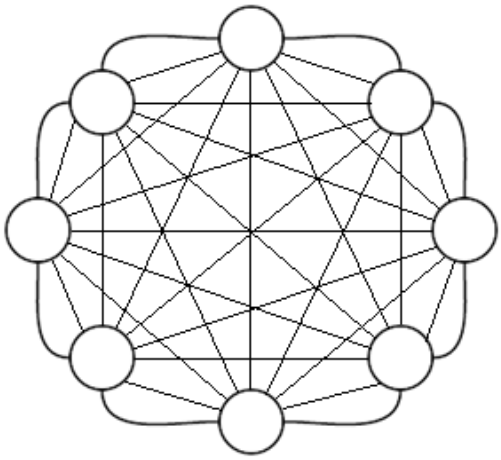


Obr. 6. Ukážka vplyvu zložiek a aktuálneho stavu na vektor pohybu častice v dvojdimenzionálnom priestore.

Model PSO je rozdelený na dva hlavné algoritmy Global Best PSO (gbest PSO) a Local Best PSO (lbest PSO), ktoré sú závislé na kognitívnom a sociálnom komponente vektora rýchlosti. Postup činnosti algoritmov je veľmi podobný, ale líšia sa v topológii sociálnej sieti častíc a v rovniciach použitých na výpočet rýchlosti a najlepších pozícii.

2.3.1.2 Global Best PSO

V gbest PSO susedí každá častica s ostatnými časticami v roji. Topológiu vzťahov v tejto sieti častíc je možné vykresliť prostredníctvom hviezdicovej topológie (Obr. 7), pre ktorú platí, že častice si aktualizujú sociálnu zložku na základe najlepšej pozície v roji. Keďže prepojenosť častíc je úplná, tak dôsledkom je rýchlejšia konvergencia k riešeniu na úkor zníženej rozmanitosti riešení.



Obr. 7. Hviezdicová topológia siete častíc v gbest PSO.

V tomto prípade je výpočet rýchlosti častice v nasledujúcom kroku realizovaný nasledovne:

$$v_{ij}(t+1) = v_{ij}(t) + c_1 r_1(t)\left[y_{ij}(t) - x_{ij}(t)\right] + c_2 r_2(t)\left[\hat{y}_j(t) - x_{ij}(t)\right] \quad (2.2)$$

kde $v_{ij}(t)$ je rýchlosť častice i pre dimenzie $j = 1, \dots, n$ v čase t, $x_{ij}(t)$ je pozícia častice i v dimenzii j v čase t, c_1 a c_2 sú pozitívne akceleračné konštanty určujúce veľkosť príspevku kognitívneho a sociálneho komponentu a $r_{1j}(t), r_{2j}(t) \sim U(0,1)$ sú náhodné hodnoty v rozsahu [0,1] získané z rovnomerného rozdelenia. Tieto náhodné hodnoty reprezentujú stochastickú časť algoritmu.

Najlepšia pozícia častice získaná na základe vlastnej skúsenosti y_i pre časticu i je najlepšia navštívená pozícia od začiatku algoritmu. Pri riešení problému maximalizácie úžitku vypočítame najlepšiu pozíciu v ďalšom kroku nasledovne:

$$y_{ij}(t+1) = \begin{cases} y_i(t), & ak\ f\big(x_i(t+1)\big) \le f(y_i(t)) \\ x_i(t+1), & ak\ f\big(x_i(t+1)\big) > f(y_i(t)) \end{cases} \quad (2.3)$$

kde f je fitnes funkcia. Podobne ako v evolučných algoritmoch, tak aj v PSO je kvalita častice (resp. kvalita riešenia) kvantifikovaná fitnes funkciou. Globálne najlepšia pozícia v roji je vybratá ako maximálna hodnota spomedzi všetkých častíc v roji:

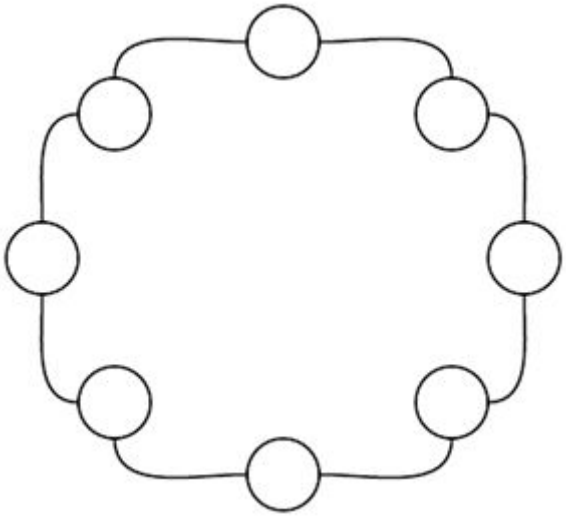
$$\hat{y}(t) = \max\{f(x_0(t), \dots, f(x_n(t))\} \quad (2.4)$$

Pseudokód algoritmu gbest PSO pre roj R:

```
repeat
    for each castica i = 1, ..., R.length do
        if f(R.x_i) > f(R.y_i) then
            R.y_i = R.x_i
        end
        if f(R.y_i) > f(R.ŷ) then
            R.ŷ = R.y_i
        end
    end
    for each castica i = 1, ..., R.length do
        vypocitaj rychlost pomocou rovnice 2.2
        vypocitaj poziciu pomocou rovnice 2.1
    end
until podmienka ukoncenia
```

2.3.1.3 Local Best PSO

Pre lbest PSO platí, že okolia susedov každej častice sú menšie ako v gbest PSO. Topológia siete vzťahov medzi časticami je prstencová (Obr. 8). Na rozdiel od gbest PSO je lbest PSO menej prepojený, čo spôsobuje zvýšenú rozmanitosť riešení. Väčšia rozmanitosť znižuje šancu na uviaznutie v lokálnom minime, avšak na úkor pomalšej konvergencie k riešeniu.



Obr. 8. Prstencová topológia siete častíc v lbest PSO.

Rýchlosť je vypočítaná podobne ako v prípade gbest PSO:

$$v_{ij}(t+1) = v_{ij}(t) + c_1 r_1(t)\left[y_{ij}(t) - x_{ij}(t)\right] + c_2 r_2(t)\left[\hat{y}_{ij}(t) - x_{ij}(t)\right] \quad (2.5)$$

s tým rozdielom, že $\hat{y}_{ij}$ určuje najlepšiu pozíciu, ktorá bola nájdená medzi susednými časticami i pre dimenziu j. Lokálne najlepšia pozícia $\hat{y}_i$ je teda najlepšia pozícia nájdená v množine susedov častice.

Dôležitou súčasťou metódy lbest PSO je dodatočné pridelenie informácie o susedoch pre každú časticu, pretože častice túto informáciu nevlastnia. Hlavným dôvodom je abstrahovanie výpočtu vzdialeností medzi každou dvojicou častíc za účelom zistenia susedského vzťahu. Druhým dôvodom je postupné rozšírenie najlepších riešení k všetkým časticiam bez ohľadu na ich aktuálnu pozíciu v priestore hľadania.

Pseudokód algoritmu lbest PSO pre roj R:

```
repeat
    for each castica i = 1, …, R.length do
        if f(R.x_i) > f(R.y_i) then
            R.y_i = R.x_i
        end
        if f(R.y_i) > f(R.ŷ_i) then
            R.ŷ_i = R.y_i
        end
    end
    for each castica i = 1, …, R.length do
        vypocitaj rychlost pomocou rovnice 2.5
        vypocitaj poziciu pomocou rovnice 2.1
    end
until podmienka ukoncenia
```

2.3.1.4 Nastavenia a parametre

Dôležitým krokom vykonávania algoritmu je inicializácia počiatočných pozícií častíc. Efektivita PSO je silne ovplyvnená práve inicializáciou, preto je odporúčané, aby boli častice rovnomerne rozprestreté v priestore hľadania. Ak sú niektoré regióny priestoru hľadania nepokryté už v inicializácii, tak je veľmi nízka pravdepodobnosť, že PSO tieto regióny odhalí. Rovnomerné rozdelenie počiatočných pozícií častíc po celom priestore hľadania je možné určiť pomocou:

$$x(t=0) = x_{min,j} + r_j\left(x_{max,j} - x_{min,j}\right), \forall j = 1, \dots, n \tag{2.6}$$

kde x_{max} a x_{min} sú ohraničenia konkrétnej dimenzie, n je počet dimenzií priestoru a $r_j \sim U(0,1)$ sú náhodné hodnoty z rozsahu [0,1] z rovnomerného rozdelenia. Počiatočný vektor pohybu musí byť pri inicializácii nenulový, aby častice dokázali preskúmať čo najviac priestoru bez vplyvu jednotlivých zložiek.

Ďalším dôležitým nastavením PSO je podmienka ukončenia, pri ktorej má algoritmus ukončiť svoje vykonávanie. Pri výbere spôsobu ukončenia je nutné vziať do úvahy fakt, aby ukončovacia podmienka nezastavila PSO predčasne, keď sa podarí nájsť suboptimálne riešenie a aby predchádzala pretrénovaniu. V reálnych aplikáciách boli použité nasledovné typy ukončení:

- ukončenie po dosiahnutí zadaného počtu iterácií – používa sa v konjunkcii s iným typom ukončenia alebo pri meraní hľadania pre konkrétnej časové dĺžky,

- ukončenie po nájdení akceptovateľného riešenia – akceptovateľné riešenie je definované výškou prahu, ktorá po prekročení ukončí vykonávanie algoritmu,
- ukončenie pri nezlepšovaní riešenia v určitom časovom úseku – je nutná parametrizácia dĺžky časového úseku a prahu priemernej odchýlky, ktorá musí byť prekonaná, aby algoritmus pokračoval,
- ukončenie pri neakceptovateľnom priemere priestoru hľadania – výpočet je realizovaný pomerom aktuálneho priemeru roja (najväčšia vzdialenosť akejkoľvek pozície častice od najlepšej globálnej pozície) k iniciálnemu priemeru roja v priestore, pričom je nutné nastaviť veľkosť hodnoty prahu, pri ktorom nastáva ukončenie,
- ukončenie pri neakceptovateľnom stúpaní cieľovej funkcie – výpočet je realizovaný prostredníctvom najlepších hodnôt v predchádzajúcom a aktuálnom kroku, pričom ukončenie spočíva v neprekročení prahu, ktorý predstavuje zlepšovanie riešenia v čase.

Základným krokom k riešeniu problémov prostredníctvom PSO je nastavenie parametrov, ktoré s nimi súvisia. Rôzne variácie modelov PSO môžu vlastniť špecifické parametre, avšak v základnom PSO modeli sú zadefinované nasledovné parametre:

- veľkosť roja – počet častíc v roji,
- veľkosť susedného okolia – priamoúmerne ovplyvňuje počet interakcií, ktoré sú vykonávané medzi časticami, pričom menšie susedné okolia sú menej náchylné na uviaznutie v lokálnom minime,
- počet iterácií – súvisí s ukončovacou podmienkou,
- akceleračné koeficienty – nastavenie vplyvu kognitívnej a sociálnej zložky pri výpočte pohybu častíc,
- stochastické parametre – náhodné vplyvy zložiek častice.

2.3.1.5 Modifikácie

I keď sa základný model PSO použil pri riešení klasických optimalizačných problémov, tak jeho problém spočíva v nekonzistencii pri konvergovaní k dobrým riešeniam. Z tohto dôvodu sú vytvorené modifikácie základného modelu za účelom zrýchlenia konvergencie a skvalitnenia nájdených riešení.

Jeden z aspektov, ktorý určuje efektivitu a správnosť optimalizačného algoritmu je vzťah medzi prieskumom a exploatáciou hľadania. Prieskum je schopnosť algoritmu hľadať cez rôzne oblasti priestoru hľadania za účelom nájsť optimálne riešenie. Exploatácia určuje

schopnosť koncentrovať hľadanie v okolí sľubných oblastí za účelom zlepšenia kandidátskych riešení. Dobrý optimalizačný algoritmus vhodne synchronizuje tieto protichodné ciele. V PSO sa pre tento účel používa modifikácia usmernenia rýchlosti pohybu (angl. *velocity clamping*), ktorá spôsobuje obmedzenie pohybu častíc pomocou ohraničení rýchlosti pohybu na základe:

$$v_{ij}(t+1) = \begin{cases} v'_{ij}(t), & ak\ v'_{ij}(t+1) < V_{max,j} \\ V_{max,j}, & ak\ v'_{ij}(t+1) \geq V_{max,j} \end{cases} \tag{2.7}$$

kde $V_{max,j}$ predstavuje maximálne ohraničenie rýchlosti častice v dimenzii j. Veľkosť je teda obmedzená podobne ako pozícia častice vo vzorci 2.3. Problémom tejto modifikácie je optimálne nastavenie ohraničenia, ktoré by nepodnecovalo pomalý alebo rýchly pohyb, resp. nezvýhodňovalo prieskum alebo exploatáciu.

Ako alternatíva pre modifikáciu usmernenia pohybu sa používa modifikácia váhou zotrvačnosti (angl. *inertia weight*). Váha je dodatočný parameter, ktorý vlastní každá častica. Predstavuje veľkosť s akou sa bude pohybovať (resp. zotrvávať) častica v aktuálnom vektore pohybu. V dôsledku dopĺňa vzorec 2.2 nasledovným spôsobom:

$$v_{ij}(t+1) = wv_{ij}(t) + c_1 r_1(t)\left[y_{ij}(t) - x_{ij}(t)\right] + c_2 r_2(t)\left[\hat{y}_j(t) - x_{ij}(t)\right] \tag{2.8}$$

kde w reprezentuje váhu zotrvačnosti. Podobne ako pri predchádzajúcej modifikácii, tak aj tu je problém s nastavením optimálnej váhy, ktorá by nezvýhodňovala prieskum priestoru hľadania pred exploatáciou alebo naopak. Časté je použitie dynamickej zmeny veľkosti váhy v čase. Dynamická zmena môže byť realizovaná niekoľkými spôsobmi:

- náhodné nastavenie váhy v každej iterácii, napr. prostredníctvom Gaussovho rozdelenia,
- lineárne klesajúcou hodnotou váhy, ktorá začína vo vysokej počiatočnej hodnote a lineárne klesá do minimálnej hodnoty váhy (správanie podobné ako pri simulovanom žíhaní),
- nelineárne klesajúcou hodnotou váhy, ktorá sa začína vo vysokej počiatočnej hodnote a rýchlosť klesania je podmienená pridaním dodatočných konštánt (správanie podobné ako pri simulovanom žíhaní),
- váha adaptujúca sa na základe fuzzy logiky, ktorá definuje konkrétne pravidlá (napr. s využitím hodnôt fitnes a iných), ktoré určujú aktuálny stav váhy,
- rastúca hodnota váhy správajúca sa opačne ako pri klesajúcich hodnotách.

2.3.1.6 Porovnanie PSO a EA

Všeobecne je PSO považovaná za techniku spadajúcu do EA, pretože s nimi zdieľa niekoľko významných čŕt. Avšak kvôli značným rozdielom je PSO v tejto práci separovaná od evolučných výpočtových techník a zaradená do rojovej inteligencie. Rozdiely a podobnosti sú opísané v nasledujúcich odsekoch.

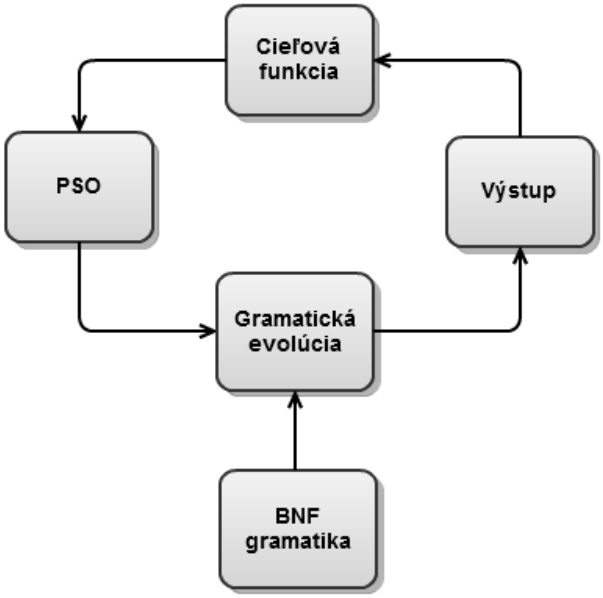
Oba prístupy sú založené na stochastickej optimalizácii, ktorá vychádza z populácie riešení. Riešenia sú získavané transformáciou populácií, ktoré sú inšpirované prírodnými fenoménmi s využitím špecifických operátorov. Hlavným rozdielom medzi PSO a EA je v tom, že zatiaľ čo v EA jedinci prehľadávajú priestor sami, tak v PSO je prehľadávanie založené na priamych sociálnych interakciách s inými jedincami, ktorí už vlastnia určitú znalosť prostredia. Ďalším dôležitým rozdielom je, že PSO má pamäť predchádzajúcich dobrých riešení, ktoré ovplyvňujú následný pohyb v priestore hľadania.

Nevýhodou PSO je z hľadiska reprezentácie nutnosť vytvorenia reprezentácie vlastností jedincov prostredníctvom vektorov s pohyblivou desatinnou čiarkou, keďže pohyb nie je diskrétny, ale spojitý. Oba prístupy používajú na kvantifikovanie kvality nájdených riešení fitnes funkciu. Hlavný rozdiel je v tom, že v EA je hľadanie priamo podmienené aktuálnou hodnotou fitnes funkcie, zatiaľ čo v PSO je hľadanie podmienené vlastnou skúsenosťou a skúsenosťou susedov. Fitnes funkcia je v PSO použitá len na kvantifikovanie kvality riešení, nie na transformáciu populácie, ako v prípade EA.

Mutácia v EA je rovnocenná so stochastickými parametrami r_1 a r_2, ktoré sú vložené do výpočtu nových pozícií častíc v PSO. Avšak kríženie nie je rovnocenné so žiadnou technikou v PSO, i keď do určitej miery je možné považovať pohyb častíc ku globálne najlepšej pozícii za takúto techniku. V konečnom dôsledku je možné tvrdiť, že PSO má slabý selektívny mechanizmus, zatiaľ čo cieľom EA je selekcia najlepších riešení.

2.4 Gramatický roj

Gramatický roj je technika, ktorá prepája optimalizáciu rojom častíc (PSO) s gramatickou evolúciou (GE) [9]. Cieľom gramatického roja je získať z oboch techník to najlepšie pri riešení dynamických problémov (Obr. 9). Pri kombinácii GE s PSO je nutné vyriešiť niekoľko problémov realizácie výpočtu, pretože kontext jednotlivých techník to nedovoľuje.



Obr. 9. Princíp činnosti gramatického roja.

Prvý problém súvisí s reprezentáciou jedinca, ktorý v gramatickej evolúcii predstavuje binárny, resp. celočíselný reťazec pevnej dĺžky. Keďže PSO hľadá riešenie v spojitom priestore, tak je nutné vykonávať zaokrúhlenie hodnoty pozície častice po každej vykonanej iterácii. Z použitej gramatiky je následne možné vytvoriť riešenie programu na základe zaokrúhlených celočíselných hodnôt.

Ďalší problém súvisí taktiež s reprezentáciou jedinca. Jedná sa o maximálnu celočíselnú hodnotu, ktorú môže jedinec v reťazci vlastniť. Maximálna hodnota musí byť použitá pri prehľadávaní priestoru pomocou PSO ako maximálna hodnota pre pozíciu a rýchlosť, ktorú môže častica nadobúdať, aby sa predchádzalo nežiaducim problémom s prepočítavaním nepovolených hodnôt, resp. pretekaním hodnôt do nepovolenej množiny hodnôt.

Implicitným problémom GE, ktorý pretrváva aj v gramatickom roji je voľba podmienky ukončenia transformácie. Ideálny spôsob je nastaviť maximálny prah transformácie, ktorý predstavuje zadefinovanie počtu možných prekladov neterminálnych znakov v sekvencii, ktoré sa vykonávajú pri transformácií genotypu na fenotyp. Problém má dopad na správanie PSO, pretože môžu nastať situácie, keď sa prekročí prah počtu transformácií a riešenie sa stane neakceptovateľným (dostane najnižšiu fitnes). To môže vytvoriť nevhodný región priestoru hľadania, pričom nie nutne všetky riešenia v tomto regióne môžu byť neakceptovateľné. Ideálne by bol stav, keby sa dokázali všetky riešenia preložiť do funkčného programu.

Spojenie PSO s GE nesie so sebou neriešiteľný problém. Jedná sa o absenciu kríženia medzi dvoma jedincami. Aj keď princíp funkčnosti PSO môže so sebou niesť znaky kríženia, tak sa nejedná o plnohodnotné nahradenia procesu kríženia. Dôsledkom absencie kríženia je nutné zadefinovanie fixnej dĺžky reprezentácie vektorov v PSO, pretože variabilitu dĺžky nie je možné dosiahnuť. Prehľadávaný priestor v PSO musí mať fixný počet dimenzií, aby nenastali problémy s konzistenciou priestoru prehľadávania a taktiež neexistuje spôsob ako implicitne meniť dĺžku bez procesu kríženia na strane GE.

PSO, ktorý je použitý v gramatickom roji je modifikáciou gbest PSO z Kap. 2.3.1.5. Prvá modifikácia spočíva v použití váh zotrvačností, pričom sa jedná o použitie metódy s lineárnym klesaním hodnoty váhy podľa nasledujúceho výpočtu:

$$w = w_{max} - \frac{w_{max} - w_{min}}{iter_{max}} * iter \tag{2.9}$$

kde w_{max} a w_{min} sú počiatočná a minimálna hodnota váhy nadobudnutá v poslednej iterácii, $iter_{max}$ je počet iterácii algoritmu a $iter$ je aktuálna iterácia. Druhá modifikácia je usmernenie pohybu, ktorá určuje maximálnu rýchlosť akú môže častica nadobudnúť.

Dôležitým krokom aplikácie gramatického roja je parametrizácia, ktorá súvisí s voľbou hodnôt jednotlivých parametrov za účelom zabezpečenia optimálneho správania sa algoritmu. Medzi parametre patria:

- počet dimenzií prehľadávaného priestoru v PSO,
- hodnoty x_{min} a x_{max}, ktoré ohraničujú pohyb v dimenzii, pričom x_{max} by mal zodpovedať maximálnej hodnote, ktorú môže nadobudnúť jedinec v genotype,
- maximálna možná hodnota rýchlosti V_{max} v PSO, ktorá by mala byť rovná hodnote x_{max},
- akceleračné koeficienty PSO c_1 a c_2,
- stochastické parametre PSO r_1 a r_2,
- počiatočnú a minimálnu hodnotu váhy zotrvačnosti w_{max} a w_{min},
- počet iterácií algoritmu $iter_{max}$.

2.5 Robocode

Robocode je multiplatformová open-source programovacia hra, ktorej cieľom je simulácia bojov robotických tankov v bojovej aréne. Ľudský hráč je programátor, ktorý naprogramuje správanie robota, pričom na prebiehajúce zápasy nemá vplyv. Cieľom hry je vytvorenie robota s umelou inteligenciou, ktorý bude schopný reagovať na udalosti vyskytujúce sa v súbojoch s inými robotmi za účelom porazenia protivníka.

Robocode beží na platforme Java a program robotov je písaný v tomto programovacom jazyku. Súboje sú simulované prostredníctvom vstavaného grafického prostredia. Hráči si môžu svojich robotov otestovať proti základnej sade robotov alebo v online ligách. Finálny výsledok je určený na základe skóre a počtu víťazstiev.

2.5.1 Online ligy

Veľká pozornosť Robocode komunity je venovaná práve ligám, pretože sa jedná o ideálne miesto na zmeranie síl proti iným robotom. Každý pokročilejší vývojár do ligy pravidelne prispieva, či už vylepšeniami starých robotov alebo kompletne novými robotmi. V Tab. 2 sú opísané existujúce ligy aj s informáciou ohľadom veľkosti ligy podľa počtu robotov, aj s počtom prispievajúcich autorov. Počet vývojárov robotov je určený podľa menných priestorov (angl. *namespace*) zapísaných robotov, kde prvá časť predstavuje autora.

Ligy umožňujú relatívne rýchle a komplexné testovanie robotov, ktorého výsledky sú postupne aktualizované a sprístupnené ihneď po zapísaní. Pozícia v lige je určená na základe skóre, ktoré sa určí z výsledkov proti iným robotom. Liga, v ktorej robot súťaží je zvolená na základe definovaných obmedzení robotov, pričom roboti sa môžu nachádzať vo viacerých ligách zároveň (napr. ľahšie váhové kategórie robotov sú testované aj v ťažších kategóriách). Ligy je možné rozdeliť podľa:

- typu súbojov (jednotlivci alebo tímy),
- obmedzenia veľkosti robotov na základe množstva vykonávaného programu špecifikované autormi Robocode (napr. volanie metódy predstavuje 3 Bajty).

Tab. 2. Online ligy (počty platné k 8.5.2013).

Názov ligy	Typ súboju	Obmedzenie	Počet robotov	Počet autorov
RoboRumble	1 proti 1	žiadne (MegaBot)	996	479
MiniRumble	1 proti 1	do 1500 Bajtov (MiniBot)	538	265
MicroRumble	1 proti 1	do 750 Bajtov (MicroBot)	404	213
NanoRumble	1 proti 1	do 250 Bajtov (NanoBot)	231	131
MeleeRumble	10 proti sebe	žiadne	360	226
MiniMeleeRumble	10 proti sebe	do 1500 Bajtov	182	108
MicroMeleeRumble	10 proti sebe	do 750 Bajtov	139	88
NanoMeleeRumble	10 proti sebe	do 250 Bajtov	82	54
TeamRumble	5 proti 5	žiadne	44	36
TwinDuel	2 proti 2	spolu do 2000 Bajtov	25	19

2.5.2 Robot

Robot je napísaný ako udalosťami riadený Java program. Princíp činnosti spočíva v cyklickom opakovaní hlavného tela programu robota kontrolujúceho správanie tanku, ktoré môže byť prerušené udalosťami posielanými zo simulačného prostredia [10]. Program robota obsahuje špecifické funkcie, ktoré sú volané pri zachytení prislúchajúcich udalostí, na ktoré sú zaregistrované. Tab. 3 obsahuje zoznam všetkých udalostí, ktoré môžu byť zaznamenané.

