



MATEMATICKO-FYZIKÁLNÍ FAKULTA Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Tomáš Karella

Evoluční algoritmy pro řízení heterogenních robotických swarmů

Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: Mgr. Martin Pilát, Ph.D.

Studijní program: Informatika

Studijní obor: Programování a Softwarové Systémy

Praha 2017

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V dne

Podpis autora

Poděkování.

Název práce: Evoluční algoritmy pro řízení heterogenních robotických swarmů

Autor: Tomáš Karella

Katedra: Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: Mgr. Martin Pilát, Ph.D., Katedra teoretické informatiky a matematické logiky

Abstrakt: Abstrakt.

Klíčová slova: klíčová slova

Title: Evolutionary Algorithms for the Control of Heterogeneous Robotic Swarms

Author: Tomáš Karella

Department: Department of Theoretical Computer Science and Mathematical Logic at Faculty of Mathematics and Physics

Supervisor: Mgr. Martin Pilát, Ph.D., Department of Theoretical Computer Science and Mathematical Logic

Abstract: Abstract.

Keywords: key words

Obsah

Úvod	2
1 Evoluční algoritmy	3
1.1 Historie	3
1.2 Co je evoluční algoritmus?	3
1.3 Části evolučních algoritmů	5
1.3.1 Reprezentace	5
1.3.2 Populace	6
1.3.3 Selekce rodičů	7
1.3.4 Variační operátory	7
1.4 Diferenciální evoluce	9
1.5 Evoluční strategie	10
2 Robotický Swarm	12
2.1 Základní vlastnosti	12
2.2 Použití	13
2.3 Řízení robotických swarmů	14
2.3.1 Genetické programování a stromy chování	14
2.3.2 Genetické algoritmy a neuronová síť	17
2.3.3 Evoluční strategie a neuronová síť	19
3 Simulátor	21
4 Experimenty	22
4.1 Použité technologie	23
4.1.1 Reprezentace Chování	23
4.1.2 Evoluční algoritmy	23
4.2 WoodScene experiment	24
4.2.1 Cíl Experimentu	24
4.2.2 Roboti	26
4.2.3 Vyhodnocování Fitness	28
4.2.4 Podíkoly	28
4.2.5 Výsledky Experimentu	41
4.3 Mineral Scene	44
4.4 Competitive Scene	44
Závěr	45
Seznam použité literatury	46
Seznam obrázků	49
Seznam tabulek	50
Seznam použitých zkratek	51
Přílohy	52

Úvod

Využití robotických hejn (robotic swarms) patří mezi rentabilní metody pro řešení složitějších úkolů. Existuje řada studií potvrzující, velký počet jednoduchých robotů dokáže plně nahradit komplexnější jedince. Dostatečná velikost hejna umožní řešení úloh, které by jednotlivec z hejna provést nesvedl. Navíc robotické hejno přináší několik výhod, díky kvantitě jsou odolnější proti poškození či zničení, neboli zbytek robotů pokračuje v plnění cílů. Dále výroba jednoduších robotů vychází levněji než komplexní jedinců, což přináší nezanedbatelnou výhodu pro práci v nebezpečném prostředí. Hejno také může pokrývat více různých úkolů než specializovaný robot, který bude při plnění úkolů, lišících se od typu úloh zamýšlených při konstrukci, mnohem více nemotorný a nejspíše pomalejší. Hejno pokryje větší plochu při plnění úkolů.

Existuje mnoho aplikací robotických hejn, většinou se používají v úlohách týkajících se průzkumu a mapování prostředí, hledání nejkratších cest, nasazení v nebezpečných místech (Jevtić a Andina de la Fuente, 2007). Jako příklad můžeme uvést asistenci záchranným složkám při požáru (Penders a kol., 2011). Mnoho projektů zabývající se řízením robotického hejna se inspiruje přírodou, používá se analogie s chováním mravenců a jiného hmyzu (David a kol., 2001). Objevují se i hardwarové implementace chování hejn, zmiňme projekty Swarm-bots (professor Marco Dorigo, 2001), Colias (Arvin a kol., n.d.)

Elementárnost senzorů i efektorů jednotlivých robotů vybízí k použití evolučních algoritmů, jelikož prostor řešení je rozlehly a plnění cílů lze vhodně ohodnotit. Vzniklo několik vědeckých prací popisující problematiku tohoto tématu (Gomes a kol., 2013) (Ivan a kol., 2013).

Cíl práce

Všechny zmíněné práce používají pro tvorbu řídicích programů evoluční algoritmy (EA) a pracují pouze s homogenními hejny. Cílem této práce je vyzkoušet využití EA na generování chování hejna heterogenních robotů, tedy skupiny agentů, ve které se objevuje několik druhů jedinců a společně plní daný úkol. V rámci práce byl vytvořen program pro simulaci různých scénářů. Pro otěstování jejich úspěšnosti v rámci EA, byly zvoleny 3 odlišné scénáře, ve kterých se objevují 2-3 druhy robotů.

Struktura práce

Rozdelení práce je následující. První kapitola je věnována obecnému úvodu do tématiky evolučních algoritmů, kde se podrobněji venuji evolučním strategiím a diferenciální evoluci, protože oba tyto postupy implementuji v programu pro řešení scénářů.

Dodat zbytek práce

1. Evoluční algoritmy

1.1 Historie

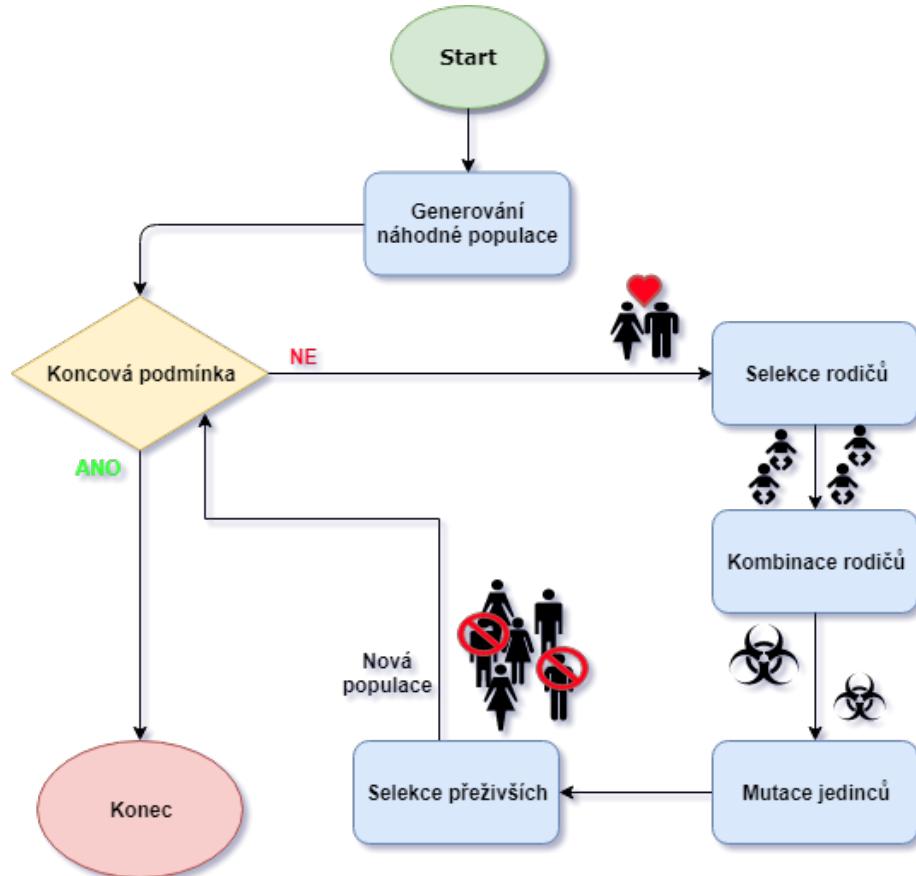
Začněme pohledem do historie Evolučních algoritmů na základě knih Mitchell (1998) a Eiben (2015). Darwinova myšlenka evoluce lákala vědce už před průlomem počítačů, Turing vyslovil myšlenku *genetického a evolučního vyhledávání* už v roce 1948. V 50. a 60. letech nezávisle na sobě vznikají 4 hlavní teorie nesoucí podobnou myšlenku. Společným základem všech teorií byla evoluce populace kandidátů na řešení daného problému a jejich následná úprava způsoby hromadně nazývány jako genetické operátory, například mutace genů, přirozená selekce úspěšnějších řešení, atp.

Rechenberg a Schwefell (1965, 1973) představili *evoluční strategie*, což je metoda optimalizující parametry v reálných číslech, v jejich práci se objevují jako prostředek pro optimalizaci tvaru letadlových křídel. Fogel, Owens, Walsh zveřejnili *evolutionary programming* (evoluční programování), technika využívající k reprezentaci kandidátů konečný automat (s konečným počtem stavů), který je vyvíjen mutací přechodů mezi stavy a následnou selekcí. *Genetické algoritmy* vynalezl Holland v 60. letech a následně se svými studenty a kolegy z Michiganské Univerzity vytvořil první implementaci. U genetických algoritmů, oproti ES a EP, nebylo hlavním cílem formovat algoritmus pro řešení konkrétních problémů, ale přenos obecného mechanismu evoluce jako metody aplikovatelné v informatickém světě. Princip GA spočívá v transformaci populace chromozomů (př. vektor 1 a 0) v novou populaci pomocí genetických operátorů křížení, mutací a inverze. V 1975 v knize *Adaptation in Natural and Artificial Systems* (Holland, 1976) Holland definoval genetický algoritmus jako abstrakci biologické evoluce spolu s teoretickým základem jejich používání. Někteří vědci ovšem používají pojmenování GA i ve významech hodně vzdálených původní Holandové definici. K sjednocení jednotlivých přístupů přispěl v 90. letech John R. Koza, dále jsou všechny metody zahrnutý pod pojmem *Evoluční algoritmy* jako jejich součásti. Dnes se věnuje tématu EA celá řada konferencí a odborných časopisů. Zmiňme ty nejvýznamnější z nich, v kontextu konferencí: GECCO, PPSN, CEC, EVOSTAR, odborné časopisy časopisy, na které bych rád upozornil: Evolutionary Computation, IEEE Transactions on Evolutionary Computation, Genetic Programming and Evolvable Machines, Swarm and Evolutionary Computation

1.2 Co je evoluční algoritmus?

Ač existuje mnoho variant evolučních algoritmů, jak jsme zmínili v krátké pohledu do historie, spojuje je společná myšlenka populace jedinců uvnitř prostředí s omezenými podmínkami. Jedinci, jinak také nazývání kandidáti, soutěží o zdroje daného prostředí, tím je docíleno přírodní selekce (přežijí jen ti nejlepší). Pokud budeme mít k dispozici kvalitativní funkci, kterou se snažíme maximizovat, pak nebude problém vytvořit náhodné jedince z definičního oboru přesně této funkce. Náhodně vzniklé jedince můžeme ohodnotit, tímto způsobem dáme vzniknout abstrakci pro měření *fitness* (z anglického fit: nejhodnější, zapadající). Z vzniklých a ohodnocených jedinců lze zvolit ty nejlepší pro tvorbu nové

generace jedinců. Tvorba nové generace probíhá kombinováním zvolených rodičů a mutacemi jedinců. Jako kombinaci uvažujeme operátor, který je aplikován na dva a více zvolených kandidátů (proto se jim také říká rodiče) a tvořící jednoho a více nových jedinců (také nazývány potomci). Mutace je aplikována pouze na jednoho jedince a její výsledkem je také pouze jednoho jedinec. Tyto dvě operace aplikované na rodičovskou generaci vedou k vytvoření nových kandidátů (potomků, offsprings). I tato nová generace je ohodnocena fitness a dále soutěží se starými jedinci na základě fitness o místo v nové generaci, občas také mimo fitness funkce bereme v potaz stáří kandidáta. Popsaný proces je opakován dokud není nalezen kandidát s dostatečně velkou fitness nebo dosáhneme maximálního počtu iterací (bylo dosaženo požadovaného počtu opakování apod.). Základy evolučního systému pohání dvě základní hnací síly: *variační operátory*, *selektivní operátory*. Variační operátory zajišťují potřebnou různorodost v populaci a tím tvoří nové cesty k úspěšnému kandidátovi. Oproti tomu selektivní operátory zvyšují průměrnou kvalitu řešení v celé populaci. Kombinací těchto operátorů obecně vede ke zlepšování fitness hodnot v následující generaci. Funkčnost evoluce ověříme snadno, stačí nám k tomu pozorovat zda se fitness v populaci blíží více a více k optimálním hodnotám vzhledem k postupu v čase. Mnoho komponent zapříčiňuje, že EA se řadí ke stochastickým metodám, selekce totiž nevybírá nejlepší jedince deterministicky, i jedinci s malou fitness mají šanci být rodiči následující generace. Během kombinování jsou části rodičů, které budou zkombinovány, také zvoleny pomocí dané pravděpodobností funkce. Podobně je tomu u mutací. Část která bude změněna, je taktéž určena náhodou, stejně tak nové rysy nahrazující staré.



Ze schéma na obrázku můžeme vyčíst, že EA patří mezi algoritmy *generate and*

test (vygeneruj a otestuj, G&T). Vyhodnocení fitness funkce poskytuje heuristický odhad kvality řešení, prohledávaní je řízen variací a selekcí. EA splňují charakteristické rysy G&T algoritmů, zpracovávají zároveň celé kolekce kandidátů. Většina EA míchá informace ze dvou a více kandidátů. EA se řadí ke stochastickým metodám.

Různé dialekty evolučního programování, zmíněné v historickém okénku, splňují všechny tyto hlavní rysy. Liší se pouze v technických detailech. Kandidáti jsou často reprezentováni různě, liší se datové struktury pro jejich uchování i jejich zakódování. Typicky se jedná o řetězce nad konečnou abecedou v případě GA, vektory reálných čísel v ES, konečné automaty v EP a stromy v GP. Důvod těchto rozdílů je hlavně historický. Technicky však lze upřednostnit jednu reprezentaci, pokud lépe odpovídá danému problému, tzn. kóduje kandidáta jednodušeji či přirozenější formou. Pro ilustraci zvolme řešení splnitelnosti (SAT) s n proměnnými, vcelku přirozeně sáhneme po použití řetězce bitů délky n a obsah $i - tého$ bitu označuje, že $i - tā$ proměnná má hodnotu (1) - pravda, (0) - nepravda, proto bychom použili jako vhodný EA genetické algoritmy. Oproti tomu evoluce programu, který hraje šachy, by bylo vhodnější použít derivační strom, kde by vrcholy odpovídaly tahům na šachovnici. Přirozenější by tedy bylo použít GP.

Neopomeňme zmínit ještě dva důležité fakty. Kombinační a mutační operátory musí odpovídat dané reprezentaci, např. v GP kombinace pracuje se stromy (kombinují podstromy ...), zatímco v GA na řetězcích (prohazují části řetězců). A dále oproti variacím musí fitness selekce záviset vždy pouze na fitness funkci, takže selekce pracuje nezávisle na reprezentaci.

1.3 Části evolučních algoritmů

Abychom vytvořili běhu schopný evoluční algoritmus, musíme specifikovat každou zmíněnou část a také inicializační funkci, která připraví první populaci. Pro konečný algoritmus ještě nesmíme opomenout a dodat koncovou podmínu.

- Reprezentace (definice individuí)
- Ohodnocující funkce (fitness funkce)
- Populace
- Selekce rodičů
- Variační operátory (kombinace, mutace)
- Selekce prostředí

1.3.1 Reprezentace

Při tvorbě EA nejdříve musíme propojit prostor původního problému s prostorem řešení, kde se bude vlastní evoluce odehrávat. K docílení propojení je většinou potřeba zjednodušit či vytvořit abstrakci nad aspekty reálného světa (prostor problému), abychom vytvořili vhodný prostor řešení, kde mohou být jednotlivá řešení ohodnocena. Neboli chceme docílit, aby možná řešení mohla být specifikována a

uložena, tak aby s nimi mohl počítač pracovat. Objekty reprezentující možná řešení v kontextu původního problému jsou nazývány *fenotyp*, zatímco kódování na straně EA prostoru se jim říká *genotyp*. Reprezentace označuje mapování z fenotypů na genotypy, používá se ve významu funkce z fenotypu na genotyp (encode) i genotypu na fenotyp (decode) a předpokládá se, že pro každý genotyp existuje nejvýše jeden fenotyp. Například pro optimalizační problém, kde množinou řešení jsou celá čísla. Celá čísla tvoří množinu fenotypu a můžeme se rozhodnout pro reprezentaci v binárních číslech. Tedy 18 by byl fenotyp a 10010 jeho genotyp. Prostor fenotypů a genotypů se zpravidla velmi liší a celý proces EA se odehrává pouze v genotypovém prostoru, vlastní řešení dostaneme rozkódováním nejlepšího genotypu po ukončení EA. Jelikož nevíme, jak vlastní řešení vypadá, je nanejvýš vhodné umět reprezentovat všechny možná řešení, v G&T bychom řekli, že generátor je kompletní.

