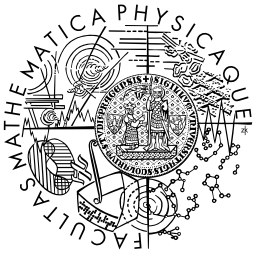


MATEMATICKO-FYZIKÁLNÍ
FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Václav Krňák

Evoluční algoritmy pro konstrukci 2D mostů ve hře Poly Bridge

Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: Mgr. Roman Neruda, CSc.

Studijní program: Informatika

Praha 2024

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

Tímto bych chtěl poděkovat vedoucímu mé bakalářské práce, panu Mgr. Romanu Nerudovi CSc. za jeho cenné rady a užitečnou zpětnou vazbu během psaní této práce. Zároveň bych chtěl poděkovat své partnerce za její podporu.

Název práce: Evoluční algoritmy pro konstrukci 2D mostů ve hře Poly Bridge

Autor: Václav Krňák

Katedra: Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: Mgr. Roman Neruda, CSc., Katedra teoretické informatiky a matematické logiky

Abstrakt: V této bakalářské práci se budeme zabývat řešením několika úrovní ze hry Poly Bridge pomocí umělé inteligence. Poly bridge je logická hra se sandboxovým prostředím, ve kterém je hráč veden k tomu, aby postavil 2D mostní konstrukci podobnou příhradovému mostu. Hráč musí dbát na stabilitu této konstrukce, ale zároveň i na materiálovou náročnost. Vzhledem ke složitosti a variabilitě problémů, které hra představuje, jsme se rozhodli použít evoluční algoritmy. Cílem práce je návrh několika genetických operátorů, které optimalizují mostní struktury, jejich porovnání a využití nalezených mostů ve hře Poly Bridge.

Klíčová slova: Evoluční algoritmy, Automatický design, Počítačové hry, Simulace,

Title: Evolutionary algorithms for 2D bridge construction in the Poly Bridge game

Author: Václav Krňák

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: Mgr. Roman Neruda, CSc., Department of Theoretical Computer Science and Mathematical Logic

Abstract: This bachelor thesis proposes an artificial intelligence approach to solve several levels from the Poly Bridge game. Poly bridge is a puzzle game with a sandbox environment in which the player is led to build a 2D bridge structure similar to a truss bridge. The player has to pay attention to the stability of this structure as well as to the material requirements. Given the complexity and variability of the problems posed by the game, we decided to use evolutionary algorithms. The aim of this thesis is to design several genetic operators that optimize the bridge structure, compare them and use the produced bridges in the Poly Bridge game.

Keywords: Evolutionary Algorithms, Automated design, Computer games, Simulation,

Obsah

Úvod

V posledních dekádách se umělá inteligence stala klíčovým prvkem v mnoha technologických a vědeckých oborech, včetně počítačových her [1] [2]. Jedním z nástrojů umělé inteligence, kterým můžeme interagovat s hrami, jsou evoluční algoritmy. Tyto algoritmy inspirované biologickou evolucí nabízejí zajímavý mechanismus pro automatizované řešení problémů a optimalizaci procesů [3].

Poly Bridge je počítačová hra zaměřená na stavební inženýrství a řešení fyzikálních hádanek. Hra žádá od hráčů, aby stavěli mosty přes různé rozpětí vodních ploch s omezenými zdroji a za dodržení specifických podmínek [4]. Záměrem této bakalářské práce je prozkoumat, jak mohou být evoluční algoritmy použity pro automatické generování a optimalizaci mostních konstrukcí v rámci hry.

Cílem práce je nejen vysvětlit teoretické aspekty evolučních algoritmů, ale především demonstrovat jejich praktické využití a potenciál v kontextu moderních počítačových her. Tato práce tedy může sloužit jako ukázka toho, jak moderní metody umělé inteligence mohou překonat tradiční přístupy používané komunitou hráčů počítačových her.

Práce nejprve představí základní koncept evolučních algoritmů a vysvětlí jejich principy a metody. Následně bude analyzovat specifika hry Poly Bridge, na jejímž základě bude možné vytvořit fyzikální simulované prostředí pro aplikaci těchto algoritmů. Praktická část práce se zaměří na implementaci vybraných algoritmů, jejich testování a hodnocení na základě efektivity a práce se zdroji.

1 Evoluční Algoritmy

Evoluční algoritmy jsou metaheuristická optimalizační metoda odvozená z teorie evoluce. Tato technika využívá principy darwinovské evoluce — selekci, mutaci a křížení k optimalizaci řešení. Tyto algoritmy jsou efektivní napříč různými problémy, protože mají velice málo předpokladů o povaze problému. Výpočetní náročnost vyplývající z vyhodnocování fitness funkce však může bránit jejich použití. Pozoruhodné je, že i jednodušší evoluční algoritmy mohou řešit složité problémy.