Robot sa skladá z 3 častí (Obr. 10) a jeho správanie môže byť definované akciami, ktoré sú v Tab. 4. Akčné členy, ktoré zabezpečujú vykonávanie akcií sú pohyb tanku, rotácia tela tanku, rotácia zbrane, rotácia radaru a streľba.

Tab. 3. Zoznam udalostí.

Názov udalosti	Opis udalosti
BulletHitEvent	zasiahnutie nepriateľa
BulletHitBulletEvent	vzájomné zasiahnutie striel
BulletMissedEvent	zasiahnutie steny bojovej arény
HitByBulletEvent	zasiahnutie nepriateľom
HitRobotEvent	kolízia s tankom
HitWallEvent	kolízia so stenou
ScannedRobotEvent	spozorovanie robota radarom
DeathEvent	ukončenie súboju prehrou
WinEvent	ukončenie súboju výhrou

Tab. 4. Zoznam akcií robotického tanku.

Názov akcie s parametrom	Opis akcie
ahead(double)	pohyb vpred o zadanú vzdialenosť
back(double)	pohyb vzad o zadanú vzdialenosť
doNothing()	vynechanie akcie (preskočenie kroku simulácie)
fire(double)	streľba so zadanou silou
resume()	povolenie zastaveného pohybu
setAdjustGunForRobotTurn(bool)	natavenie závislosti pohybu zbrane od tela tanku
setAdjustRadarForGunTurn(bool)	nastavenie závislosti pohybu radaru od zbrane
setAdjustRadarForRobotTurn(bool)	nastavenie závislosti pohybu radaru od tela tanku
stop()	zastavenie pohybu agenta
turnGunLeft()	otočenie zbrane doľava
turnGunRight()	otočenie zbrane doprava
turnLeft()	otočenia tela tanku doľava
turnRight()	otočenie tela tanku doprava
turnRadarLeft()	otočenie radaru doľava
turnRadarRight()	otočenie radaru doprava

Obr. 10. Anatómia robotického tanku.

Výsledok zápasu je vyhodnotený na základe počtu súbojov, ktoré sa vykonávajú v rade. Súboj je ukončený, keď zostane v aréne len jeden tank. Tanky sú zničené, ak ich úroveň energie klesne na nulu. Energia môže počas súboja stúpať a klesať, čo závisí od nasledujúcich pravidiel:

- tank získa energiu, ak zasiahne strelou iný tank,
- tank stratí energiu, ak je zasiahnutý strelou,
- tank stratí energiu, ak je v kolízii s iným tankom alebo stenou,
- tank stratí energiu, ak vykoná streľbu, pričom úroveň straty je proporcionálna úrovne sily strely.

Streľba je obmedzená zahrievaním hlavne, ktorej teplota rastie proporcionálne silou streľby. Teplota časom klesá a streľba je umožnená, až keď má teplota nulovú hodnotu.

Počas hry je možné získavať informácie prostredníctvom indikátorov stavu tanku. Tie sa dajú následne použiť pri vykonávaní programu robota a zvýšiť tak komplexnosť jeho správania. Zoznam indikátorov je opísaný v Tab. 5.

Tab. 5. Zoznam indikátorov tanku.

Názov indikátoru	Opis indikátoru
battlefieldHeight	výška arény
battlefieldWidth	šírka arény
energy	aktuálna energia tanku
gunHeading	smer natočenia zbrane
gunHeat	teplota hlavne
heading	smer natočenia tela tanku
radarHeading	natočenie radaru
time	čas kola
velocity	aktuálna rýchlosť tanku
x	horizontálna pozícia tanku v aréne
y	vertikálna pozícia tanku v aréne

Dôležitou súčasťou dynamického prispôsobovania správania je možnosť reagovania na správanie nepriateľského tanku, ktoré zaznamenávame pomocou radaru. V prípade zaznamenania zachytávame udalosť *ScannedRobotEvent*, ktorá obsahuje niekoľko použiteľných indikátorov stavu nepriateľského tanku (Tab. 6).

Tab. 6. Zoznam indikátorov pre udalosť *ScannedRobotEvent*.

Názov indikátoru	Opis indikátoru
bearing	natočenie nepriateľského tanku vzhľadom na vlastné natočenie
distance	vzdialenosť nepriateľského tanku
energy	aktuálna energia nepriateľského tanku
heading	natočenie nepriateľského tanku
velocity	rýchlosť nepriateľského tanku

2.5.3 Pohyb

Pohyb je jednou z dvoch najdôležitejších súčastí tvorby správania robota. Návrh pohybu je nutné realizovať so zámerom splnenia troch požiadaviek:

- vyhýbanie sa strelám protivníka,
- udržiavanie si vhodnej pozície pre vykonanie streľby na protivníka,
- veľkosť programu.

V Robocode sa počas rokov vyvinulo viacero rozšírených techník, ktoré sú použité vo veľkom množstve existujúcich robotov. Každá z techník zvláda vyššie definované požiadavky na rôznych úrovniach, ktoré následne podmieňuje ich použitie. Podrobné informácie k existujúcim technikám pohybu používaných v Robocode sú uvedené v prílohe C.1.1.

2.5.4 Streľba

Streľba sa týka najmä vhodnej voľby zameriavania protivníka so zámerom jeho úspešného zasiahnutia. Taktiež berieme do úvahy veľkosť programu. Podobne ako v prípade pohybu sa vyvinulo mnoho riešení, ktoré sú podrobne rozpracované v prílohe C.1.2. Návrh vhodnej metódy zameriavania spočíva najmä v predikcii protivníkovho pohybu a určenie jeho budúcej pozície vzhľadom na jeho vzdialenosť, pohyb a ďalšie indikátory. Pri použití pokročilejších metód pohybu sa môže jednať o riešenie náročného problému. V takom prípade je dôležitý výber metódy, ktorá by minimalizovala protivníkovu nepredvídateľnosť.

2.6 Analýza existujúcich riešení

Robocode obsahuje vlastný repozitár, do ktorého umiestňujú autori svojich robotov. Veľké množstvo z nich obsahuje zdrojový kód programu, vlastnosti stratégií a techník použitých pri tvorbe, dosiahnuté výsledky a ďalšie dodatočné informácie. Pre účely práce bolo cieľom nájsť robotov, ktorí vznikli prostredníctvom automatického generovania programu robota evolučnými výpočtovými technikami. Ako podklad pre odborne zdokumentované riešenia v nasledujúcich kapitolách boli použité informácie z práce [11], ktorá opisuje prvé pokusy automatickej tvorby programov robotov.

2.6.1 GP-Bot

GP-Bot bol vytvorený za účelom porovnania stratégií získaných evolučnými výpočtovými technikami proti stratégiám, ktoré boli vytvorené manuálne [12]. GP-Bot sa úspešne zúčastnil ligy pre kategóriu HaikuBots, ktorá bola zameraná na vytvorenie robota, v ktorom sú súboje realizované jeden proti jednému, pričom program robota je obmedzený na 4 riadky kódu. Dôvod voľby kategórie tohto typu je generovanie riešení genetickým programovaním, ktoré produkuje dlhé riadky kódu. Na základe tohto faktu boli ostatné typy kategórii zamietnuté ako nevyhovujúce.

Princíp evolúcie robota spočíva v použití populácie jedincov, ktorí sú reprezentovaní pomocou LISP výrazov pozostávajúcich z funkcií a terminálnych znakov. Použitými funkciami sú logické a aritmetické výrazy, ktoré majú vstupné parametre a vracajú numerickú hodnotu. Terminálne znaky pozostávajú z matematických funkcií, hodnôt stavu robota získaných z indikátorov a konštánt. Evolúcia riešenia spočíva v použití genetického programovania.

Pri vývoji boli použité rôzne konfigurácie evolúcie, ako napríklad STGP (Strongly Typed Genetic Programming), kde sa typy vstupných parametrov a terminálnych znakov líšia a ADF (Automatically Define Functions), ktorá podporuje evolúciu podmnožín riešení. V konečnom dôsledku sa tieto techniky vynechali, pretože ich efekt nebol použiteľný pre hru Robocode.

Architektúra programu musela byť prispôsobená obmedzeniam veľkosti programu robota, takže sa muselo upustiť od niektorých funkcionalít robota. Ako prvé sa vynechalo používanie rotácie radaru, ktorý sa otáčal zároveň s hlavňou tanku. Druhou výnimkou bola streľba, ktorá bola zadefinovaná ako numerická konštanta vyskytujúca sa kdekoľvek v kóde. Hlavný cyklus programu obsahoval jeden riadok kódu, ktorý otáčal hlavňou tanku v jednom smere za účelom zaznamenania protivníka radarom. Zvyšné tri riadky kontrolovali jeden akčný člen, ktorého vstup bol získaný evolúciou. Štruktúra programu vyzerala nasledovne:

```
while(true){
   turnGunRight(value); // hodnota otocenia je maximalna mozna
}

onScannedRobot(){
   moveTank(<GP#1>); // posunutie tanku smerom k evolvovanej hodnote
   turnRight(<GP#2>); // otocenie tanku smerom k evolvovanej hodnote
   turnGunRight(<GP#3>); // otocenie hlavne smerom k evolvovanej hodnote
}
```

Fitnes jednotlivých riešení bola vypočítaná na základe dvoch prístupov:

1. Výber protivníkov bol realizovaný zo sady existujúcich tankov, pričom pre testovanie každého jedinca v populácii boli z tejto sady náhodne vybraté 3 tanky, na ktorých sa realizovalo testovanie.
2. Výpočet fitnes funkcie bol realizovaný na základe výpočtu aký sa používal v turnaji: $F = \frac{S_P}{S_P+S_A}$, kde F je výsledné pomerové skóre, S_P je skóre získané robotom a S_A náhodne vybraným protivníkom. Konečná hodnota fitnes je určená priemernou hodnotou získaného skóre so všetkými protivníkmi.

Keďže výpočet fitnes prostredníctvom evolučného algoritmu s parametrami opísanými v Tab. 7. bol výpočtovo náročný, tak sa použilo distribuované počítanie fitnes hodnôt medzi viac ako 20 počítačov. Výber najlepšieho riešenia po ukončení evolučného algoritmu spomedzi všetkých riešení spočíval v ich otestovaní proti skupine 12 rozličných robotov. Na základe výsledkov sa vybralo najvhodnejšie riešenie, ktoré sa označilo ako GP-Bot.

Tab. 7. Parametre evolučného algoritmu.

Parameter	Hodnota
populácia	256 jedincov
podmienka ukončenia	zastavenie pri nezlepšovaní fitnes funkcie
mutácia	zmena podstromu v uzle alebo liste
kríženie	výmena podstromov dvoch riešení
selekcia	turnaj o veľkosti 5
elitizmus	2 najlepší jedinci postupovali bez modifikácie

Výsledkom GP-Bota v súťaži HaikuBots bolo 3. miesto, pričom programy ostatných robotov boli manuálne vytvorené. Z všeobecného hľadiska je GP-Bot úspešný, pretože:

- je konkurencie schopný v súboji s manuálne vytvorenými robotmi,
- učenie GP-Bota je podmienené súbojmi s inými typmi robotov.

2.6.2 Koevolúcia robotov s využitím SCALP

SCALP (Spatial Co-Evolution in Age Layered Plans) je používaný ako evolučné prostredie za účelom podpory koevolúcie. SCALP pozostáva z niekoľkých oddelených vrstiev v priestore a využíva koncept starnutia v jednotlivých vrstvách. SCALP vrstva pozostáva z mriežky prepojených uzlov so 4 priamo susediacimi a 4 diagonálne susediacimi uzlami [13].

V každom uzle spoločne žije hostiteľ a parazit. Pre každú generáciu je v každom uzle realizovaný boj medzi parazitom so susediacimi uzlami, v ktorých dostávajú obaja rôzne skóre. Myšlienka tejto implementácie spočíva v tom, že prostredníctvom koevolúcie môže predátor dobehnúť niektorú z koristí. Na druhej strane, pre korisť je menej podstatné, že dokáže predbehnúť viacej predátorov ako to, že dokáže predbehnúť viacej koristí.

Skóre hostiteľa je vypočítané prostredníctvom súčtu skóre z bojov proti parazitom, pričom parazit získava skóre na základe jeho najhoršieho boja. Po vyhodnotení každého uzla je možné pre hostiteľov a parazitov preniesť potomka do nového uzla v ďalšej generácii. 40% uzlov v novej generácii sú potomkami, ktorí majú jedného z rodičov na rovnakom mieste v predchádzajúcej generácii a ďalší je náhodne vybraný susediaci rodič, 10-20% riešení je zmutovaných.

Pre SCALP pozostávajúceho z vrstiev platí, že hostiteľ je testovaný aj proti parazitom, ktorí sa nachádzajú v uzle na nižšej vrstve s jeho susedným uzlom. Každá vrstva má určitý generačný limit, ktorý sa zvyšuje vyššími vrstvami. To v konečnom dôsledku spôsobuje, že nižšie vrstvy slúžia ako dodatočný genetický materiál pre riešenia na vyšších vrstvách. Na základe Robocode bola koevolúcia realizovaná spôsobom, že paraziti predstavovali manuálne napísané programy robotov a hostitelia boli automaticky vytvorené riešenia, ktoré im mali konkurovať.

Výsledkom koevolúcie s využitím SCALP bolo vybranie niekoľkých robotov s najlepším fitnes v jednotlivých generáciách za účelom otestovania ich úspešnosti bojovať so základnou množinou robotov. Roboti boli úspešní v súboji s väčšinou robotov s úspešnosťou nad 70%, pričom jediný problém bol s robotom Walls (jeho stratégia spočíva v pohybe blízkosti stien), ktorého nedokázali úspešne poraziť. Proti robotom, ktorí sú umiestnení v druhej polovici rebríčka ligy RoboRumble sa podarilo vytvoriť robota, ktorý ich porážal s úspešnosťou nad 50%, avšak proti robotom v prvej polovici sa nepodarilo vytvoriť konkurencie schopné riešenie.

3 Opis riešenia

3.1 Špecifikácia požiadaviek

Cieľ práce sa zameriava na splnenie dvoch hlavných požiadaviek. Prvou je použitie gramatického roja ako optimalizačnej evolučnej techniky za účelom úspešného vyriešenia definovaného problému, ktorý súvisí s programovacou hrou Robocode. Druhou požiadavkou, ktorá zároveň preverí úspešnosť splnenia prvej časti, je vyvinutie úspešného agenta v hre Robocode, ktorého konkurencieschopnosť by dosahovala úroveň analyzovaných agentov vytvorených pomocou iných optimalizačných evolučných techník (kap. 2.6).

3.2 Návrh

Kapitola návrhu je venovaná opisu návrhu riešenia, ktorý je rozdelený do jednotlivých podkapitol na základe aspektu riešených problémov. Cieľom je poskytnúť základný pohľad na koncept riešenia s použitím vybraných algoritmov, ktorý bude úspešne riešiť stanovené požiadavky finálnej práce. Návrh postupne prechádza cez všetky úrovne riešenia, od komplexných problémov až po elementárne.

3.2.1 Robot

Pri návrhu správania robota si je nutné uvedomiť, že zložitosť správania je priamoúmerná výpočtovej zložitosti a je podmienená zvolenou metódou generovania programu robota, ktorá opisuje možnosti jeho správania sa v hre. To znamená, že najdôležitejšou súčasťou návrhu správania robota je výber vhodnej metódy generovania finálneho programu tak, aby bolo hľadanie výpočtovo nenáročné s tým, že vygenerované správanie bude komplexné a hlavne konkurencieschopné voči vybratej množine protivníkov. Pri výbere všeobecnej metódy generovania sme identifikovali dva hlavné prístupy (Tab. 8).

Tab. 8. Všeobecné metódy generovania programu robota.

Metóda	Zložitosť správania	Výpočtová zložitosť	Informácia
úplná	vysoká	vysoká	cieľom je dosiahnuť správanie manuálne vytvorených robotov
parciálna	vysoká	normálna (nie nutne)	generujeme len špecifikovanú časť robota

Úplná metóda generovania predstavuje riešenie, ktorého cieľom je vytvorenie finálneho správania robota, pričom sa snaží o paralelné vytváranie celého správania robota naprieč všetkými jeho modulmi. Výsledok tejto metódy je možné porovnať s manuálnou tvorbou programov robotov, pretože roboty vytvorené týmto prístupom majú taktiež definované úplné správanie.

Druhou metódou je parciálna metóda, ktorú je možné použiť vo viacerých situáciách, pričom ju je možné použiť aj ako nahradenie úplnej metódy. Dôvod je ten, že parciálna metóda umožňuje postupné generovania programu pre jednotlivé moduly robota a aj cielené generovanie programu do existujúcich programov. Oproti úplnej metóde generovania vlastní nezanedbateľnú výhodu, ktorá sa týka kratšej realizácie výpočtu, pretože jej cieľom je generovať len špecifikované časti robota, nie jeho úplné správanie. To spôsobuje výrazné ohraničenie stavového priestoru možných riešení. Vzhľadom na charakter zadania sa budeme pri návrhu riešenia zaoberať len parciálnou metódou generovania, pretože poskytuje viacero výhod:

- nižšiu výpočtovú zložitosť,
- usmernenie vývoja pre konkrétne moduly robota,
- podporu použitia viacerých druhov parciálnych metód,
- dostatočne rozsiahly priestor pre úspešnú realizáciu riešenia.

3.2.2 Optimalizácia

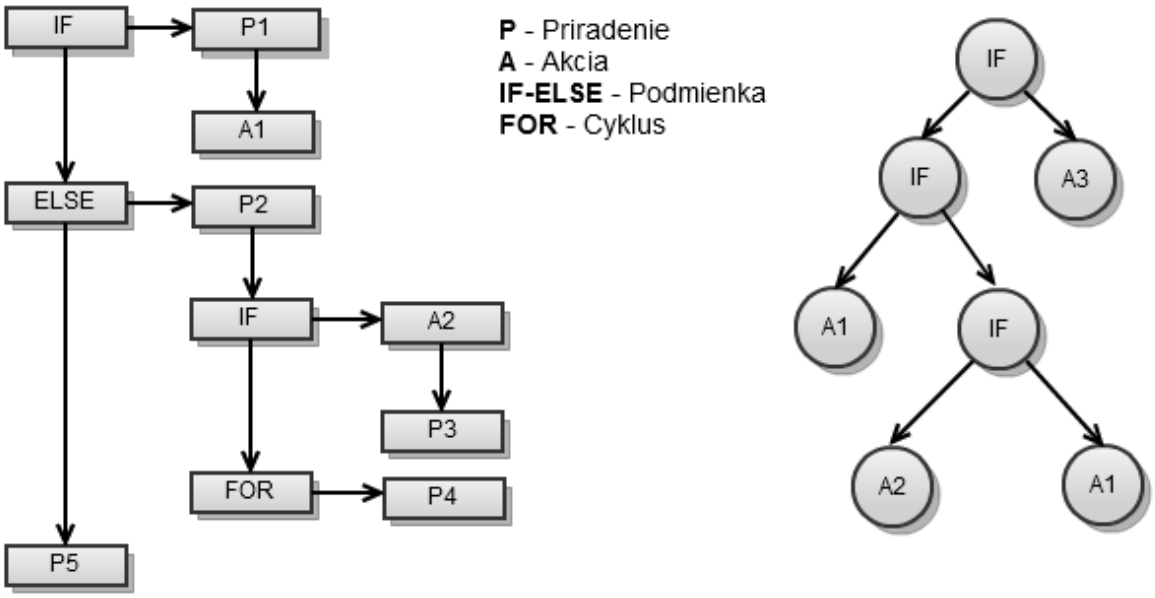
Návrh optimalizácie predstavuje hlavnú časť riešenia, ktorá stavia na použití parciálnej metódy generovania robotov. Návrh sa skladá z použitia vhodnej vyhľadávacej metódy v stavovom priestore možných riešení a komunikácie s ostatnými časťami aplikácie. Optimalizačný algoritmus je reprezentovaný postupnosťou ucelených krokov, ktoré sú vykonávané za účelom nájdenia takých robotov, ktorí by boli konkurencieschopní v boji proti základnej sade robotov poskytnutej hrou Robocode a inými robotmi. Kapitola sa skladá z návrhu základného algoritmu gramatického roja a jeho možnými rozšíreniami.

3.2.2.1 Stromová reprezentácia správania

Manuálnu tvorbu programov robotov je možné asociovať so skriptovaním umelej inteligencie v jednoduchých počítačových hrách. To znamená, že ak máme množinu povolených akcií, ktoré je možné v hre vykonať, tak výsledné správanie je možné špecifikovať rozhodovacím stromom (if-else), v ktorom uzly stromu obsahujú rozhodovacie prvky ovplyvňované stavom prostredia hry a listami sú povolené akcie.

Aj keď stromová metóda abstrahuje použitie častí syntaxe a logiky programov vyskytujúcich sa v sekvenčnom vykonávaní programov (funkcie, cykly, objekty, stavové premenné, postupnosti akcií), tak získavame rámec pre jednoduchú implementáciu optimalizačných techník založených na gramatickom roji s použitím netriviálnych gramatík. Podobný princíp bol použitý pri tvorbe správania GP-Bota. Ďalším zámerom zavedenia stromovej metódy je zmenšenie stavového priestoru možných riešení, tým pádom zníženie výpočtovej zložitosti hľadania riešenia. Na Obr. 11 je ukážka vygenerovaného sekvenčného a stromového programu.

Pri riešení je potrebné zvoliť vhodnú metodológiu vývoja robotov, ktorá predstavuje taký spôsob vývoja robotov, aby sme minimalizovali chyby a čas vzniknutý pri ich vývoji. Vhodným prístupom je progresívny prístup, ktorý znamená, že vývoj je zameraný na postupnú tvorbu od jednoduchších robotov, až k zložitejším. Minimalizácia chýb je zabezpečená použitím techník pre jednoduchšie problémy, kde je ľahšie overenie ich správnosti a minimalizácia času spočíva v nižšej výpočtovej zložitosti.



Obr. 11. Model sekvenčných (vľavo) a stromových programov (vpravo).

3.2.2.2 Gramatický roj

Gramatický roj je technika, ktorá je totožná s gramatickou evolúciou, ale ako metódu prehľadávania stavového priestoru používa optimalizáciu rojom častíc (PSO). Kap. 2.4 obsahuje vysvetlenie a teoretické základy ohľadom činnosti práce gramatického roja. Nasledujúce riadky sa venujú niektorým aspektom použitia gramatického roja,

od elementárnych rozhodnutí tvorby algoritmu cez parametrizáciu jeho hodnôt a vysvetlením konfigurácie jednotlivých parametrov, až po opísanie hlavných princípov činnosť algoritmu a detailne rozobratie jeho jednotlivých krokov.

Gramatický roj používa ako vyhľadávaciu funkciu PSO, ktorá môže vyhľadávať na základe globálne najlepšej pozície (gbest) alebo najlepšej hodnoty suseda (lbest). Podľa [9] sa ukázal gramatický roj úspešný už pri riešení problémov s nižším počtom častíc, pričom nižší počet častíc je predpokladom aj pri riešení nášho problému. Dôsledkom tohto tvrdenia je, že použitie lbest vyhľadávania nie je vyhovujúce, pretože definícia veľkosti susedstva pre veľký počet dimenzií a menší počet častíc nemusí byť vhodne nastavená, čo spôsobí, že susedné častice sa nemusia ovplyvňovať a teda budú konvergovať veľmi rýchlo do lokálnych miním. Preto je odporúčané použiť gbest vyhľadávanie.

Tab. 9 obsahuje zoznam parametrov s odporúčaným intervalom hodnôt a vysvetlením, ktoré je potrebné vhodne nastaviť pred vykonávaním algoritmu. Intervaly hodnôt nemusia byť pevné, ale vyjadrujú hodnoty, ktoré boli nastavené pri riešení iných problémov týkajúcich sa klasického PSO alebo gramatického roja.

Tab. 9. Parametre gramatického roja.