- V kontextu původního problému jsou následující výrazy ekvivalentní: fenotyp, kandidát (na řešení), jedinec, individuum (množina všech možných řešení = fenotypový prostor)
- V kontextu EA: genotyp, chromozom, jedinec, individuum (množina, kde probíhá EA prohledávání = genotypový prostor)
- Části individuí jsou nazývány gen, locus, proměnná a dále se dělí na alely, či hodnoty

Neuronové sítě

Pro reprezentaci jedinců v oblasti robotiky, rozpoznávání obrazů a dalších oblastí umělé inteligence se v poslední době používají nejčastěji neuronové sítě. Neuronová síť se strukturou podobá neuronovým sítím v mozku. Základní sítě se skládají z jednotlivých neuronů. V informatickém světě se jmenují perceptrony. Perceptron je funkce $\mathbb{R}^n \rightarrow \mathbb{R}$ a definovaná $\sum_{i < n+1} w_i x_i$, kde pro $i < n$ x_i je i -tý prvek vstupního vektoru, pro $i = n+1$ je $x_i = 1$. w_i se označuje jako váha a většinou $w_i \in \mathbb{R}$.

Jednovrstvou neuronovou sítí pak myslíme n perceptronů, tedy funkci $\mathbb{R}^n \rightarrow \mathbb{R}^n$, kde $i - t$ prvek výstupního vektoru dostaneme aplikací $i - t$ perceptronové funkce neuronové sítě.

Pokud dovolíme skládání perceptronů, vzniknou nám tzv. vícevrstvé neuronové sítě. Podmnožiny výstupů z první vrstvy neuronů neurčují přímo výstup, ale jsou opět zvoleny jako vstupní vektory pro další jednovrstvou neuronovou síť.

Skrytá vrstva (hidden layer) je taková jednovrstvá neuronová síť, jejíž výstup je pouze vstupem jiné neuronové sítě.

1.3.2 Populace

Populace je multimnožina genotypů, slouží jako jednotka evoluce. Populace uvádí adaptaci a změny, zatímco vlastní jedinci se nijak nemění, jen vznikají noví a nahrazují předešlé. Pro danou reprezentaci je definice populace velmi jednoduchá, charakterizuje ji pouze velikost. U některých specifických EA má populace další prostorové struktury, definované vzdáleností jedinců nebo relacemi sousedních jedinců, což by se dalo připodobnit reálným populacím, které se vyvíjejí v

různých geografických prostředích. U většiny EA se velikost populace nemění, což vede k soutěživosti mezi jedinci („zůstanou ti nejlepší“). Na úrovni populací pracují právě selektivní operátory. Například zvolí nejlepší jedince aktuální populace jako rodiče následující a nahradí nejhorský jedince novými. Rozmanitost populace - vlastnost, které chceme z pravidel docílit, je měřena jako počet různých řešení v multimnožině. Neexistuje však jediné hledisko, podle kterého lze tuto vlastnost měřit, většinou se používá počet rozdílných hodnot fitness, rozdílných fenotypů či genotypů a také entropie (míra neuspořádanosti). Ovšem musíme mít na paměti, že jedna hodnota fitness v populaci neznamená, že populace obsahuje pouze jeden fenotyp, stejně tak jeden fenotyp nemusí odpovídat jednomu genotypu apod.

1.3.3 Selekce rodičů

Rodičovská selekce, někdy také partnerská selekce, vybírá lepší jedince jako rodiče pro příští generaci. Jedinec se stává rodičem, pokud byl zvolen k aplikaci variačních operátorů a tím dal vzniknout novým potomkům. Společně s selekcí přeživších je rodičovská selekce zodpovědná za zvedání kvality v populacích. Rodičovská selekce je v EA většinou pravděpodobnostní metoda, která dává jedincům s větší kvalitou mnohem větší šanci být vybrán než těm s nízkou. Nicméně i jedincům s nízkou kvalitou je často přidělena malá nenulová pravděpodobnost pro výběr, jinak by se celé prohledávání mohlo vydat slepou cestou a zaseknout se na lokální optimu.

1.3.4 Variační operátory

Hlavní úlohou variačních operátorů (mutace, rekombinaci) je vytváření nových individuí ze starých. Z pohledu Generate Test algoritmů spadají variační operátory do právě do Generate části. Obvykle je dělíme na dva typy podle jejich arity, jedná se o unární (Mutace) a n-arní (rekombinace) operátory.

Mutace

Ve většině případů myslíme unárním variačním operátorem mutace. Tento operátor je aplikován pouze na jeden genotyp a výsledkem je upravený potomek. Mutace řadíme mezi stochastické metody, výstup (potomek) totiž závisí na sérii náhodných rozhodnutí. Však mutace není vhodné pojmenování pro všechny variační operátory, například pokud se jedná o heuristiku závislou na daném problému, která se chová systematicky, hledá slabé místo a následně se jej snaží vylepšit, v tomto případě se nejedná o mutaci v pravém slova smyslu. Obecně by mutace měla způsobovat náhodné a nezaujaté změny. Unární variační operátory hrají odlišnou roli v rozdílných EA opět díky historicky oddělenému vývoji. Zatímco v GP se nepoužívají vůbec, v GA mají velmi důležitou roli a v EP se jedná o jediný variační operátor. Díky variačním operátorům aplikovaným v jednotlivých evolučních krocích dostává prohledávací prostor topologickou strukturu. Existují teorie podporující myšlenku, že EA (s dostatečným časem) naleznou globální optimum daného problému opírající se právě o tuto topologickou strukturu a také spoléhají na fakt, že může vzniknout každý genotyp reprezentující možné řešení. Nejjednodušší cesta ke splnění těchto podmínek vede právě přes variační operátory. U mutací tohoto například dosáhneme, když povolíme, aby mohly skočit

kamkoliv, tzn. každá alela může být zmutována na jakoukoli další s nenulovou pravděpodobností. Většina vědecké společnosti považuje tyto důkazy za nepříliš použitelné v praktickém využití a proto tuto vlastnost většina EA neimplementuje.

Rekombinace

Rekombinace, také nazývána křížení, je binární variační operátor. Jak název napovídá, spojuje informace ze dvou rodičů (genotypů) do jednoho nebo dvou potomků. Stejně jako mutace patří rekombinace k stochastickým operátorům. Rozhodnutí, jaké části budou zkombinovány a jakým způsobem toho bude docíleno, závisí na náhodě. Role rekombinace se znovu liší v rozličných EA, v GP se jedná o jediné variační operátory, v EP nejsou použity vůbec. Existují rekombinační operátory s vyšší aritou (, tzn. používající více než dva rodiče) a jsou také jednoduché na implementaci. Dokonce několik studií potvrdilo, že mají velmi pozitivní vliv na celou evoluci, ale nemají tak hojně zastoupení, nejspíše proto, že neexistuje biologický ekvivalent. Rekombinace funguje na jednoduchém principu, slučuje dvě individua a produkuje jednoho či více potomků, kteří kombinují jejich výhodné vlastnosti, tedy potomek by měl být úspěšnejší než jeho rodiče. Tento princip podporuje fakt, že po tisíciletí se aplikací rekombinace na rostliny a zemědělská zvířata mnohokrát podařilo vytvořit nové jedince s vyšším výnosem či jinými výhodnými vlastnostmi (odolnost proti škůdcům atd.). Evoluční algoritmy vytváří množství potomků náhodnou rekombinací a doufáme, že zatímco malá část nové generace bude mít nežádoucí vlastnosti, většina se nezlepší ani si nepohorší a konečně další malá část předčí jejich rodiče. Na naší planetě se sexuálně (kombinací dvou jedinců) rozmnožují pouze vyšší organismy, což vzbuzuje dojem, že rekombinace je nejvyšší forma reprodukce. Neopomeňme, že variační operátory jsou závislé na reprezentaci (genotypu) jedinců. Např. pro genotypy bitových řetězců může použít bitovou inverzi jako vhodný mutační operátor bude-li ovšem genotyp strom musíme zvolit nějaký jiný.

Selekce prostředí

Selekce prostředí, někdy také nazývána selekce přeživších, jak název napovídá má blízko k selekci rodičů, odlišuje jednotlivá individua na základě jejich fitness hodnoty. Oproti rodičovské selekci ji však používáme v jiné části evolučního cyklu. Selekcí prostředí spouštíme hned po vytvoření nových potomků. Jelikož velikost populace kandidátů bývá skoro vždy konstantní, je nutné rozhodnout, které kandidáty zvolíme do další generace. Toto rozhodnutí většinou závisí na hodnotě fitness jednotlivých kandidátů, také se často hledí na dobu vzniku daného kandidáta. Dobou vzniku myslím generaci v které daný jedinec vznikl. Na rozdíl od rodičovské selekce, která bývá stochastická, bývá selekce prostředí deterministickou metodou. Uvedeme dvě nejběžnější metody, obě kladou největší důraz na fitness. První vybírá nejlepší segment z množiny nových potomků i z původní generace nezávisle, druhá dělá totéž jen z množiny nových potomků. Selekce prostředí je uváděna i pod názvem "záměnná" strategie (replacement strategy). Selekce prostředí (preživších) se víceméně používá, pokud je množina nových potomků větší než velikost generace a zámena využíváme když je nových potomků velmi málo.

Inicializace populace

Inicializace populace zastává důležitý úkol a to vytvořit první generaci kandidátů. Její implementace bývá ve většině EA její implementace velmi jednoduchá. První generace je vygenerována čistě náhodně. Principiálně by se zde dalo využít nějaké heuristiky k vytvoření vyšší fitness v první generaci, ovšem musela by zohledňovat řešený problém. Jestli tento postup stojí za výpočetní čas, velmi záleží na konkrétní aplikaci EA. Ovšem existují obecná doporučení, která se touto otázkou zabývají.

Ukončovací podmínka

Rozlišme dvě varianty vhodné ukončovací podmínky. Pokud řešený problém má známou optimální hodnotu fitness, potom v ideální případě ukončovací podmínkou je řešení s touto fitness. Pokud jsme si vědomi, že náš model oproti modelovanému prostředí obsahuje nutná zjednodušení nebo by se v něm mohly vyskytovat nežádoucí šumy, lze se spokojit s řešením, které dosáhlo optima fitness s přesností $\epsilon > 0$. EA díky své stochastičnosti většinou nemohou garantovat dosažení takového hodnoty fitness, takže ukončovací podmínka by nemusela být nikdy splněna a algoritmus by běžel věky. Kdybychom vyžadovali konečnost algoritmu, musíme rozšířit ukončovací podmínku. Zatím-to účelem se nejběžněji používají následující rozšíření:

1. Byl překročen čas vyhrazený pro počítání na CPU
2. Celkový počet evaluací fitness přesáhl svůj předem daný limit
3. Zlepšení fitness v dané časovém bloku (měřenému počtem generací, či evaluací) klesla pod přípustný práh
4. Rozmanitost v populaci nedosahuje předem určených hodnot

Ukončovací podmínka bývá prakticky disjunkce následujících tvrzení: dosáhli jsme optima or podmínka X ze seznamu. Může se stát, že optimum známé není. Pak se používá pouze zmíněných podmínek nebo se smíříme s nekonečností algoritmu.

1.4 Diferenciální evoluce

Diferenciální evoluce (DE) se objevila v roce 1995, kdy Storn a Price vydali technický report Storn a Price (1997). Tento evoluční algoritmus popíše trochu více, protože je jedním z klíčových algoritmů mého řešení.

Charakteristické vlastnosti

DE dostala své jméno hlavně díky změnám obvyklých reprodukčních operátorů v EA. Diferenciální mutace, jak jsou mutace v DE nazývány, z dané populace kandidátů vektorů v \mathbb{R}^n vzniká nový mutant \bar{x}' přičtením pertubačního vektoru (perturbation vector) k existujícímu kandidátovi.

$$\bar{x}' = \bar{x} + \bar{p}$$

Kde pertubační vektor \bar{p} je vektor rozdílů dvou náhodně zvolených kandidátů z populace $\bar{y} a \bar{z}$.

$$\bar{p} = F(\bar{y} - \bar{z})$$

Faktor škálování $F > 0$, $F \in \mathbb{R}$, který kontroluje míru mutace v populaci. Jako rekombinační operátor v DE slouží uniformní křížení, pravděpodobnost křížení je dána $C_r \in [0,1]$, definuje šanci jakékoli pozice v rodiči, že odpovídající alela potomka bude brána z prvního rodiče. DE také upravuje křížení, neboť jedna náhodná alela je brána vždy z prvního rodiče, aby nedocházelo k duplikaci druhého rodiče.

V hlavních implementacích DE reprezentují populace spíše listy, odpovídají lépe než množiny, umožňují referenci i-tého jedince podle pozice $i \in 1, \dots, \mu$ v listu. Pořadí kandidátů v této populaci $P = < \bar{x}_1 \dots \bar{x}_i \dots \bar{x}_\mu >$ není závislé na hodnotě fitness. Evoluční cyklus začíná vytvořením populace vektorů mutantů $M = < \bar{v}_1 \dots \bar{v}_\mu >$. Pro každého nového mutantu \bar{v}_i jsou zvoleny 3 vektory z P , base vektor a dva definující pertubační vektor. Po vytvoření populace mutantů V , pokračujeme tvorbou populace $T = < \bar{u}_1, \dots, \bar{u}_\mu >$, kde \bar{u}_i je výsledek aplikace křížení \bar{u}_i je výsledek křížení \bar{v}_i a \bar{x}_i (všimněme si, že je zaručené, že nezreplikujeme \bar{x}_i). Jako poslední aplikujeme selekci na každý pář \bar{x}_i a \bar{u}_i a do další generace vybereme \bar{u}_i pokud $f(\bar{u}_i) \leq f(\bar{x}_i)$ jinak \bar{x}_i .

DE algoritmus upravují 3 parametry, škálovací faktor F , velikost populace μ (obvykle značen NP v DE literatuře) a pravděpodobnost křížení C_r . Na C_r lze také pohlížet jako na míru mutace, tzn. alela nebude převzata od mutantu.

Vlastnosti Diferenciální Evoluce:

Reprezentace:	vektor \mathbb{R}^n
Rekombinace:	uniformní křížení
Mutace:	diferenční mutace
Rodičovská selekce:	uniformní náhodné selekce 3 nezbytných vektorů
Selekce prostředí:	deterministická selekce elity (dítě vs rodič)

Varianty DE:

Během let, vzniklo a bylo publikováno mnoho variant DE. Jedna z modifikací zahrnuje možnost base vektoru, když vytváříme mutantské populace M. Může být náhodně zvolen pro každé v_i , jak bylo řečeno, ale také lze využít jen nejlepšího vektoru a nechat změny na pertubačním vektoru. Další možnost otevírá použití více pertubačních vektorů v mutačním operátoru. Což by vypadalo následovně:

$$\bar{p} = F(\bar{y} - \bar{z} + \bar{y}' - \bar{z}'), \text{ kde } \bar{y}, \bar{z}, \bar{y}', \bar{z}' \text{ jsou náhodně vybrány z původní populace.}$$

Abychom odlišili různé varianty, používá se následující notace DE/a/b/c, kde a specifikuje base *vektor(rand, best)*, b specifikuje počet diferenčních vektorů při vzniku pertubačního vektoru, c značí schéma křížení (bin=uniformní). Takže podle této notace by základní verze DE byla zapsána takto: DE/rand/1/bin.

1.5 Evoluční strategie

Evoluční strategie (evolution strategies) byly představeny na počátku 60. let pány Rechenbergem a Schwefelem Beyer a Schwefel (2002), oba jmenovaní pánové se zabývali optimalizací tvarů křídel na Technické univerzitě v Berlíně. Jedná se

o evoluční algoritmu cílící na optimalizaci vektorů reálných čísel. Objevili si dvě schémata (1+1) a (1, 1). Starší (1+1) (one plus one ES) vytvářejí potomka mutací a to přičtením náhodných nezávislých hodnot ke složkám rodičovského vektoru, následně je potomek přijat, pokud získal lepší fitness než jeho rodič. Jako další vznikl (1,1) (one comma one ES), které neimplementovalo elitismus, tzn. měnila vždy rodiče potomkem. Mutační funkce bere náhodná čísla z Normálního rozdělení s se střední hodnotou 0 a odchylkou σ . Pro velikost mutačního operátoru (mutation step size) se také používá symbol: σ . Vylepšení původního algoritmu bylo představeno v 70. letech, jedná se o koncept multisložkových ES, které se skládají ze μ jedinců (velikost populace) a λ jedinců vygenerovaných během jednoho cyklu. Opět tento koncept existuje ve dvou verzích (μ, λ) a $(\mu + \lambda)$. Zatímco běžné rekombinační schéma zahrnuje dva rodiče a jednoho potomka, intermediate křížení, které se u ES používá, zprůměruje hodnoty z rodičovských alel. Tímto způsobem můžeme používat i rekombinační operátory s použitím více než dvou rodičů, tyto operátory se nazývají globální rekombinace. Konkrétně se na potomkovi podílí λ rodičů. V praxi se preferuje (μ, λ) před $(\mu + \lambda)$, protože zahazuje všechny rodiče, tím pádem se snadno nezastaví na lokálním optimu, (μ, λ) se zvládá lépe adaptovat i při hledání pohyblivého optima. Pro typické použití se používá poměr 1:4 a 1:7.