Evoluční algoritmy mají v informatice kořeny už ve 40. letech 20. století. První myšlenky o simulované evoluci představil Alan Turing v roce 1948 [5]. Prakticky se začaly používat v 60. letech 20. století. K tomuto oboru významně přispěli různí průkopníci z celého světa, včetně Johna Hollanda [6] a Johna Kozy [7]. V průběhu let se tento obor vyvinul a stal se předmětem několika vědeckých časopisů a specializovaných konferencí, jako jsou GECCO a CEC.

1.1 Zakládní definice

V této části práce bychom chtěli definovat některé užitečné pojmy. Dále ukážeme základní komponenty evolučních algoritmů a zároveň vysvětlíme, jakou hrají roli při návrhu a běhu algoritmu.

1.1.1 Genom a Jedinec

Genom představuje jednoho řešení pro daný problém. Je důležité zvolit vhodné kódování tohoto řešení. J. Holland si ve svém původním návrhu evolučního algoritmu představoval, že každé řešení bude kódovano binárně [6]. Později se však ukázalo, že je možné dosáhnout lepších výsledků, když kódováním reprezentujeme jakési stavební bloky [8].

Jedincem myslíme zastoupení genomu v populaci.

1.1.2 Populace a Generace

Populace je seznam jedinců. Z počátku běhu algoritmu obvykly naplníme populaci jedinci s náhodnými genomy. Postupnou aplikací genetických operátorů v ní budeme evolvovat lepší a lepší řešení. *Generace* představuje stav populace v konkrétním čase.

1.1.3 Genetické operátory

Genetické operátory jsou funkce, které můžeme aplikovat na jednoho a více jedinců nebo na celou populaci s cílem vybrat lepší jedince do další generace. Pomocí těchto operátorů můžeme vyvažovat exploraci a exploataci algoritmu a zároveň celou populaci směrovat k optimálnímu řešení [3].

V kontextu genetických operátorů budeme často mluvit o *rodičích* a *potomcích*. Rodiči myslíme ty jedince, kteří se v populaci nachází před aplikací genetických operátorů. Potomci jsou ti, kteří se v populaci vysktují po jejich aplikaci.

1.1.4 Křížení

Křížení je genetický operátor, kterým můžeme ze dvou nebo více jedinců (rodičů) vytvořit nového jedince (potomka), jenž strukturou připomíná oba (všechny) svoje rodiče. Tento proces je inspirován biologickou reprodukcí, ve které potomci zdědí vlastnosti obou rodičů, což může vést k vyšší genetické variabilitě v populaci. Je důležité, aby nový potomek nebyl pouze náhodnou kombinací částí genů svých rodičů, ale aby křížení dávalo z hlediska struktury smysl [8].

1.1.5 Mutace

Mutace je genetický operátor, který může měnit náhodné části genomu. Od křížení se liší hlavně tím, že je nezávislá na ostatních genomech v populaci. Význam mutací spočívá v udržení genetické diverzity populace. To je zásadní pro průzkum širšího prostoru řešení a předcházení předčasné konvergenci.

1.1.6 Fitness

Fitness funkci budeme značit písmenem $f : G \to \mathbb{R}$, kde G je množina všech možných genomů. Fitness nabízí měřitelnou kvalitu daného genomu a pomáhá algoritmu rozlišovat mezi více a méně vhodnými jedinci. Je třeba dbát na správný návrh fitness funkce, aby se předešlo běžným problémům, jako je předčasná konvergence nebo uváznutí v lokálních maximech. Není neobvyklé, že do fitness funkce je zahrnuto více různých komponent, které do výsledné hodnoty přispívají různými vahami. Tímto způsobem můžeme lépe rozlišit kvalitní řešení od těch nekvalitních a poskytnout algoritmu více informací pro efektivnější průzkum prostoru řešení. V praxi to může znamenat, že vedle hlavního kritéria, kterým je například výkon nebo efektivita, mohou být do fitness funkce zahrnuty i sekundární kritéria, jako je cena, estetičnost nebo další metaheuristiky.

V problémech, které budeme chtít řešit evolučními algoritmy, se funkci f obvykle snažíme maximalizovat. Jinými slovy, hledáme takový genom $g^* \in G$, že

$$g^* = argmax_{g \in G} f(g).$$

1.1.7 Selekce

Selekce, neboli také environmentální selekce, je genetický operátor, který simuluje proces přírodního výběru. Tento operátor přiřazuje jednotlivým jedincům jejich schopnosti přežívat a reprodukovat se na základě jejich fitness. Ti nejúspěšnější jedinci jsou vybíráni pro reprodukci, zatímco ti méně úspěšní jsou buď eliminováni nebo mají menší šanci přispět svými geny do další generace. Díky selekci se algoritmus soustředí na oblasti vyhledávacího prostoru s vysokým potenciálem, což vede k rychlejší konvergenci k optimálním řešením.

V kontextu selekce se často mluví o $(\mu + \lambda)$ a (μ, λ) strategiích selekce.