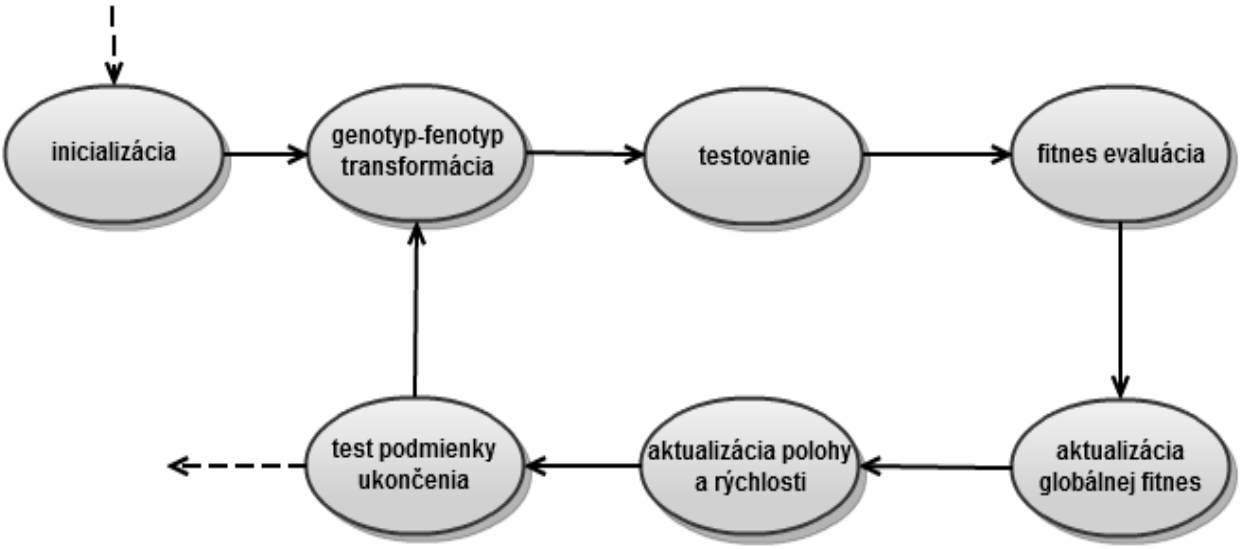
Parameter	Hodnota	Vysvetlenie
veľkosť roja	30 – 100	počet častíc v roji
dĺžka vektora častice	10 – 150	počet dimenzií polohového vektora častice
dolné ohraničenie hodnôt polohového vektora	0	minimálna veľkosť hodnoty v polohovom vektore
horné ohraničenie hodnôt polohového vektora	100/255	maximálna veľkosť hodnoty v polohovom vektore
dolné ohraničenie hodnôt vektora rýchlosti	-32	minimálna veľkosť hodnoty vo vektore rýchlosti
horné ohraničenie hodnôt vektora rýchlosti	32	maximálna veľkosť hodnoty vo vektore rýchlosti
kognitívny koeficient	0.5 – 1	rýchlosť konvergencie častice k najlepšej lokálnej pozícii
sociálny koeficient	0.5 – 1	rýchlosť konvergencie častice k najlepšej globálnej pozícii
počiatočný koeficient zotrvačnosti	0.9 – 1	iniciálna hodnota váhy zotrvačnosti, ktorú má častica pridelenú v prvej iterácii
koncový koeficient zotrvačnosti	0.2 – 1	finálna hodnota váhy zotrvačnosti, ktorú nadobudne častice v poslednej iterácii
počet iterácií	100 – 10000	počet iterácií PSO algoritmu
maximálny počet transformácií prekladu	50 – 500	maximálna dovolená veľkosť stromu, ktorú je možné vygenerovať algoritmom

Polohový vektor častice v roji reprezentuje genotyp kandidátskeho riešenia. V gramatickom roji sa vykonáva transformácia genotypu na fenotyp prostredníctvom definovanej gramatiky. Preto je potrebné pri generovaní programu vyriešiť problém hĺbky rozvetvenia rozhodovacieho stromu, ktorý predstavuje finálne správanie kandidátskeho riešenia vo forme programu robota. Pre účely nášho riešenia je ideálne zvoliť opakované použitie vektoru častice (po spracovaní hodnoty v poslednej dimenzii sa pokračuje znova od začiatku), ktoré je obmedzené maximálnym počtom transformácií prekladu neterminálnych znakov za účelom vygenerovania akceptovateľného riešenia. V prípade prekročenia prahu sa riešenie stane neakceptovateľným a ohodnotí sa najnižšou fitnes.

Algoritmus obsahuje dve modifikácie, ktorých cieľom je vylepšenie prehľadávania priestoru možných riešení. Prvou je pridanie váhy zotrvačnosti, ktorá predstavuje veľkosť s akou sa bude zotrvávať častica v aktuálnom vektore pohybu. Druhou je ohraničenie veľkostí hodnôt vektoru rýchlosti, ktorá zabraňuje príliš rýchlemu pohybu častíc v priestore.

Keďže pri hľadaní riešenia nášho problému nepoznáme finálne riešenie a v konečnom dôsledku ich môže byť viacej, tak je algoritmus ukončený len v prípade, ak je splnená jedna z nasledujúcich podmienok:

1. počet iterácií presiahne maximálny počet iterácii,
2. riešenie sa nevyvíja (uviazne v lokálnom alebo globálnom minime).



Obr. 12. Postupnosť krokov optimalizácie gramatickým rojom.

Algoritmus optimalizácie gramatického roja je znázornený na Obr. 12 a pozostáva z nasledujúcich krokov:

1. *Inicializácia* – vygenerovanie náhodných hodnôt pre polohový vektor a vektor rýchlosti každej častice.
2. *Genotyp-fenotyp transformácia* – vytvorenie programov robotov na základe polohového vektoru častíc.
3. *Testovanie* – testovanie kvality robotov v súbojoch.
4. *Fitnes evaluácia* – určenie fitnes hodnoty každej častice na základe výsledkov testovania.
5. *Aktualizácia globálnej fitnes* – v prípade nutnosti sa určí nová globálna fitnes hodnota.
6. *Aktualizácia polohy a rýchlosti* – výpočet nového polohového vektoru a vektoru rýchlosti každej častice.
7. *Test podmienky ukončenia* – ak je podmienka splnená, tak sa ukončí vykonávanie algoritmu.

3.2.2.3 Modulárne generovanie

Modulárne generovanie (MG) je optimalizácia, ktorá rozširuje optimalizáciu gramatickým rojom. Cieľom modulárneho generovania je vývoj len určitej časti správania robota. V prípade generovania robotov pomocou parciálnej stromovej metódy ide o generovanie len tých častí rozhodovacieho stromu, ktoré sú na začiatku algoritmu zvolené.

Základným predpokladom optimalizácie je príprava ostatných modulov programu robota, tak aby generovanie zlepšovalo kvalitu robota. To môžeme zabezpečiť prostredníctvom cielenej alebo striedavej metódy generovania (Tab. 10). Pre jednotlivé metódy môžeme definovať špecifický typ a formát testovania, to znamená, že môžeme priamo ovplyvňovať ciele vývoja a modelovanie správania robota.

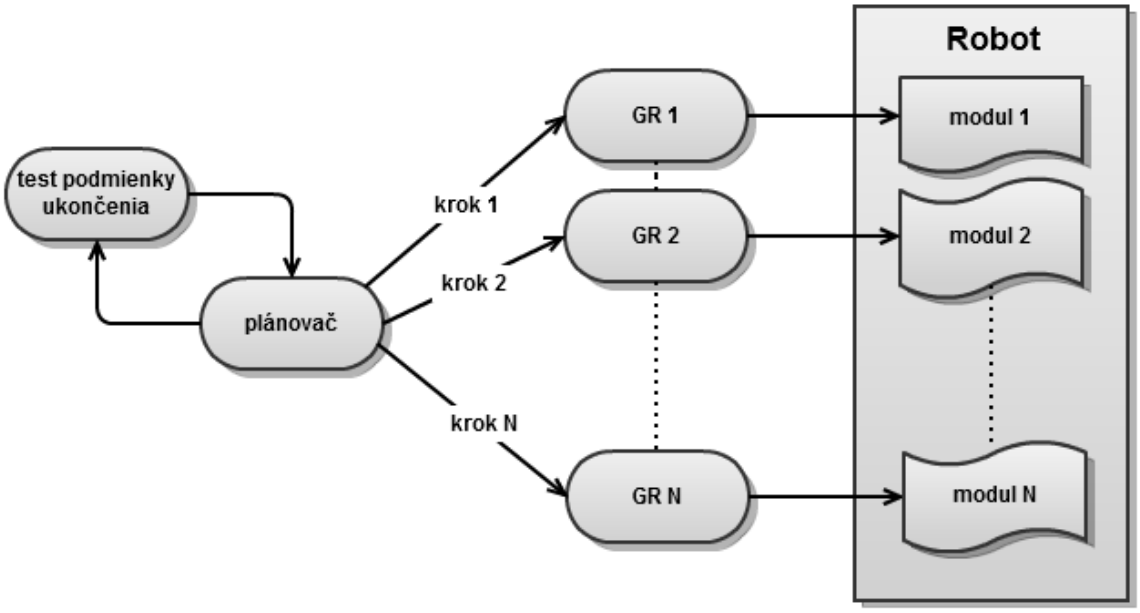
Tab. 10. Metódy MG.

Metóda	Výhoda	Nevýhoda	Princíp
cielené MG	pokračovanie vo vývoji robota	preddefinované správanie pre ostatné moduly robota	generujeme len konkrétnu časť robota, ostatné časti majú preddefinované správanie
striedavé MG	vývoj viacerých modulov robota zároveň	-	striedavo generujeme jednotlivé časti robota

3.2.2.4 Paralelné gramatické roje

Paralelné gramatické roje sú algoritmickým riešením optimalizácie pracujúcej na princípe striedavého MG. Základná myšlienka aplikácie striedavého MG je, že každý vyvíjaný modul musí byť reprezentovaný samostatným nezávislým gramatickým rojom. Algoritmus pre prácu s paralelnými gramatickými rojmi je plánovačom nad množinou gramatických rojov s tým, že jeho jedinou funkcionalitou bude prepínanie vývoja medzi jednotlivými modulmi (resp. rojmi).

Prepínanie práce medzi jednotlivými modulmi môže byť definované prostredníctvom viacerých metód. Pre účely riešenia nášho problému sme zvolili metódu postupného generovania s jednotkovým krokom (Obr. 13). Princíp tejto metódy spočíva v prepínaní vývoja modulov plánovačom, ktorý aktivuje v každom kroku iný gramatický roj. Jednotlivé gramatické roje vyvíjajú určitý modul finálneho robota a pre ostatné moduly používajú najlepšie riešenia rojov získané od začiatku behu algoritmu. Podmienka testu ukončenia algoritmu gramatických rojov je generalizovaná do plánovača.



Obr. 13. Postupné generovanie modulov s jednotkovým krokom.

3.2.3 Gramatika

Cieľ návrhu metódy generovania správania spočíva vo výbere gramatiky, ktorá musí spĺňať dve základné požiadavky. Prvou je menšia veľkosť množiny rozvetvujúcich sa produkčných pravidiel, aby sa predišlo problému s generovaním veľmi hlbokých stromov. Druhou je vhodná voľba terminálnych akcií a neterminálnych funkcií vzhľadom na ich počet, aby sa dalo pomocou gramatiky úspešne vygenerovať komplexné správanie robota.

Uvedené metódy generovania správania vychádzajú v použití špecifických gramatík, ktoré sú opísané a vychádzajú z poznatkov získaných zo zdroja [4]. Takýmto spôsobom sme identifikovali 3 regresívne metódy generovania programov:

1. behaviorálna regresia,
2. symbolická regresia,
3. parametrická regresia.

Všetky uvedené gramatiky je možné jednoducho rozšíriť o nové terminálne symboly a funkcie. V prípade rozšírenia je potrebné sledovať pomer počtu rozhodovacích a ostatných elementov, aby nenastala situácia, že rozvetvovanie stromu správania bude minimálne (nastane s malou pravdepodobnosťou) alebo nekonečné (ohodnotenie nulovou fitnes). Rozšírenie gramatiky môže byť realizované pomocou:

- stochastických funkcií – náhodné generovanie čísel z intervalu,
- matematických funkcií – násobenie, delenie, modulo a iné,
- numerických konštánt – hodnoty nemenné počas generovania gramatiky,
- goniometrických funkcií – sínus, kosínus a ich inverzné funkcie,
- indikátorov získaných z udalostí – rýchlosť nepriateľa, natočenie zbrane nepriateľa a iné,
- pomocných funkcií – maximálne možné hodnoty, vzdialenosť od steny a iné.

3.2.3.1 Behaviorálna regresia

Cieľom behaviorálnej regresie je vygenerovanie výrazu s numerickou návratovou hodnotu kombinujúc podmienené príkazy, funkcie a indikátory stavov prostredia, ktoré sú v našom prípade použité ako vstupné hodnoty pre konkrétne akcie. Prispôsobovanie správania robotov je umožnené prostredníctvom podmienených príkazov, ktoré rozvetvujú správanie robotov a vytvárajú tak rámec pre vygenerovanie komplexnejšieho správania robotov. Generovanie programu je riešené pomocou parciálnej metódy, pričom základná štruktúra programu je rovnaká ako v prípade GP-Bota:

```
while(true){
  turnRadarLeft(value); // hodnota otocenia je maximalna mozna
}

onScannedRobot(){
  moveTank(<vyraz>); // posunutie tanku
  turnLeft(<vyraz>); // otocenie tanku
  turnGunLeft(<vyraz>); // otocenie hlavne
}
```

Uvedená štruktúra programu nám umožňuje jednoduchým spôsobom ohraničiť stavový priestor hľadania, pričom ponechať relatívne vysoké možnosti generovania správania robotov. Taktiež je možné uvedenú štruktúru rozšíriť so zámerom pridania konkrétnych funkcionalít robota alebo pridať správanie aj pre iné herné udalosti. Vychádzajúc z analýzy Robocode prostredia v Kap. 2.5 a povolenej množiny akcií GP-Bota v Kap. 2.6.1 sa definovali niektoré funkcie a terminály, ktoré sa použijú ako základ pre vývoj robotov v BNF gramatike (Tab. 11) pomocou behaviorálnej regresie.

Tab. 11. Základné funkcie (neterminály) a terminály pre behaviorálnu regresiu.

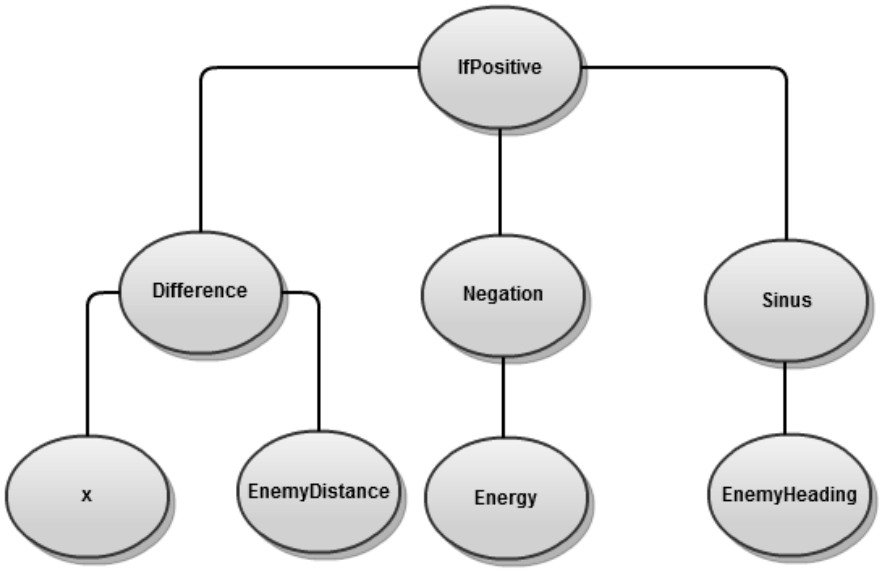
Názov	Typ	Informácia
Energy()	terminál	vráti aktuálnu energiu tanku
EnemyEnergy()	terminál	vráti energiu nepriateľského tanku
Heading()	terminál	vráti natočenie tanku
EnemyHeading()	terminál	vráti natočenie nepriateľského tanku
X()	terminál	vráti aktuálnu horizontálny polohu
Y()	terminál	vráti aktuálnu vertikálnu polohu
Width()	terminál	vráti veľkosť bojiska
EnemyDistance()	terminál	vráti vzdialenosť nepriateľského tanku
Fire(x)	funkcia	vykoná streľbu, ak je $x > 0$
IfGreater(x, y, left, right)	funkcia	ak je $x > y$, tak vráti výraz *left*, inak *right*
IfPositive(x, left, right)	funkcia	ak je $x > 0$, tak vráti výraz *left*, inak *right*
Sum(x, y)	funkcia	vráti súčet *x* a *y*
Difference(x, y)	funkcia	vráti rozdiel *x* a *y*
Abs(x)	funkcia	vráti absolútnu hodnotu *x*
Negation(x)	funkcia	vráti negáciu *x* (vynásobi hodnotu -1)
Sinus(x)	funkcia	vráti sínus *x*

Formálny zápis gramatiky vyzerá nasledovne (kvôli ukážke nie sú vyjadrené terminálne symboly definícií funkcií):

```
G = {N, T, P, S}
N = {<vyraz>, <strelba>, <if>, <if-nula>, <sucet>, <rozdiel>,
<abs>, <negacia>, <sin>}
T = {energia, e_energia, smer, e_smer, x, y, sirka, vzdialenost}
S = {<vyraz>}
P:
<vyraz>::= <strelba> | <if> | <if-nula> | <sucet> | <rozdiel> |
           <abs> | <negacia> | <sin> | energia | e_energia |
           smer | e_smer | x | y | sirka | vzdialenost
<strelba>::= strelba(<vyraz>)
<if>::= if(<vyraz>, <vyraz>, <vyraz>, <vyraz>)
```

```
<if-nula>::=  if-nula(<vyraz>, <vyraz>, <vyraz>)
<sucet>::=  sucet(<vyraz>, <vyraz>)
<rozdiel>::= rozdiel(<vyraz>, <vyraz>)
<abs>::= abs(<vyraz>)
<negacia>::= negacia(<vyraz>)
<sin>::= sin(<vyraz>)
```

Výsledná gramatika obsahuje základné funkcie, ktoré je možné použiť pri generovaní programu robota. Na Obr. 14 je ukážka výrazu vytvoreného použitím úplnej gramatiky a stromovej reprezentácie. V prípade použitia rozširujúcich funkcií by bola množina neterminálnych funkcií doplnená o zvolené funkcie, avšak na úkor zväčšenia stavového priestoru možných riešení. To isté platí aj pre zväčšenie množiny terminálov, ktoré je možné vykonať prostredníctvom stochastických funkcií alebo konštánt.



Obr. 14. Ukážka vygenerovaného výrazu prostredníctvom behaviorálnej regresie.

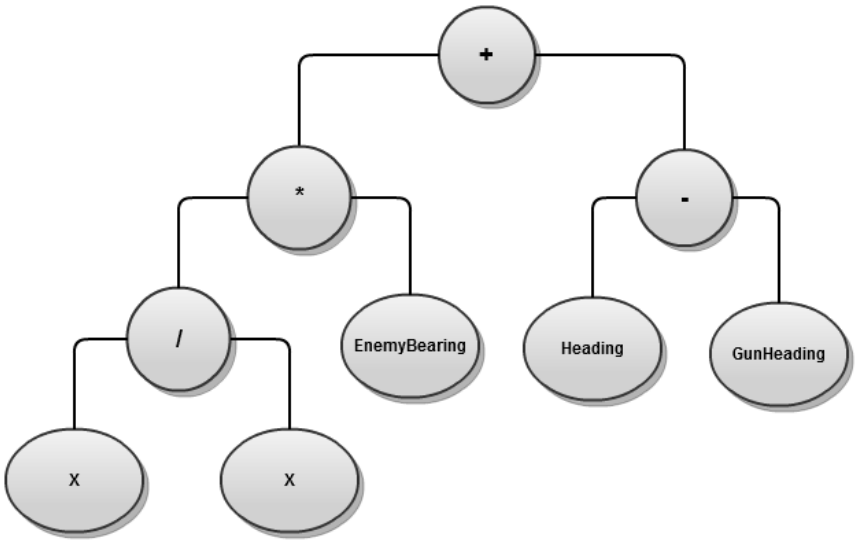
3.2.3.2 Symbolická regresia

Pri použití symbolickej regresie je hlavným cieľom objavenie regresnej funkcie, ktorá by čo najlepšie opisovala výstupné dáta. Symbolická regresia sa zvyčajne používa pre menej rozmerné stavové priestory riešení, avšak v našom prípade je priestor prehľadávania viacdimenzionálny. Dôvod je ten, že pre úspešné nájdenie vhodnej funkcie musíme používať viacero vstupných dát, aby sme mali vo výsledku zahrnuté informácie všetkých podstatných stavov prostredia alebo robotov.

Vygenerované regresné funkcie budú v podstate matematické formuly s numerickou návratovou hodnotou, ktorá sa následne použije ako vstupný argument pre konkrétny modul alebo funkciu. Formálny zápis gramatiky by mohol vyzerať nasledovne:

```
G = {N, T, P, S}
N = {<vyraz>, <operand>, <operator>, <indikator>}
T = {+, -, *, /, x, y, vzdialenost, smer, e_smer, e_natocenie, sirka, natocenie_zbrane}
S = {<vyraz>}
P:
<vyraz>::=  <vyraz><operator><vyraz> | <operand>
<operator>::=  + | - | * | /
<operand>::= x | y | vzdialenost | smer | e_smer | e_natocenie | sirka | natocenie_zbrane
```

Na Obr. 15 je zobrazená ukážka jednej z možných matematických formúl vygenerovaných symbolickou regresiou s vyššie uvedenou gramatikou. Ukážka sa dá ešte zjednodušiť na reprezentáciu Head-On metódy zameriavania uvedenú v prílohe C.1.2.



Obr. 15. Ukážka vygenerovanej matematickej formuly pomocou symbolickou regresie.

3.2.3.3 Parametrická regresia

Parametrická regresia spočíva v optimalizácií parametrov pre fixne zadanú funkciu. Metódu je možné zjednodušiť abstrahovaním gramatiky, čím sa vrátime k optimalizácií rojom častíc, kde hodnoty polohového vektoru môžu predstavovať optimalizované parametre. Avšak, na rozdiel od optimalizácie rojom častíc môžeme vykonať transformáciu polohových vektorov do vlastnej množiny parametrov, ktorých veľkosť nemusí nutne súvisieť s veľkosťou hodnôt

v polohovom vektore. Pri použití parametrickej regresie si je nutné uvedomiť, že počet dimenzií stavového priestoru riešení a výpočtová zložitosť je priamoúmerná počtu parametrov, ktoré sa snažíme optimalizovať.

Formálny zápis gramatiky by mohol vyzerať nasledovne:

```
G = {N, T, P, S}
N = {<vyraz>, <mnozina1>, <mnozina2>, mnozina3>}
T = {celociselne hodnoty z intervalu 0-100}
S = {<vyraz>}
P:
<vyraz>::=  <mnozina1>, <mnozina1>, <mnozina2>, <mnozina3>
<mnozina1>::=  0 | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8
| 0.9 | 1.0
<mnozina2>::=  2 | 3 | 5 | 7 | 9 | 11 | 13 | 17 | 21 | 23
<mnozina3>::=  0 | 10 | 20 | 40 | 60 | 80 | 100 | 150 | 200 | 500
```

Vygenerované parametre môžu byť použité napríklad pre výpočet funkcie F:

$$F = p_1 * f_1 + p_2 * f_2 + p_3 * f_3 + p_4 * f_4,$$

ktorá pozostáva zo súčtu optimalizovaných funkcií f_1, f_2, f_3, f_4, s parametrami p_1, p_2, p_3, p_4.

3.2.3.4 Porovnanie regresií

Každá z uvedených regresií má špecifické vlastnosti, ktoré je možné sledovať na základe definovaných kritérií relevantných pre problémovú oblasť. Prostredníctvom porovnania môžeme zvoliť vhodnú metódu pre účely nášho riešenia. Tab. 12 obsahuje súhrnné porovnanie regresií na základe nasledovných kritérií:

- spôsob reprezentácie – predstavuje finálny výstup vygenerovaného riešenia,
- veľkosť programu – predstavuje počet použitých volaní funkcií a premenných v kóde,
- počet pravidiel gramatiky – predstavuje veľkosť gramatiky,
- zložitosť riešenia – predstavuje veľkosť stavového priestoru riešení a je priamoúmerná výpočtovej zložitosti.

Tab. 12. Porovnanie regresií.

Typ regresie	Spôsob reprezentácie	Veľkosť programu	Počet pravidiel gramatiky	Zložitosť riešenia
behaviorálna	sekvencia príkazov	vysoká	vysoký	vysoká
symbolická	matematická formula	priemerná (nie nutne)	priemerný	nízka (nie nutne)
parametrická	konštanty	nízka	nízky	nízka

3.2.4 Fitnes funkcia

Základnou súčasťou evolučných optimalizačných techník je správny návrh fitnes funkcie. V Tab. 13 sú opísané hodnoty, ktoré sú poskytnuté hrou Robocode a je ich možné použiť po skončení súboja. Každá z hodnôt má definované výhody a nevýhody, ktoré by sa mali pri návrhu fitnes vziať do úvahy.

Tab. 13. Hodnoty merania fitnes funkcie.

Hodnota	Typ hodnotenia	Nevýhoda
umiestnenie (rank)	jednoduché hodnotenie	ignoruje špecifické aspekty hodnotenia
počet víťazstiev	jednoduché hodnotenie	ignoruje špecifické aspekty hodnotenia
skóre (score)	komplexné hodnotenie	pri dlhých a vyrovnaných súbojoch môže byť skreslená
percentuálne skóre	komplexné hodnotenie	môže byť náchylné na špecifické chyby
prežitie (survival)	špecifické hodnotenie	zvýhodňuje silných robotov
poškodenie (bullet damage)	špecifické hodnotenie	pri dlhých a vyrovnaných súbojoch môže byť skreslená
nárazové poškodenie (ram damage)	špecifické hodnotenie	pri dlhých a vyrovnaných súbojoch môže byť skreslená

Ďalším aspektom, ktorý je nutné zahrnúť do výpočtu a testovania robotov je postupný vývoj týchto robotov, ktorý vyplýva z použitia evolučnej optimalizačnej techniky. Preto je vhodné navrhnúť také riešenie, ktoré by testovalo robotov proti dynamickej množine nepriateľských robotov na základe ich kvality, pričom je nutné zobrať do úvahy riešený problém, pretože hľadáme robota, ktorého konkurencieschopnosť môže byť definovaná viacerými spôsobmi. Tab. 14 obsahuje identifikované prístupy testovania kvality robotov.