Vlastnosti Evolučních strategií:

Reprezentace:	vektor \mathbb{R}^n
Rekombinace:	intermediary křížení
Mutace:	přičítání náhodných hodnot z norm. rozdělení
Rodičovská selekce:	uniformní náhodná
Selekce prostředí:	(sigma, lambda) nebo (sigma + lambda)

Následuje krátký jednoduchý příklad ES v jazyku Pythonu, pro lepší pochopení jsem použil kód ze stránky: <https://blog.openai.com/evolution-strategies> (2018). Cílem tohoto velmi jednoduché programu je nalézt vektor čísel, který se co nejvíce blíží vektoru solution.

```

solution = np.array([0.5, 0.1, -0.3])
def f(w): return -np.sum((w - solution)**2)

npop = 50
sigma = 0.1
# learning rate
alpha = 0.001
w = np.random.randn(3)
for i in range(300):
    N = np.random.randn(npop, 3)
    R = np.zeros(npop)
    for j in range(npop):
        w_try = w + sigma*N[j]
        R[j] = f(w_try)

    A = (R - np.mean(R)) / np.std(R)
    w = w + alpha/(npop*sigma) * np.dot(N.T, A)

```

2. Robotický Swarm

V češtině se také používá výraz Rojová Robotika nebo Robotický Roj, v angličtině je známý pod pojmem Swarm Robotics. Myšlenka Robotického Swarmu pochází podobně jako u Genetických Algoritmů z inspirace matkou Přírodou. Podle souhrnu Tan a Zheng (2013) popíší základní myšlenku RS.

2.1 Základní vlastnosti

Motivací pro použití RS může být chování živočichů na Zemi. Zaměříme se na skupiny živočichů jako jsou mravenci, včely, ryby dokonce i některé savce. Pokud bychom vložili do prostředí jednotlivce z některé ze zmíněných skupin, nebude schopen konkurovat nepřátelskému prostředí a nejspíše příliš dlouho nepřežije. Na druhou stranu, když budeme uvažovat celé společenství, tak se nám ze slabého jedince stane velmi adaptivní, odolný a rychle se vyvýjející roj. Podobnému účinku bychom se chtěli přiblížit v RS. Pro relativně jednoduchého robota, který není schopen plnit obtížný úkol, se pokusíme použít vícero robotů stejného typu, kteří společně zadáný úkol vyřeší. Navíc chceme těžit ze všech výhod hejna.

Jako nejčastější výhody RS oproti jednomu robotovi se nejčastěji uvádějí:

1. Paralelita - Díky malé ceně jedince, si můžeme dovolit velkou populaci jedinců. Malou cenou jedince v ES myslíme, že se jedná o jednoduchého robota s nízkou pořizovací cenou. V kontextu živočichů můžeme uvažovat množství energie, jídla pro tvorbu takového jedince. Velká populace nám umožňuje řešit vícero úkolů naráz, také na velké ploše. Zvláště pro vyhledávací úkoly ušetříme nemalé množství času.
2. Škálovnatelnost - Změna velikosti populace hejna neovlivní chování ostatních jedinců. Samozřejmě plnění úkolu bude rychlejší či pomalejší, ale původní hejno bude stále plnit původní úkol. Tím pádem můžeme celkem snadno upravovat velikost populace bez větší obtíží. V přírodě můžeme pozorovat, že smrt jednotlivých mravenců-dělníků znatelně neovlivní práci celého mraveniště. Nově narození mravenci se mohou vydat do práce, zatímco zbytek mraveniště nemění činnost.
3. Houževnatost - Související se škálovatelností, jen v tomto případě máme na mysli necílenou změnu populace. Jako v předchozím příkladu, u smrti mravenců, část robotů ES může selhat z rozličných důvodů, zbytek hejna však bude pokračovat k cíli, i když ve výsledku jim bude jeho dosáhnutí trvat o něco déle. Což se nám může vyplatit v nebezpečných prostředích.
4. Ekonomické výhody - Cena návrhu a konstrukce jednoduchých hejn robotů vyjde většinou levněji než jeden specializovaný robot schopný uspokojit stejné požadavky. V dnešním světě vychází výroba ve velkém množství mnohem levněji než tvorba jednoho drahého konkrétního robota.
5. Úspora energie - Díky menší velikosti a složitosti jednotlivých robotů vyžadují mnohem menší množství energie. Což má za důsledek, že si u nich můžeme dovolit energetickou rezervu na delší čas. Navíc když je pořizovací

cena jednoho robota menší než náklady na dobití, můžeme díky škálovatelnosti pouze připojit nové roboty, což u drahého robota jde málokdy.

6. Autonomie a Decentralizace - V kontextu RS musí každý jedinec hejna jednat autonomně, jedinci nejsou řízeny žádnou autoritou. Takže umí pracuje i při ztrátě komunikace. Opět se vychází z chování živých organismů. Pokud se chovají jedinci hejna dostatečně kooperativně, mohou pracovat bez centrálního řízení, důsledkem toho se stává celé hejno ještě flexibilnější a odolnější, hlavně v prostředích s omezenou komunikací. Navíc hejno mnohem rychleji reaguje na změny.

Mimo RS existuje i řada jiných přístupů, které se inspirovaly životem hejn v přírodě. Občas jsou zaměňovány za RS, nejčastěji se jedná o multi-agentní systémy a senzorové sítě (sensor networks). V následující tabulce jsou popsány jejich nejklíčovější vlastnosti.

	Robotická hejna	Multi-robotické systémy	Senzorové sítě	Multi-agentní systémy
Velikost populace	Variabilní ve velkém rozsahu	Malá	Fixní	V malém rozsahu
Rízení	decentralizované a autonomní	centralizované	centralizované	centralizované
Odlíšnosti	většinou homogenní	většinou heterogenní	homogenní	homogenní nebo heterogenní
Flexibilita	vysoká	nízká	nízká	střední
Škálovnatelnos	vysoká	nízká	střední	střední
Prostředí	neznámé	známé nebo neznámé	známé	známé
Pohyblivost	ano	ano	ne	vyjímceně

Tabulka 2.1: Porovnání systémů s více agenty

Také se liší svojí aplikací Robotická hejna se nejčastěji používají v vojenských, nebezpečných úkolech a také pro řešení ekologických katastrof. Multi-robotické systémy potkáváme v transportních, skenovacích úkolech, dále pro řízení robotických fotbalových hráčů. Oproti tomu nejčetnější využití senzorových sítí zasahuje do medicínské oblasti, ochrany životního prostředí. Multi-agentní systémy zase nejvíce zasahují do řízení síťových zdrojů a distribuovaného řízení.

2.2 Použití

Existuje několik vědeckých prací, které studují a navrhují použití RS v reálném nasazení.

Některé jsem zmínil už v úvodu této práce, jako například hasičům asistující roboty (Penders a kol., 2011). Kde si robotické hejno klade za cíl usnadnit a podpořit navigaci lidem v nebezpečném prostředí. Jejich využití je ilustrováno záchranou misí ve velkém skladišti. Hasiči mají díky kouři velmi omezenou viditelnost. Tím pádem se lokalizace přeživších, epicenter požáru a další důležitých bodů stává obtížným a zdraví ohrožujícím úkolem. Robotické hejno tedy může prozkoumat celý prostor před vlastním zásahem. Při zásahu ještě asistovat hasičům při orientaci v prostoru. Nezřídka kdy se stává, že zasahující hasič zahyne, protože se v hustém dýmu v objektu ztratil.

Robotická hejna se také ukázala jako užitečná u ekologický pohrom. Španělští vědci testovali jejich použití při úniku ropy (Aznar a kol., 2014), či při hledání centra radiace (Bashyal a Venayagamoorthy, 2008).

V prvně zmíněném příkladu autoři mapovat znečištění mořské vody. Dokonce při plavbách bez defektů se do moře uvolňuje palivo, ropa a další nebezpečné látky. Očekává se, že s rostoucí námořnickou dopravou, rozrostou se tyto lokální znečištění v vážnou hrozbu. Aktuální systémy monitorující znečištění při katastrofách tankerů jsou pro toto využití příliš drahé. Autoři proto navrhují použití hejno dronů, které bude dokumentovat velké vodní plochy a bude schopno odhalovat případné nebezpečí a větší koncentrace cizích látek. Dokonce budou moci na základě získaných informací sledovat znečistovatele.

Po jaderné katastrofě se stává explorace zamořené oblasti v podstatě nemožným úkolem pro lidské průzkumníky. Právě monitorování radiací postiženým územím se stala motivací pro práci (Bashyal a Venayagamoorthy, 2008). Ve zmiňované práci se autoři soustředí na porovnávání autonomních robotických hejn a RH komunikující s člověkem. V rámci výsledků ukazují, výhody použití robotického hejna pro hledání centra radiace a také že RH interagující s lidmi dosahují lepších výsledků.

Některé neskončili u simulací a také využívali RS u fyzických robotů. Hlavní motivací pro tuto práci byl fakt, že většinou se řízení RS vytváří a hlavně testuje pouze v uzavřeném a simulovaném prostředí. Tvůrci se rozhodli pro reálné použití na moři, kde nemohou prostředí jakkoli ovládat či kontrolovat. Snaží se tím ukázat, že i přes šumy a neočekávané situace, RS je stabilní a použitelné pro aplikaci ve skutečném světě. Celé hejno se skládalo z deseti robotů. Jednalo se o malé lodičky s délkou přibližně 60 cm a nízkou pořizovací cenou okolo 300 eur. Každý robot byl vybaven GPS, WIFI, kompasem. Chování bylo připraveno pomocí evolučních algoritmů v simulaci, konkrétně autoři používají neuroevoluci NEAT. Simulace obsahovala 4 podúkoly: navádění, shlukování, rozptylování a monitorování prostředí. Poté bylo stvořené řízení ohodnoceno na vodní ploše. Prezentované výsledky vypadají velmi slibně, chování a úspěšnost řízení se velmi blíží mezi simulací a reálným nasazením. Také potvrzují přítomnost výhodných vlastností ze simulaci v reálném světě, jedná se o robustnost, flexibilitu, škálovatelnost. V neposlední řadě také úspěšnost skládání jednoduchých podúkolů do komplexního chování v rámci hejna, které řeší složitý hlavní cíl. (Duarte a kol., 2016).

Rozšířit o další příklady použití?

2.3 Řízení robotických swarmů

Chování swarmů se řadí mezi velmi obtížné úkoly pro svět informatiky. Pro reprezentaci chování se využívá *neuronových sítí*, které se optimalizují pomocí nastavování vah jednotlivých perceptronů, neboť se jedná o velký prostor vstupních informací ze senzorů a prostor pro interakci s prostředím je taktéž velmi rozsáhlý. Přímé prohledávání takto obřího prostoru nepřichází v úvahu, proto v poslední době získávají na oblibě evoluční algoritmy. Mezi nejvíce používané patří evoluční strategie, či genetické programování.

2.3.1 Genetické programování a stromy chování

V práci "Evolving behaviour trees for Swarm robotics" Jones a kol. Alan F T Winfield s kolegy podporují myšlenku evoluce chování v hejnové robotice. Pro řízení navrhují vcelku zajímavé využití stromů chování (SC)(Behaviour tree), které

mají uplatnění především v herním průmyslu pro chování charakterů, které nejsou ovládány hráčem. Jako optimalizační algoritmus zvolili Genetické programování.

Strom chování je strom, jehož listy interagují s prostředím, vnitřní vrcholy spojují tyto akce dohromady a tvoří rozhodovací a závislostní pravidla. Celý strom je vyhodnocován v pravidelných intervalech, v práci se značí jako *tick*. Opírají se o článek Shoulson a kol. (2011), kde bylo ukázáno, že SC může plnohodnotně reprezentovat konečné automaty, dokonce i když budeme používat pravděpodobnosti konečné automaty. Jako jedince z hejna zvolili Kilobot, které byl představen Rubenstein v Rubenstein a kol. (2014). Kilobot se pohybuje pomocí dvou vibračních motorů, komunikují přes infracervený kanál, v prostředí se orientují pomocí foto detektoru a signalizují pomocí LED diod s barevným spektrem. Následující části zjednoduší komunikaci s efektory robota a nad nimi optimalizováno SC.

Efektor/Senzor	Read or Write	Popis
motor	W	vypnut, vřed, vlevo, vpravo
přídavná paměť	R&W	libovolná hodnota
vysílač signálu	R&W	vysílá při větší hodnotě než 0.5
přijímač signálu	R	1 pokud přijímá signál
detektor potravy	R	1 pokud světelný snímač vidí potravu dno
nosič jídla	R	1 pokud nese jídlo
hustota robotů	R	hustota Kilobotů v blízkosti
$\delta_{hustota}$	R	změna v hustotě
$\delta_{vzdálenost_{potrava}}$	R	změna ve vzdálenosti k potravě
$\delta_{vzdálenost_{hnízdo}}$	R	změna ve vzdálenosti k hnizdu

Tabulka 2.2: Parameterizing behavior trees, Motion in Games - podoba stromů

Tyto akce pak odpovídají listům v SC, vnitřní vrcholy mohou být kompoziční: *seqm*, *selm*, *probm* a tyto mohou mít 2 až 4 syny. Informace procházející mezi vrcholy mohou být následujícího druhu *success*, *failure*, *running*. Zpracovávají informace následujícím způsobem posílají tik do každého syna dokud od nějakého nepřijde hodnota *failure* nebo tik proběhne na všech synech. Pokud proběhne úspěšně tik u všech synů vrací *success*, *failure* jinak. Oproti tomu *selm* vysílá tik, dokud mu nějaký syn nevrátí hodnotu *success* nebo všichni synové provedli tik, pokud se nevrátí jediná hodnota *success*, tak vrací *failure*, v opačném případě *success*. Od obou se liší *probm*, ten s danou pravděpodobností vybere jednoho syna a vrátí jeho odpověď. Vrchol, který má alespoň jednoho syna se statusem *running* vrací stejnou hodnotu.

Vrcholy jen s jedním synem patří do jedné z následujících kategorií: *repeat*, *successd*, *failured*. Vrchol *repeat* vrací tik svým synům, s daným počtem pokusů, dokud nedostane hodnotu *success*. Následující dva vrcholy vrací konstantní odpověď na tik dle svého jména, i přesto pošlou tik svému následníkovi.

Poslední skupinu vrcholu tvoří tzv. akční vrcholy *ml*, *mr*, *mf*, což ve stejném pořadí jsou: zatoč vlevo, zatoč vpravo, jed kupředu. Vrcholy pohybu při první tiku vrací *running* při druhém *success*. K akčním vrcholům také patří *if*, který implementuje porovnávání synů a vrací *success*, pokud porovnání platí. Poslední z akčních vrcholů je *set*, který nastavuje danou hodnotu synům.

V prostředí jsou s konstantní frekvencí prováděny tzv. update cykly. Jeden takový cyklus se skládá z:

1. Spočítají se hodnoty v synech z vysílaných signálů a prostředí.
2. Na SC je proveden tik, což čte a zapisuje hodnoty do synů.
3. Pohybové motory jsou aktivovány a vysílání je upraveno, oboje dle zapsaných hodnot do synů.

Jako zátěžový test používají obvyklý scénář, který spočívá v hledání potravy a její odvážení zpět do hnízda. Fitness se hodnotí podle vzdálenosti doneseného jídla, čím blíže k hnizdu tím lépe. Jako optimalizační metodu autoři vybrali genetické programování a používají DEAP knihovnu Fortin a kol. (2012). Celá populace velikosti n_{pop} je ohodnocena fitness, každý jedinec se hodnotí podle 10 simulací, každá simulace má jinou startovní konfiguraci. Roboti startují vždy na poli o velikosti 5x5, jejich orientace je vybírána náhodně. Běží 300 simulovaných sekund, frekvence update cyklu 8Hz pro vnímání prostředí a u ovladačů s 2Hz. Používané Genetické programování implementuje elitismus přenáší n_{elite} nejlepších jedinců do další generace, zatímco zbylá část je zvolena pomocí turnajovou selekcí s velikostí t_{size} . A křížící(rekombinační) operátor, který kříží části stromů, je aplikován s pravděpodobností p_{xover} na všechny páry vybrané turnajovou selekcí. Na vzniklé páry se aplikují 3 mutační operátory.