$(\mu + \lambda)$ *selekce* spočívá v tom, že z μ rodičů generujeme λ potomků. Tyto potomky následně spojíme s původními rodiči a z této kombinované skupiny vybereme nejlepších μ jedinců pro další generaci. Tento přístup zajišťuje zachování nejlepších genetických vlastností z předešlých generací a poskytuje pojistku proti

ztrátě kvalitních genů v případě, že nová generace by byla průměrně horší než její předchůdce [3].

Na druhou stranu, (μ, λ) *selekce* vychází z principu, ve kterém μ rodičů generuje λ potomků, ale pouze tito potomci postupují do další generace, což znamená úplné nahrazení původní populace. Tento proces pomáhá efektivněji překonávat lokální minima prostoru řešení, což je velmi cenné v situacích, ve kterých prostor řešení obsahuje mnoho lokálních minim [3].

Jako selekci nejčastěji používáme *ruletovou selekci*, nebo *turnajovou selekci*. Při ruletové selekci jedince g_a vybereme do další generace s pravděpodobností $\frac{f(g_a)}{\sum_g f(g)}$. Při turnajové selekci vybereme náhodně k jedinců z populace (většinou $k = \{2,3,5,10\}$). Z těchto k jedinců postupuje do další populace pouze ten nejlepší.

1.2 Evoluční algoritmus

Příklad velmi jednoduchého evolučního algoritmu může můžeme vidět v algoritmu 1 11.

Algoritmus 1 Jednoduchý evoluční algoritmus

```
1: function EA(Selekce, Křížení, Mutace, Fitness)
2:     p ← náhodně inicializujeme populaci
3:     f ← Fitness(p_1), ..., Fitness(p_n)        ▷ ohodnotíme fitness pro každého jedince
4:     while není dosaženo zastavovací kriterium do
5:         p ← Selekce(p, f)
6:         p ← Křížení(p)
7:         p ← Mutace(p)
8:         f ← Fitness(p_1), ..., Fitness(p_n)
9:     end while
10:    Vrátíme nejlepšího jedince z p
11: end function
```

1.3 Příklad

V této části bychom chtěli ukázat, jak lze navrhnout evoluční algoritmus pro řešení některých vybraných netriviálních problémů.

1.3.1 Problém batohu

Problém batohu je generalizace mnoha industriálních optimalizačních problémů [3]. Představme si, že se balíme na několikadenní túru do hor a s sebou bychom si chtěli zabalit batůžek. Chtěli bychom s sebou mít co nejužitečnější věci, ale zároveň si nemůžeme vzít všechno, protože bychom to neunesli. Problém batohu spočívá v tom, jak si zabalit věci, abychom maximalizovali užitek a zároveň se vešli do našeho stanoveného limitu.

Formálněji se tento problém definuje následovně. Je dána množina n předmětů s hmotnostmi $h_1, \ldots, h_n \in \mathbb{N}$, cenami $c_1, \ldots, c_n \in \mathbb{N}$ a maximální hmotnost $H \in \mathbb{N}$.

Hledáme takovou podmnožinu $p \subseteq \{1, \dots n\}$, pro kterou platí, že $\sum_{i \in P} h_i \leq H$ a zároveň $\sum_{i \in P} c_i$ je co největší.

Jako ilustrační řešení tomuto problému jsme navrhli tyto evoluční operátory a reprezentace:

- Gen bude řetězec $\alpha \in \{0,1\}^n$, kde α_i (i-tý prvek z řetězce) značí, zda jsme se rozhodli vybrat i-tý prvek do množiny p nebo ne.
- Jako selekci jsme zvolili turnajovou selekci.
- Jako mutaci jsme zvolili jednobodové křížení. To v praxi znamená, že nejdříve vybereme náhodný bod sdílený pro oba rodiče. Potomek pak zdědí od jednoho rodiče část řetězce před tímto bodem a od druhého část za ním.
- Fitness funkce f v tomto případě bude $f(\alpha) = \sum_{i=1}^{n} \alpha_i \cdot c_i$ pokud $\sum_{i=1}^{n} \alpha_i \cdot h_i < W$ jinak 0

Výsledky běhu takového algoritmu můžeme vidět na obrázku 4.1

2 Hra Poly Bridge

Poly Bridge je logická simulovaná hra, která byla vyvinuta nezávislým vývojářským týmem Dry Cactus v roce 2016 [4]. Hráči mají za úkol navrhovat mosty, které musí odolat zátěži projíždějících vozidel a zároveň splňovat omezení daná rozpočtem a dostupnými materiály. Díky svému základu v realistické fyzice nabízí Poly Bridge možnost experimentovat s různými stavebními technikami a materiály.

Zajímavým aspektem hry je také komunitní prvek. Hráči mohou sdílet své vlastní návrhy mostů a úrovní online, kde je mohou ostatní hráči testovat a hodnotit. Díky tomu je možné sdílet techniky a nápady mezi hráči z celého světa.