Tab. 14. Metódy testovania kvality robotov.

Metóda	Princíp	Výhoda	Nevýhoda
jednoduchá	testovanie proti konkrétnemu robotovi	jednoduchosť, rýchlosť	náchylnosť na chyby, špecializácia
komplexná	testovanie proti pevnej množine robotov	komplexnosť	zložitosť
progresívna	postupné odomykanie robotov na základe kvality	postupné učenie, nižšia zložitosť	definícia kvality
dynamická	množina robotov sa mení v čase	overenie komplexnosti riešenia	náhodný charakter konvergencie riešení
cielená	testovanie proti množine robotov za účelom vyvinúť konkrétnu časť riešenia	vhodné pre MG	definícia dosiahnutia cieľov

Pri výbere metódy testovania je hlavným rozhodnutím zameranie robota, to znamená, že či je našim cieľom vyvinúť špecializované alebo komplexné riešenie. Úroveň jednotlivých metód sa pohybuje medzi týmito dvoma typmi riešenia, pričom obe strany majú určité úskalia.

Problémom komplexnosti riešenia je, že kvalita robota klesá veľkosťou testovacej množiny nepriateľských robotov, pretože vývoj kvalitných úzko špecializovaných čŕt správania robota (konkrétnych techník) je relatívne pomalý. Na druhej strane, komplexné riešenie je robustné, a preto je vyššia šanca, že uspeje aj proti novým robotom, ktorý pracujú na podobnej mechanike. Problém úzkej špecializácie riešenia je evidentný a súvisí s tým, že správanie robota bude vytvorené len na jeden účel, čo je problematické pri testovaní proti testovacej množine skladajúcej sa z viacerých robotov pracujúcich na rozdielnej mechanike.

Ďalší problém súvisí s organizáciou hľadania riešenia v hre Robocode, pretože hra umožňuje definíciu počtu súbojov. Počet súbojov musí byť dostatočne veľký nato, aby sme minimalizovali chybu výsledku (čím viac súbojov, tým je výsledok presnejší) a dostatočne malú hodnotu nato, aby výpočet nebol príliš zložitý.

Posledný problém, ktorý je potrebné vyriešiť súvisí s výberom najlepšieho riešenia spomedzi všetkých nájdených riešení počas celého behu výpočtu, keďže je nutné vziať do úvahy situáciu, v ktorej môže byť fitnes hodnota niektorých jedincov vyššia dôsledkom stochastických vplyvov alebo sa riešenie až príliš mapuje na trénovaciu množinu a teda fitnes hodnota nebude opisovať kvalitu nájdeného jedinca tak, ako by sme si predstavovali. Riešením je racionálny výber množiny najlepších riešení, ktoré budú následne samostatne otestované proti testovacej množine robotov a ich výsledky spriemerované. Na základe výslednej priemernej hodnoty sa vyberie najlepšie riešenie nájdené počas výpočtu.

4 Implementácia

Nasledujúca kapitola sa venuje architektúre finálnej aplikácie, ktorá bude zabezpečovať výpočet a generovanie programov s následným testovaním. Softvérovú realizáciu architektúry z pohľadu programovacích jazykov a prostriedkov.

4.1 Architektúra

Prvým a základným problémom, ktorý je nutné vyriešiť je voľba architektúry softvérovej aplikácie riešiacej špecifikované požiadavky. Architektúra musí spĺňať funkcionálne požiadavky týkajúce sa efektívnosti a rozšíriteľnosti. Efektívnosť aplikácie súvisí s rýchlosťou hľadania riešenia, keďže úroveň časovej zložitosti evolučných výpočtových techník je priamoúmerná komplexnosti riešených problémov.

Pod rozšíriteľnosťou sa rozumie jednoduché pridávanie nových funkcionalít algoritmov za účelom jednoduchého doplnenia alebo úpravy určitého aspektu výpočtu (optimalizácie, experimenty). Rozšíriteľnosť je realizovaná pomocou modulárneho rozdelenia systému a definície spôsobu komunikácie medzi jednotlivými modulmi a externými systémami. Výpočtové časti systému môžeme rozdeliť do nasledovných modulov:

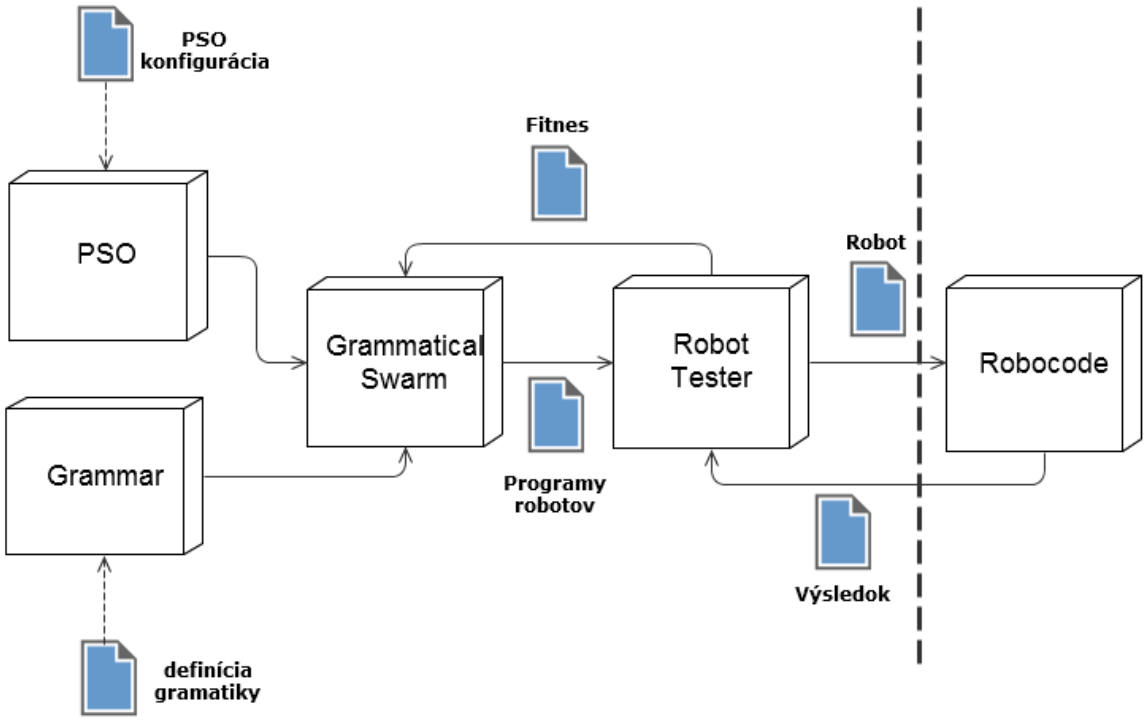
1. *PSO* – zabezpečuje výpočet pre algoritmus optimalizácie rojom častíc,
2. *Grammar* – zabezpečuje prácu s gramatikou a jej definíciu,
3. *Gramatical Swarm* – zabezpečuje výpočet algoritmu gramatického roja,
4. *Robot Tester* – zabezpečuje výpočet fitnes hodnoty robotov.

Pre jednotlivé moduly je nutné definovať vstupné a výstupné údaje, ktoré si budú medzi sebou posielať. Výsledný návrh aplikácie je zobrazený na Obr. 16.

Na začiatku výpočtu je nutné pripraviť parametre algoritmov a definíciu gramatiky na základe informácií v príslušných konfiguračných súboroch. Po inicializácii sa spustí vykonávanie hlavného algoritmu gramatického roja, ktorý v jednotlivých iteráciách generuje programy robotov, ktoré sú následne priamo testované v súbojoch v externom prostredí hry Robocode. Výstupom je výsledok súbojov, ktorý sa použije pre výpočet fitnes hodnoty robota. Algoritmus následne pokračuje ďalšou iteráciou.

Znázornený návrh nám spĺňa obe funkcionálne požiadavky, pretože hlavný algoritmus je možné jednoducho rozšíriť, modifikovať alebo prispôsobiť novým potrebám a v testovacom module je možné vykonávať optimalizáciu časovej zložitosti výpočtu, teda je možné splniť aj

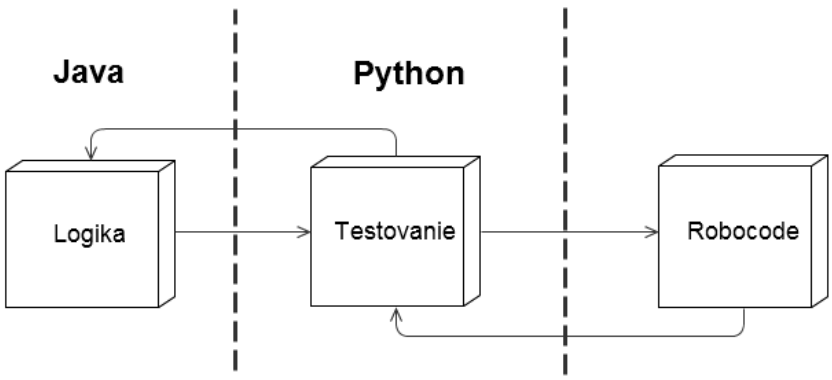
požiadavku efektívnosti. V prípade potreby je taktiež možné nahradiť celý modul bez toho, aby sa muselo zasahovať do iných modulov, pričom jedinou potrebou je dodržanie vstupno-výstupného protokolu posielaných dát medzi príslušnými modulmi.



Obr. 16. Modulárny návrh finálnej aplikácie.

4.2 Realizácia

Implementácia aplikácie je realizovaná v programovacích jazykoch Java a Python. Algoritmické moduly systému sú implementované v Jave (PSO, gramatický roj) a podporný modul pre paralelný výpočet, testovanie a komunikáciu s externým prostredím Robocode v Pythone (Obr. 17). Hlavným dôvodom je skúsenosť s paralelizáciou výpočtu pomocou Pythonu, teda išlo o efektívnejšiu implementáciu z pohľadu časových prostriedkov.



Obr. 17. Komunikácia hlavných modulov systému.

Hlavné problémy implementácie riešenia môžeme rozdeliť do nasledovných kategórií:

- overenie správnej činnosti algoritmov – konvergovanie algoritmov k riešeniu,
- paralelizácia – urýchlenie výpočtu pomocou paralelných výpočtových techník,
- logovanie – zápis výsledkov a ich vizualizácia.

Konvergovanie algoritmov k riešeniu sa podarilo úspešne vykonať pri jednoduchých príkladoch pre PSO a generovanie gramatiky. Pre komplexné problémy nie je možná automatická kontrola, takže je nutné ju vykonávať manuálne krokovaním cez jednotlivé logy iterácií. Programy robotov, výsledky súbojov a logy výpočtov sú zapisované do špeciálne navrhnutej hierarchickej štruktúry priečinkov tak, aby bolo jednoduché spracovávanie výpočtových informácií. Testovací priečinok obsahuje priečinky s informáciami každej iterácie algoritmu s detailnými informáciami ohľadom iterácie. Iterácia obsahuje priečinok pre každého robota (resp. časticu), kde sú uložené detailné informácie robotov. Takýmto spôsobom bolo možné jednoduché vytvorenie grafického používateľského rozhrania, ktoré bolo zamerané na textovú a grafickú vizualizáciu výsledkov.

4.3 Výpočtová zložitosť

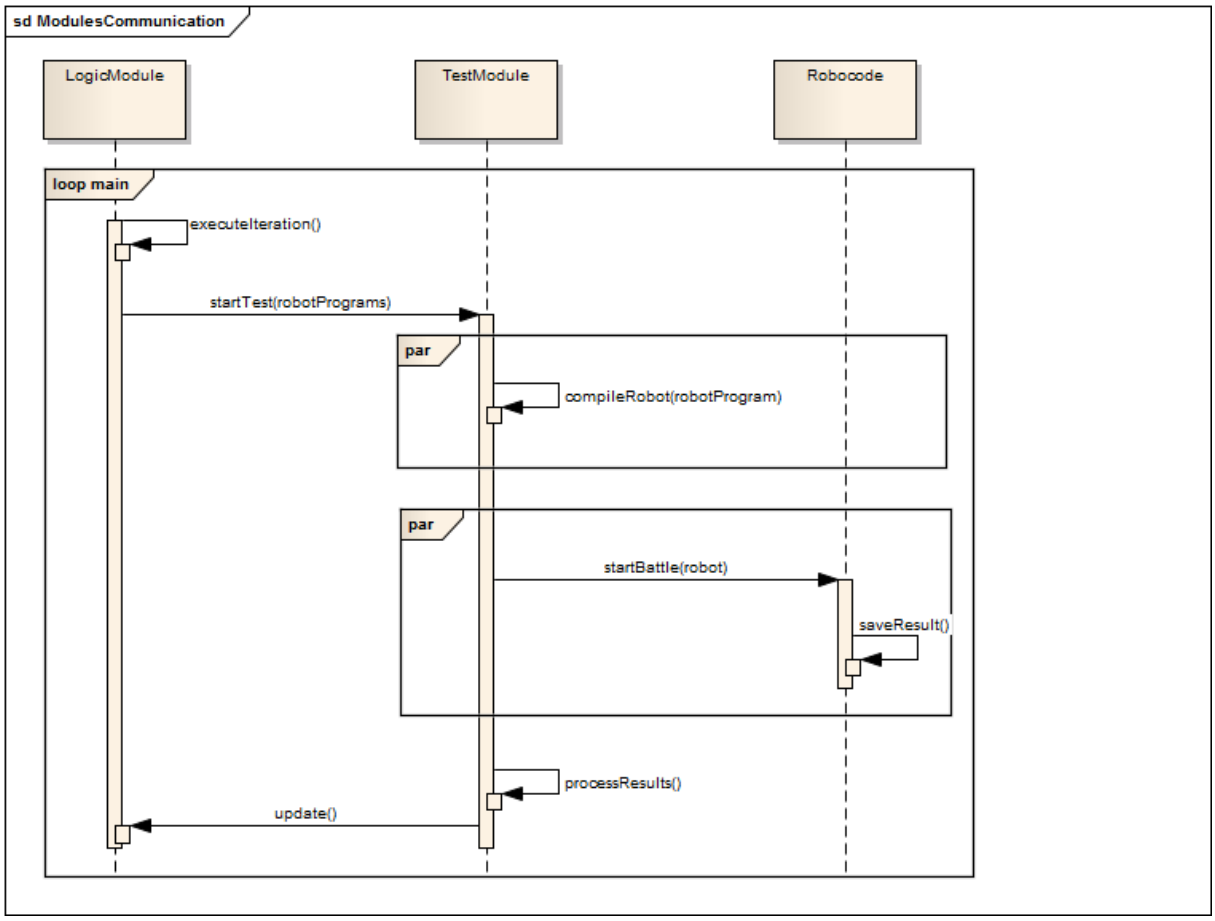
Najväčším problémom implementovaného riešenia je jeho výpočtová zložitosť, ktorú spôsobuje hra Robocode pri realizácii simulácie súbojov, ktoré sú potrebné pri výpočte fitnes hodnoty jednotlivých robotov. Zložitosť výpočtu T môžeme definovať pomocou nasledujúcej formuly [12]:

$$T = CasKola * PocetKol * PocetProtivnikov * VelkostRoja * PocetIteracii,$$

kde:

- $CasKola$ – je pevne nastavený na jednu sekundu (kvôli zjednodušeniu výpočtu),
- $PocetKol, VelkostRoja$ a $PocetIteracii$ – závisí od konfigurácie výpočtu,
- $PocetProtivnikov$ - závisí od veľkosti trénovacej množiny.

V rámci implementácie sa problém časovej zložitosti podaril čiastočne odstrániť paralelizáciou výpočtového procesu. Základný princíp aplikovanej paralelizácie je zobrazený na Obr. 18, ktorý spočíva v paralelizovaní kompilácie robotov a následného procesu testovania robotov v Robocode, kde je pre každého robota vyčlenené jedno vlákno (angl. *thread*). Dôsledkom optimalizácie bolo zníženie času kompilácie približne o 45% a času trvania súbojov približne o 65%.



Obr. 18. Sekvenčný diagram komunikácie modulov a paralelizácie výpočtu.

5 Testovanie

Testovanie implementovanej aplikácie a vývoj finálnych robotov prebiehal v troch fázach (pre každú regresívnu metódu). Výsledkom každej fázy je niekoľko najlepších typovo odlišných robotov, ktorí sa počas výpočtov vyvinuli. Každá kapitola obsahuje opis krokov a metód pri realizácii výpočtu s referenciami na príslušnú parametrizáciu, testovaciu a trénovaciu množinu robotov, grafy vývoja fitnes a zhrnuté výsledky vyvinutých riešení pomocou komparatívnej metódy s inými existujúcimi robotmi. Pre účely publikovania výsledkov práce bola zverejnená krátka dokumentácia týkajúc sa aplikácie gramatického roja na dokumentačnom webe Robocode (RoboWiki) so zámerom rozšíriť použiteľné techniky zamerané na vývoj robotov [14].

5.1 Testovanie behaviorálnej regresie

5.1.1 Rumbler

Cieľom prvej etapy vývoja bolo overenie správnosti práce optimalizácie pomocou gramatického roja, správnosť generovania programov na základe triviálnej gramatiky s unárnymi operátormi, správnosť komunikácie s Robocode a spracovania výsledkov za účelom korektného výpočtu fitnes proti jednoduchému robotovi. Testovanie bolo prekvapením, pretože sa podarilo vyvinúť riešenie (Rumbler), ktoré porazilo robota MyFirstRobot s viac ako 95% úspešnosťou. I keď strom riešenia má minimálnu veľkosť, tak sa potvrdzuje tvrdenie, že aj jednoduchý robot môže byť silný. Rumbler má pridelené označenie na základe jeho metódy útoku, teda postupný útok streľbou s nárazom.

5.1.2 Taurus

Pri vývoji Taura sa použila komplexnejšia gramatika definujúca viacero indikátorov stavu, podmienených premenných a matematických operátorov. Taktiež bola použitá menšia veľkosť roja na základe empirického výskumu podľa [9] a progresívna metóda výpočtu fitnes, ktorá odomyká robotom nové zápasy proti silnejším súperom pomocou získaných výsledkov s predchádzajúcimi súpermi.

Princíp činnosti Taura spočíva na podobnej mechanike ako v prípade Rumblera, čo môže byť čiastočne spôsobené aj tým, že Rumbler sa použil v testovacej množine ako nepriateľský robot. Taurus bol vyvinutý v 88. iterácii algoritmu, pričom lepšie riešenie sa nepodarilo nájsť až po 150. iteráciu, v ktorom bolo hľadanie ukončené.

Fitnes hodnota je vypočítaná na základe získaného skóre proti robotom v testovacej množine. Ak bol robot úspešný v súboji so slabšími robotmi (jeho skóre prekročilo definovaný limit pre ďalšiu úroveň), tak sa mu odomkol zápas s nasledujúcim silnejším súperom. Finálna fitnes robota je súčtom skóre, ktoré získal vo všetkých zápasoch. Detailný opis parametrov výpočtu s grafom vývoja fitnes je v prílohe C.2.1.

5.1.3 Focus

Pri vývoji Focusa bolo použité modulárne generovanie (kap. 3.2.2.3) s využitím paralelných gramatických rojov (kap. 3.2.2.4), ktorého princíp funkčnosti spočíva v automatickom striedaní vývoja modulov robota. Správanie robota je tak podmienené stavom troch nezávislých častíc v jednotlivých paralelných gramatických rojoch, z ktorých každý generuje riešenia pre konkrétny modul. Dôvodom zavedenia bolo odstránenie problému s nerovnomerným vývojom stromov riešenia v rámci jednotlivých modulov. Detailný opis parametrov výpočtu s grafom vývoja fitnes je v prílohe C.2.2.

Pri prístupe modulárneho generovania bolo nutné zvoliť vhodný spôsob výberu správania pre moduly, ktoré nie sú aktuálne vyvíjané. V našom prípade sme použili výber správania na základe globálne najlepšej častice v ostatných gramatických rojoch. Dôsledok výberu bolo použitie najlepšieho riešenia správania modulov (na základe výsledkov), pričom správanie vyvíjaného modulu môže kooperovať už s existujúcim správaním ostatných modulov.

Problém, ktorý nastal pri vývoji Rumblera a Taura s nerovnomerným generovaním stromu správania sa podarilo odstrániť, ale bolo nutné zaviesť aj definovanie počiatočného stavu stromu správania pre gramatiku, ktorá namiesto prázdneho výrazu obsahuje podmienený výraz IF-ELSE. To spôsobuje, že vždy budú použité prvé štyri stavy vo vektore častíc. Focus dokázal takýmto spôsobom vyvinúť riešenie pre svoje tri moduly (pohyb, natočenie tela a hlavne) správania s veľkosťou programov 4, 10 a 24.

Testovanie proti nepriateľským robotom prebiehalo prostredníctvom progresívnej metódy, pričom sa použil zoznam robotov s príslušne špecifikovaným limitom. Focus dokázal získať proti uvedenej skupine robotov 129 víťazstiev.

5.1.4 Carousel

Výsledky vývoja Carousela sú porovnateľné s výsledkami existujúcich robotov, keďže sa mu podarilo poraziť základnú sadu robotov s viac ako 80% úspešnosťou, aj keď robot Walls zostáva stále neprekonaný. Označenie Carousel bolo zvolené na základe jeho vyvinutého krúživého pohybu, ktoré je charakteristické aj pre GP-Bota. Vývoj robota sprevádzala zmena

použitých funkcií, ktoré menia stav robota z relatívneho natočenia tela a zbrane na absolútne. Pri testovaní bola použitá progresívna metóda, pričom zoznam nepriateľov sa oproti zoznamu použitého pri vývoji robota Focus rozšíril o pokročilejších robotov a limit na prechod k ďalšiemu robotovi bol zmiernený. Detailný opis parametrov výpočtu s grafom vývoja fitnes je v prílohe C.2.3.

5.1.5 Phoenix

V rámci testov s použitím behaviorálnej regresie sa dá konštatovať, že vývoj Phoenixa sa vydaril, pretože sa podarilo vyvinúť riešenie, ktoré je relatívne lepšie (vzhľadom na výsledky proti testovacej množiny) ako GP-Bot. Výpočet obsahoval na rozdiel od posledného výpočtu mnoho zmien, ktoré súviseli najmä s vyriešením problému časovej zložitosti. Detailný opis parametrov výpočtu s grafom vývoja fitnes je v prílohe C.2.5. Vývoj sprevádzali nasledovné zmeny:

- odobratie generovania správania pre modul posunu a jeho nahradenie stálym posunom vpred, keďže sa takéto správanie vyvinulo pri všetkých robotoch v kap. 5.1.1 – 5.1.4,
- zavedenie množiny konštánt vychádzajúcich z čiastkových hodnôt π,
- zmena veľkosti trénovacej množiny protivníkov kvôli výpočtovej zložitosti,
- zmena spôsobu testovania z progresívnej na komplexnú kvôli menšej veľkosti trénovacej množiny,
- oddelenie metódy streľby od generovania správania, keďže sa takéto správanie vyvinulo pri všetkých robotoch v kap. 5.1.1 – 5.1.4,
- manuálne nastavenie radaru pomocou jednouchej techniky Head-On kvôli nezávislosti pohybu radaru od zbrane.

5.2 Testovanie parametrickej regresie

Pri testovaní prostredníctvom parametrickej regresie je hlavným cieľom vytvorenie konkurencieschopného riešenia pre ľahšie váhové kategórie robotov. Dôvod je ten, že vlastnosťou parametrickej regresie je nízka veľkosť programov, pretože sa snažíme optimalizovať parametre manuálne vytvorených metód so zámerom nájdenia vhodnej kombinácie hodnôt.

5.2.1 Ringo 1.0

Pri návrhu manuálneho správania Ringa bolo cieľom použitie takej techniky, na ktorej by bolo možné názorne ukázať možnosti parametrickej regresie. Hlavným ťažiskom experimentu

je parametrizácia vplyvu hodnôt na pohyb robota, ktorý je v neustálom pohybe vpred. Nasledovné indikátory ovplyvňujú smer pohybu:

1. vzdialenosť k najbližšej stene (najkratšia kolmica na stenu vzhľadom na polohu),
2. západný dotykový senzor (vo vzdialenosti 150 pixelov od robota),
3. východný dotykový senzor (vo vzdialenosti 150 pixelov od robota),
4. severný dotykový senzor (vo vzdialenosti 150 pixelov od robota),
5. južný dotykový senzor (vo vzdialenosti 150 pixelov od robota),
6. vzdialenosť protivníka,
7. natočenie protivníka vzhľadom na aktuálne natočenie robota.

Indikátory vzdialeností musia byť normalizované, aby bolo možné použiť rovnakú gramatiku pre všetky parametre. Gramatika definuje rozsah hodnôt parametrov z intervalu $< -1.0, 1.0 >$.

Ringo je jeden z prípadov, kde sa vybralo riešenie s nižšou nájdenou fitnes (v skorších iteráciách výpočtu), pretože riešenia s vyššími fitnes nedosahovali stabilné výsledky proti testovacej množine robotov. Výsledné riešenie muselo byť ešte zoptimalizované, aby sa výsledný program zmestil do váhovej kategórie NanoRumble. Detailný opis parametrov výpočtu s grafom vývoja fitnes je v prílohe C.2.5.