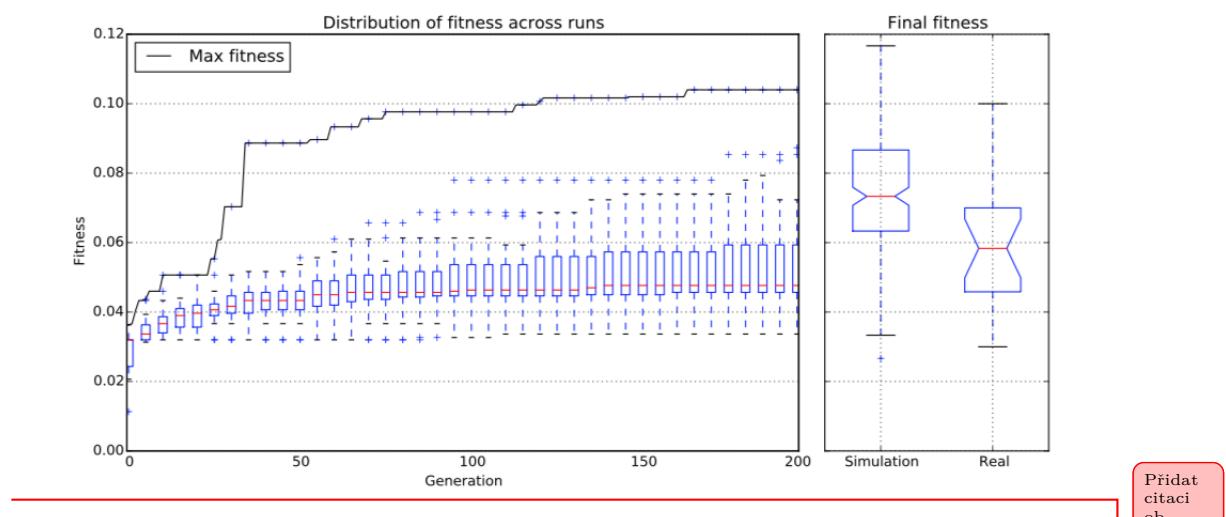
1. S pravděpodobností p_{mutu} je náhodný vrchol stromu vyměněn za nový náhodně vytvořený
2. S pravděpodobností p_{muts} je náhodná větev stromu a je vyměněna za náhodně zvolený terminál(vyskytující se na této větvi)
3. S pravděpodobností p_{mutn} je náhodný vrchol vyměněn za nový, ale se stejným počtem argumentů
4. S pravděpodobností p_{mute} je náhodná konstanta vyměněna za jinou náhodnou hodnotu

Krom simulace 25 nezávislých běhů evoluce. Také byly otestovány algoritmy na reálných strojích, vytrénované chování bylo otestováno 20 běhy s rozdílnou startovací pozicí a ohodnocení stejnou fitness.

Výsledky simulace byly více než uspokojivé v simulační části si hejno vedlo o trochu lépe 0.075 z maximální hodnoty 0.12(minimum 0) a co se týče reálného nasazení vygenerovaného chování 0.058. Což opravdu není velký rozdíl, když přihlédneme k tomu, že evoluce probíhala čistě na simulační rovině. Následující graf zachycuje výslednou fitness.

Konkrétní nastavení parametrů		
Parametr	Hodnota	Popis
n_{gen}	200	Generace
t_{test}	300	Délka testu ve vteřinách
n_{pop}	25	Velikost populace
n_{elite}	3	Velikost Elity
p_{xover}	0.8	Pravděpodobnost křížení
p_{mutu}	0.05	Pravděpodobnost výměny podstromu
p_{muts}	0.1	Pravděpodobnost sevrknutí podstromu
p_{mutn}	0.5	Pravděpodobnost výměny vrcholu
p_{mute}	0.5	Pravděpodobnost výměny konstanty

Tabulka 2.3: Parameterizing behavior trees, Motion in Games - nastavení parameterů

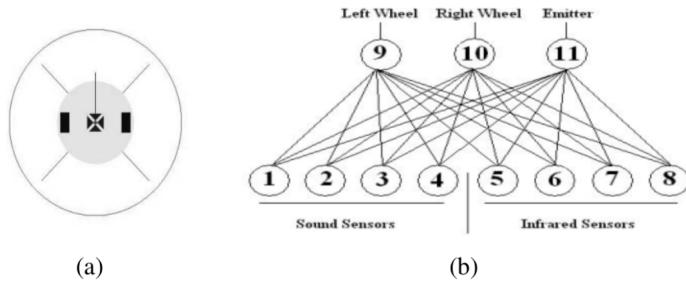


2.3.2 Genetické algoritmy a neuronová síť

Cagri a Yalcin používají ve své práci (Yalcin, 2008) práci neuronových sítí místo SC a pro nastavení vah Genetického algoritmu. Shodují se s Winfieldem a jeho kolegy, že evoluce mnoha chování robotického hejna přináší zajímavých strategií, která mohou být mnohem komplexnější než explicitně vytvořené chování. Popisují však také obtížnosti použití Evoluce, zvláště volbu Evolučního algoritmu a efektivnost celého výpočtu.

Využívají již existujícího simulátoru Cobot2D, pro všechny experimenty bylo použita mapa bez překážek velikosti 400x400. Roboti se pohybují pomocí dvoukolečkových motorů, orientují se 4 infračervenými senzory a 4 zvukovo-směrovými senzory, v jejich středu je umístěn vše směrový zvukový vysílač. Vysílače mají pevný rozsah kruhu vysílání a dynamickou sílu. Zvukové senzory se rozhodují pouze podle signálů, jejichž vysílače spadají do 90° výseče od senzoru a také jejich vzdálenost musí být menší než daná konstanta. Síla signálu se zmenšuje směrem od vysílače a senzory vrací součet sil signálů. Infračervené senzory úsečku dané velikosti a orientace vzhledem k robotovi, a vrací vzdálenost k nejbližšímu průsečíku. V rámci simulace jsou generovány náhodné šumy pro každou interakci s prostředím, aby se simulace přiblížila, co nejvíce reálnému nasazení. Pro ovládání robota byla navržena neuronová síť, která má 8 vstupů (4 pro infra-senzory a 4

pro zvukové) a 3 výstupy a není zde žádná skrytá vrstva. Nákres robota a jeho ovládací neuronové sítě můžete vidět na obrázku.



Genetický algoritmus, jak jej nazývají, má následující podobu.

Genetický algoritmus:

1. Inicializuj populace P 100 jedinců a každý jedinec reprezentován váhovým vektorem
2. Nastav všem váhovým vektorům z populace náhodné floating point hodnoty v intervalu $\in (-1,1)$
3. $G_{current}$ nastav na 0(id generace)
4. Dokud $G_{current} < 100$ Prováděj
 - (a) Pro každý vektor $w \in P$
 - i. Vytvoř 5 simulací prostředí s 10 roboty a orientovanými náhodně
 - ii. Přiřaď každému robotu jako ovladač neuronovou síť s váhami w
 - iii. Spust simulaci pro 5000 kroků
 - iv. Každé simulaci spočítej fitness pro skupinku robotů
 - v. Průměr z vypočítaných fitness z předchozího kroku přiřaď jako fitness vektoru w
 - (b) Setříd vektory podle jejich fitness
 - (c) Zvol nejlepších 20 jako elitu P_e
 - (d) Zvol náhodně 80 vektorů P_c a aplikuj na ně křížení(prohazuje 1/3 vektoru a shodné páry)
 - (e) Zvol náhodně 40 vektorů z P_c a aplikuj mutační operátor(přičtení náhodného čísla z $(-1,1)$)
 - (f) Zvol $P = P_c \cup P_e$
 - (g) $G_{current} += 1$
5. Vrat jedince, který má z P největší fitness

Fitness Funkce: První použitá $fitness_1$ z tohoto experimentu, obrácená hodnota průměrné vzdálenosti do středu robotické skupiny.

Přidat citaci obrazku, případně vyndat

Todle je asi špatně

$$fitness_1 = \frac{1}{n \sum_{r=1}^n d_{rc}}^{-1}$$

Kde n je počet robotů v robotické skupině, r je robotův index, d_{rc} je euklidovská vzdálenost mezi r středem robotické skupiny c .

Druhá $fitness_2$ používá metodu *inverse of hierarchical social entropy* (Balch, 2000). Tato metoda počítá kompaktnost skupiny, tím že hledá každý možnou skupinku(cluster) pomocí změn maximální vzdálenosti h mezi jedinci ze stejného clusteru. Přidávají ještě rozšíření od *Shannon's information entropy*, jenž používá pevné h . Toto rozšíření je definováno:

$$H(h) = - \sum_{k=1}^M p_k \log_2(p_k)$$

H se nazývá entropie, p_k je proporce jedince z clusteru k , M je počet clusterů pro dané h . Konečně celý předpis daný Balchem vypadá následovně:

$$fitness_2 = \int_0^\infty \frac{1}{H(h)dh}$$

Použití neuronových sítí a genetického algoritmu se ve výsledku ukázalo jako vhodný prostředek pro učení robotického hejna, neboť se vygenerované chování se obstojně shlukuje do úzkých skupin. Definují další tzv. cost funkci pro měření úspěchu nalezených chování, aby mohli porovnat funkce fitness. A $fitness_2$ se ukazuje jako účinější.

2.3.3 Evoluční strategie a neuronová síť

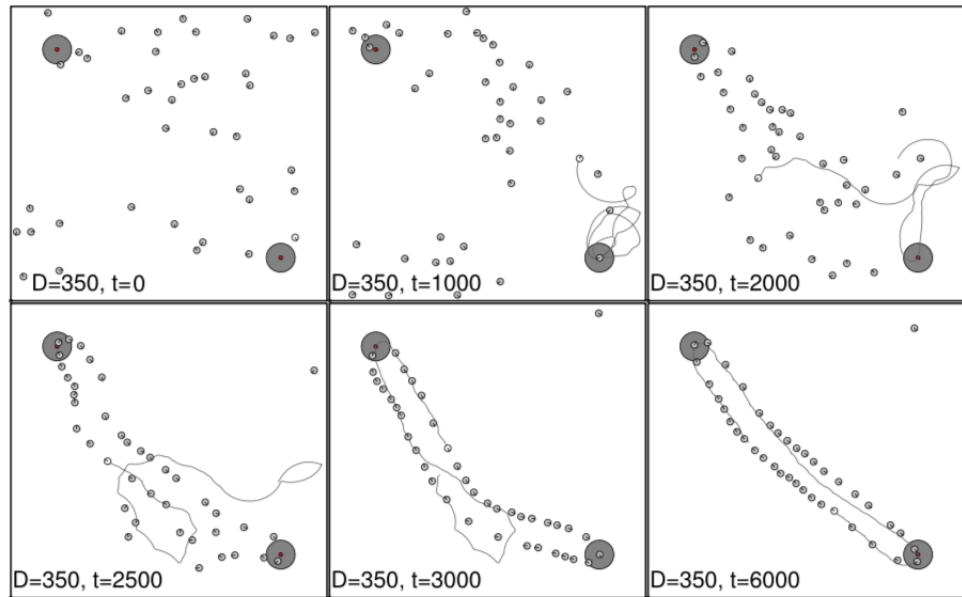
V článku *Self-organised path formation in a swarm of robots* (Sperati a kol., 2011) aplikují pro řízení robotických hejn Evoluční strategie. Jako cíl si článek klade problém průzkumu a navigace v neznámém prostředí v kontextu robotických hejn. Experiment, který měl otestovat uvedené vlastnosti robotického hejna, spočíval v co nejrychlejším přesunu celého hejna mezi dvěma prostory v neznámém prostředí.

Pro simulaci bylo využito upravené verze OS Evorobota a jako model jedince z hejna e-puck robot (Mondada a kol., 2009). Tento robot se pohybuje pomocí dvoukoleček, má 8 infračervených senzorů, navíc jeden infračervený senzor na povrch a jeden rozpoznávací barvy vpředu (, v tomto případě černobílé prostředí). Navíc mu byla přidělána LED vpředu s modrou barvou a červenou vzadu, která může zapínat a vypínat dle potřeby, a také má snímač barev na vrchu.

Pro ovládání robota zvolili autoři neuronovou síť se 13 vstupy (8 infračervené senzory, 1 binární podlahový senzor (bílá vs. černá), 4 binární vizuální snímače), dále 3 skryté neurony a 4 výstupní neurony (2 ovládající kolečka, 2 aktivující přední a zadní led). Formou jsou podobné předchozím modelům robotů.

Fitness funkce je vyhodnocena po nasazení do robotů a provedení simulace, vlastní fitness je pak průměr z 15 běhů. Ve vyznačených místech se roboti nabíjí, což trvá daný čas a roboti s lepší efektivitou přesunů z jednoho místa do druhého stihnout cestu tam a zpět mnohem rychleji.

Výsledky prokazatelně ukazují úspěšné použití Evolučních strategií na optimalizaci chování robotického hejna. Pro většinu prostředí dokázali najít efektivní řešení a jak lze vidět na nákresech, dráhy se optimalizují ve dvousměrnou cestu pro přesun z A do B.



Přidat
citaci
ob-
rázku,
pří-
padně
vyndat

3. Simulátor

Udělat výcuc z dokumentace a připravit spouštění jednoduchým scriptem

Zmínit optimalizaci grid map a parallelní zpracování

4. Experimenty

Všechny práce zmíněné v úvodní kapitole, sice používají evoluční algoritmy k vytvoření řízení chování homogenního robotického hejna, tzn. s jedním druhem robotů. V následující kapitole podrobně popíší postup hledání optimální chování pro heterogenního hejna. Optimalizaci jsem navrhl a otestoval na třech rozličných scénářích. Hlavní motivací při tvorbě scénářů bylo vytvořit obtížnější úkoly než se obvykle používají jako například: shlukování, vyhýbání překážkám... Navrhnut je natolik komplexně, aby nebylo možné, že část hejna se nebude podílet na jeho plnění. Také jsem volil scénáře, aby se přiblížily situacím z reálného světa. Každému z nich jsem věnoval samostatnou kapitolu, která zahrnuje popis hlavního úkolu scénáře, seznam robotů i s jejich senzory a efektory, způsob hodnocení fitness, rozdělení do podúkolů s průběhem fitness u ES a DE, vizualizaci a rozbor chování nejlepšího jedince.

Pracovní názvy scénářů:

1. Wood Scene - zpracování dřeva
2. Mineral Scene - přetvoření minerálů na palivo
3. Competitive Scene - soubojový scénář

Pro řešení problému jsem navrhl řadu postupů, proto v tomto odstavci zmíním ty nejvíce přímočaré a slibné, které se ovšem ukázaly jako neúspěšné. V kapitolách zabývajícími se konkrétními scénáři už budu pouze popisovat jen konečné, úspěšné postupy.

Nedostatečný se ukázal pokus provádět evoluci pro fitness hlavního úkolu scénáře. Většina hodnocení náhodných chování byl roven nule, proto EA nedostaly dostatek informací k vhodné exploraci a díky malé pravděpodobnosti vygenerování chování alespoň částečně řešící hlavní úkol nedocházelo ani k exploataci. Což mělo za důsledek neefektivní DE a ES, takže ani jeden z EA nedošel k úspěšnému řešení.

Posun zaznamenala více obecná fitness i když sama o sobě také nebyla úspěšná. Do fitness jsem zahrnul i menší pozitivní znaky, které byly součástí hlavního úkolu. Například jsem záporně ohodnotil pokusy o pohyb končící kolizi, kladně počet nalezených entit či vhodných objektů v kontejnerech, aktuální stav paliva. Optimalizovaná chování opravdu zaznamenala posun. Ovšem oba EA obtížně hledaly cestu z lokálního optima a ve většině případů optimalizovali pouze jednoduché části úkolu. I přes přidávání složitějších matematických funkcí do fitness nebyly schopny dosáhnout uspokojivého řešení hlavního úkolu scénáře.

Pro finální řešení jsem zvolil metodu, kterou nazývám metodou podúkolů. U každého scénáře podrobně popíší její průběh a nastavení, zde pouze nastíním základní myšlenku. Rozdělil jsem hlavní cíl na několik menších podúkolů (metaúkolů). Každému z nich vytvoříme fitness funkci odpovídající nutné části hlavního cíle. Fitness metaúkolu jsem navrhoval, tak aby necílila na již optimalizované úkony a v každém podúkolu jsem se vždy soustředil pouze na jeden jednoduchý úkon. Díky tomuto principu jsem dosáhl mnohem vyšší odolnosti proti uvíznutí v lokálním minimu. Explorace se tímto procesem také zlepšila, protože hlavní cíl závisí

na podíkolech a pokud bylo chování rozmanité a úspěšné, přenesly se tyto vlastnosti i dále. Poté jsem generaci úspěšnou v průzkumu optimalizoval na sbírání materiálů pouze požadované barvy a takto jsem rozděloval až k finálnímu úkolu scénáře.

Každý robot má připojen paměťový slot, neboť roboti s nimi dosahovaly ve všech úkolech znatelně lepších výsledků a pomáhaly vytvářet složitější chování.

4.1 Použité technologie

4.1.1 Reprezentace Chování

Pro ovládání robotů jsem zvolil v poslední době velmi oblíbené neuronové sítě. Použil jsem jejich definici podle (Marsalli).

Definice 1 (Perceptron). *Perceptron je funkce $z \mathbb{R}^n \rightarrow \mathbb{R}$, která je dáná následujícím předpisem:*

$$Y = S(\Theta + \sum_{i=0}^n w_i x_i)$$

n : velikost vstupu

x_i : i -tý vstup perceptronu

w_i : váha i -tého vstupu

Θ : práh, (váha s konstatním vstupem 1)

$S(x)$: přenosová funkce

Y : výstup perceptronu

Jako aktivační funkce se mi nejvíce osvědčila často používaná funkce hyperbolického tangentu se změněným oborem hodnot pro konkrétní výstup.

Jelikož v mé práci bylo prioritou nalézt funkční řešení jednotlivých scénářů a nikoliv optimalizaci pro co nejlepší výsledek. Zvolil jsem jednoduchou architekturu jednovrstvé neuronové sítě, což se ukázalo jako dostatečné. Také pro každé reálné číslo, které očekává robot jako vstup pro efektor, byl připojen neuron do kterého vstupuje vektor reálných čísel odpovídající každé hodnotě ze senzorů. Pro ještě lepší řešení by zde bylo možné nasadit NEAT algoritmus či hledat více specifickější architektury, případně vyzkoušet vliv vícero vrstev.