Autoři svojí hru na svých stránkách popsali následovně: *„Popusťte uzdu své inženýrské kreativitě v poutavém a neotřelém simulátoru stavby mostů se všemi zvonky a píšťalkami. Užijte si hodiny řešení fyzikálních hádanek v kampani a pak se vrhněte do sandboxu a vytvářejte vlastní návrhy mostů a hádanky!“* (Dry Cactus přeloženo [4])

2.1 Vlastní hra

Samotná hra se skládá ze 7 kampaní. V každé kampani se nachazí 15 různých úrovní, jejichž obtížnost se postupně zvyšuje.

Po výběru úrovně je hráči představeno prostředí, které se ve většině případů skládá z levého a pravého břehu řeky a jednoho nebo více vozidel. K dispozici je řada materiálů jako je dřevo, vozovka, ocel, hydraulika a další. Materiály se od sebe liší svou nosností, maximální délkou a cenou.

Hráč nejdříve vytvoří svůj návrh a realizuje ho klikáním na předem určené body v prostředí. Kliknutím na dva body se mezi nimi vytvoří vybraný materiál. Pokud dva materiály končí nebo začínají na stejném bodě, vytvoří se mezi nimi volně pohyblivý kloub. V každé úrovni jsou předem některé body vybrané jako kotvy. Tyto kotvy jsou nepohyblivé a hráč na ně může navázat stavbou svého mostu a získat tím oporu pro jeho výtvor.

V momentě, kdy je hráč se svým návrhem a realizací spokojen, může spustit simulaci. Během simulace není možné přidávat další materiály. Vozidla začnou jezdit z jedné strany mostu na druhou a hráč sleduje, zda konstrukce vydrží. Pokud most selže, hráč musí identifikovat slabá místa a provést potřebné úpravy. Ilustraci simulace můžeme vidět na obrázcích 2.1 a 2.1.

Proces návrhu, testování a úprav se opakuje dokud most nevydrží požadované zatížení bez překročení rozpočtu.

Obrázek 2.1 Ukázka ze hry Poly Bridge [4].

Obrázek 2.2 Ukázka ze hry Poly Bridge [4].

3 Implementace

V této části bakalářské práce se zaměříme na praktickou implementaci evolučních algoritmů. Zároveň je nutné implementovat i fyzikální prostředí, podobné tomu, které je ve hře Poly Bridge. V první řadě se zaměříme, aby se naše simulace co nejvěrněji podobala hře, což umožní použít řešení navržené evolučními algoritmy i ve hře. Následně navrhneme několik různých podob evolučního algoritmu pro stavbu mostu a ty mezi sebou porovnáme. Jako programovací jazyk jsme zvolili Python. Kompletní implementaci i se stručnou dokumentací můžeme nalézt na githubu [9].

3.1 Související literatura

Pro návrh evolučních operátorů využijeme inspiraci z existujících studií, které se zaměřily na podobný problém. Konkrétně z diplomové práce Huga Lispectora [10] adoptujeme metodu chytré inicializace mostů. Dále použijeme některé evoluční operátory z programovacího projektu autora AstroSam [11]. Je třeba poznamenat, že obě zmíněné práce se nevěnují přesně stejnému problému, což nám znemožňuje přímé porovnání našich výsledků s těmito studiemi.

3.2 Fyzikální engine

Jako fyzikální engine pro simulaci jsme zvolili Box2D [12]. Box2D je open-source fyzikální engine, který poskytuje simulaci pohybu objektů ve 2D prostoru. Je často využíván ve vývoji počítačových her ale také simulací a umožňuje snadné zpracování kolizí, gravitace, tuhosti objektů a dalších fyzikálních jevů. Tento engine používáme dostupnými knihovnami v Pythonu, ale původně byl implementován v jazyce C++, což snižuje výpočetní náročnost. To nám umožňuje iterovat přes rozsáhlé množství simulací. Pro tento engine jsme se rozhodli, jelikož z dostupných zdrojů víme, že stejný engine použili i vývojáři hry Poly Bridge [13].

3.3 Aproximace hře Poly Bridge

V naší simulaci jsme implementovali různé aspekty hry pomocí následujících komponent knihovny Box2D [14]:

- *Materiály*: Ty jsou modelovány jako dynamické objekty, pro které používáme `Box2D.b2DynamicBody`.
- *Klouby*: Pro spoje různých materiálů jsme využili `Box2D.b2RevoluteJoint`.
- *Zátež na prvky*: Abychom zjistili síly působící na jednotlivé elementy v simulaci, používáme metodu `b2body.GetReactionForce()`, která vrací reakční sílu vzniklou v důsledku interakce těles.