5.2.2 Flex 1.0

Cieľom vývoja Flexa bolo rozšírenie myšlienok aplikovaných v existujúcom open-source robotovi sheldor.jk.Yatagan 1.0.5, ktorý je umiestnený medzi NanoBotmi na druhej pozícii, pričom rozšírené riešenie bude zapísané do turnaja MicroRumble. Vylepšenie spočíva:

- v zavedení metódy zameriavania prostredníctvom techník virtuálnych zbraní (angl. *Virtual Guns*) a výberu z možností (angl. *Multiple Choice*),
- v reprezentácií virtuálnych zbraní prostredníctvom stratégií, ktoré sú opísané parametrami ovplyvňujúcimi pohyb, zameriavanie a udržiavanie si konštantnej vzdialenosti od protivníka, ktorú si robot snaží udržať,
- výber vhodnej stratégie na základe ukladania si informácie úspešnosti jednotlivých stratégií v priebehu súboja.

Parametrická regresia bola použitá pri vývoji jednej zo stratégií ovládanie pohybu zbrane. Trénovacia množina pozostávala z náhodne zvolenej vzorky robotov nachádzajúcich sa na popredných miestach v súťaži MicroRumble. Výsledné riešenie muselo byť ešte čiastočne

zoptimalizované, aby sa zmestilo do váhovej kategórie MicroRumble. Detailný opis parametrov výpočtu s grafom vývoja fitnes je v prílohe C.2.6.

Prepínanie medzi zbraňami pracuje na princípe prepočítavania úspešnosti aktuálne zvolenej zbrane po ukončení súboja podľa formuly:

$$X_i = \frac{1+N_{il}}{1+N_{iw}},$$

kde X_i je úspešnosť i-tej stratégie, N_{il} je počet prehier a N_{iw} je počet víťazstiev i-tej stratégie. Aby sme zvýšili počiatočné šance každej stratégie, tak sme prirátali do čitateľa a do menovateľa 1. Na začiatku každého súboja sa zvolí stratégia, ktorá má najvyššiu úspešnosť X. Po ukončení súboja sa podľa výsledku inkrementuje príslušná hodnota N_{il} alebo N_{iw} pre aktuálne zvolenú i-tu stratégiu. Flex 1.0 obsahuje tri stratégie:

1. evolvovaná stratégia,
2. metóda pohybu *Orbit* v kombinácii s technikou zameriavania *Pattern Matching*,
3. metóda pohybu *Oscillation* v kombinácii s technikou zameriavania *Pattern Matching*.

5.3 Testovanie symbolickej regresie

Testovanie symbolickej regresie sa realizovalo ako nezávislý vývoj úrovne sily streľby pre robota Ringo 1.0. Výsledné riešenie bolo použité aj v robotovi s označením Flex 1.5.

5.3.1 Ringo 2.0

Úroveň sily streľby môže byť z intervalu $< 0.1, 3.0 >$. Cieľom vývoja je nájdenie vhodnej matematickej formuly, ktorá by sa použila pre výpočet sily streľby na základe indikátorov stavu hry. Pre tento účel je ideálne použiť symbolickú regresiu. Trénovacia množina pozostávala z náhodne vybratej množiny robotov nachádzajúcich sa v rebríčku NanoRumble pred robotom Ringo 1.0 a GP-Bota. Detailný opis parametrov výpočtu s grafom vývoja fitnes je v prílohe C.2.7.

Pri vývoji sa mnoho riešení zahodilo ako nevyhovujúcich, pretože symbolická regresia je náchylná na generovanie nekonečných stromov, čo spôsobovalo chyby pri kompilácii zdrojového kódu. Riešenie bolo vybraté z viacerých možností, ale okrem kvalitatívnej stránky riešenia sa do úvahy brala aj požiadavka veľkosti matematickej formuly, pretože výsledok mal byť použitý pri rozšírení robotov vystupujúcich v nízkych váhových kategóriách, kde je dôležité šetrenie miestom.

5.3.2 Flex 1.5

Rozšírenie robota Flex 1.0 sa týkal nasledovných bodov:

1. pridanie rovnakej matematickej formuly pre výpočet sily streľby ako v prípade Ringa 2.0,
2. pridanie 3 manuálne vytvorených stratégií,
3. pridanie dodatočného konceptu prepínania zbraní, ktorá limituje použitie rovnakej zbrane, ak počet prehier v posledných súbojov s touto zbraňou prekročí definovaný limit.

5.4 Testovanie kvality robotov

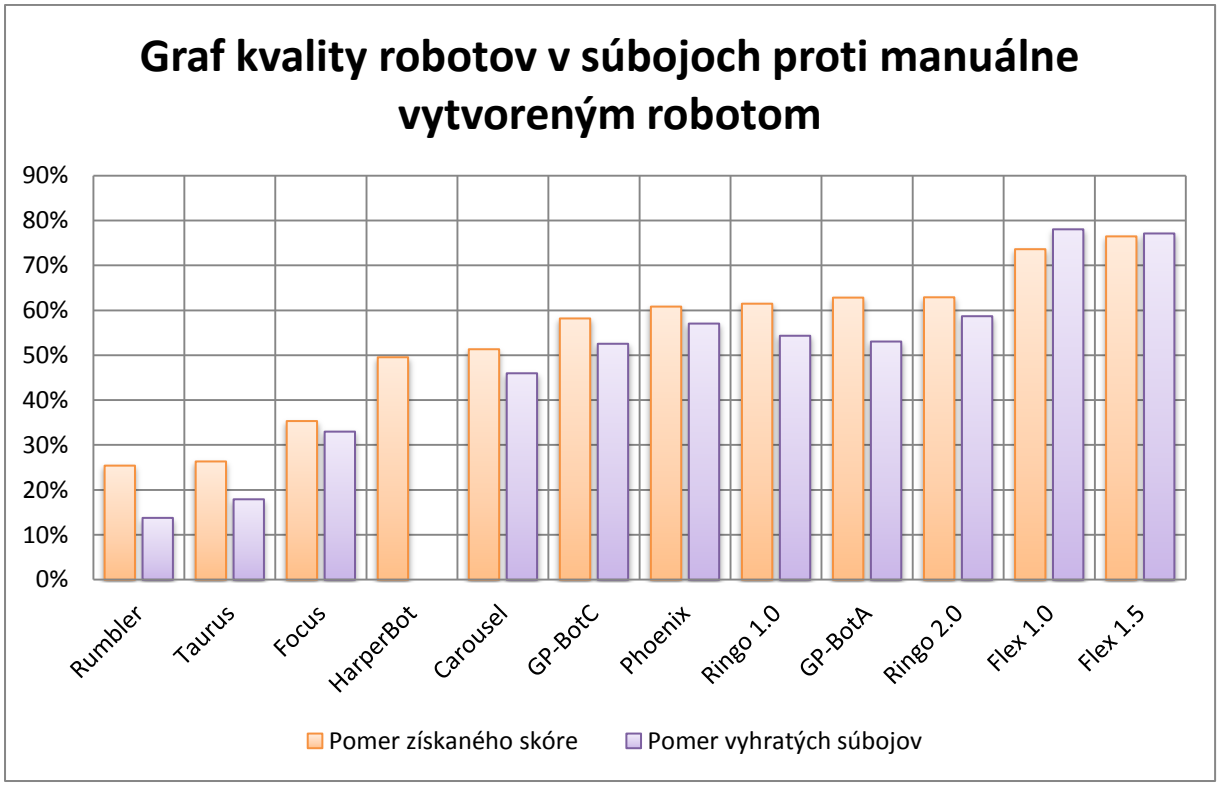
Nasledujúce kapitoly sú venované výsledkom získaných počas testovania kvality konkrétnych robotov. Výsledky a poradia v online ligách boli získané dňa 30.04.2013. Hodnota *úspešnosť* v grafoch predstavuje percentuálny pomer vyhraných súbojov a hodnota *skóre* predstavuje percentuálny pomer získaného skóre v súbojoch.

5.4.1 Testovanie proti manuálne vytvoreným robotom

Testovacia množina robotov (Tab. 15) bola zvolená na základe výsledkov testovania HarperBota [13], pričom jeho výsledky boli z tejto publikácie prebraté. Dôvod je ten, že robot nie je voľne prístupný a nebolo tak možné ho otestovať proti inej množine robotov. Na Obr. 19 je znázornený finálny graf výsledkov pre dva nezávislé testy zamerané na hodnotenie prostredníctvom percentuálneho skóre a počtu víťazstiev. Každá dvojica odohrala počas testovania medzi sebou 100 súbojov. Finálna tabuľka výsledkov sa nachádza v prílohe C.3.

Tab. 15. Testovacia množina manuálne vytvorených robotov.

Robot	Poradie v RoboRumble	Váhová kategória
sample.Crazy 1.0	975	NanoBot
sample.Tracker 1.0	964	MicroBot
sample.TrackFire 1.0	955	NanoBot
sample.RamFire 1.0	951	NanoBot
sample.VelociRobot 1.0	946	NanoBot
sample.SpinBot 1.0	934	NanoBot
sample.Walls 1.0	918	NanoBot
gg.Peryton 1.1	-	MegaBot
NDH.GuessFactor 1.0	698	MiniBot
dummy.micro.Sparrow 2.5	504	MicroBot
apv.Cannibal 1.1	431	NanoBot
apv.Aspid 1.7	187	MiniBot
abc.Tron 2.02	172	MegaBot
cx.mini.Cigaret 1.31	123	MiniBot



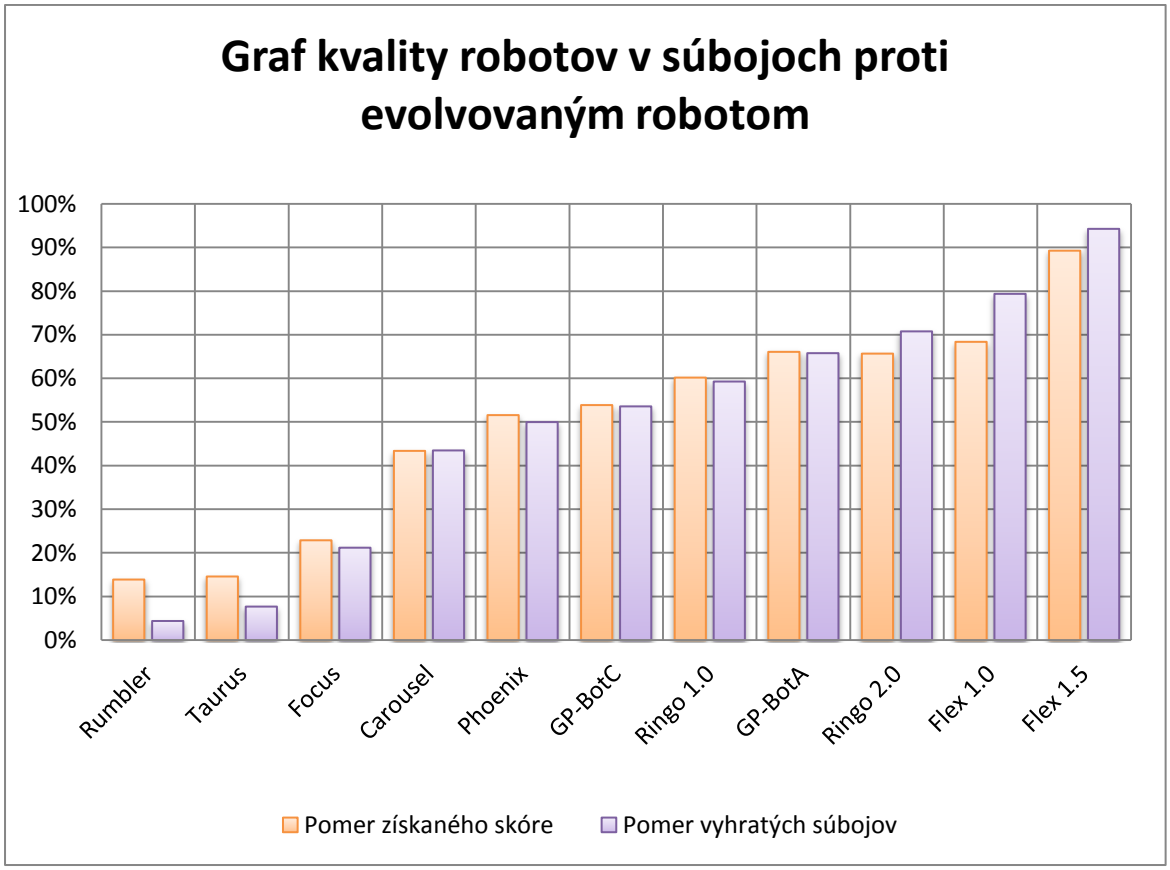
Obr. 19. Graf kvality robotov v súbojoch proti manuálne vytvoreným robotom.

5.4.2 Testovanie proti evolvovaným robotom

Testovacia množina evolvovaných robotov je znázornená v Tab. 16. GP-Bot má priradených dvoch robotov, pričom analyzované riešenie v kap. 2.5 sa týka len robota GPBotC. GPBotA je riešenie, ktoré používa manuálne pripravené funkcie podobne ako v prípade robotov Ringo a Flex. Finálna tabuľka výsledkov sa nachádza v prílohe C.3. Na Obr. 20 je znázornený finálny graf výsledkov pre dva nezávislé testy zamerané na hodnotenie prostredníctvom percentuálneho skóre a počtu víťazstiev.

Tab. 16. Testovacia množina evolvovaných robotov.

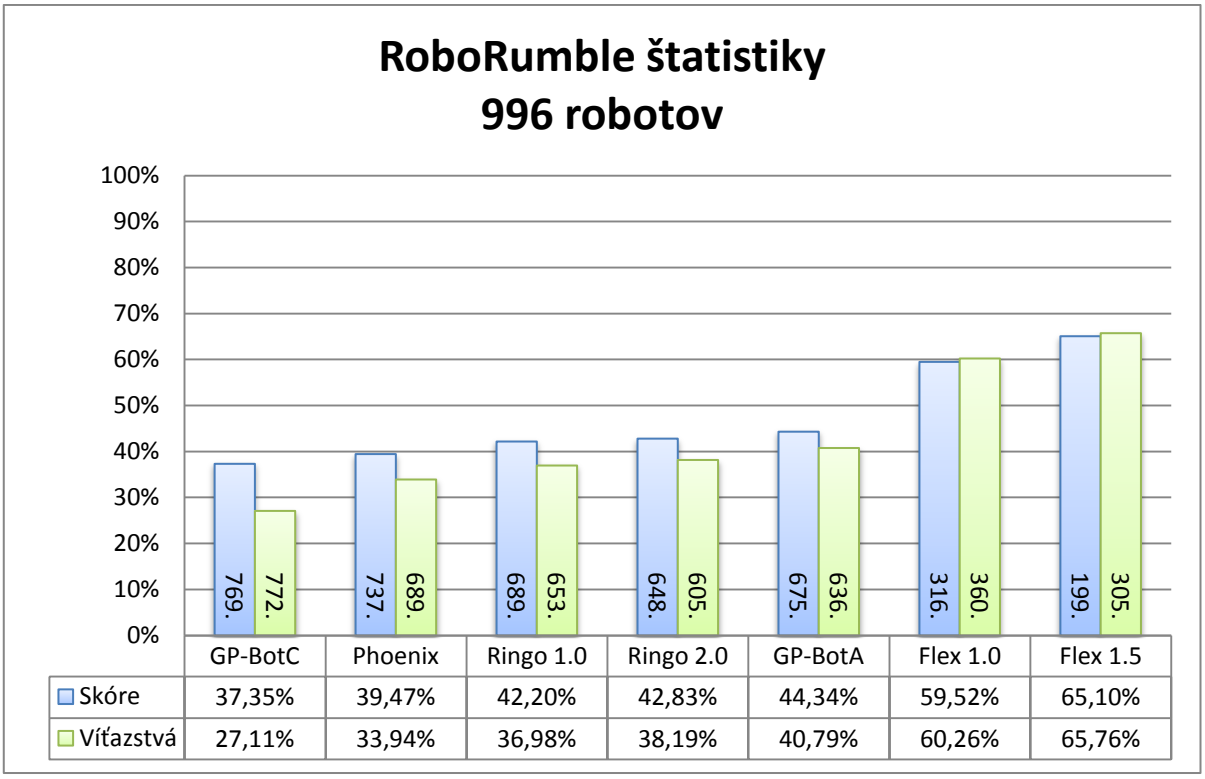
Robot	Poradie v RoboRumble	Váhová kategória
hlavko.nano.Rumbler 1.0	-	NanoBot
hlavko.nano.Taurus 1.0	-	NanoBot
hlavko.nano.Focus 1.0	-	NanoBot
hlavko.nano.Carousel 1.0	-	NanoBot
hlavko.nano.Phoenix 1.0		NanoBot
geep.haiku.GPBotC 1.0	770	MicroBot
hlavko.nano.Ringo 1.0d	689	NanoBot
hlavko.nano.Ringo 2.0	677	NanoBot
geep.haiku.GPBotA 1.0	649	MiniBot
hlavko.micro.Flex 1.0	317	MicroBot
hlavko.micro.Flex 1.5	200	MicroBot



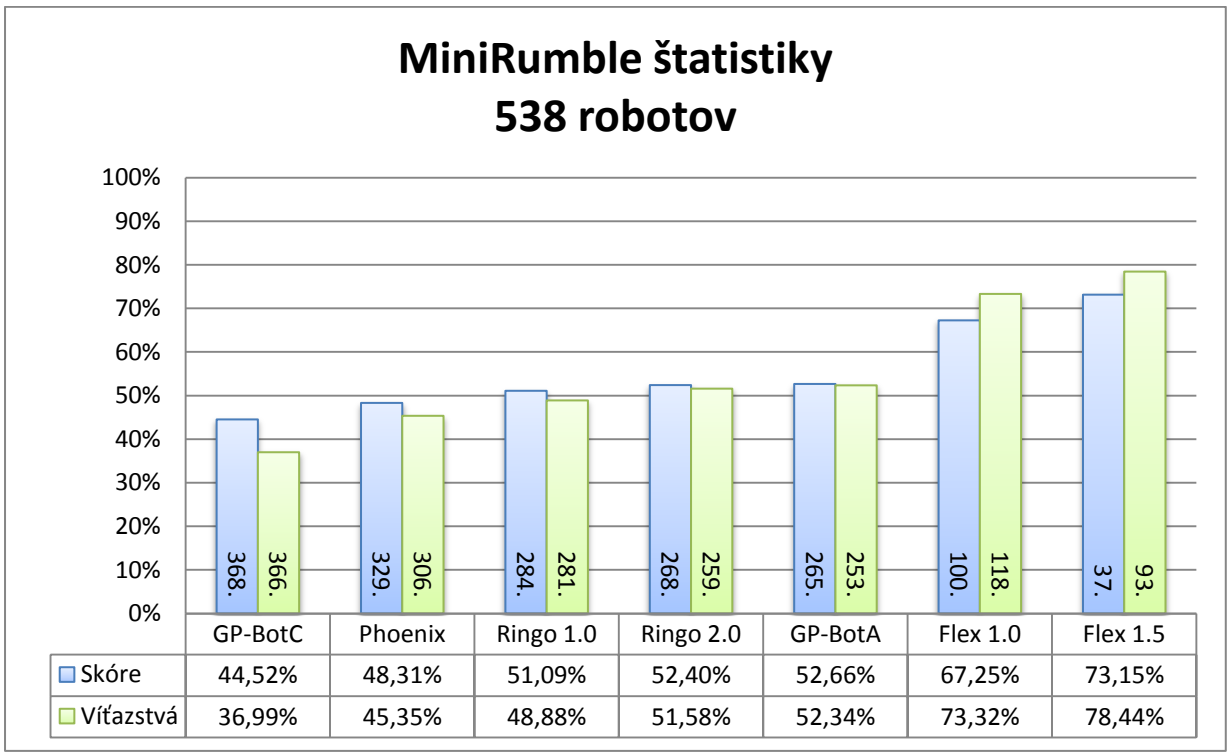
Obr. 20. Graf kvality robotov v súbojoch proti evolvovaným robotom.

5.4.3 Testovanie v súťaži LiteRumble

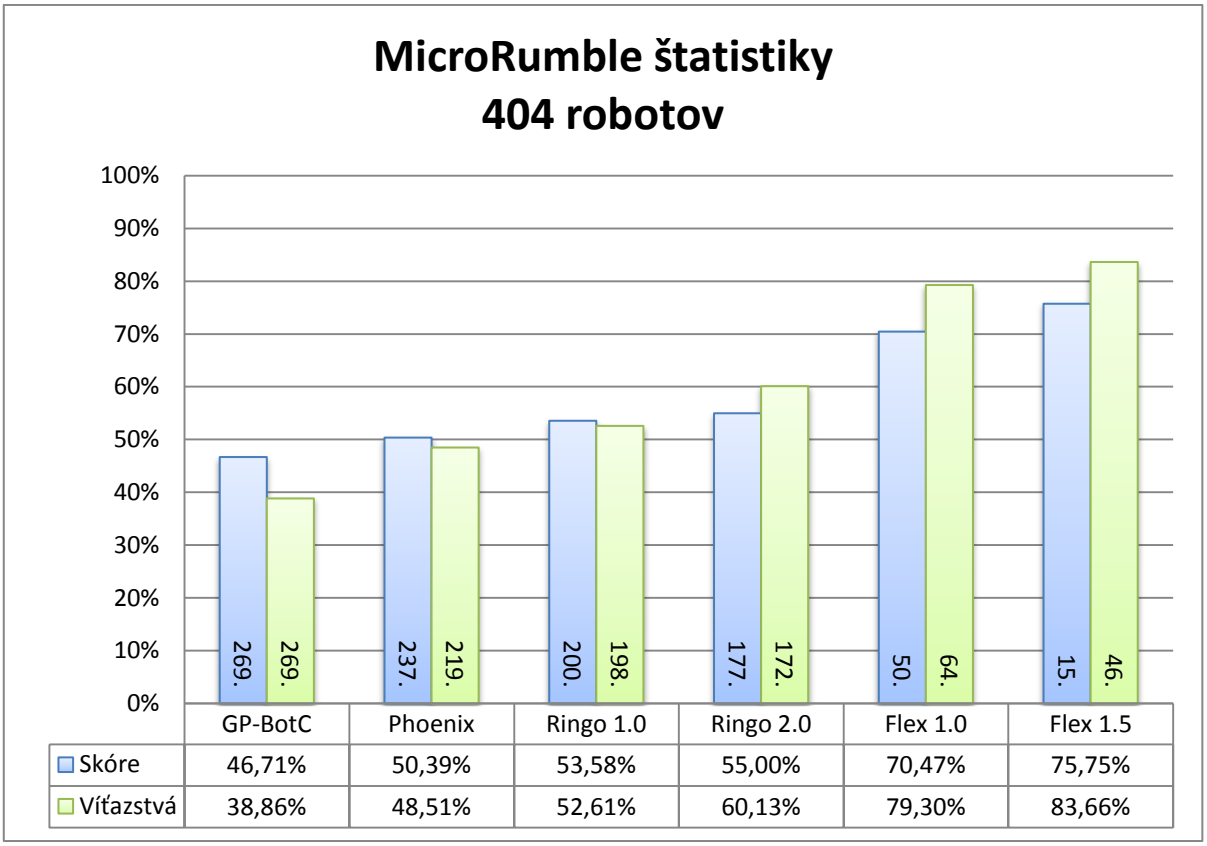
Výsledky publikované v tejto kapitole sú získané z oficiálnej Robocode ligy LiteRumble. Súboje v ligách sú realizované medzi všetkými zapísanými robotmi. Výsledky sú automaticky aktualizované a kategorizované do líg pre jednotlivé váhové kategórie (Obr. 21 – Obr. 24), ako aj podľa ďalších parametrov opisujúcich ich kvalitu.



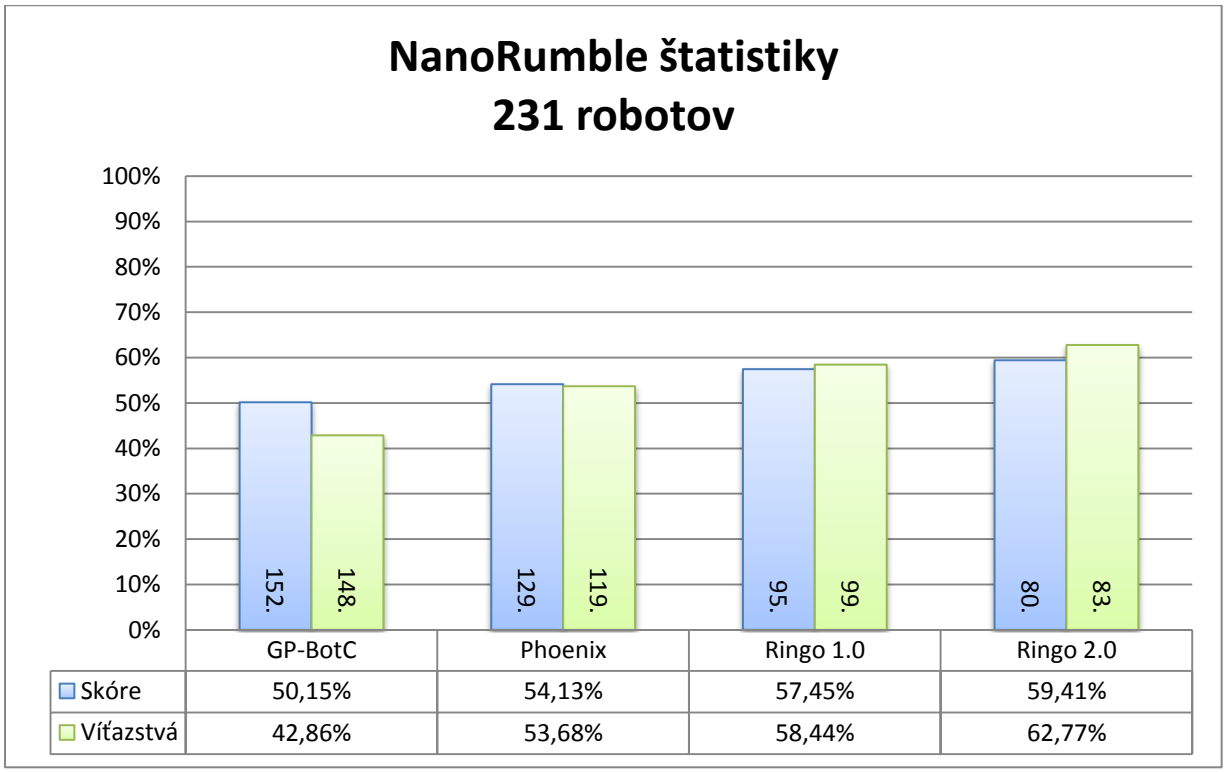
Obr. 21. Štatistika robotov zapísaných v lige RoboRumble
(číslo v stĺpci predstavuje umiestnenie v rebríčku spomedzi 996 zapísaných robotov).