Mám sem přidat obrázek funkce?

To úplně nevím, zda je pravda

Mohu to na- psat sem? nebo do diskuze

4.1.2 Evoluční algoritmy

Neuronovou sítě si lze představit jako množinu vektorů, kde jeden perceptron odpovídá vektoru reálných čísel (vah vstupů + práh $v = (x_0, x_1, \dots, x_n, \Theta)$). V kontextu evolučních algoritmů se pro optimalizaci vektorů reálných čísel nejčastěji používají ES a DE, I z tohoto důvodu jsem zvolil zmíněné algoritmy jako zástupce pro optimalizaci chování heterogenní skupiny robotů. Oba zmíněné algoritmy popisují v kapitole 1.4 a 1.5 a má implementace se od popisu v úvodu liší pouze v malý detailech. Do detailu jsou popsány v přiložené dokumentaci.

Nezapomenout přidat do docu

Nastavení parametrů

Parametr:	Hodnota:
F(DE):	0,8
CR(DE):	0,5
alpha(ES)	0,05
sigma(ES)	0,1

Tabulka 4.1: Nastavení parametrů u EA

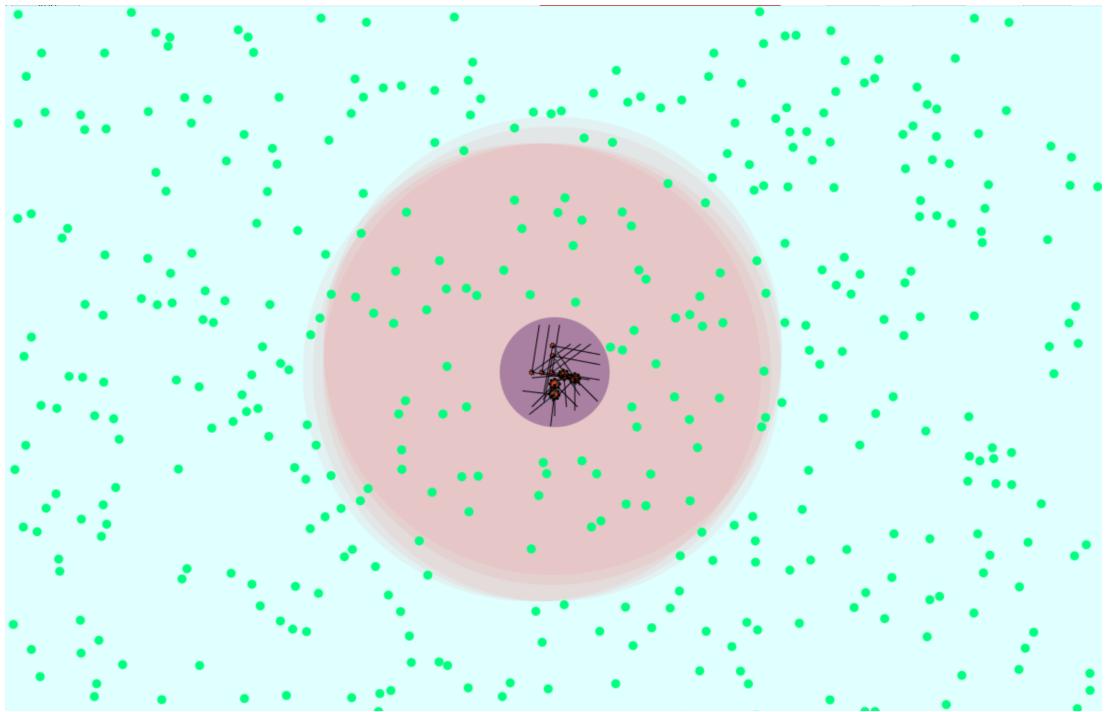
4.2 WoodScene experiment

4.2.1 Cíl Experimentu

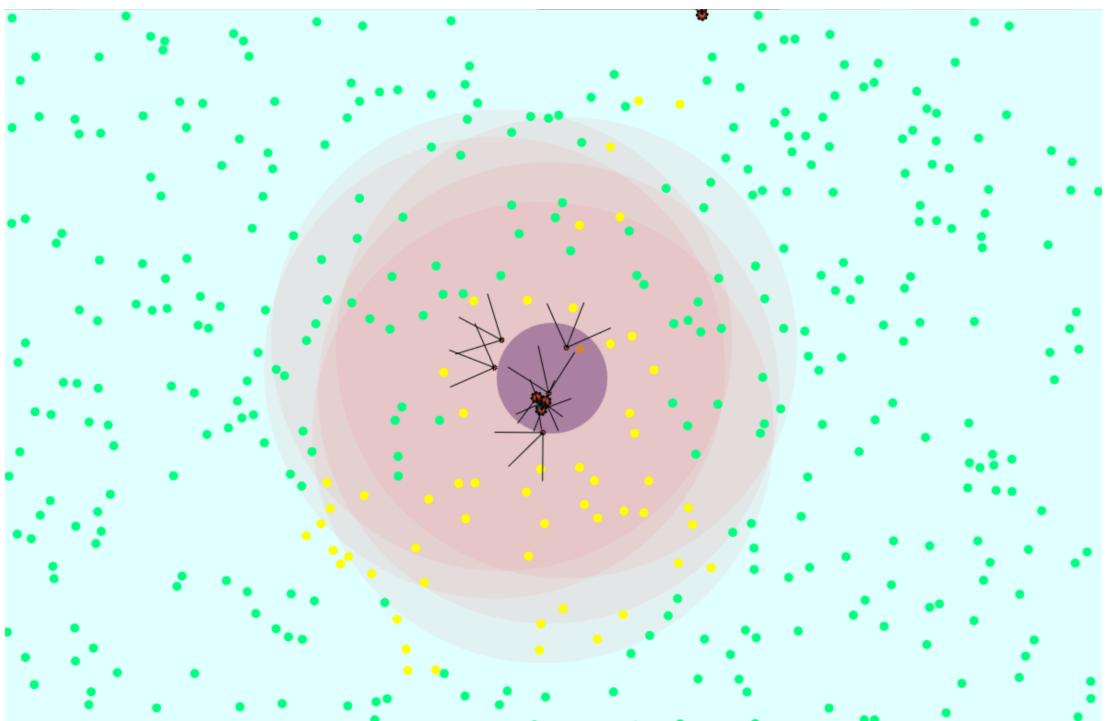
Tento scénář je analogií pro kácení lesa, kdy se roboti snaží maximalizovat množství zpracované dřevo na předem vyznačené ploše. Plocha pro skládání dřeva je označena rádiovým signálem s hodnotou signálu 2. V experimentu se dohromady celkem vyskytuje 9 robotů dvou různých druhů. Roboti jsou na začátku simulace umístěni uprostřed mapy na skládacím prostoru. Na náhodných pozicích jsou rozmištěny stromy. Scout roboti musí nejdříve strom nalézt a zpracovat, disponují totiž efektorem zpracovávající dřevo. Ovšem Scout roboti neuvezou žádné entity, proto Worker roboti mají za úkol zpracovaný strom naložit a odvézt do skládacího prostoru.

V rámci bakalářské práce byla připraven program zajišťující jednoduchou vizualizaci chování robotů. Jeho vizuální výstup můžete vidět na obrázku 4.1. Na ní si vysvětlíme jednotlivé entity nacházející se na mapě. Mapa je ohraničena čtvercovými hranicemi, které se chovají jako zed. Zelené kroužky znázorňují stromy, které ještě nebyly objeveny. Objevený strom změní barvu na žlutou. Modré označený prostor je určen pro uskladnění zpracovaného dřeva. Hnědé kolečka zastupují pokácené dřevo. V některých ech se objevuje dřevo už při inicializaci, proto ještě neobjevené má tmavší barvu a objevené světlejší. Pro roboty je v tomto prostoru vysílán rádiový signál. Roboti jsou vyplněny červenou barvou, jejich senzory a efektory mají černou barvu. Pro každý rádiový signál je určena jedna unikátní barva s alfa kanálem.

Pro potvrzení, že scénář není triviálně řešitelný. Bylo vygenerováno tisíc náhodných chování. Hodnoceny byly dle fitness funkce podúkolu kooperace popsaný níže. Nejlepší z nich můžete vidět těsně po inicializaci mapy 4.1. Výsledek krátce před 10 000 iterací je zachycen v obrázku 4.2



Obrázek 4.1: Příklad WoodScene mapy: start náhodného chování



Obrázek 4.2: Příklad WoodScene mapy: po 9000 iteracích náhodného chování

4.2.2 Roboti

Scout robot

Jedná se o robota, který má na starosti průzkum mapy a kácení nalezených stromů. Pro komunikaci s ostatními roboty má možnost vysílat rádiový signál s hodnotou 0. Oproti Worker robotovi je menší, rychlejší, jeho senzory mají větší dosah, navíc proti němu disponuje type senzorem a refaktorem nalezených stromů. Type senzor představuje formu radaru, říká robotovi s jakou četností se vyskytuje v dosahu senzoru. Refaktor reprezentuje techniku kácení mění strom na dřevo.

Scout Robot	
Tvar:	<i>Kruh</i>
Poloměr:	2,5
Název:	<i>WoodCutterM</i>
Velikost kontejner:	0
Efektor	
Motor:	<i>Dvoukolečkový</i>
Maximální rychlosť:	3
Kód rádiového signálu:	0
Poloměr signálu:	200
Refaktor:	<i>Strom ⇒ Dřevo</i>
Dosah refaktoru:	10
Počet paměťových slotů:	10
Obsah slotu:	<i>float</i>
Senzory	
Počet line senzorů:	3
Délka line senzorů:	50
Orientace line senzorů:	$0^\circ, 45^\circ, -45^\circ$
Poloměr type senzoru:	50
Poloměr rádiového přijímače:	100
Počet touch senzorů:	8
Lokátor senzor	

Tabulka 4.2: Wood - Scout robot popis

Worker robot

Worker robot se stará o transport objektů na mapě. Ke komunikaci využívá signálů s kódem 1. Sebrané objekty ukládá do kontejneru, kam se vejde 5 entit. Zvedání a pokládání probíhá skrze efektor Picker.

Worker Robot	
Tvar:	<i>Kruh</i>
Poloměr:	5
Název:	<i>WoodWorkerM</i>
Velikost kontejner:	5
Efektory	
Motor:	<i>Dvoukolečkový</i>
Maximální rychlosť:	2
Kód rádiového signálu:	0
Poloměr signálu:	200
Dosah pickeru:	10
Počet paměťových slotů:	10
Obsah slotu:	<i>float</i>
Senzory	
Počet line senzorů:	3
Délka line senzorů:	30
Orientace line senzorů:	$0^\circ, 45^\circ, -45^\circ$
Poloměr rádiového přijímače:	100
Počet touch senzorů:	8
Lokátorový senzor	

Tabulka 4.3: Wood - Worker robot popis

4.2.3 Vyhodnocování Fitness

Fitness funkce pro ohodnocení WoodScene scénáře probíhá vždy až na konci simulace. I když se úspěšnost v ech vždy posuzuje jinak, celou fitness funkci lze shrnout do následujícího cílů. Roboti jsou odměňováni za:

1. kolize = počet pokusů o pohyb při kterém by došlo ke kolizi
2. nalezené stromy = stromy o které zavadil line senzor
3. pokácené stromy = stromy, které refaktor změnil
4. sebrané dřevo = zpracované dřevo, které mají roboti uvnitř kontejnerů
5. uskladněné dřevo = dřevo, které dovezli na vyznačené místo

4.2.4 Podúkoly

Pro oba použité algoritmy jsem používal stejné úlohy pro naučení robotů postupně těžších a těžších cílů. Hlavně také, abych mohl porovnat oba evoluční algoritmy.

1. vygenerování robotů = Na začátku je vygenerováno chování robotů naprosto náhodně. Pro každého robota, je vygenerována náhodná jednovrstvá neuronová síť.
2. učení chůze = Pro oba roboty je velmi důležité, aby se pohybovali bez kolizí po celé mapě a objevovali, co největší prostor. Roboti jsou vyvíjeni odděleně a fitness se soustředí na počet kolizí(záporným ohodnocením) a na nalezené stromy(kladným ohodnocením).
3. těžba stromů = Scout roboty, kteří se už obstojně po mapě pohybují, je třeba naučit káct stromy. Proto dalším cílů ve fitness funkci na počet pokácených stromů. Nicméně stále také na počet stromů nalezených.
4. převoz dřeva = Správně pohybující chceme naučit sbírat vytěžené dřevo. Fitness hodnotí počet sebraného dřeva, případně i uskladněné dřevo. Na těchto mapách jsou už na začátku připraveny pouze entity zpracovaného dřeva.
5. kooperace = V posledním experimentu, se hodnotí pouze sebrané a uskladněné dřevo. A evolvují se oba druhy robotů současně.

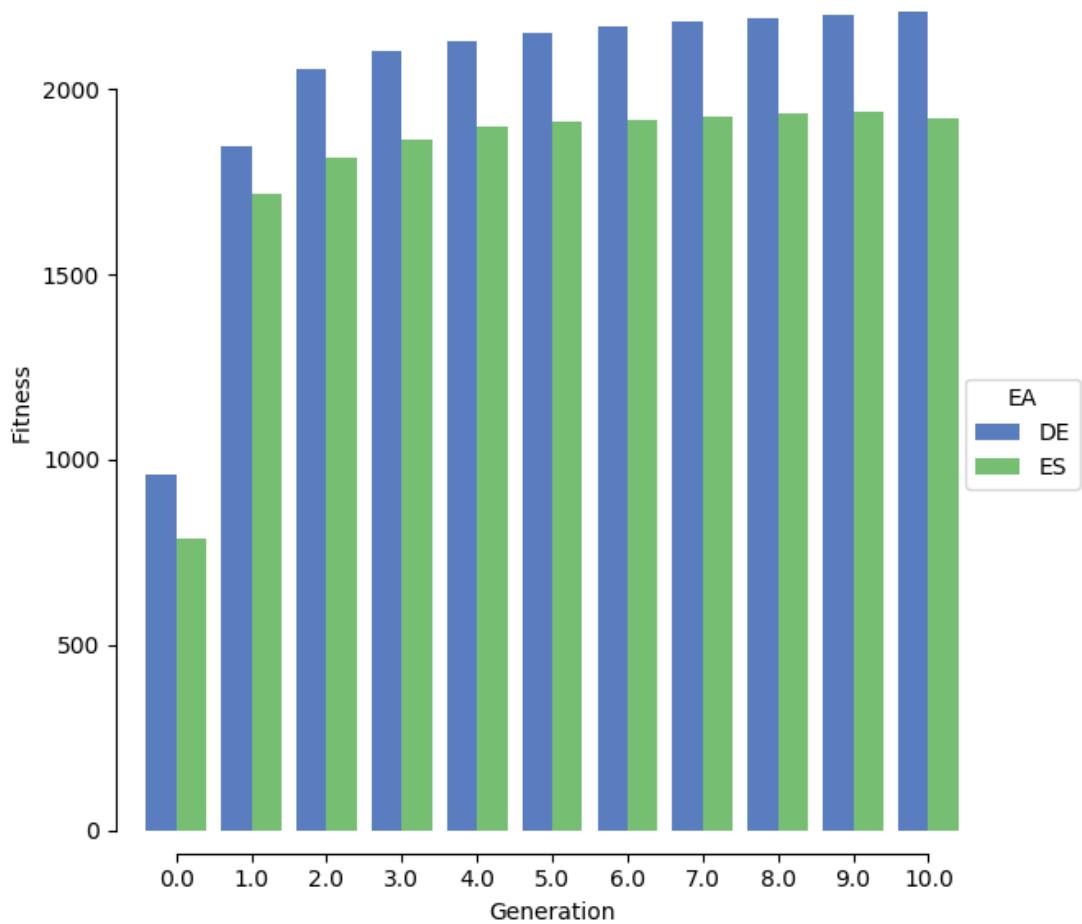
Scout chůze - nastavení experimentu

Vlastnost:	Hodnota:
Roboti:	<i>Scout</i> – 5
Počet generací:	1000
Počet iterací map	1000
Velikost generace(DE)	200
Počet jedinců(ES)	10
Počet mutovaný potomků(ES)	20
Elitismus(ES)	<i>Ano</i>
Elitismus(DE)	<i>Ne</i>