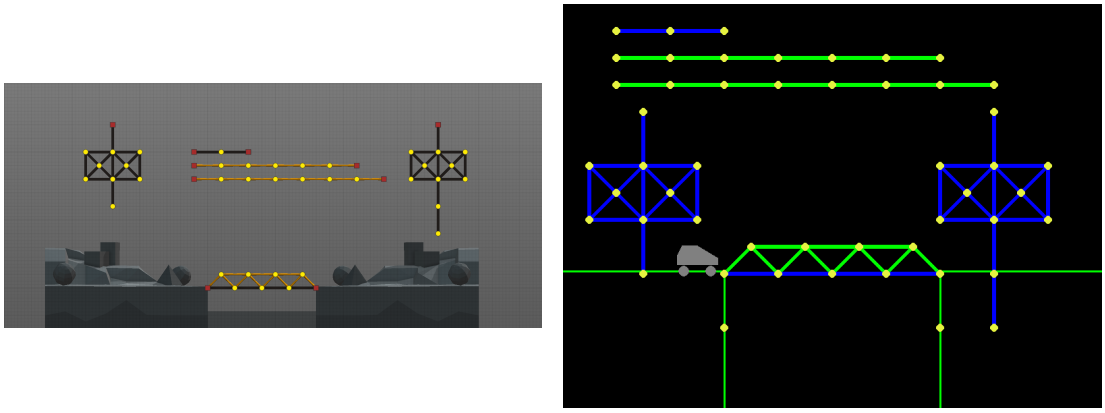
3.3.1 Testy

Abychom v naší simulaci co nejvěrněji napodobili chování fyzikálních prvků jako ve hře, zavedli jsme šest různých testů. Tyto testy zkoumají aspekty fyzikální simulace, jako jsou odolnost materiálů v proměnlivých podmínkách, hmotnost materiálů a interakci sil mezi objekty.

Zvolené testy jsou následující:

1. **2 vozovky mezi dvěma pevnými body:** Očekávaným výsledkem je, že konstrukce praskne pod zatížením samotných vozovek.
2. **6 dřevěných dílů mezi dvěma pevnými body:** Očekáváme, že konstrukce vydrží bez prasknutí.
3. **7 dřevěných dílů mezi dvěma pevnými body:** V tomto testu očekáváme, že konstrukce pod tíhou praskne.
4. **Symetrický obrazec z 13.66 metrů vozovky, zavěšený na jednom kusu vozovky:** Testujeme, zda vozovka unese zatížení bez prasknutí.
5. **Symetrický obrazec z 14.66 metrů vozovky, zavěšený na jednom kusu vozovky:** V tomto případě testujeme, zda konstrukce nevydrží zatížení a praskne.
6. **Komplexní most z vozovek a dřeva, po kterém přejede auto:** Cílem tohoto testu je prozkoumat interakci sil mezi různými materiály, kdy očekáváme, že most vydrží přejetí auta.

Vizualizaci testů můžeme vidět na obrázku 3.1



Obrázek 3.1 Vizualizace testů ve hře polybridge (vlevo) a v simulaci (vpravo). Test č.1 nahoře uprostřed. Test č. 2 a 3 pod test č.1. Nalevo test č.4. Napravo test č.5. Most z 6. testu uprostřed. Vozidlo jede z levého břehu na pravý.

Naše implementace simulace má 4 různé parametry.

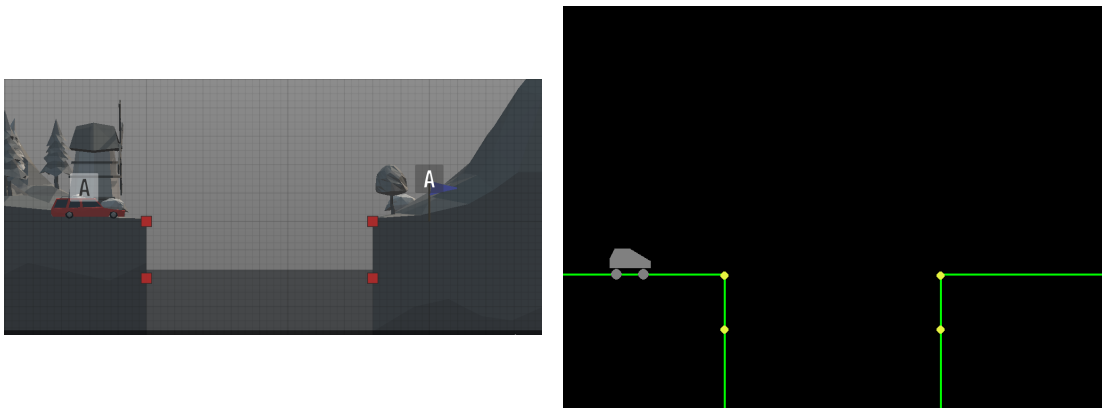
- *Hustota dřeva*: Hustota materiálů pro dřevo (`Box2D.b2Density`).
- *Koeficient hustoty*: Násobek hustoty dřeva, který bude použit pro hustotu vozovky.

- *Max. zatížení dřeva*: Maximální zatížení dřeva.
- *Keoficient zatížení*: Násobek maximálního zatížení dřeva, který bude použit pro maximální zatížení vozovky.

Pro nalezení takových parametrů, které splní nejvíce testů jsme zvolili *Random search* [15]. Prohledaná rozmezí těchto parametrů a jejich nejlepší nalezené hodnoty můžeme vidět v tabulce 3.1. Bohužel se nám nepodařilo najít takové parametry, abychom splnili všechny testy. Rozhodli jsme se že 5. test vyřadíme.