Obr. 22. Štatistika robotov zapísaných v lige MiniRumble
(číslo v stĺpci predstavuje umiestnenie v rebríčku spomedzi 538 zapísaných robotov).



Obr. 23. Štatistika robotov zapísaných v lige MicroRumble
(číslo v stĺpci predstavuje umiestnenie v rebríčku spomedzi 404 zapísaných robotov).



Obr. 24. Štatistika robotov zapísaných v lige NanoRumble.
(číslo v stĺpci predstavuje umiestnenie v rebríčku spomedzi 231 zapísaných robotov).

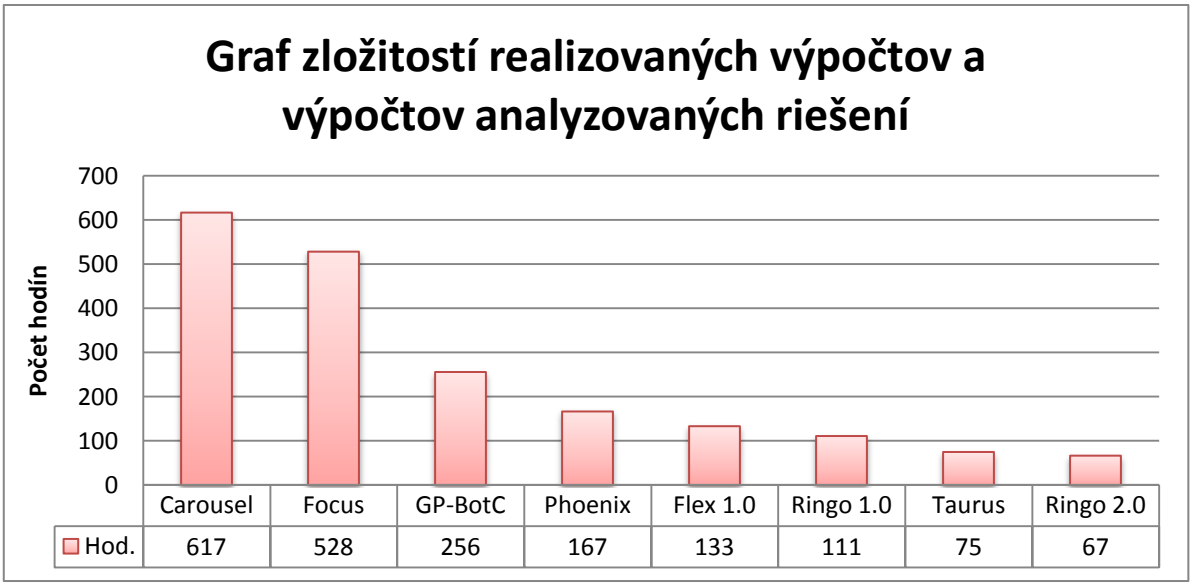
5.4.4 Testovanie výpočtovej zložitosti

Tab. 17 obsahuje zložitosti výpočtov pri ktorých boli vytvorené kompletné roboty alebo niektoré ich moduly. Formula pre výpočet zložitosti je použitá z kap. 4.3, kde pre uľahčenie výpočtu predpokladáme, že čas jedného kola trvá jednu sekundu. Parametrizácia výpočtu pre robota GP-BotC je získaná zo zdroja [12]. V prípade robotov vyvíjaných prostredníctvom progresívnej metódy bola aplikovaný logaritmus, keďže maximálny počet protivníkov nebol takmer nikdy dosiahnutý.

Na Obr. 25 môžeme vidieť, že sa vzhľadom na výpočtovú zložitosť existujúceho robota GP-BotC podarilo vyvinúť riešenia, ktorých výpočtová zložitosť bola nižšia. Ak zoberieme do úvahy výsledky z kap. 5.4.1 – 5.4.3, tak sa pri týchto riešeniach podarilo dosiahnuť lepšie výsledky aj po kvalitatívnej stránke.

Tab. 17. Tabuľka zložitostí výpočtov.

Robot	Počet kôl	Počet protivníkov	Veľkosť roja (populácie)	Počet iterácií (generácií)	Spolu [sekundy]	Spolu [hodiny]
Carousel	20	4	300	100	2220264	617
Focus	20	3	300	100	1901955	528
GP-BotC	3	3	256	400	921600	256
Phoenix	20	3	100	100	600000	167
Flex 1.0	20	4	60	100	480000	133
Ringo 1.0	20	4	50	100	400000	111
Taurus	10	6	30	150	270000	75
Ringo 2.0	20	4	30	100	240000	67



Obr. 25. Graf zložitostí realizovaných výpočtov a výpočtov v analyzovaných riešeniach.

5.5 Výsledky optimalizácie

Použitie optimalizácie gramatickým rojom si môže nájsť uplatnenie najmä pri riešení dynamických problémov, pre ktoré je možné navrhnúť takú gramatiku, ktorej veľkosť, ako aj celkový priestor možných riešení opísaný touto gramatikou nie je príliš rozsiahly, aby bolo možné výpočet realizovať v reálnom čase.

Na základe znalostí získaných prostredníctvom vykonaných testovaní môžeme tvrdiť, že rýchlosť konvergencie algoritmu je relatívne rýchla, čo je zapríčinené najmä optimalizáciou rojom častíc, na ktorej je princíp gramatického roja postavený. Pre jednoduchšie problémy je tak odporúčané opakovať výpočtový proces, pričom pri zložitejších je odporúčané použitie len pre také problémy, ktoré majú rovnomernú distribúciu kvalitných riešení naprieč celým priestorom riešení, aby konvergencia smerovala práve k týmto riešeniam.

Pri použití gramatickej evolúcie v zložitejších problémoch je nutné si uvedomiť, že správna parametrizácia algoritmu, návrh gramatiky, výber fitnes funkcie a ďalších konfigurácii použitých pri výpočte nie je triviálny proces a mnohokrát je nutné postupovať metódou pokus-omyl s následnou analýzou získaných výsledkov a odstránením vzniknutých problémov so zámerom vylepšenia výpočtového procesu.

Vzhľadom na výsledky testovaní je možná bližšia špecifikácia vlastnosti jednotlivých regresívnych metód použitých v realizovaných výpočtoch (Tab. 18).

Tab. 18. Porovnanie regresívnych metód.

Metóda	Výhody	Nevýhody
behaviorálna	komplexné správanie robotov	mŕtve vetvy, veľkosť definovanej gramatiky, relatívne rozsiahle programy, nekonečné vetvenie
parametrická	jednoduché použitie, minimálna veľkosť programov	nutná manuálna príprava kódu
symbolická	tvorba matematických formúl, jednoduché použitie	nie vždy použiteľné, môžu byť rozsiahlejšie programy, nekonečné vetvenie

6 Zhodnotenie

Práca obsahuje podrobnú analýzu zameranú na vysvetlenie evolučných optimalizačných techník používaných pri riešení dynamických problémov. Postupne prechádza cez hlavné teoretické princípy evolučných algoritmov, gramatickej evolúcie, rojovej inteligencie, optimalizácie rojom častíc a gramatického roja, ktoré sú použité pri realizovaní riešenia.

V kapitole návrhu sú detailne opísané možnosti aplikácie a rozšírenia prostredníctvom optimalizácie gramatickým rojom. Podarilo sa charakterizovať regresívne metódy generovania správania programov robotov a porovnať možnosti ich použitia. Taktiež sme identifikovali viaceré možnosti výpočtu fitnes hodnôt pri testovaní kvality nájdených robotov.

Prostredníctvom navrhnutej a implementovanej softvérovej aplikácie s jednoduchou paralelizáciou sa podarilo úspešne vyvinúť rámec pre realizáciu výpočtov postavených na optimalizácii gramatickým rojom, ako aj paralelných gramatických rojov, ktorých princíp je založený na modulárnom generovaní programov robotov. Rámec je možné použiť aj pri iných typoch problémov, avšak je nutné doprogramovanie modulov, ktoré sú špecifické pre vybrané problémy a neboli vytvorené pri realizácii riešenia v tejto práci.

Na základe výsledkov robotov získaných pri lokálnom testovaní a prostredníctvom online líg, kde je zapísaných 996 robotov vytvorených 479 autormi môžeme tvrdiť, že sa nám po kvalitatívnej stránke podarilo úspešne vyvinúť konkurencieschopných robotov. Dôvod je ten, že naši roboti dokázali uspieť proti veľkej množine manuálne naprogramovaných robotov s rôznymi technikami riadenia a taktiež sa im podarilo dosiahnuť lepšie výsledky v porovnaní s robotmi, ktorí sú vyvinutí pomocou evolučných optimalizačných techník.

Robot Phoenix, ktorý je vyvinutý prostredníctvom behaviorálnej regresie dosahuje lepšie výsledky v porovnaní s analyzovaným robotom GP-Bot, ktorý taktiež pracuje na rovnakom princípe. Robot Ringo, ktorý kombinuje výsledky získané pomocou parametrickej a symbolickej regresie sa v lige NanoRumble umiestnil na 80. mieste spomedzi 231 robotov. Robota Flex, ktorý je čiastočne postavený na existujúcom riešení sa podarilo vylepšiť rovnakou metódou a s využitím dodatočných techník riadenia. V lige MicroRumble sa tak umiestnil na 15. mieste, čím predstihol rozširujúceho robota sheldor.jk.Yatagan 1.0.5 o desať miest a v rebríčku všetkých robotov RoboRumble je na 199. mieste.

Ďalším pozitívom realizovaných výpočtov je, že výpočtová zložitosť optimalizácie gramatickým rojom je v porovnaní so zložitosťou evolučnej optimalizačnej techniky použitej pri vývoji GP-Bota niekoľkonásobne efektívnejšia.

6.1 Možné vylepšenia

Vylepšenia vytvorenej softvérovej aplikácie a použitých evolučných optimalizačných techník by mohol spočívať v nasledujúcich možnostiach:

- pridanie optimalizačných techník pre behaviorálnu regresiu so zámerom odstránenia mŕtvych vetiev programov a zníženia veľkosti programov,
- kombinovanie použitých optimalizačných techník s existujúcimi overenými technikami, ako napríklad horolezecký algoritmus a iné,
- optimalizácia výpočtovej zložitosti prostredníctvom distribuovaného výpočtového systému,
- vyhľadanie a definovanie novej regresívnej metódy (napr. prostredníctvom multiplexerov pre generovanie siete správania podobne ako v neurónových sieťach),
- vylepšenie existujúcich open-source riešení prostredníctvom parametrickej regresie alebo pridaním, či výmenou modulov, ktorých správanie je vygenerované na základe behaviorálnej alebo symbolickej regresie.

7 Použitá literatúra

[1] Eiben, A.E., Smith, J.E.: *Introduction to Evolutionary Computation*. Springer, Natural Computing Series, (2008), s. 15–24.

[2] Dempsey, I., O`Neill, M., Brabazon, A.: *Foundations in Grammatical Evolution for Dynamic Environments*. Springer, Studies in Computational Intelligence Series, (2009), s. 2–20.

[3] Hugosson, J. et al.: Genotype Representations in Grammatical Evolution. *Applied Soft Computing Journal,* (2010), Zv. 10, s. 36–43.

[4] O`Neill, M. et al.: Crossover in Grammatical Evolution. *Journal of Genetic Programming and Evolvable Machines,* (2003), Zv. 4, s. 67–84.

[5] Keijzer, M.: Ripple Crossover in Genetic Programming. *Proceedings of 4th European Conference on Genetic Programming (EUROGP2001),* Como, Italy, (2001), Zv. 2038, s. 74–86.

[6] Liu, Y., Passino, K.M.: *Swarm Intelligence: Literature Overview*. Technical report, Ohio State University, USA, (2000), Zv. 2015, s. 1–8.

[7] Engelbrecht, A.P. *Fundamentals of Computational Swarm Intelligence.* Wiley, (2005), s. 2–129.

[8] Millonas, M.M.: Swarms, Phase Transitions, and Collective Intelligence. *Proceedings of Artifical Life III.* Santa Fe Institute: Addison-Wesley, USA, (1992), s. 417–445.

[9] O`Neill, M., Brabazon, A.: Grammatical Swarm: The generation of programs. *Natural Computing: An International Journal,* (2006), Zv. 5, s. 443–462.

[10] Nelson, M.: *Robocode*. [Online] 11. 08. 2009. [Dátum: 07. 05. 2013.] http://robowiki.net/wiki/Robocode.

[11] Eisenstein, J.: *Evolving Robocode Tank Fighters*. Technical report AIM-2003-023 AI Lab, Massachusetts Institute of Technology, USA, (2003).

[12] Shichel, Y., Ziserman, E., Sipper, M.: GP-Robocode: Using Genetic Programming to Evolve Robocode Players. *Proceedings of 8th European Conference on Genetic Programming (EUROGP2005),* Lausanne, Switzerland, (2005), s.143–154.

[13] Harper, R.: Co-evolving robocode tanks. *GECCO '11 Proceedings of the 13th annual conference on Genetic and evolutionary computation*, ACM New York, USA, (2011), s. 1443–1450.

[14] Hlaváč, M.: *RoboWiki: Grammatical Swarm Evolution*. [Online] 05. 05. 2013. [Dátum: 07. 05. 2013] http://robowiki.net/wiki/Grammatical_Swarm_Evolution

Príloha A: Inštalačná príručka

Pre korektné spustenie aplikácie je nutné vykonávať inštaláciu na operačnom systéme Windows a mať nainštalované nasledujúce softvérové prostredia:

- Java 1.6.0 a viac
- Python 3.0 a viac
- Robocode 1.8.1.0 a viac (odporúčané v priečinku *C:\Robocode*)

Inštaláciu potrebných prostredí je možné vykonať pomocou otvorenia priečinku *install* na priloženom elektronickom médiu a spustením nasledujúcich inštalačných súborov:

1. Java (32-bit) *jre-7u21-windows-i586.exe,* (64-bit) *jre-7u9-windows-x64.exe*
2. Python (32-bit) *python-3.0.msi*, (64-bit) *python-3.0.amd64.msi*
3. Robocode (32/64-bit) *robocode-1.8.1.0-setup.jar*

Aplikácia s konfiguračnými súbormi sa nachádza v priečinku *app*, ktorý skopírujeme z elektronického média. Pre testovacie spustenie aplikáciu spustíme súbor *g-swarm-demo.jar*, ktorý sa nachádza v priečinku *app*. Pre spustenie hlavnej aplikácie *g-swarm.jar* s možnosťou vykonávania výpočtov prostredníctvom hry Robocode je nutné nakonfigurovať cesty v konfiguračnom súbore *app/conf/path.conf*, ktorý môže mať nasledovnú formu:

```
TEST_SCRIPT="C:/g-swarm/app/script/robot_tester.py"
JAVAC="C:/Program Files/Java/jdk1.7.0_03/bin/javac"
ENEMY_LIST="C:/g-swarm/app/conf/robots.lst"
BATTLE_TEMPLATE="C:/g-swarm/app/conf/battle/gswarm.battle"
ROBOCODE="C:/Robocode"
```

1. *TEST_SCRIPT* – cesta k testovaciemu skriptu, ktorý vykonáva testy kvality robotov vygenerovaných počas výpočtu,
2. *JAVAC* – cesta k Java kompilátoru,
3. *ENEMY_LIST* – cesta k zoznamu testovacích robotov,
4. *BATTLE_TEMPLATE* – cesta k šablóne súbojov,
5. *ROBOCODE* – cesta k hre Robocode.

Posledným krokom je rozbalenie množiny existujúcich robotov v *robots/robots.zip* do *ROBOCODE/robots*. Po úspešnom spustení aplikácie je možné prehliadať existujúce výpočty s možnosťou tvorby nových výpočtov, ktoré sú detailne opísané v prílohe B.

Príloha B: Používateľská príručka

Na priloženom CD sa nachádzajú tri spustiteľne Java aplikácie (JAR súbory) s názvami:

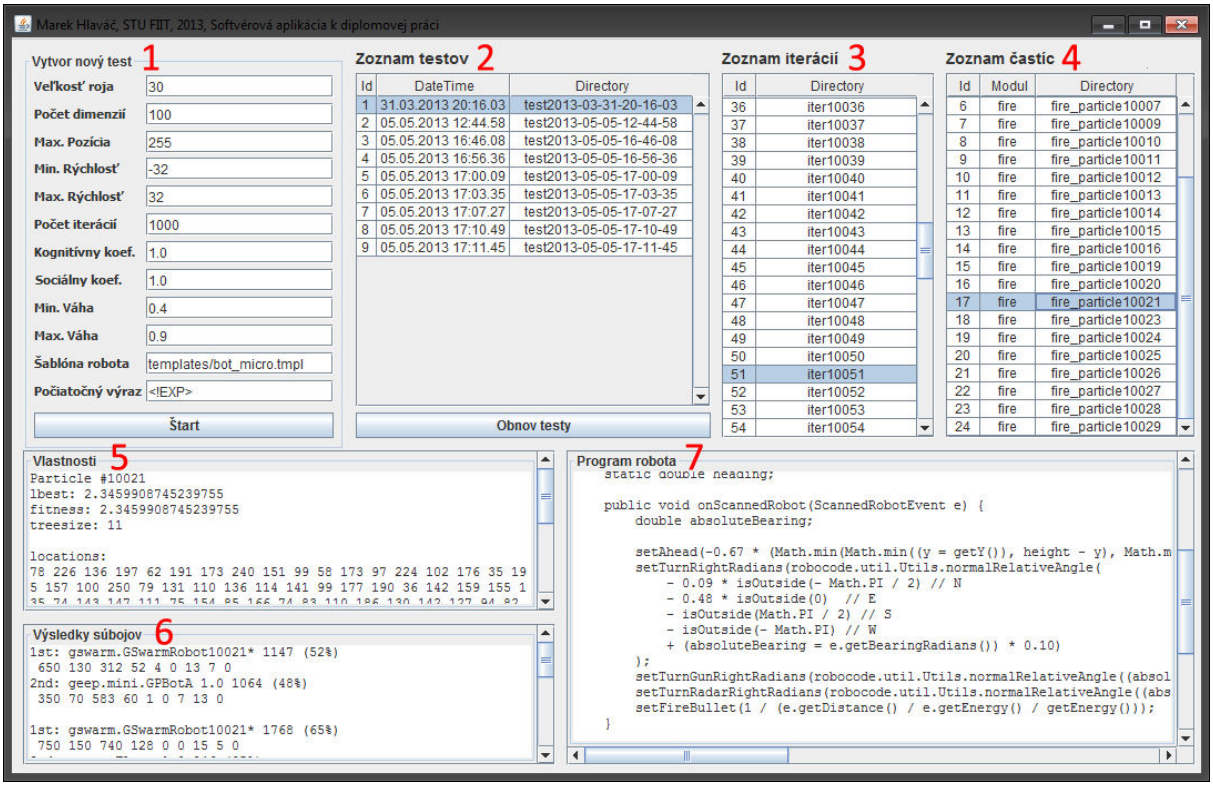
- *g-swarm-demo.jar* – aplikácia vytvorená na testovacie účely obsahujúca len funkčnosť symbolickej regresie pre fixne zadaný príklad,
- *g-swarm.jar* – hlavná aplikácia podporujúca testovanie robotov pomocou optimalizácie gramatického roja,
- *g-swarm-parallel.jar* – vedľajšia aplikácia podporujúca testovanie robotov prostredníctvom algoritmu paralelných gramatických rojov.

Aplikácie umožňujú prezeranie prebehnutých testov, ako aj vytvorenie a spustenie nového testu. Pre spustenie je odporúčané použiť príkazový riadok kvôli konzolovým výstupom nasledujúcim príkazom:

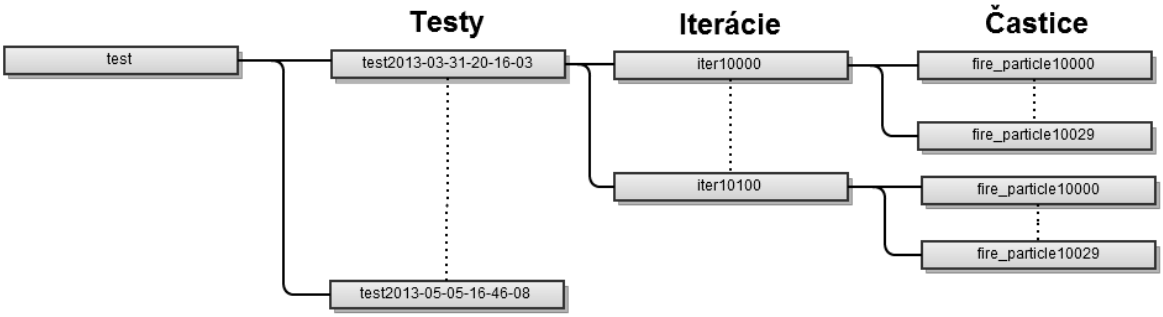
java -jar g-swarm.jar

Po spustení sa zobrazí hlavné okno aplikácie, ktoré je rozdelené do siedmych častí vyznačených na Obr. 26:

1. formulár pre vytvorenie testu s požadovanými parametrami a s možnosťou jeho spustenia,
2. zoznam existujúcich testov nachádzajúcich sa v priečinku *test* (štruktúra priečinka je zobrazená na Obr. 27.) s možnosťou obnovenia stavu testov pomocou tlačidla *Obnov testy*,
3. zoznam iterácií zvoleného testu,
4. zoznam častíc zvolenej iterácie s označením vyvíjaného modulu (názov modulu je relevantný len pre paralelné gramatické roje),
5. môže obsahovať 2 výstupy:
    a. v prípade označenej iterácie obsahuje vlastnosti označenej iterácie,
    b. v prípade označenej častice obsahuje vlastnosti označenej častice,
6. zobrazenie výsledkov súbojov robota reprezentovaného časticou v prípade zvolenej častice,
7. výsledný program robota, ktorý vznikol na základe definovanej šablóny a aplikáciou gramatického roja použitý pri realizovaní súbojov.



Obr. 26. Ukážka hlavného okna aplikácie.



Obr. 27. Hierarchia testov v súborovom systéme.

V prípade, že bola inštalácia úspešná a aplikáciu je možné spustiť, tak môžeme vytvoriť nový výpočet, na ktorého realizáciu potrebujeme vykonať kroky, ktoré sú opísané v nasledujúcich odsekoch:

1. vytvorenie šablóny robota,
2. vytvorenie príslušných súborov s gramatikou použitou v šablóne,
3. vytvorenie zoznamu testovacích robotov,
4. vytvorenie šablóny súbojov,
5. spustenie skriptu, ktorý pripraví zápasy v hre Robocode,
6. parametrizácia testu cez používateľské rozhranie a jeho spustenie.

1. **Príprava šablóny robota**

Šablóna robota predstavuje program robota, ktorý budeme dopĺňať vygenerovaným programom prostredníctvom gramatického roja. Šablóna môže vyzerať nasledovne:

```
package gswarm;
import robocode.AdvancedRobot;
import robocode.ScannedRobotEvent;
public class GSwarmRobot#ROBOTCLASS# extends AdvancedRobot {
  public void run() {
        while (true) {
              turnRadarLeft(360);
        }
  }
  public void onScannedRobot(ScannedRobotEvent e) {
        ahead(#gswarm-simple.grm);
  }
}
```

Označenie *#ROBOTCLASS#* predstavuje miesto pre identifikáciu robota pomocou jeho poradia v iterácií. Identifikátor musí byť prítomný v každom teste, inak výpočet zlyhá. Označenie *#gswarm-simple.grm* predstavuje použitie gramatického roja s gramatikou v súbore s názvom, ktorý sa nachádza za symbolom #. Šablóna použitá pri výpočte sa definuje prostredníctvom GUI v aplikácii.

2. **Príprava gramatiky**

Gramatika je definovaná v špecifickom súbore, ktorý musí byť umiestnený v priečinku *conf*. Gramatika je použitá len vtedy, ak použitá šablóna robota obsahuje referenciu na túto gramatiku. Gramatika je definovaná prostredníctvom pravidiel v BNF forme, ktorá môže mať nasledujúcu formu:

```
<!EXP> = 1 / (<!VAL>)
<!VAL> = <!VAL> <!OPP> <!VAL> | <!VAR>
<!OPP> = + | - | * | /
<!VAR> = e.getEnergy() | getEnergy() | e.getDistance()
```

Gramatika obsahuje terminálne a neterminálne symboly, pričom terminálne symboly predstavujú finálnu podobu určitej časti programového výrazu a neterminálne sú označené pomocou konštrukcie *<!XXX>*, kde *XXX* by malo byť dlhé práve 3 znaky. Počiatočný stav gramatiky sa definuje prostredníctvom GUI v aplikácii.