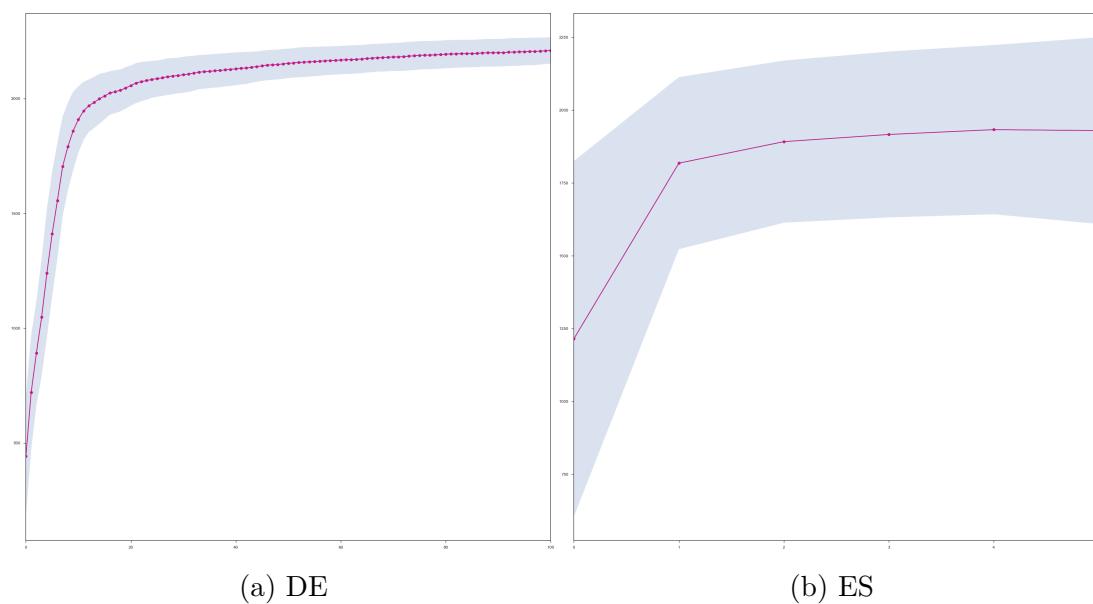
Vlastnost:	Hodnota:
Hodnota nalezeného stromu	10
Ostatní hodnoty:	0
Počet stromů:	300
Počet už pokácených stromů	100

Tabulka 4.4: Scout chůze - nastavení experimentu

Opravit obrázky - rozsahy a popisy



Obrázek 4.3: Scout chůze - porovnání průměrné fitness ES a DE



Obrázek 4.4: Scout chůze - průběh fitness

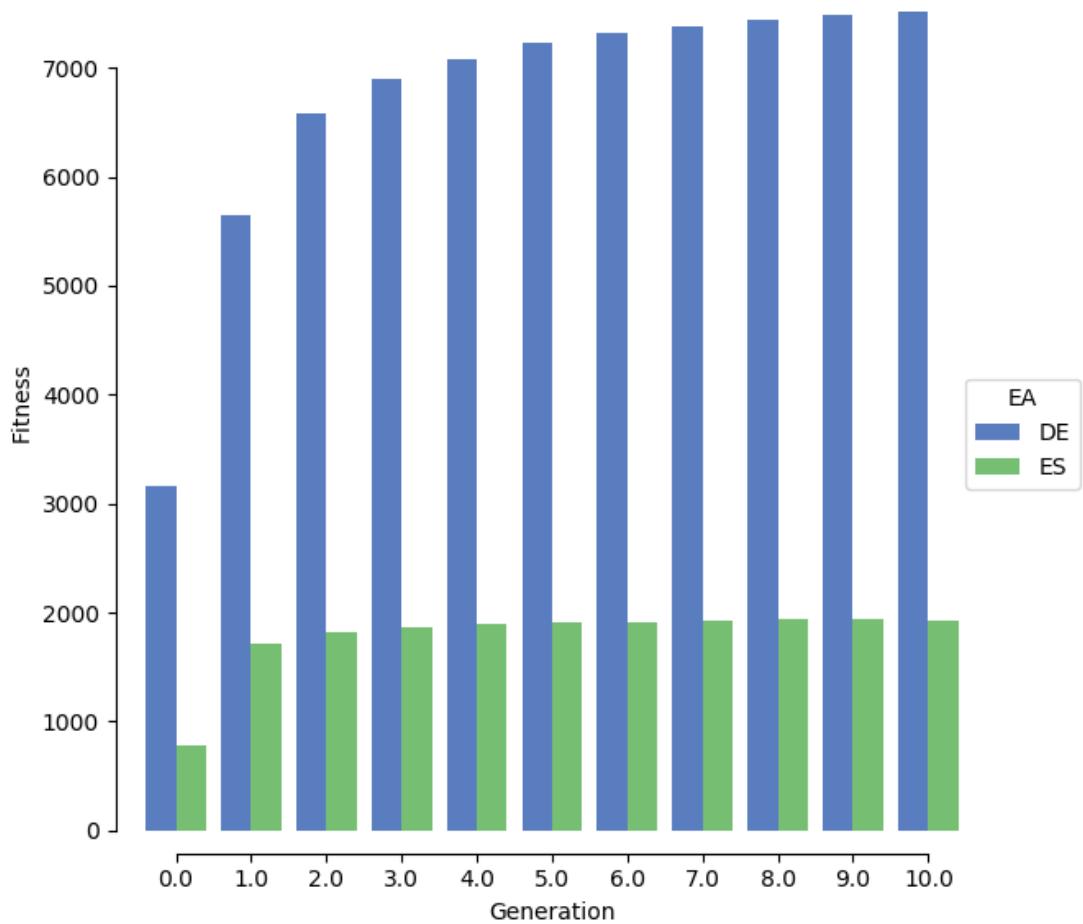
Worker chůze - nastavení experimentu

Vlastnost:	Hodnota:
Roboti:	<i>Worker</i> – 4
Počet generací:	1000
Počet iterací map	1000
Velikost generace(DE)	200
Počet jedinců(ES)	10
Počet mutovaný potomků(ES)	20
Elitismus(ES)	<i>Ano</i>
Elitismus(DE)	<i>Ne</i>

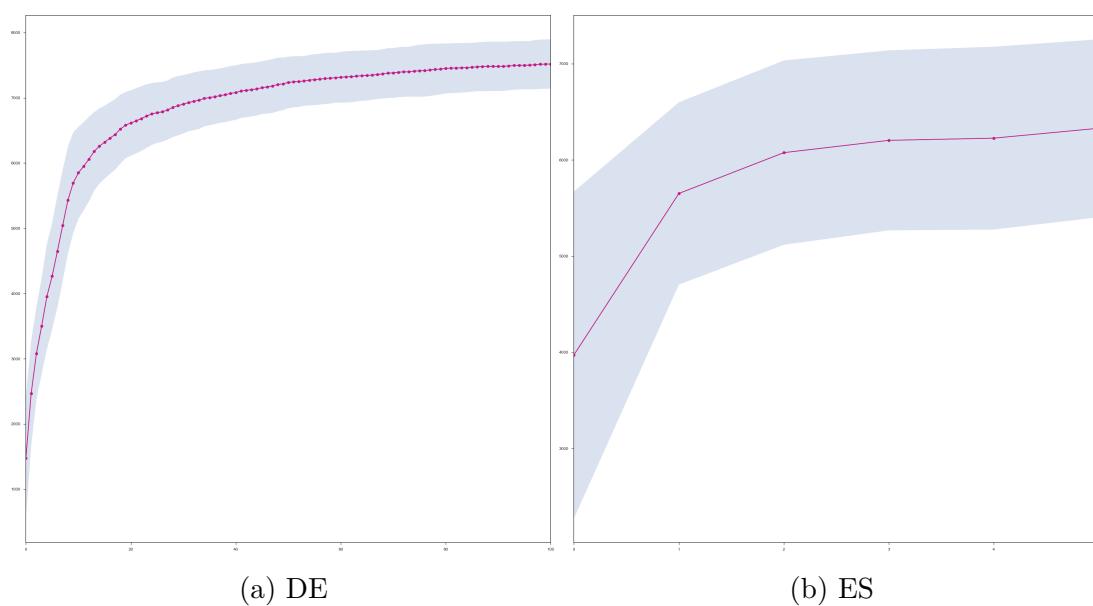
Vlastnost:	Hodnota:
Hodnota nalezeného pokáceného stromu	100
Ostatní hodnoty:	0
Počet stromů:	200
Počet už pokácených stromů	200

Tabulka 4.5: Worker chůze - nastavení experimentu

Opravit rozsahy a rozlišení



Obrázek 4.5: Worker chůze - porovnání průměrné fitness ES a DE



Obrázek 4.6: Worker chůze - průběh fitness v rámci generací

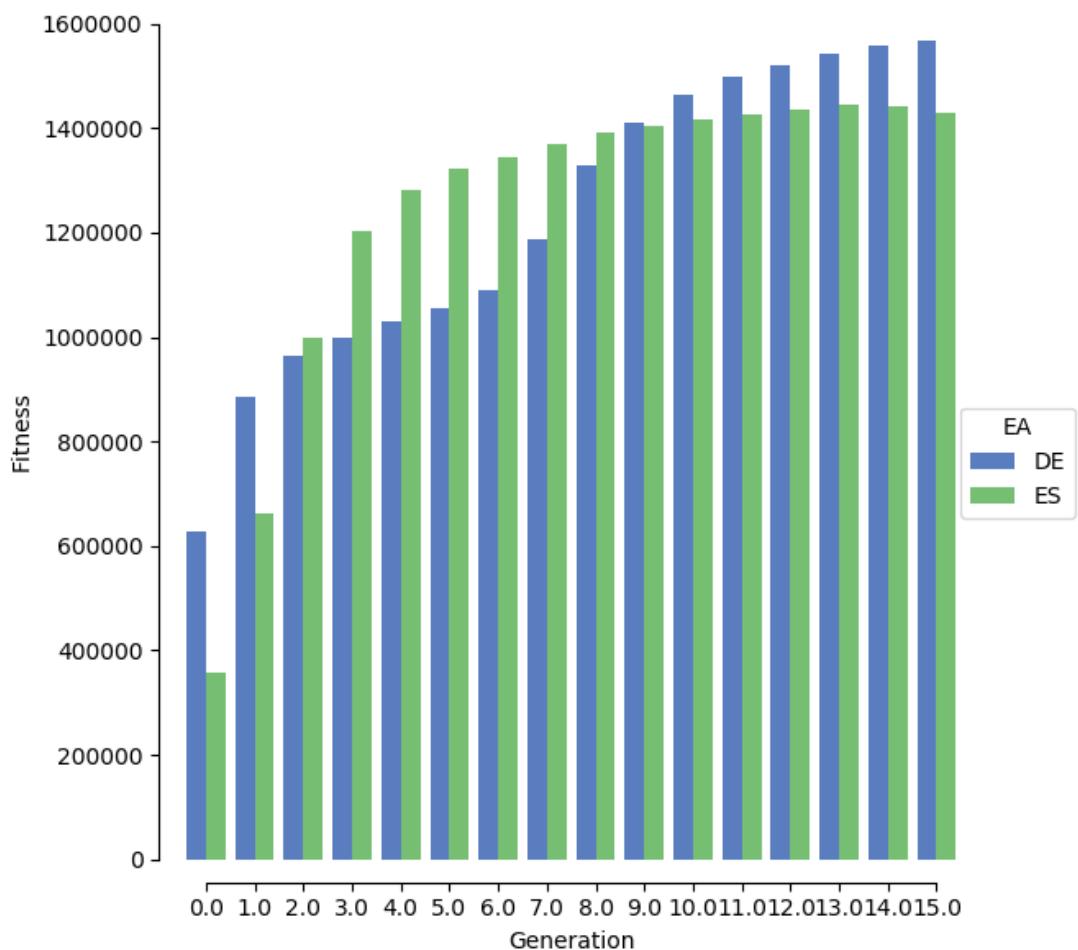
Scout kácení - nastavení experimentu

Vlastnost:	Hodnota:
Roboti:	<i>Scout</i> – 5
Počet generací:	1500
Počet iterací map	1000
Velikost generace(DE)	200
Počet jedinců(ES)	10
Počet mutovaný potomků(ES)	20
Elitismus(ES)	<i>Ano</i>
Elitismus(DE)	<i>Ne</i>

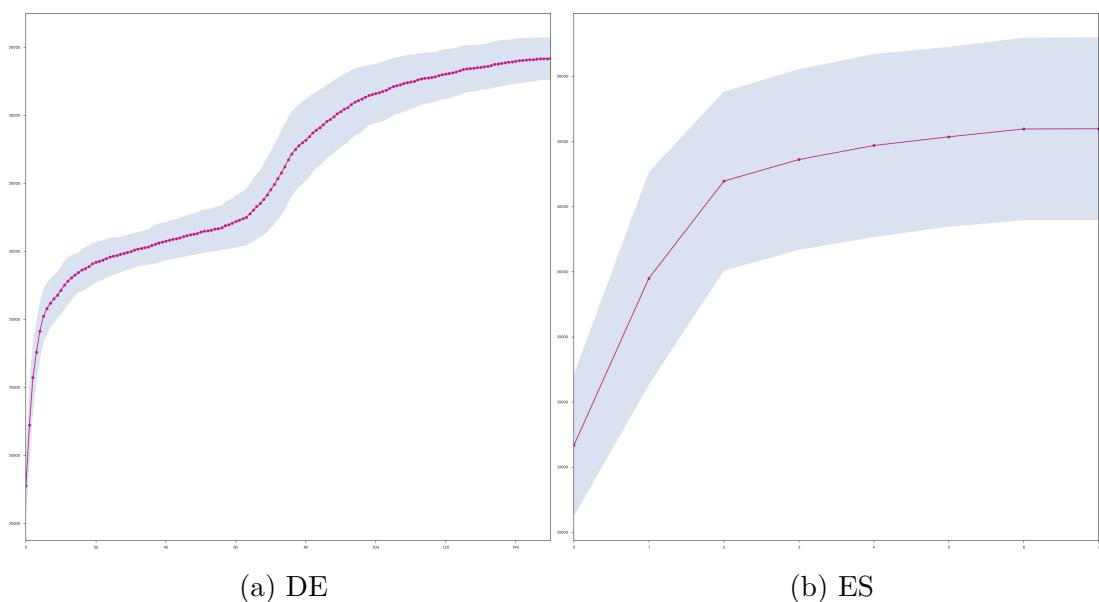
Vlastnost:	Hodnota:
Hodnota nalezeného stromu	1000
Hodnota pokáceného stromu	10000
Hodnota kolize	-1
Ostatní hodnoty:	0
Počet stromů:	400
Počet už pokácených stromů	0

Tabulka 4.6: Scout kácení - nastavení experimentu

Opravit obrázky



Obrázek 4.7: Scout kácení - porovnání průměrné fitness ES a DE



Obrázek 4.8: Scout kácení - průběh fitness v rámci generací

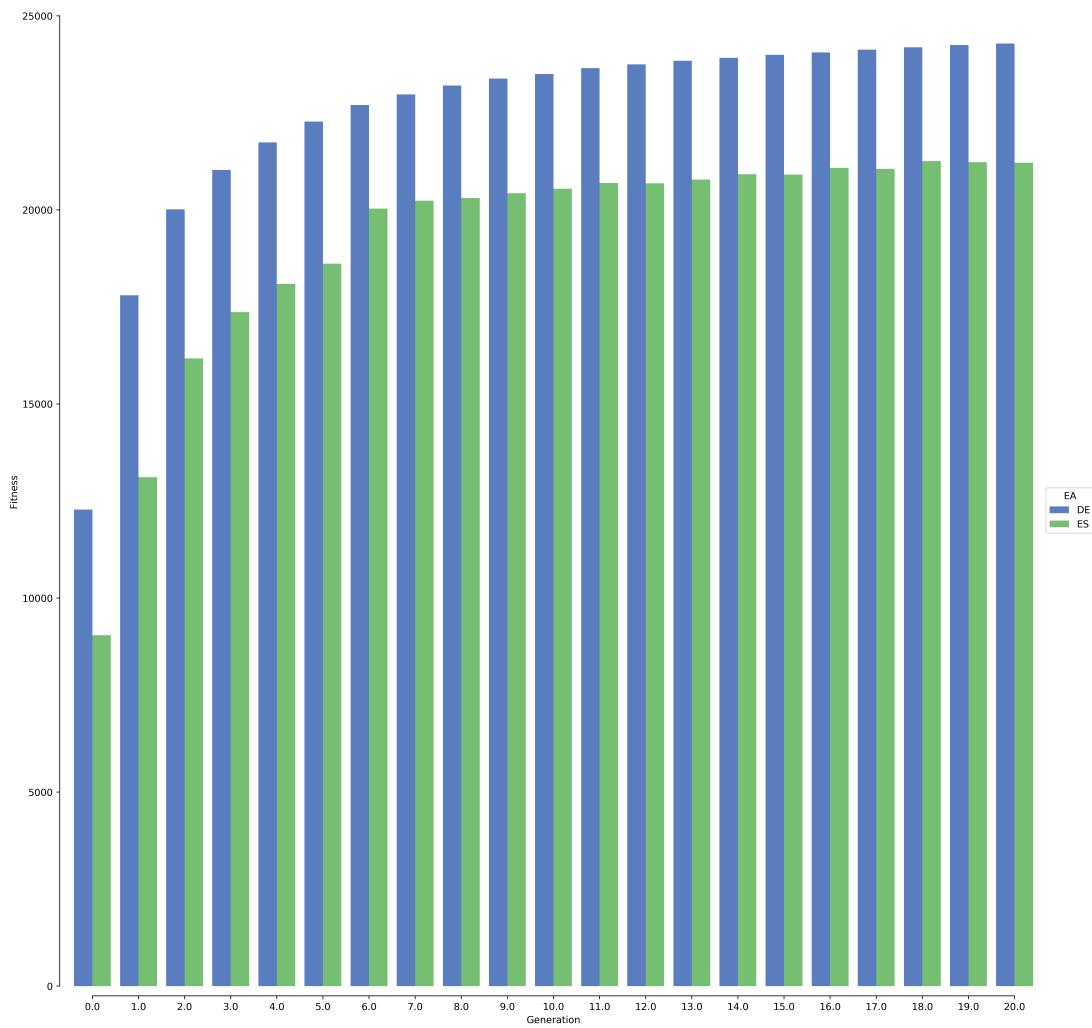
Worker sbírání - nastavení experimentu

Vlastnost:	Hodnota:
Roboti:	<i>Worker</i> – 4
Počet generací:	2000
Počet iterací map	1000
Velikost generace(DE)	200
Počet jedinců(ES)	10
Počet mutovaný potomků(ES)	20
Elitismus(ES)	<i>Ano</i>
Elitismus(DE)	<i>Ne</i>