3.3.2 Úrovně

Jako testovací prostředí pro evoluční algoritmus jsme zvolili první 4 úrovně z původní hry (viz obrázky 3.2, 3.3, 3.4 a 3.5). Více úrovní jsme kvůli jejich komplexitě nezahrnuli.



Obrázek 3.2 Vizualizace 1. úrovně ve hře polybridge (vlevo) a v simulaci (vpravo).

3.4 Aplikace evolučních algoritmů

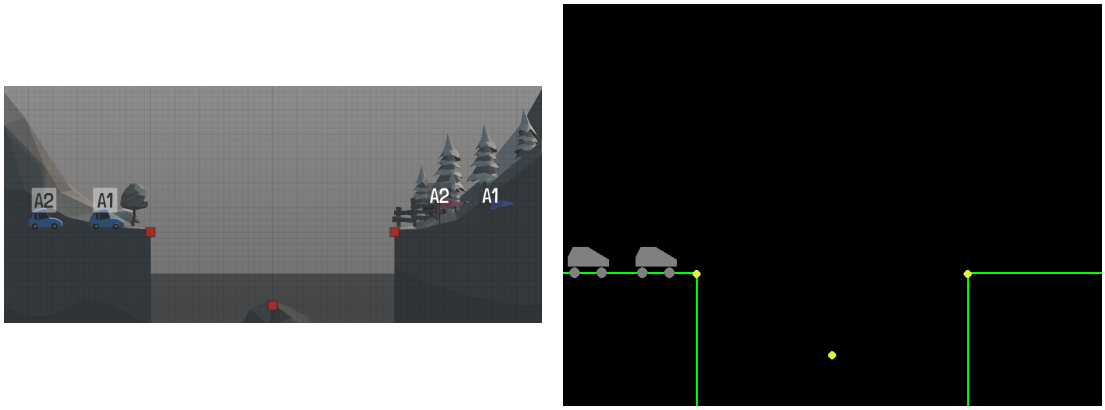
V následující sekci ukážeme, jak jsme navrhli různé typy genetický operátorů. Budeme používat následující značení.

- l_{max} je maximální délka materiálu
- T je množina všech materiálů, které můžeme použít (vozovka, dřevo, nic)

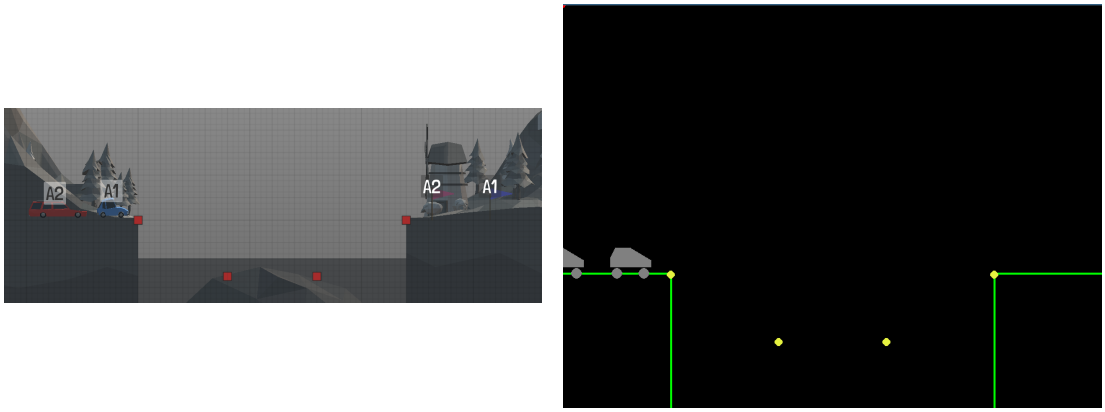
Parametr	**Rozmezí**	**Nalezená hodnota**[a]
Hustota dřeva	$[0{,}01 - 3{,}01]$	1,348
Max. zatížení dřeva	$[50{,}0 - 2050{,}0]$	765,0
Koeficient hustoty	$[0{,}3 - 9{,}3]$	3,992
Koeficient zatížení	$[0{,}1 - 5{,}1]$	0,821

Pozn: [a] Zaokrouhleno na 3 desetinná místa

Tabulka 3.1 Parametry pro simulaci, prohledávaná rozmezí a jejich nejlepší nalezená hodnoty.



Obrázek 3.3 Vizualizace 2. úrovně ve hře polybridge (vlevo) a v simulaci (vpravo).



Obrázek 3.4 Vizualizace 3. úrovně ve hře polybridge (vlevo) a v simulaci (vpravo).