3. Príprava zoznamu testovacích robotov

Zoznam testovacích robotov pre určenie fitnes hodnoty robotov sa definuje v separátnom súbore, ktorý sa následne použije pri testovaní. Cesta k súboru sa definuje po inštalácii v konfiguračnom súbore, ktorý je opísaný v prílohe A. Formát súboru môže mať nasledovnú štruktúru:

```
1 jk.micro.Toorkild-0.4.5 0
2 simonton.micro.WeeklongObsession-3.4.1 0
3 kc.micro.WaveShark-0.31 0
```

Prvý stĺpec reprezentuje identifikátor robota. Druhý stĺpec obsahuje úplný názov robota, tak aby ho bolo prostredie Robocode schopné rozpoznať, pričom robot musí byť v hre prítomný, inak testovanie zlyhá. Názov robota môže obsahovať medzery, ktoré sú v našom prípade nahradené pomlčkou a počas realizácie výpočtu nahradená naspäť na medzeru. Tretí stĺpec predstavuje kritérium používané pri progresívnej metóde testovania na určenie, či robot pokračuje v testovaní s ďalším robotom alebo nie. Pre účely jednoduchej alebo komplexnej metódy testovania nastavujeme na nulu.

4. Príprava šablóny súbojov

Šablóna súbojov sa nastavuje v špecifickom súbore, ktorá obsahuje konfiguračné dáta pre vykonanie súboju v hre Robocode. Cesta k súboru sa definuje po inštalácii v konfiguračnom súbore, ktorý je opísaný v prílohe A. Formát súboru môže mať nasledovnú štruktúru:

```
#Battle Properties
#Thu Oct 25 17:48:57 CEST 2012
robocode.battle.numRounds=20
robocode.battle.gunCoolingRate=0.1
robocode.battleField.width=800
robocode.battle.selectedRobots=#ENEMY#,#ROBOT#*
robocode.battle.rules.inactivityTime=300
robocode.battle.hideEnemyNames=true
robocode.battleField.height=600
```

Dôležitými parametrami sú *numRounds* (počet súbojov realizovaných medzi dvoma robotmi), *width* (šírka bojiska), *height* (výška bojiska) a *selectedRobots*, ktorý obsahuje označenie robotov, medzi ktorými bude súboj vykonávaný. Časti *#ENEMY#* a *#ROBOT#* sú nahradené pri generovaní súbojov v ďalšom kroku prípravu výpočtového prostredia.

5. Príprava súbojov v hre Robocode

Príprava súbojov predstavuje vygenerovanie konfiguračných súborov súbojov, ktoré sú vytvorené v prostredí Robocode. Generovanie je zabezpečené python skriptom

nachádzajúcim sa na ceste *script/create_battles.py*. Skript nájde definovanú šablónu súbojov pomocou konfiguračného súboru obsahujúcu definované cesty a nakopíruje súboje do hry Robocode. Dôsledkom je, že pri spustení hry Robocode sa môžeme priamo odkazovať na tieto konfigurácie súbojov v súboroch a meniť testovanie dynamicky. Spustenie skriptu vyzerá nasledovne:

```
python script/create_battles.py
```

6. Parametrizácia testu

Parametrizácia gramatického roja a voľba šablóny robota sa realizuje prostredníctvom formuláru v grafickom používateľskom rozhraní v spustenej aplikácii (Obr. 28). Formulár obsahuje nasledovné položky:

1. počet častíc v gramatickom roji,
2. veľkosť polohového vektoru a vektoru rýchlosti častice,
3. maximálnu pozíciu častice v intervale < 0, N >,
4. minimálnu rýchlosť častice v priestore hľadania,
5. maximálnu rýchlosť častice v priestore hľadania,
6. limit počtu iterácií výpočtu,
7. koeficient pre kognitívny komponent v PSO,
8. koeficient pre sociálny komponent v PSO,
9. minimálnu váhu zotrvačnosti (nadobudnutá v poslednej iterácii výpočtu),
10. počiatočnú váhu zotrvačnosti,
11. relatívnu cestu k šablóne programu robota,
12. počiatočný výraz aplikovaný v použitých gramatikách.

Obr. 28. Ukážka formuláru pre vytvorenie nového testu.

Analýza výsledkov výpočtu

Progres výpočtu je možné sledovať v otvorenej aplikácii pomocou tlačidla *Obnov Testy* a časti *Vlastnosti*, kde sa zobrazuje výpis vlastností zvolenej iterácie, ktorý môže mať formu:

```
Iteration #14
gbest: particle10019 (from iteration 7) with fitness 1.9267
actual max: 1.8477 (particle10014)
lbest mean: 1.7399
actual mean: 1.5851 (only valid particles are counted)
valid particles: 25
invalid particles: 5
```

1. *Iteration* predstavuje číslo zvolenej iterácie.
2. *gbest* zobrazuje informácie ohľadom najlepšieho globálneho riešenia *gbest* reprezentovaného identifikátor častice *particle10019* v rámci konkrétnej iterácie *iteration 7* a s príslušnou fitnes častice *1.9267*.
3. *actual max* je maximálna hodnota fitnes *1.8477* nadobudnutá vo zvolenej iterácii s informáciou ohľadom častice, ktorá túto hodnotu dosiahla *particle10014*.
4. *lbest mean* je hodnota priemernej lokálnej fitnes získaná spriemerovaním súčtu najlepších fitnes, ktoré jednotlivé časti v doterajšom priebehu výpočtu nadobudli.
5. *actual mean* je priemerná fitnes nadobudnutá vo zvolenej iterácii, pričom do výpočtu sú zahrnuté len platné častice (tie ktoré majú akceptovaný program robota).
6. *valid/invalid particles* zobrazuje počet platných, resp. neplatných častíc.

Ďalšie možnosti analýzy sa týkajú informácií ohľadom častíc, ktoré je možné vidieť v časti *Vlastnosti* alebo *Výsledky súbojov* po zvolení konkrétnej častice. Časť *Vlastnosti* môže mať nasledovnú formu:

```
Particle #10004
lbest: 1.6707780650816624
fitness: 1.6707780650816624
treesize: 3
locations: 69 171 184 161 139 ...
velocities: -3 13 -5 18 -8 ...
```

1. *Particle* obsahuje identifikačné číslo častice.
2. *lbest* obsahuje najlepšiu fitnes, ktorú častica získala v doterajšom priebehu výpočtu.
3. *fitness* predstavuje hodnotu udelenú v aktuálne zvolenej iterácii.
4. *treesize* predstavuje hĺbku vygenerovaného stromu (relevantné len pre behaviorálnu regresiu)
5. *locations* obsahuje polohový vektor častice.
6. *velocities* obsahuje vektor rýchlostí častice.

Časť *Výsledky súbojov* obsahuje zoznam výsledkov vygenerovaných hrou Robocode, ktoré boli realizované pri testovaní kvality častice. Výsledok zo zoznamu môže mať nasledovnú formu:

```
1st: geep.mini.GPBotC 1.0 1093 (52%) 200 40 718 48 64 24 4 16 0
2nd: gswarm.GSwarmRobot10004* 1028 (48%) 800 160 21 3 43 0 16 4 0
```

Každá hodnota v riadku definuje určitú hodnotu získanú v súboji. Jednotlivé stĺpce reprezentujú nasledovné charakteristiku:

1. *Rank* – poradie v realizovaných súbojov na základe získaného skóre,
2. *Robot Name* – názov robota v realizovanom súboji,
3. *Total Score* – skóre,
4. *Percentual Score* – pomerové skóre,
5. *Survival* – skóre prežitia (ak je niektorý robot zničený, tak ostatní získavajú body),
6. *Surv Bonus* – bonus za prežitie (získaný, ak robot zostane ako jediný v aréne)
7. *Bullet Dmg* – skóre za spôsobené zničenie na protivníkovi,
8. *Bullet Bonus* – bonus získaný pri zničený nepriateľa strelou,
9. *Ram Dmg * 2* – skóre za spôsobené nárazové poškodenie,
10. *Ram Bonus* – bonus získaný pri zničený nepriateľa nárazom,
11. *1sts* – počet prvých miest v súbojoch,
12. *2nds* – počet druhých miest v súbojoch,
13. *3rds* – počet tretích miest v súbojoch (irelevantné pre naše účely).

Import priložených výpočtov

1. na elektrickom médiu otvoríme priečinok *app/backup*,
2. v otvorenom priečinku si nájdeme výpočet týkajúci sa požadovaného robota,
3. príslušný *zip* súbor výpočtu rozbalíme do priečinku *test*, kde sa nachádza aplikácia,
4. otvoríme aplikáciu alebo obnovíme stav testov pomocou tlačidla *Obnov testy* a v zozname testov pribudne nová položka pre pridaný výpočet, ktorú môžeme následne prehľadávať.

Príloha C: Technická dokumentácia

C.1 Existujúce techniky používané v hre Robocode

C.1.1 Metódy pohybu

Tab. 19. Metódy pohybu.

Názov	Opis
Anti-Gravity	vytvorenie antigravitačných miest na základe pozícií objektov a udalostí, ktoré následne pôsobia silou na tank
Bullet Shadow	vykonávanie pohybu na základe vlastnej streľby, ktorá definuje bezpečné miesta pri vlnách nepriateľskej streľby
Circle	pohyb v kruhoch
Flattener	snaha o vyhýbanie sa navštívených miest na základe pamätanie si histórie vlastného pohybu
GoTo	priamy pohyb k zvolenej pozícii
Linear	pohyb pozdĺž priamky
Mirror	kopírovanie pohybu protivníka
Multi-Mode	prepínanie medzi viacerými typmi pohybu
Orbit	lineárny pohyb s udržiavaním si konštantnej vzdialenosti od protivníka
Oscillation	opakovaný pohyb podľa špecifikovaného vzoru
Random	forma pohybu, kde je smer pohybu vygenerovaný náhodným generátorom čísel
Ramming	narážanie do protivníka
Stop And Go	krátky posun v prípade zaznamenania streľby protivníkom
Wall	pohyb pozdĺž stien
Wall Smoothing	vyhýbanie sa stenám bez nutnosti použitia reverzného pohybu
Wave Surfing	vyhýbanie sa zaznamenaným vlnám nepriateľských striel, kde vlna predstavuje všetky možné polohy striel

Tab. 20. Porovnanie metód pohybu.

Názov	Výhody	Nevýhody
Anti-Gravity	malá veľkosť implementácie, vhodné pre Melee robotov	nastavenie váh síl pre veľký počet miest
Bullet Shadow	používané najlepšími robotmi	rozsiahla veľkosť implementácie
Circle	malá veľkosť implementácie	predvídateľnosť
Flattener	zmätenie protivníkov, ktorí vykonávajú streľbu na základe histórie pohybu	možnosť nahradenia náhodným pohybom
GoTo	malá veľkosť implementácie	predvídateľnosť
Linear	malá veľkosť implementácie	predvídateľnosť
Mirror	malá veľkosť implementácie, efektívne proti protivníkom so zložitejším pohybom	nepoužiteľné proti jednoduchým pohybom protivníka
Multi-Mode	vyššia pravdepodobnosť použitia vhodnejšej metódy pohybu	relatívne väčšia veľkosť implementácie
Orbit	malá veľkosť implementácie, vhodne zvolená vzdialenosť zlepšuje výsledky	voľba vzdialenosti
Oscillation	malá veľkosť implementácie	malá kontrola nad absolútnou pozíciou v aréne
Random	ťažké predvídanie pohybu protivníkom, malá veľkosť implementácie	nastavenie váh a spúšťačov pre zmenu
Ramming	dodatočný bonus za náraz	jednoduché zasiahnutie nepriateľskou streľbou
Stop And Go	malá veľkosť implementácie, účinné proti jednoduchším metódam streľby	neúčinné proti pokročilejším metódam streľby
Wall	problematické pre určité druhy streľby	neúčinné proti väčšine metód streľby
Wall Smoothing	vyhýbanie sa pozíciám, ktoré sú náchylné na zmenšenie priestoru pohybu	čiastočné zmenšenie priestoru a možností pohybu
Wave Surfing	používané najlepšími robotmi	rozsiahla veľkosť implementácie

C.1.2 Metódy zameriavania

Tab. 21. Metódy zameriavania protivníka.

Názov	Opis
Angular	štatistické zameriavanie, ktoré spočíva v hľadaní faktora korelácie medzi uhlovou rýchlosťou protivníka a uhlom streľby
Area	zameriavanie na určitú oblasť arény namiesto robota
Circular	metóda predpokladajúca, že robot bude pokračovať v pohybe rovnakým zatočením a rovnakou rýchlosťou
Dynamic Clustering	vyhľadávanie vhodných výsledkov v záznamoch, ktoré sú podobné aktuálnej situácii pomocou klastrovania
Guess Factor	štatistické zameriavanie, ktoré berie do úvahy smer pohybu protivníka a možnosti úteku v momente vykonania streľby
Head-Fake	zameriavanie s predpokladom, že protivník náhle zmení smer pohybu
Head-On	jednoduchá metóda, ktorá zameriava na miesto, kde bol protivník posledný krát zaznamenaný
Linear	metóda predpokladajúca, že robot bude pokračovať v pohybe rovnakým smerom a rovnakou rýchlosťou
Mean	metóda používajúca priemerovanie zaznamenaných dát za účel vykonania predikcie pohybu
Multiple Choice	štatistický výber z viacerých možností metód zameriavania
Neural	použitie neurónovej siete za účelom predpovedať pohyb protivníka
Pattern Matching	logovacia metóda, ktorá sa snaží vykonať predikciu na základe vyhľadania vhodnej vzorky z poslednej pohybov protivníka
Play It Forward	logovacia metóda, ktorá vygeneruje uhol streľby na základe histórie pohybu protivníka simuláciou jeho budúceho pohybu
Random	jednoduchá metóda, ktorá vyberie náhodný smer streľby z určitého intervalu
Virtual Guns	výber streľby zo sady virtuálnych zbraní pomocou váh, ktoré sú dynamicky prepočítavané na základe možných zásahov
Wiki	hybridná metóda, ktorá zbiera a spracováva dáta nepriateľských vĺn a následne ich používa ako tabuľku pre vykonávanie streľby

Tab. 22. Porovnanie metód zameriavania protivníka.

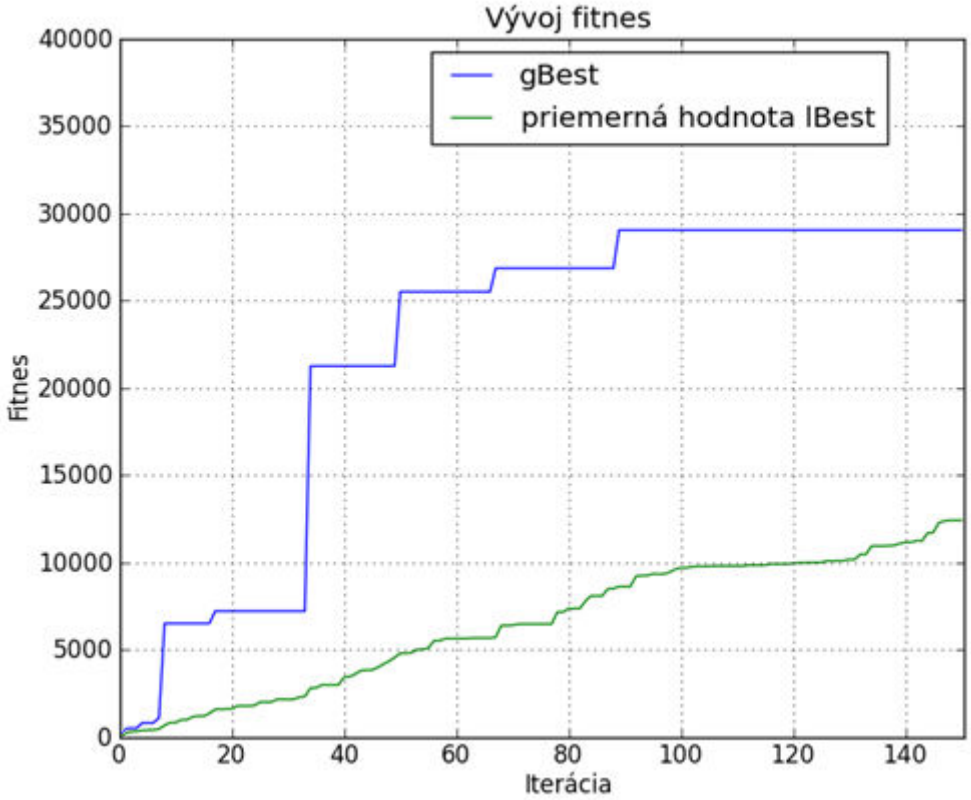
Názov	Výhody	Nevýhody
Angular	relatívne efektívne proti väčšine metód pohybu	nevhodné proti náhodným pohybom, potreba štatistickej vzorky
Area	vhodné v súbojoch proti viacerým robotom	v súboji 1 proti 1
Circular	vhodné proti krúživému pohybu	nevhodné proti zložitejším metódam pohybu
Dynamic Clustering	vysoká úspešnosť proti väčšine metód pohybu	rozsiahla veľkosť implementácie
Guess Factor	vysoká úspešnosť proti väčšine metód pohybu	rozsiahla veľkosť implementácie
Head-Fake	malá veľkosť implementácie	nevhodné proti zložitejším metódam pohybu
Head-On	malá veľkosť implementácie	nevhodné proti zložitejším metódam pohybu
Linear	malá veľkosť implementácie	nevhodné proti zložitejším metódam pohybu
Mean	rozširuje konkrétne metódy zameriavania	logovanie je možné nakaziť zlými dátami, tvorba váh
Multiple Choice	dosiahnutie vysokej úspešnosti proti konkrétnym typom pohybu	tvorba váh, relatívne rozsiahla veľkosť implementácie
Neural	prispôsobovanie streľby na základe rôznych indikátorov	rozsiahla veľkosť implementácie
Pattern Matching	malá veľkosť implementácie	logovanie je možné nakaziť zlými dátami
Play It Forward	proti jednoduchším typom pohybu sa dosahuje vysoká úspešnosť	logovanie je možné nakaziť zlými dátami
Random	malá veľkosť implementácie	neúčinné proti zložitejším metódam pohybu
Virtual Guns	možnosť okamžitého prepnutia na účinnejšiu metódu streľby	relatívne rozsiahla veľkosť implementácie
Wiki	vysoká úspešnosť proti väčšine metód pohybu	rozsiahla veľkosť implementácie

C.2 Technické aspekty vývoja robotov

C.2.1 Taurus

Tab. 23. Opis parametrizácie a metód použitých vo výpočte.

Názov	Opis
algoritmus hľadania	gramatický roj
regresia	behaviorálna
veľkosť roja	30 častíc
veľkosť vektora častice	100 dimenzií
rozmer stavového priestoru	255
maximálna rýchlosť častice	32
sociálny a kognitívny koeficient	1.0
váha zotrvačnosti	od 0.9 do 0.4
metóda výpočtu fitnes	progresívna na základe totálneho skóre z 10 kôl (kap. 3.2.4)
trénovacia množina	Target, MyFirstRobot, Rumbler, Crazy, Tracker, Walls
iterácia nálezu	88
počet iterácií výpočtu	150
veľkosť programu (stromu)	59
princíp činnosti	v prípade zaznamenania nepriateľa radarom sa natočí k nemu, následne strieľa a posúva sa až k stene
hlavný nedostatok vývoja	nerovnomerné generovanie stromu správania

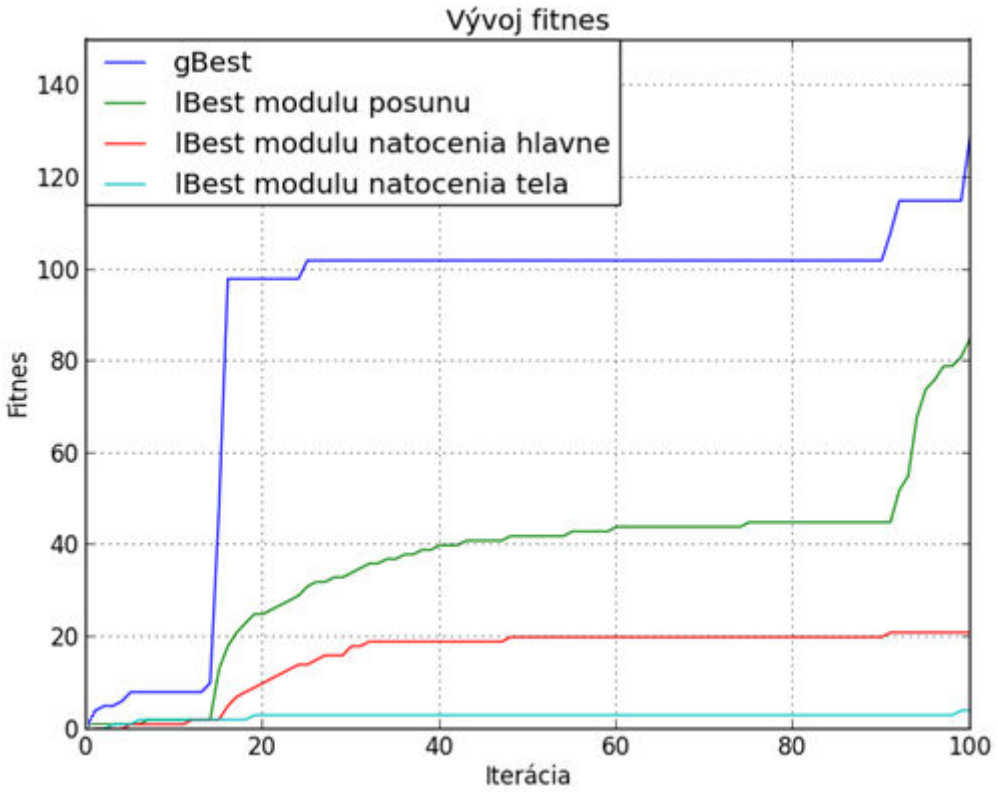


Obr. 27. Vývoj globálnej (gBest) a priemernej lokálnej (lBest) fitnes počas výpočtu.

C.2.2 Focus

Tab. 24. Opis parametrizácie a metód použitých vo výpočte.

Názov	Opis
algoritmus hľadania	paralelné gramatické roje
regresia	behaviorálna
veľkosť roja	300 častíc (pre každý roj 100)
veľkosť vektora častice	100 dimenzií
rozmer stavového priestoru	255
maximálna rýchlosť častice	32
sociálny a kognitívny koeficient	1.0
váha zotrvačnosti	od 0.9 do 0.4
metóda výpočtu fitnes	progresívna na základe počtu víťazstiev z 20 kôl (kap. 3.2.5)
trénovacia množina	MyFirstRobot, RamFire, Crazy, Tracker, Fire, TrackFire, VelociRobot, SpinBot, GuessFactor
iterácia nálezu	99
počet iterácií výpočtu	100
veľkosť programu (stromu)	39
princíp činnosti	okamžité natočenie tela aj zbrane k zaznamenanému robotovi so streľbou
hlavný nedostatky vývoja	dlhší čas výpočtu



Obr. 28. Vývoj globálnej (gBest) a priemerných lokálnych (lBest) fitnes počas výpočtu.

Tab. 25. Zoznam trénovacej množiny robotov pri vývoji Focusa.

Robot	Limit počtu víťazstiev pre prechod k nasledujúcemu súperovi (z 20 súbojov)
MyFirstRobot	14
RamFire	14
Crazy	14
Tracker	14
Fire	14
TrackFire	14
VelociRobot	10
SpinBot	5
GuessFactor	-

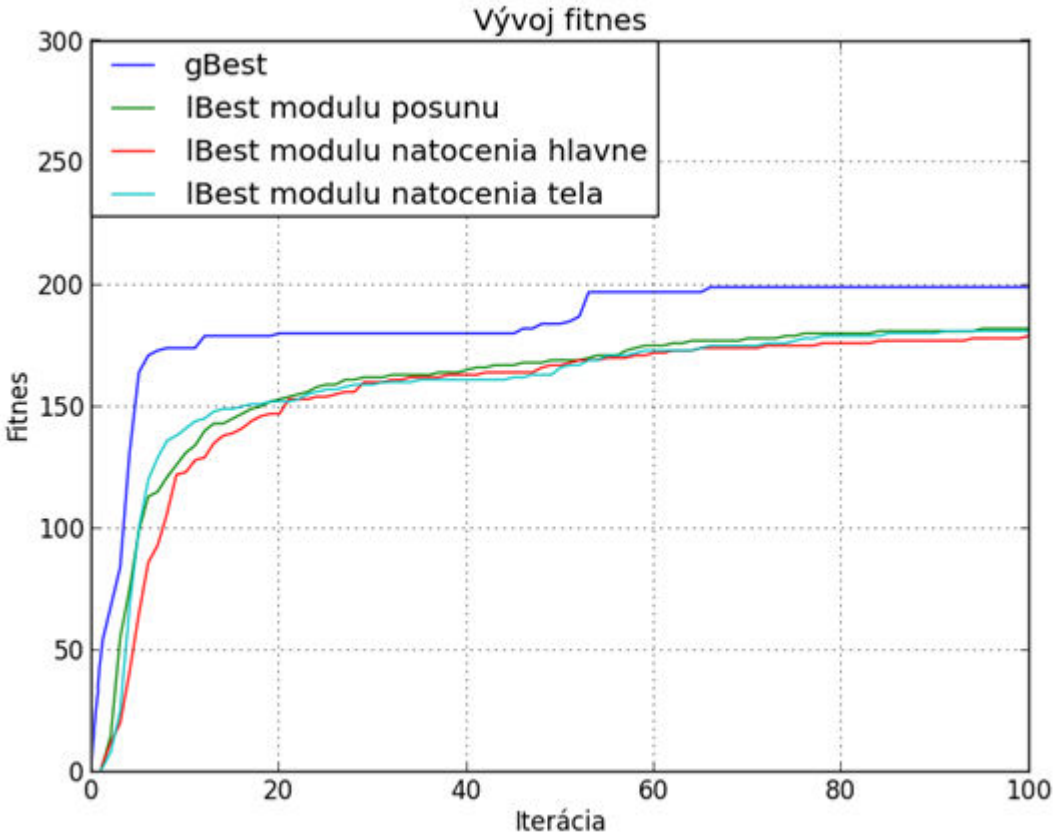
C.2.3 Carousel

Tab. 28. Opis parametrizácie a metód použitých vo výpočte.