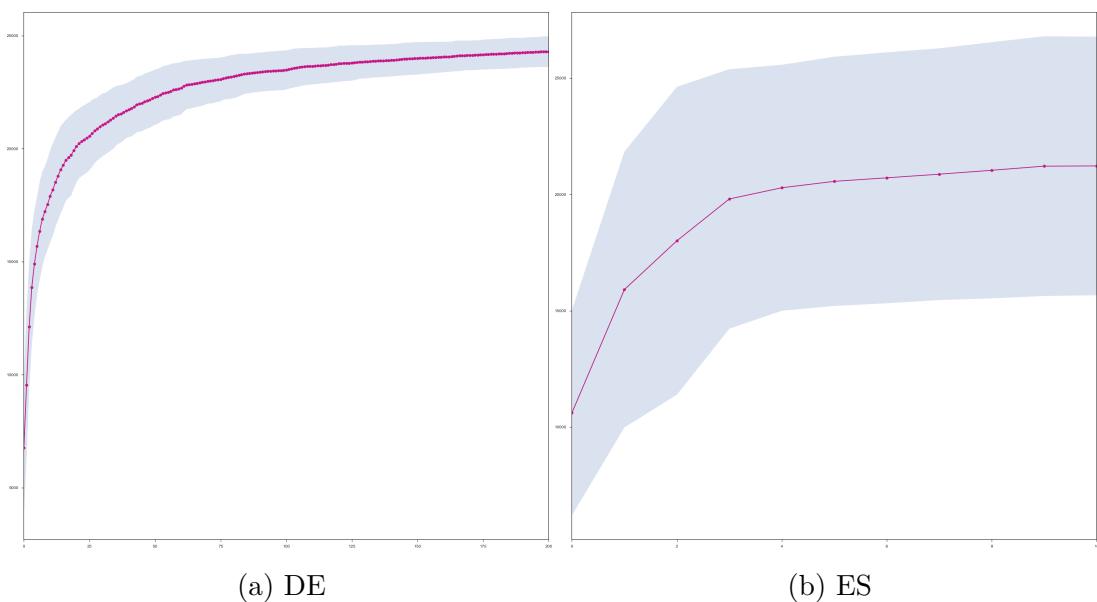
Vlastnost:	Hodnota:
Hodnota nalezeného pokáceného stromu	100
Hodnota uloženého dřeva	1010
Hodnota dřeva v kontejneru	1000
Hodnota jiné entity v kontejneru	-100
Hodnota kolize	-1
Ostatní hodnoty:	0
Počet stromů:	200
Počet už pokácených stromů	200

Tabulka 4.7: Worker sbírání - nastavení experimentu

Opravit obrázky



Obrázek 4.9: Worker sbírání - porovnání průměrné fitness ES a DE



Obrázek 4.10: Worker sbírání - průběh fitness v rámci generací

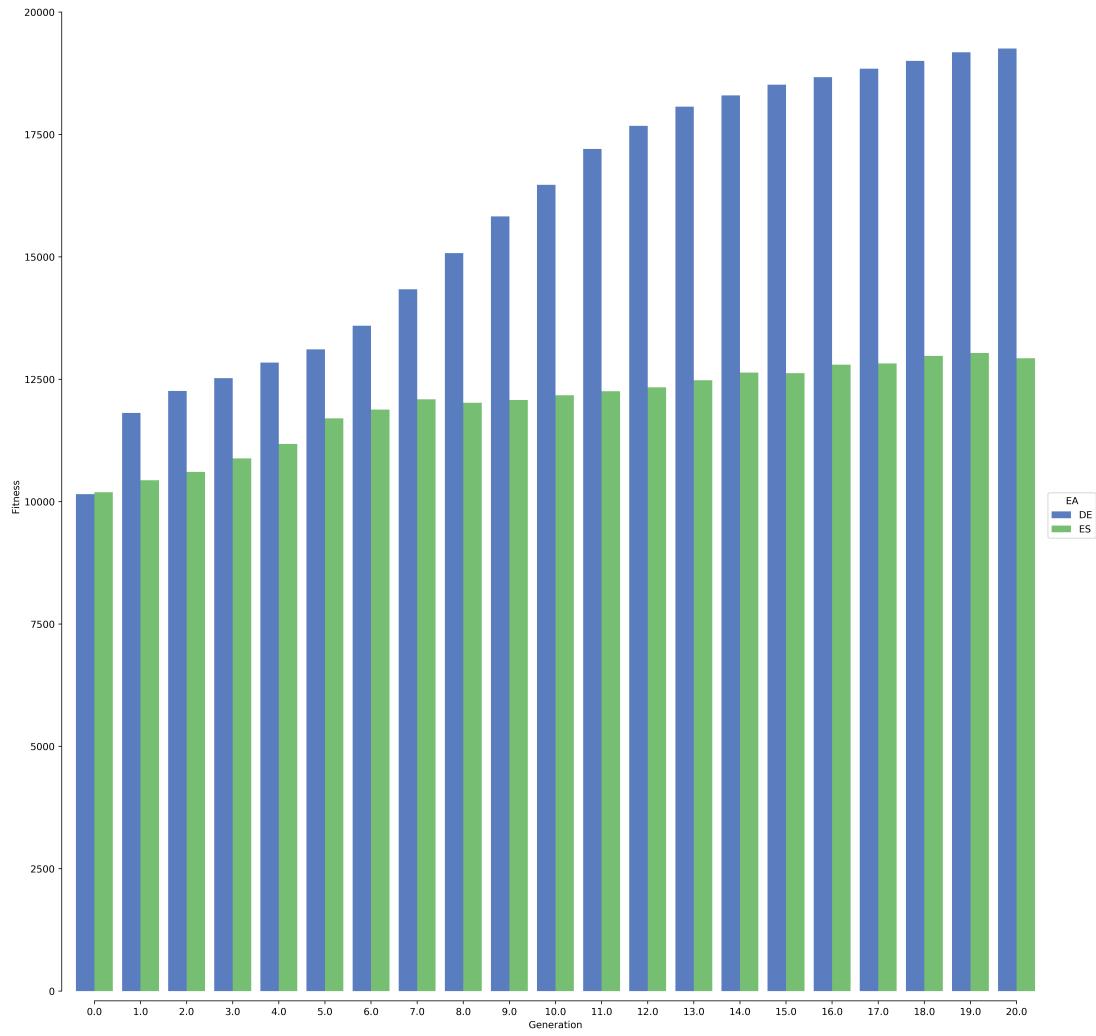
Worker ukládání doprostřed - nastavení experimentu

Vlastnost:	Hodnota:
Roboti:	<i>Worker</i> – 4
Počet generací:	2000
Počet iterací map	2000
Velikost generace(DE)	200
Počet jedinců(ES)	10
Počet mutovaný potomků(ES)	20
Elitismus(ES)	<i>Ano</i>
Elitismus(DE)	<i>Ne</i>

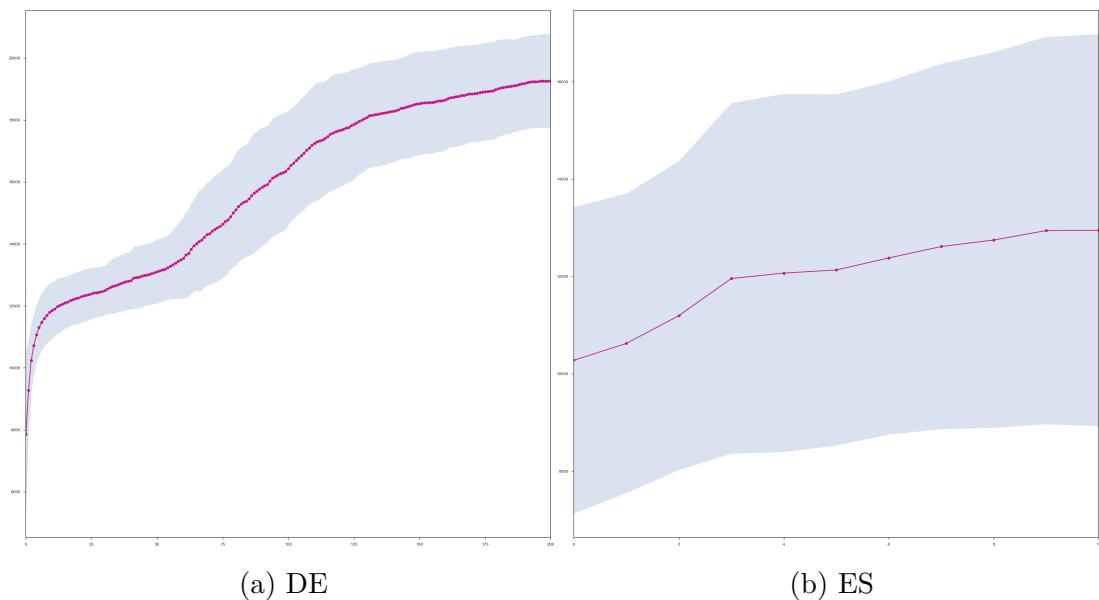
Vlastnost:	Hodnota:
Hodnota nalezeného pokáceného stromu	100
Hodnota uloženého dřeva	1000
Hodnota dřeva v kontejneru	100
Hodnota jiné entity v kontejneru	-100
Hodnota kolize	-1
Ostatní hodnoty:	0
Počet stromů:	200
Počet už pokácených stromů	200

Tabulka 4.8: Worker ukládání doprostřed - nastavení experimentu

Opravit obrázky



Obrázek 4.11: Worker ukládání doprostřed - porovnání průměrné fitness ES a DE



Obrázek 4.12: Worker ukládání doprostřed - Průběh fitness v rámci generací

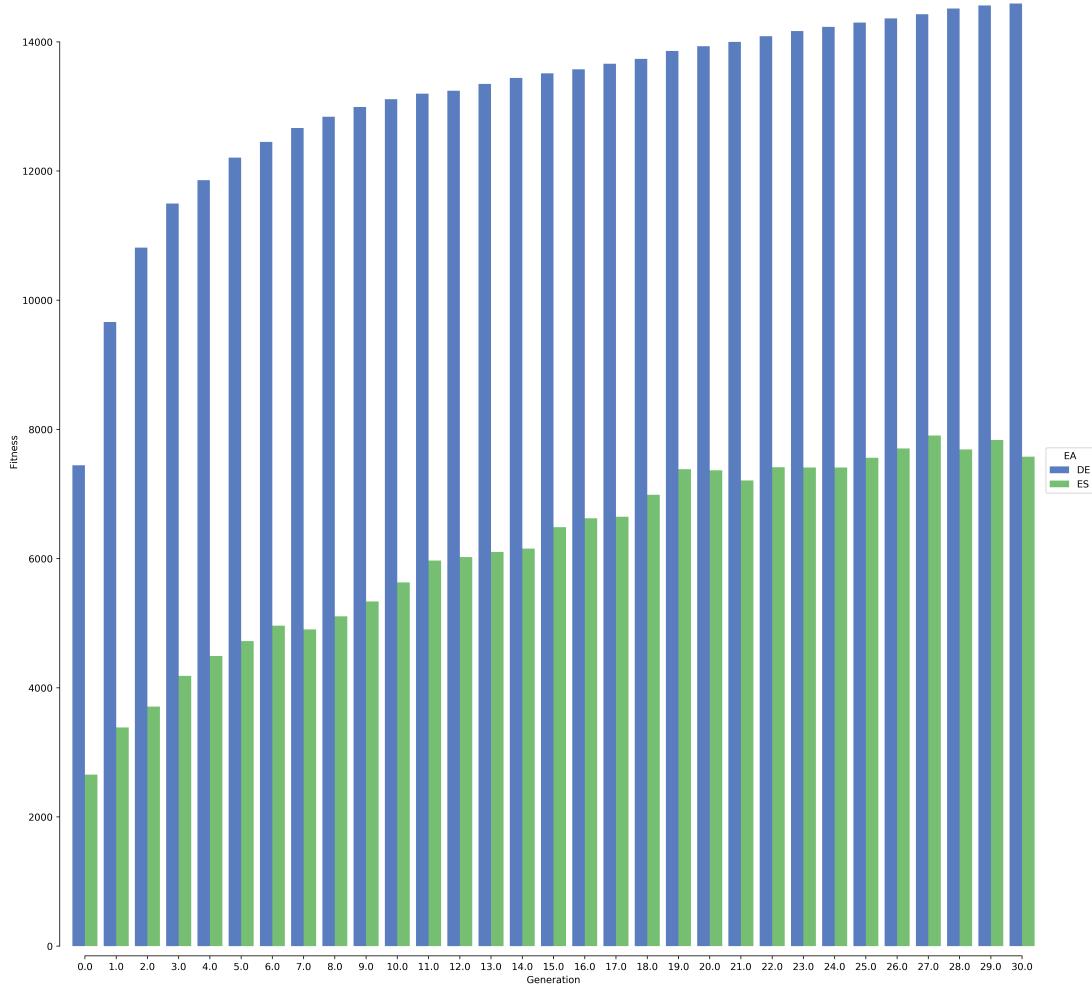
Kooperace hlavní úkol - nastavení experimentu

Vlastnost:	Hodnota:
Roboti:	<i>Scout – 5, Worker – 4</i>
Počet generací:	4000
Počet iterací map	2000
Velikost generace(DE)	200
Počet jedinců(ES)	10
Počet mutovaný potomků(ES)	20
Elitismus(ES)	<i>Ano</i>
Elitismus(DE)	<i>Ne</i>

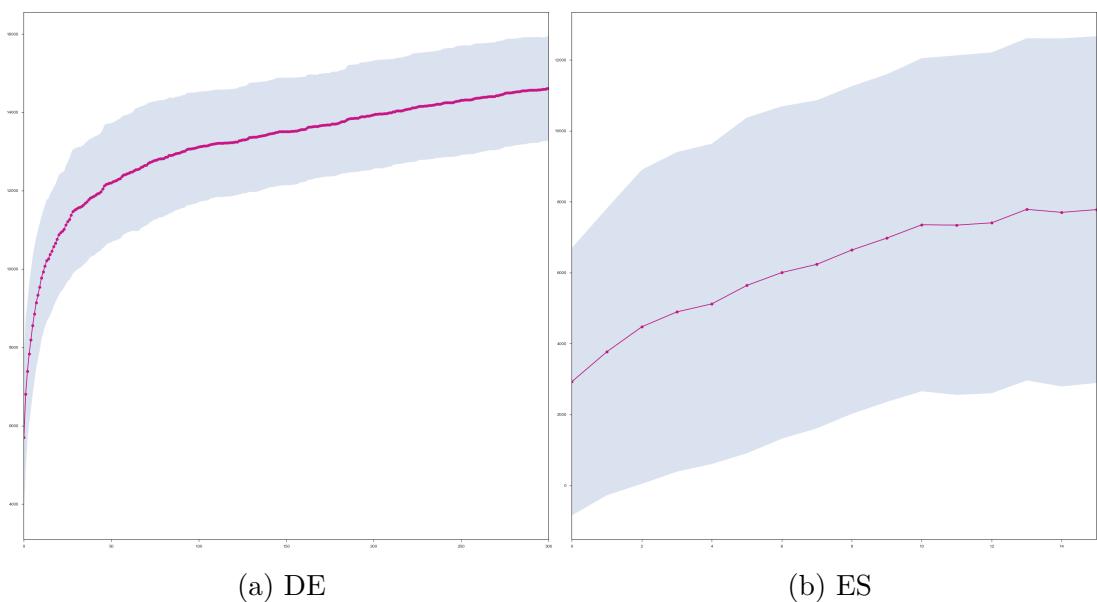
Vlastnost:	Hodnota:
Hodnota nalezeného pokáceného stromu	100
Hodnota uloženého dřeva	1000
Hodnota dřeva v kontejneru	100
Hodnota jiné entity v kontejneru	-100
Hodnota kolize	-1
Ostatní hodnoty:	0
Počet stromů:	400
Počet už pokácených stromů	0