- g je genom jedince, nebo také jeden konkrétní most
- $d_{min}(g)$ minimální vzdálenost vozidla od úrovní definovaného bodu na druhé straně řeky, které se podařilo dosáhnout v simulaci
- $cost(g)$ cena mostu g

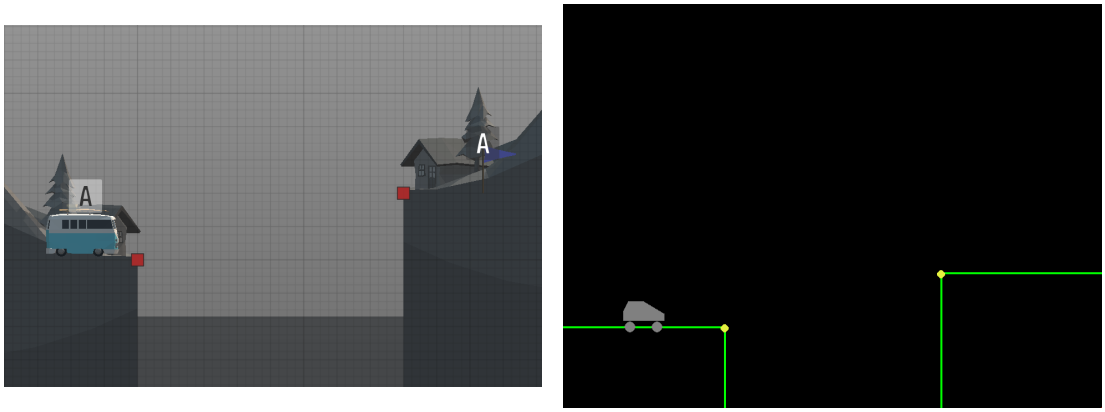
Ve všech případech se snažíme optimalizovat dvě hodnoty a to jak daleko naše vozidlo dojelo a cenu mostu. V rámci algoritmu se tedy primárně snažíme maximalozovat $-d_{max}(g)$ a sekundárně $-cost(g)$.

Naše fitness funkce f bude tedy udávaná dvojicí čísel

$$f(g) = (-d_{min}(g), -cost(g)).$$

3.4.1 Jednoduchý návrh

Nejjednodušší návrh danému problému by mohl vypadat následovně. Reprezentace genu je vektor dvojic čísel $c \in \{([0, \ldots, x_{max}] \times [0, \ldots, y_{max}])\}^n$ kde $x_{max} \in \mathbb{N}$ a $y_{max} \in \mathbb{N}$ je šířka a výška úrovně a vektor $t \in T^n$. Most z genomu poté postavíme následovně. Iterujeme přes všechny dvojice z c a zároveň i přes materiály z t. Mezi bod tvořený současnou dvojicí a poslední bod, na který jsme materiál přidali, se snažíme položit současný materiál. Pokud je současný bod od toho minulého příliš daleko tak jej přiblížíme aby jeho vzdálenost byla l_{max}. Na začátku



Obrázek 3.5 Vizualizace 4. úrovně ve hře polybridge (vlevo) a v simulaci (vpravo).

jako poslední bod vybereme nejvyšší kotvu na levém břehu. Tímto způsobem se snažíme napodobit posloupnost kliknutí, které by hráč normálně provedl.

Jako mutaci jsme zvolili náhodné posunutí pozice kliknutí (bodu) o ± 1 s pravděpodobností $\frac{1}{n}$ a náhodnou změnu materiálu s pravděpodobností $\frac{1}{n}$.Použijeme jednobodové křížení vektoru c a t podle stejně zvoleného náhodného bodu. Pro selekci jsme zvolili turnajovou selekci.

Jak můžeme vidět v experimentu 4.2, algoritmu se nedaří stavět příliš kvalitní mosty. Domníváme se, že tomu tak je z následujících důvodů:

- z principu reprezentace jedince je nepravděpodobné, aby vznikaly krátké hrany, které mohou být klíčové pro kvalitní řešení.
- I malá mutace na začátku genu může mít velký vliv na celkovou strukturu mostu.
- Křížení v naší reprezentaci nedává smysl.
- Fitness funkce nevrací dobrou zpětnou vazbu o kvalitě jedince (viz. obrázek 3.6)