Názov	Opis
algoritmus hľadania	paralelné gramatické roje
regresia	behaviorálna
veľkosť roja	300 častíc (pre každý roj 100)
veľkosť vektora častice	100 dimenzií
rozmer stavového priestoru	255
maximálna rýchlosť častice	32
sociálny a kognitívny koeficient	1.0
váha zotrvačnosti	od 0.9 do 0.4
gramatika	základná (kap. 3.2.3.1)
metóda výpočtu fitnes	progresívna na základe počtu víťazstiev z 20 kôl (kap. 3.2.5)
trénovacia množina	MyFirstRobot, RamFire, Crazy, Tracker, Fire, TrackFire, VelociRobot, SpinBot, GuessFactor, Peryton, Walls, GP-Bot, Duelist
iterácia nálezu	64
počet iterácií výpočtu	100
veľkosť programu (stromu)	33 (5, 20, 8)
princíp činnosti	krúživý pohyb a okamžitá streľba
hlavný nedostatky vývoja	dlhší čas výpočtu

Tab. 26. Zoznam nepriateľských robotov progresívnej metódy testovania.

Robot	Limit počtu víťazstiev pre prechod k nasledujúcemu súperovi (z 20 súbojov)
MyFirstRobot	10
RamFire	10
Crazy	10
Tracker	10
Fire	10
TrackFire	10
VelociRobot	10
SpinBot	5
GuessFactor	5
Peryton	5
Walls	5
GP-Bot	5
Duelist	-

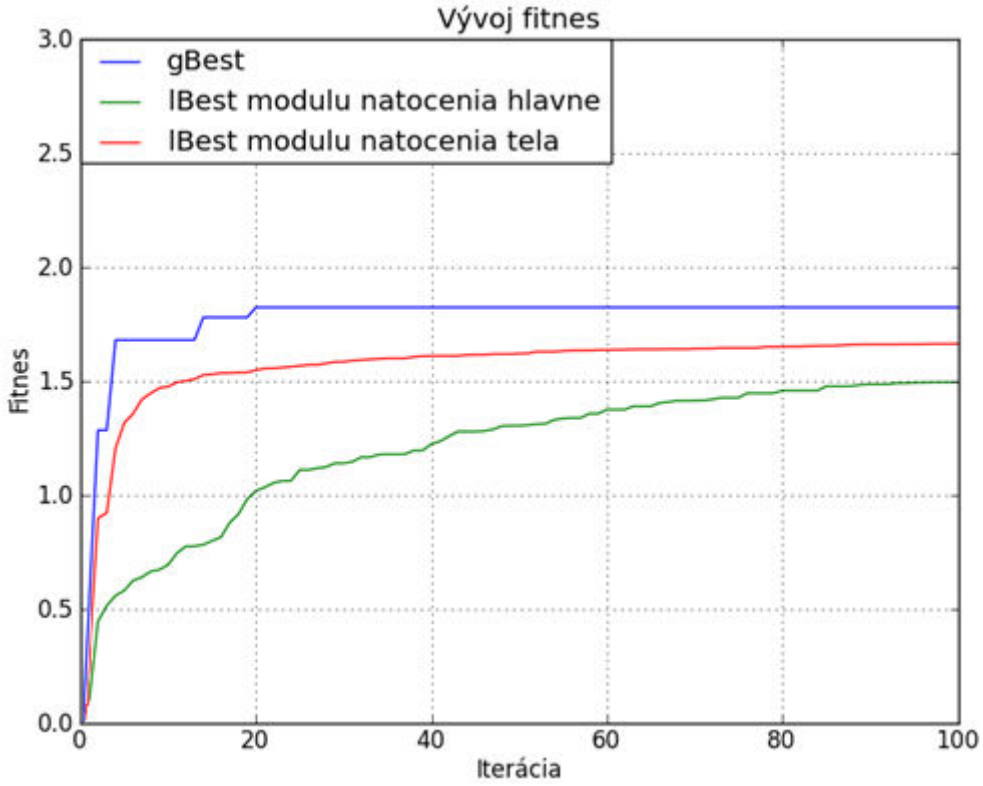


Obr. 29. Vývoj globálnej (gBest) a priemerných lokálnych (lBest) fitnes počas výpočtu.

C.2.4 Phoenix

Tab. 27. Opis parametrizácie a metód použitých vo výpočte.

Názov	Opis
algoritmus hľadania	paralelné gramatické roje
regresia	behaviorálna
veľkosť roja	100 častíc (pre každý roj 50)
veľkosť vektora častice	100 dimenzií
rozmer stavového priestoru	255
maximálna rýchlosť častice	32
sociálny a kognitívny koeficient	1.0
váha zotrvačnosti	od 0.9 do 0.4
metóda výpočtu fitnes	komplexná na základe priemerného pomerového skóre z 20 zápasov
trénovacia množina	Peryton, GP-Bot, Sparrow
iterácia nálezu	19
počet iterácií výpočtu	100
veľkosť programu (stromu)	32 (8, 24)
princíp činnosti	krúživý pohyb, okamžitá streľba a technika zameriavania pre streľbu Head-On
hlavný nedostatky vývoja	nízka konkurencieschopnosť proti robotom mimo základnú množinu

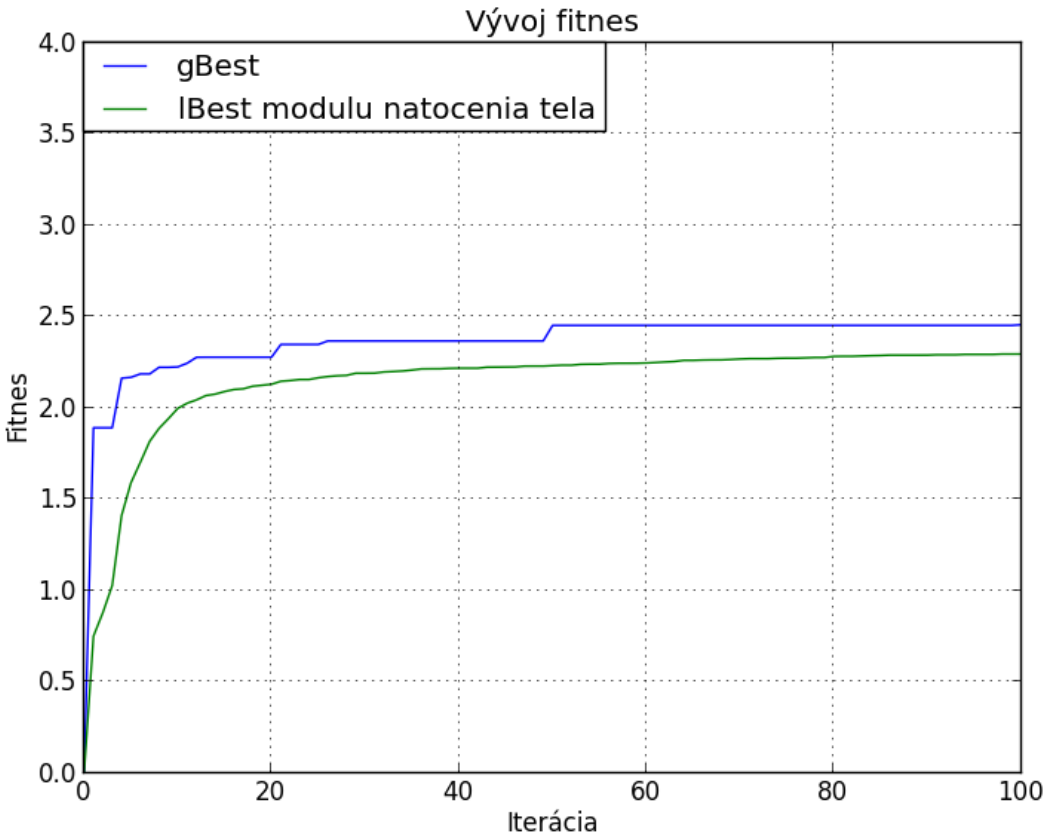


Obr. 30. Vývoj globálnej (gBest) a priemerných lokálnych (lBest) fitnes počas výpočtu.

C.2.5 Ringo 1.0

Tab. 28. Opis parametrizácie a metód použitých vo výpočte.

Názov	Opis
algoritmus hľadania	gramatický roj
regresia	parametrická
veľkosť roja	50 častíc
veľkosť vektora častice	7 dimenzií
rozmer stavového priestoru	100
maximálna rýchlosť častice	32
sociálny a kognitívny koeficient	1.0
váha zotrvačnosti	od 0.9 do 0.4
metóda výpočtu fitnes	komplexná na základe priemerného pomerového skóre z 20 zápasov
trénovacia množina	Peryton 1.1, GP-Bot, Sparrow, DuelistMicro
iterácia nálezu	25
počet iterácií výpočtu	100
princíp činnosti	striedavý krúživý pohyb, čiastočné vyhýbanie sa stenám, prispôsobovanie pohybu protivníkovi
hlavný nedostatky vývoja	len priemerná konkurencieschopnosť voči existujúcim NanoBotom

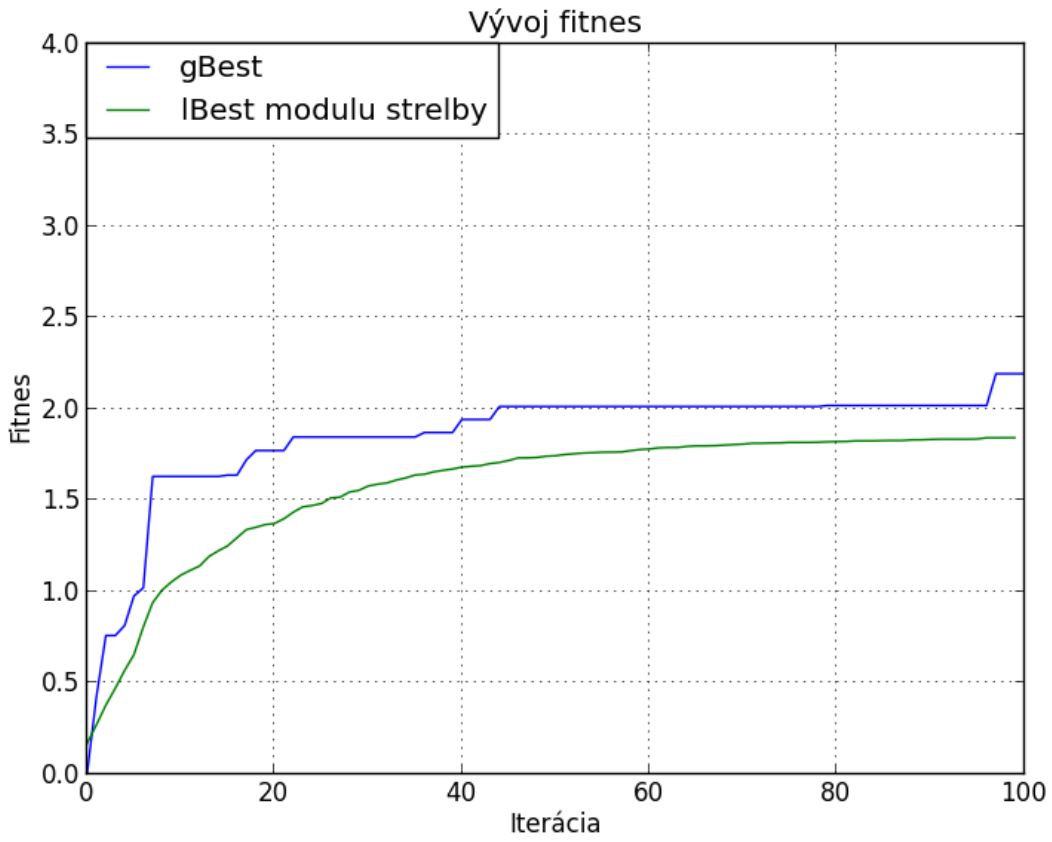


Obr. 31. Vývoj globálnej (gBest) a priemernej lokálnej (lBest) fitnes počas výpočtu.

C.2.6 Flex 1.0

Tab. 29. Opis parametrizácie a metód použitých vo výpočte.

Názov	Opis
algoritmus hľadania	gramatický roj
regresia	parametrická
veľkosť roja	60 častíc
veľkosť vektora častice	7 dimenzií
rozmer stavového priestoru	100
maximálna rýchlosť častice	30
sociálny a kognitívny koeficient	1.0
váha zotrvačnosti	od 0.9 do 0.4
metóda výpočtu fitnes	komplexná na základe priemerného pomerového skóre z 20 zápasov
trénovacia množina	jk.micro.Toorkild 0.4.5, jam.micro.RaikoMicro 1.44, voidios.micro.Jen 1.11, rz.GlowBlow 2.31
iterácia nálezu	43
počet iterácií výpočtu	100
princíp činnosti	streľba na základe váh virtuálnych zbraní, výber stratégie streľby a pohybu na základe úspešnosti

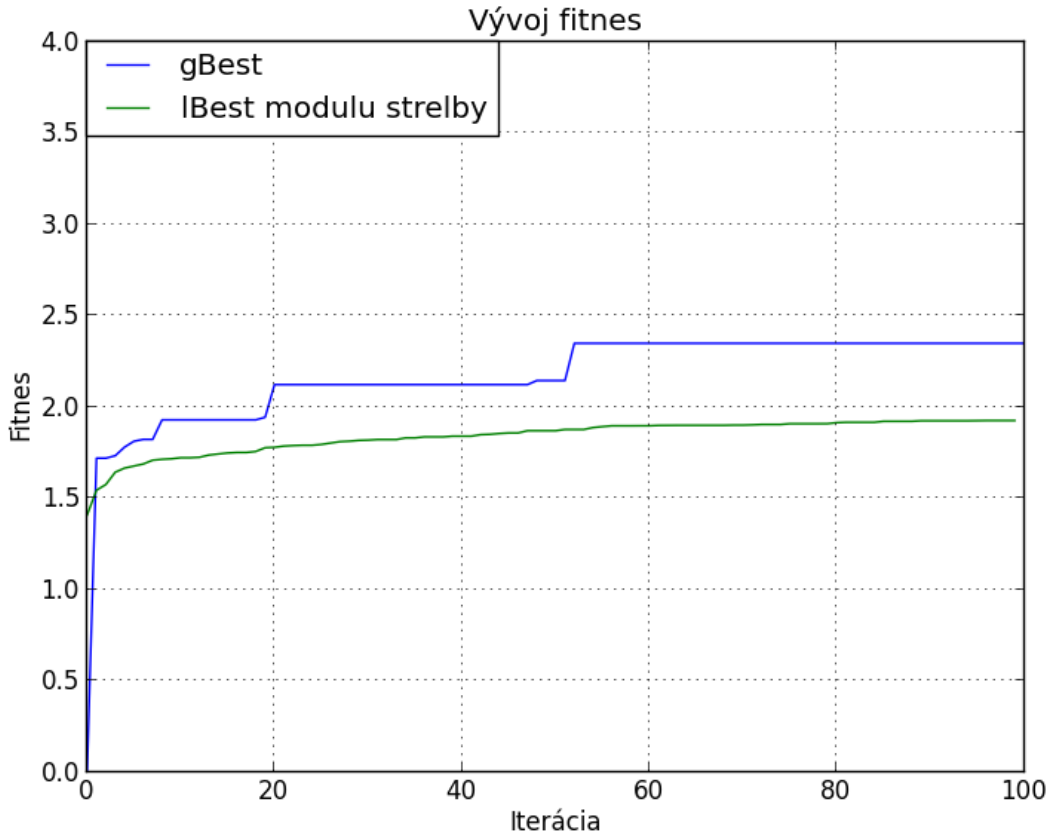


Obr. 32. Vývoj globálnej (gBest) a priemernej lokálnej (lBest) fitnes počas výpočtu.

C.2.7 Ringo 2.0

Tab. 30. Opis parametrizácie a metód použitých vo výpočte.

Názov	Opis
algoritmus hľadania	gramatický roj
regresia	symbolická
veľkosť roja	30 častíc
veľkosť vektora častice	255 dimenzií
rozmer stavového priestoru	100
maximálna rýchlosť častice	32
sociálny a kognitívny koeficient	1.0
váha zotrvačnosti	od 0.9 do 0.4
metóda výpočtu fitnes	komplexná na základe priemerného pomerového skóre z 20 zápasov
trénovacia množina	GPBot, casey.Flump 1.0, ratosh.Wesco 1.4, robar.nano.BlackWidow 1.3
iterácia nálezu	51
počet iterácií výpočtu	100
princíp činnosti	rovnaký ako v prípade Ringa 1.0, ale sila streľby závisí od viacerých indikátorov stavu



Obr. 33. Vývoj globálnej (gBest) a priemernej lokálnej (lBest) fitnes počas výpočtu.

C.3 Podrobné výsledky testovania robotov

Tab. 31. Výsledky získaného skóre v testovaní proti manuálne vytvoreným robotom

(stĺpec *RoboRumble* predstavuje umiestnenie robota v tejto lige).

Súper	RoboRumble	Rumbler	Taurus	Focus	HarperBot	Carousel	GP-BotC	Phoenix	Ringo 1.0	GP-BotA	Ringo 2.0	Flex 1.0	Flex 1.5
sample.Crazy 1.0	975.	51%	53%	70%	86%	83%	86%	86%	77%	80%	89%	94%	99%
sample.Tracker 1.0	964.	42%	52%	67%	83%	85%	93%	93%	99%	98%	99%	96%	99%
sample.TrackFire 1.0	955.	33%	31%	59%	64%	81%	87%	94%	98%	96%	97%	94%	99%
sample.RamFire 1.0	951.	40%	52%	68%	67%	80%	89%	88%	97%	96%	97%	99%	99%
sample.VelociRobot 1.0	946.	48%	44%	63%	74%	73%	85%	89%	92%	95%	96%	86%	98%
sample.SpinBot 1.0	934.	44%	47%	41%	72%	77%	83%	87%	93%	96%	94%	92%	99%
sample.Walls 1.0	918.	4%	12%	11%	65%	58%	43%	37%	25%	51%	27%	84%	94%
gg.Peryton 1.1	-	11%	10%	17%	55%	42%	64%	72%	73%	54%	72%	82%	81%
NDH.GuessFactor 1.0	698.	10%	13%	16%	51%	49%	64%	73%	69%	60%	72%	65%	61%
dummy.micro.Sparrow 2.5	504.	7%	3%	6%	25%	25%	43%	51%	46%	55%	46%	72%	62%
apv.Cannibal 1.1	431.	37%	32%	47%	21%	23%	30%	34%	31%	34%	34%	43%	53%
apv.Aspid 1.7	187.	9%	5%	12%	8%	17%	16%	19%	24%	27%	21%	44%	42%
abc.Tron 2.02	172.	14%	9%	11%	8%	14%	17%	15%	18%	20%	18%	41%	43%
cx.mini.Cigaret 1.31	123.	6%	6%	7%	15%	12%	15%	14%	19%	18%	19%	39%	42%
Priemer	**-**	**25%**	**26%**	**35%**	**50%**	**51%**	**58%**	**61%**	**62%**	**63%**	**63%**	**74%**	**77%**

Tab. 32. Výsledky počtu výhier v testovaní proti manuálne vytvoreným robotom

(stĺpec *RoboRumble* predstavuje umiestnenie robota v tejto lige).

Súper	RoboRumble	Rumbler	Taurus	Focus	Carousel	GP-BotC	Phoenix	Ringo 1.0	GP-BotA	Ringo 2.0	Flex 1.0	Flex 1.5
sample.Crazy 1.0	975.	34%	44%	62%	72%	76%	75%	55%	61%	81%	100%	100%
sample.Tracker 1.0	964.	28%	49%	95%	99%	100%	100%	100%	100%	100%	99%	100%
sample.TrackFire 1.0	955.	6%	3%	98%	92%	88%	100%	100%	100%	100%	100%	100%
sample.RamFire 1.0	951.	25%	62%	93%	96%	99%	100%	100%	100%	99%	100%	100%
sample.VelociRobot 1.0	946.	50%	43%	58%	67%	86%	95%	93%	97%	100%	98%	99%
sample.SpinBot 1.0	934.	33%	41%	24%	75%	87%	94%	98%	99%	99%	100%	100%
sample.Walls 1.0	918.	0%	3%	3%	33%	14%	7%	1%	18%	3%	95%	93%
gg.Peryton 1.1	-	0%	0%	1%	27%	62%	74%	66%	33%	68%	91%	85%
NDH.GuessFactor 1.0	698.	0%	0%	0%	54%	74%	85%	79%	64%	85%	78%	71%
dummy.micro.Sparrow 2.5	504.	0%	0%	0%	13%	34%	46%	35%	46%	44%	81%	65%
apv.Cannibal 1.1	431.	17%	6%	26%	9%	12%	19%	18%	11%	27%	44%	55%
apv.Aspid 1.7	187.	0%	0%	2%	4%	1%	2%	12%	9%	8%	41%	39%
abc.Tron 2.02	172.	0%	0%	0%	1%	1%	0%	0%	2%	1%	32%	36%
cx.mini.Cigaret 1.31	123.	0%	0%	0%	2%	2%	2%	4%	3%	7%	34%	37%
Priemer	**-**	**14%**	**18%**	**33%**	**46%**	**53%**	**57%**	**54%**	**53%**	**59%**	**78%**	**77%**

Tab. 33. Výsledky získaného skóre v testovaní proti evolvovaným robotom.

Súper	Rumbler	Taurus	Focus	Carousel	Phoenix	GP-BotC	Ringo 1.0	GP-BotA	Ringo 2.0	Flex 1.0	Flex 1.5
Rumbler	-	54%	71%	84%	89%	89%	94%	96%	95%	92%	97%
Taurus	46%	-	72%	85%	91%	88%	95%	96%	96%	90%	95%
Focus	29%	28%	-	75%	91%	83%	96%	96%	95%	81%	97%
Carousel	16%	15%	25%	-	63%	59%	70%	73%	73%	78%	94%
Phoenix	11%	9%	9%	37%	-	48%	69%	70%	72%	65%	94%
GP-BotC	11%	12%	17%	41%	52%	-	58%	64%	63%	51%	92%
Ringo 1.0	6%	5%	4%	30%	31%	42%	-	61%	62%	66%	91%
GP-BotA	4%	4%	4%	27%	30%	36%	39%	-	53%	55%	87%
Ringo 2.0	5%	4%	5%	27%	28%	37%	38%	47%	-	63%	89%
Flex 1.0	8%	10%	19%	22%	35%	49%	34%	45%	37%	-	57%
Flex 1.5	3%	5%	3%	6%	6%	8%	9%	13%	11%	43%	-
Priemer	**14%**	**15%**	**23%**	**43%**	**52%**	**54%**	**60%**	**66%**	**66%**	**68%**	**89%**

Tab. 34. Podrobné výsledky počtu výhier v testovaní proti evolvovaným robotom.

Súper	Rumbler	Taurus	Focus	Carousel	Phoenix	GP-BotC	Ringo 1.0	GP-BotA	Ringo 2.0	Flex 1.0	Flex 1.5
Rumbler	-	63%	99%	98%	98%	99%	99%	100%	100%	100%	100%
Taurus	37%	-	96%	96%	100%	97%	99%	100%	100%	100%	98%
Focus	1%	4%	-	91%	100%	92%	100%	100%	100%	100%	100%
Carousel	2%	4%	9%	-	63%	60%	75%	75%	80%	97%	100%
Phoenix	2%	0%	0%	37%	-	47%	76%	75%	83%	80%	100%
GP-BotC	1%	3%	8%	40%	53%	-	61%	71%	73%	58%	96%
Ringo 1.0	1%	1%	0%	25%	24%	39%	-	64%	74%	81%	98%
GP-BotA	0%	0%	0%	25%	25%	29%	36%	-	65%	65%	97%
Ringo 2.0	0%	0%	0%	20%	17%	27%	26%	35%	-	72%	95%
Flex 1.0	0%	0%	0%	3%	20%	42%	19%	35%	28%	-	59%
Flex 1.5	0%	2%	0%	0%	0%	4%	2%	3%	5%	41%	-
Priemer	**4%**	**8%**	**21%**	**44%**	**50%**	**54%**	**59%**	**66%**	**71%**	**79%**	**94%**

Príloha D: Obsah elektronického média

Elektronické médium má nasledovnú štruktúru:

- **/app** – obsahuje spustiteľnú aplikáciu,
  - **/backup** – obsahuje zbalené dáta realizovaných výpočtov robotov,
  - **/conf** – obsahuje konfiguračné súbory použité pri výpočte,
  - **/script** – obsahuje Python skripty,
  - **/templates** – obsahuje šablóny programov,
  - **/test** – obsahuje dáta výpočtov, ktoré je možné prehliadať v aplikácii,
- **/docs** – obsahuje dokumentáciu diplomovej práce,
- **/install** – obsahuje inštalačné súbory potrebných prostredí,
- **/robots** – obsahuje balík s existujúcimi a vyvinutými robotmi (*hlavko.**),
- **/src** – obsahuje zdrojový kód Java aplikácie.