Tabulka 4.9: Kooperace hlavní úkol - nastavení experimentu

Opravit obrázky



Obrázek 4.13: Kooperace hlavní úkol - porovnání průměrné fitness ES a DE

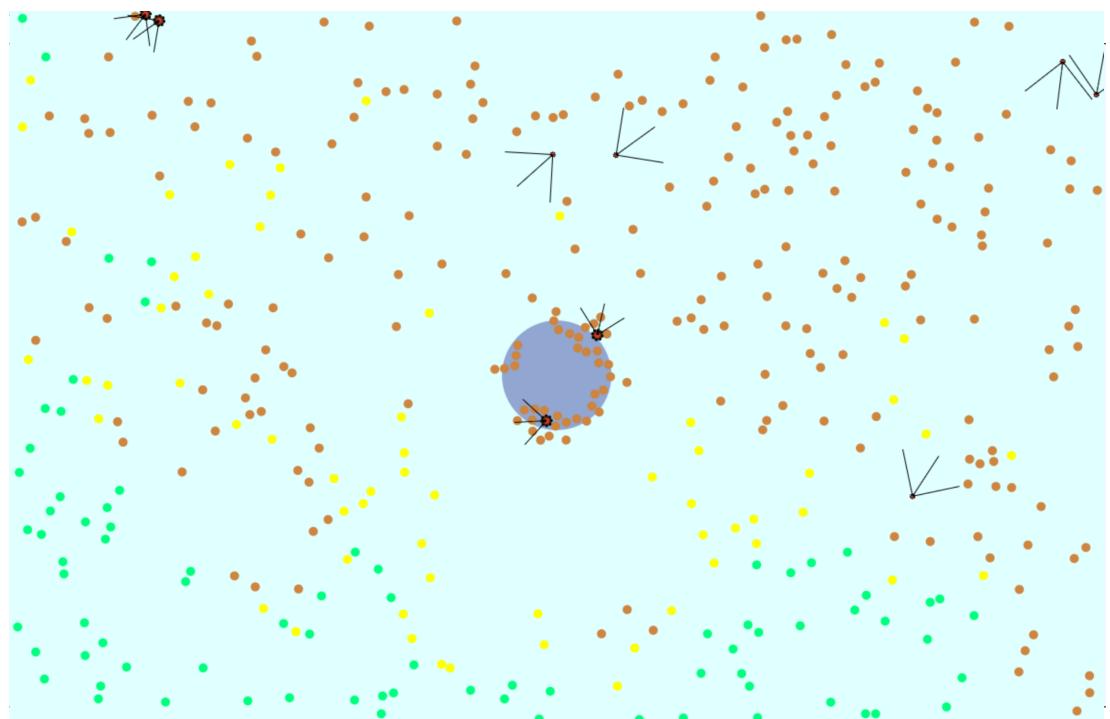


Obrázek 4.14: Kooperace hlavní úkol - průběh fitness v rámci generací

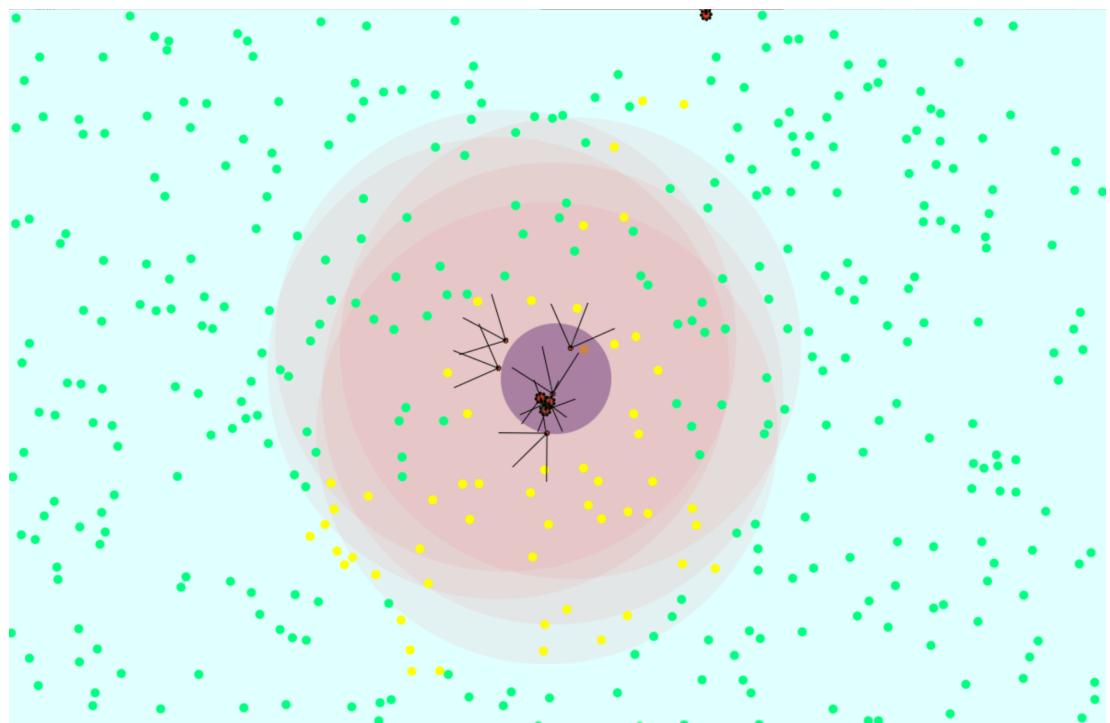
4.2.5 Výsledky Experimentu

Výsledkem posloupnosti všech podúkolů vzniklo poměrně velmi komplexní chování. Finální neuronové sítě ať u Worker robotů, či Scout robotů se zvládají vyhýbat překážkám. Scout roboti kácí stromy, které naleznou. Worker roboti nakládají zpracované dřevo, pokud na něj narazí, když zachytí signál úložiště, tak vyloží aktuální náklad. Některé chování byli také schopny předejít zaseknutí o nějaký shluk objektů, pokud byl jejich pohyb vpřed neúspěšný, tak po několika pokusech vycouvali a vydali se cestou okolo kritického místa. U většiny se také objevilo použití rádiových signálů jako prostředku pro největší možné rozptýlení po mapě, jakmile zachytí cizí signál vydají se opačným směrem. Průběh fitness jednotlivých podúkolů je zachycena na předchozích grafech 4.4 až 4.12b.

Ač se jedná o nejlepší dosažené chování objevují se nějaké nedostatky. Worker robot se občas dostane do pozice z které není schopen vyjet, jedná se především o kolize s vícero entitami. Tento problém by mohlo vyřešit použití vícevrstvých sítí či evolučního algoritmu evolvujícího i architekturu sítě. Skládání zpracovaného materiálu po obvodu skladiště není efektivní způsob, jak do něj naskládat maximální množství dřeva. V tomto případě jsem se snažil vylepšit tento nedostatek promítnutím vzdálenosti dřeva od středu skladiště do celkové fitness, ovšem bez znatelného zlepšení v chování. Nejspíše by bylo třeba použít rádiový senzor poskytující více informací o směru k zachycenému signálu.



Obrázek 4.15: Nejlepší jedinec - 10000 iterací



Obrázek 4.16: Nejlepší jedinec - 10000 iterací

Inicializační nastavení:

Výška	800
Šířka	1200
Počet iterací	10000
Počet stromů	400
Počet Scout robotů	5
Počet Worker robotů	4

Tabulka 4.10: WoodScene - nastavení mapy pro statistické údaje

Výsledky

Zpracované dřevo zanechané v mapě	225,22
Stromy v mapě	156,22
Z toho nalezené	52,84
Dřevo v kontejnerech	18,48
Uskladněné dřevo	17,56

Tabulka 4.11: WoodScene - průměrné výsledky nejlepšího chování pro 100 pokusů

Shrnutí

Nějak shrnout, úplně nevím, co by to mělo obsahovat.

4.3 Mineral Scene

Jedná se o scénář reprezentující sběr surovin pro výrobu paliva a jeho následné využití. Figurují zde 3 rozliční roboti, všichni potřebují pro pohyb dané množství paliva. Úspěšnost daného hejna se měří množstvím paliva. Nejmenší robot (Mineral scout) disponuje pouze senzory k exploraci prostředí a rádiovým vysílačem pro komunikaci se skupinou. Robot prostřední velikosti (Mineral Worker) se pohybuje o něco pomaleji než Mineral Scout, ale umí přesouvat objekty i více najednou. Robot pro přeměnu minerálu („suroviny na výrobu paliva“) označen ve frameworku jako Mineral Refactor se přemisťuje nejpomaleji, má možnost přeměnit minerál na palivo. Tento scénář si bere jako inspiraci strategické hry a hypotetické přežití robotů na cizí planetě, kde si budou muset obstarat vlastní nerostné suroviny pro běh.

4.4 Competitive Scene

Poslední ze scénářů se týká soutěže dvou týmů (hejn) ve kterých figurují jeden malý průzkumný robot (Competitive Scout) a jeden větší bojový robot (Competitive Fighter). Úspěšnost týmu je dána zachovanými jednotkami zdraví robotů a uděleným poškozením do nepřátelské skupiny robotů. Competitive Scout se pohybuje značně rychleji než Competitive Fighter, ale uděluje menší poškození. Což lze opět vztáhnout na chování rozdílných skupin nepřátel např. ve strategických hrách, kde se jejich chování adaptuje, co nejlépe na dané prostředí.

Závěr

Seznam použité literatury

- (2018). Evolution strategies as a scalable alternative to reinforcement learning.
<https://blog.openai.com/evolution-strategies>. Accessed: 21.2.2018.
- ARVIN, F., MURRAY, J., ZHANG, C. a YUE, S. (n.d.). Colias: An autonomous micro robot for swarm robotic applications. *INTERNATIONAL JOURNAL OF ADVANCED ROBOTIC SYSTEMS*, **11**. ISSN 17298806. URL <http://search.ebscohost.com/login.aspx?authtype=shib&custid=s1240919&profile=eds>.
- AZNAR, F., SEMPERE, M., PUJOL, M., RIZO, R. a PUJOL, M. J. (2014). Modelling oil-spill detection with swarm drones. *Abstract & Applied Analysis*, pages 1 – 14. ISSN 10853375. URL <http://search.ebscohost.com/login.aspx?authtype=shib&custid=s1240919&profile=eds>.
- BALCH, T. (2000). Hierachic social entropy: An information theoretic measure of robot group diversity. *Autonomous robots*, **8**(3), 209–238.
- BASHYAL, S. a VENAYAGAMOORTHY, G. K. (2008). Human swarm interaction for radiation source search and localization. pages 1–8.
- BEYER, H.-G. a SCHWEFEL, H.-P. (2002). Evolution strategies – a comprehensive introduction. *Natural Computing*, **1**(1), 3–52. ISSN 1572-9796. doi: 10.1023/A:1015059928466. URL <https://doi.org/10.1023/A:1015059928466>.
- DAVID, P., MIKE, D., REGINA, E., MIKE, H. a CRAIG, L. (2001). Pheromone robotics. *Autonomous Robots*, **(3)**, 319. ISSN 0929-5593. URL <http://search.ebscohost.com/login.aspx?authtype=shib&custid=s1240919&profile=eds>.
- DUARTE, M., COSTA, V., GOMES, J., RODRIGUES, T., SILVA, F., OLIVEIRA, S. M. a CHRISTENSEN, A. L. (2016). Evolution of collective behaviors for a real swarm of aquatic surface robots. *PLoS ONE*, **11**(3), 1 – 25. ISSN 19326203. URL <http://search.ebscohost.com/login.aspx?authtype=shib&custid=s1240919&profile=eds>.
- EIBEN, A. (2015). *Introduction to evolutionary computing*. Springer, Berlin. ISBN 978-3662448731.
- FORTIN, F.-A., RAINVILLE, F.-M. D., GARDNER, M.-A., PARIZEAU, M. a GAGNÉ, C. (2012). Deap: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, **13**(Jul), 2171–2175.
- GOMES, J. C., URBANO, P. a CHRISTENSEN, A. L. (2013). Evolution of swarm robotics systems with novelty search. *CoRR*, **abs/1304.3362**. URL <http://arxiv.org/abs/1304.3362>.
- HOLLAND, J. H. (1976). *Adaptation in Natural and Artificial Systems*, volume 18. URL <http://search.ebscohost.com/login.aspx?authtype=shib&custid=s1240919&profile=eds>.

- IVAN, T., TÜZE, K. a KATSUNORI, S. (2013). Hormone-inspired behaviour switching for the control of collective robotic organisms. *Robotics*, Vol 2, Iss 3, Pp 165-184 (2013), (3), 165. ISSN 2218-6581. URL <http://search.ebscohost.com/login.aspx?authtype=shib&custid=s1240919&profile=eds>.
- JEVTIĆ, A. a ANDINA DE LA FUENTE, D. (2007). Swarm intelligence and its applications in swarm robotics.
- JONES, S., STUDLEY, M., HAUERT, S. a WINFIELD, A. Evolving behaviour trees for swarm robotics.
- MARSALLI, M. McCulloch-pitts neurons. URL <http://www.mind.ilstu.edu/curriculum/modOverview.php?modGUI=212>.
- MITCHELL, M. (1998). *An introduction to genetic algorithms*. Complex adaptive systems. Cambridge : MIT Press, 1998. ISBN 0-262-13316-4. URL <http://search.ebscohost.com/login.aspx?authtype=shib&custid=s1240919&profile=eds>.
- MONDADA, F., BONANI, M., RAEMY, X., PUGH, J., CIACI, C., KLAPOCZ, A., MAGNENAT, S., ZUFFEREY, J.-C., FLOREANO, D. a MARTINOLI, A. (2009). The e-puck, a robot designed for education in engineering. In *Proceedings of the 9th conference on autonomous robot systems and competitions*, volume 1, pages 59–65. IPCB: Instituto Politécnico de Castelo Branco.
- PENDERS, J., ALBOUL, L., WITKOWSKI, U., NAGHSH, A., SAEZ-PONS, J., HERBRECHTSMEIER, S. a EL-HABBAL, M. (2011). A robot swarm assisting a human fire-fighter. *Advanced Robotics*, **25**(1/2), 93 – 117. ISSN 01691864. URL <http://search.ebscohost.com/login.aspx?authtype=shib&custid=s1240919&profile=eds>.
- PROFESSOR MARCO DORIGO (2001). Swarm-bots. URL <http://www.swarm-bots.org>.
- RUBENSTEIN, M., AHLER, C., HOFF, N., CABRERA, A. a NAGPAL, R. (2014). Kilobot: A low cost robot with scalable operations designed for collective behaviors. *Robotics and Autonomous Systems*, **62**(Reconfigurable Modular Robotics), 966 – 975. ISSN 0921-8890. URL <http://search.ebscohost.com/login.aspx?authtype=shib&custid=s1240919&profile=eds>.
- SHOULSON, A., GARCIA, F., JONES, M., MEAD, R. a BADLER, N. (2011). Parameterizing behavior trees. *Motion in Games*, pages 144–155.
- SUPERATI, V., TRIANNI, V. a NOLFI, S. (2011). Self-organised path formation in a swarm of robots. *Swarm Intelligence*, **5**(2), 97–119.
- STORN, R. a PRICE, K. (1997). Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, **11**(4), 341–359.
- TAN, Y. a ZHENG, Z.-Y. (2013). Research advance in swarm robotics. *Defence Technology*, **9**, 18 – 39. ISSN 2214-9147. URL <http://search.ebscohost.com/login.aspx?authtype=shib&custid=s1240919&profile=eds>.

YALCIN, C. (2008). Evolving aggregation behavior for robot swarms: A cost analysis for distinct fitness functions. pages 1–4.

Seznam obrázků

4.1	Příklad WoodScene mapy: start náhodného chování	25
4.2	Příklad WoodScene mapy: po 9000 iteracích náhodného chování .	25
4.3	Scout chůze - porovnání průměrné fitness ES a DE	30
4.4	Scout chůze - průběh fitness	30
4.5	Worker chůze - porovnání průměrné fitness ES a DE	32
4.6	Worker chůze - průběh fitness v rámci generací	32
4.7	Scout kácení - porovnání průměrné fitness ES a DE	34
4.8	Scout kácení - průběh fitness v rámci generací	34
4.9	Worker sbírání - porovnání průměrné fitness ES a DE	36
4.10	Worker sbírání - průběh fitness v rámci generací	36
4.11	Worker ukládání doprostřed - porovnání průměrné fitness ES a DE	38
4.12	Worker ukládání doprostřed - Průběh fitness v rámci generací .	38
4.13	Kooperace hlavní úkol - porovnání průměrné fitness ES a DE .	40
4.14	Kooperace hlavní úkol - průběh fitness v rámci generací	40
4.15	Nejlepší jedinec - 10000 iterací	42
4.16	Nejlepší jedinec - 10000 iterací	42

Seznam tabulek

2.1	Porovnání systémů s více agenty	13
2.2	Parameterizing behavior trees, Motion in Games - podoba stromů	15
2.3	Parameterizing behavior trees, Motion in Games - nastavení parametrů	17
4.1	Nastavení parametrů u EA	24
4.2	Wood - Scout robot popis	26
4.3	Wood - Worker robot popis	27
4.4	Scout chůze - nastavení experimentu	29
4.5	Worker chůze - nastavení experimentu	31
4.6	Scout kácení - nastavení experimentu	33
4.7	Worker sbírání - nastavení experimentu	35
4.8	Worker ukládání doprostřed - nastavení experimentu	37
4.9	Kooperace hlavní úkol - nastavení experimentu	39
4.10	WoodScene - nastavení mapy pro statistické údaje	43
4.11	WoodScene - průměrné výsledky nejlepšího chování pro 100 pokusů	43

Seznam použitých zkratek

Přílohy