3.4.2 Polární kódování

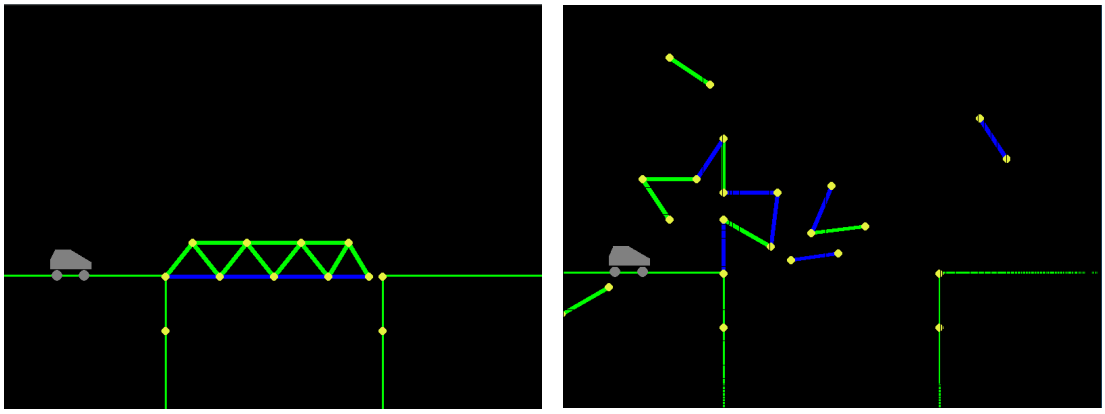
Kvůli tomu, jak jsme reprezentovali jedince v předchozím návrhu, je nepravděpodobné, že budou vznikat krátké hrany, které mohou být zásadní pro dobré řešení. To, že náhodně zvolíme bod blízko od posledního kliknutí je méně pravděpodobné, než že zvolíme bod daleko. Proto jsme navrhli kódování genu, ve kterém dvojce z vektoru c představují délku a úhel přidaného materiálu $c \in \{([0, l_{max}] \times [0, 2\pi])\}^n$. Vektor t zůstává stejný jako v předchozím případě.

Jako mutaci jsme zvolili přepsání hodnoty z c na novou náhodně zvolenou s pravděpodobností $\frac{1}{n}$ a náhodnou změnu materiálu s pravděpodobností $\frac{1}{n}$. Křížení a selekci použijeme stejnou jako v předchozím případě.

3.4.3 Vylepšená fitness funkce

Jedním z problémů, se kterým se náš současný návrh potýká je ten, že naše fitness funkce moc dobře nerozlišuje, jak je dané řešení kvalitní. Na obrázku 3.6

můžeme vidět dva různé jednice, kteří mají stejnou fitness a v kvalitě se značně liší.



Obrázek 3.6 Dva mosty se stejnou fitness, ale rozdílných kvalit.

Navrhli jsme proto dvě různé penalizace, které můžeme do fitness zapojit.

- Penalizace za umisťování materiálů, který se nespojí s dalším materiálem. Lépe propojený most by měl mít lepší stabilitu.
- Penalizace za všechny kotvy, které jedinec nepoužil. V praxi použijeme vzdálenost každého kliknutí ke všem nepoužitým kotvám. Most který používá více kotev by měl být stabilnější.

Nová fitness funkce vypadá následovně:

$$f(g) = (-d_{min}(g) + \alpha \cdot mat + \beta \cdot anch, -cost(g)).$$

Koeficienty $\alpha, \beta \in \mathbb{R}$ značí váhu penalizace a $mat, anch$ jsou hodnoty penalizace za nespojený materiál a nevyužité kotvy.

Použijeme stejnou selekci, mutaci a reprezentaci jedince jako v předchozím návrhu.

3.4.4 Měnící se fitness

V rámci našeho přístupu jsme narazili na specifický problém spojený se stabilitou mostu. Spočívá v tom, že dokud není most zcela dokončen, nedosahuje potřebné stability, které je potřeba pro přejetí vozidlem. Abychom se s tímto omezením vypořádali, rozhodli jsme se pro zjednodušení problému a využití techniky zvané *inkrementální evoluční alogritmus* navržené ve článku Mansouryho Nashaata et al. [16]. Na začátku experimentu proto začínáme s břehy blíže umístěnými k sobě a jakmile dosáhneme dostatečně nízké průměrné fitness v celé populaci, vzdálenost postupně zvětšujeme. To opakujeme dokud nedosáhneme vzdálenosti definové úrovní.

Použijeme stejnou selekci, mutaci a reprezentaci jedince jako v předchozím návrhu.

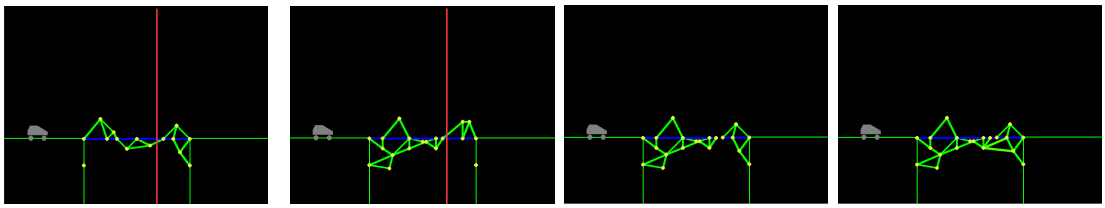
3.4.5 Grafové kódování

V této části bychom chtěli představit odlišný způsob, jak kódovat jednotlivce. Naše dosavadní kódování značně trpí tím, že je nepravděpodobné, aby se v jednom bodě spojilo více než dva kusy materiálu. Tento problém se pokusíme vyřešit tím, že jedince budeme kódovat jako graf, tedy pomocí vrcholů a hran. To v praxi znamená, že gen jedince se skládá z množiny vrcholů V, množiny hran $E \subseteq V \times V$, funkce $\sigma_v : V \to \mathbb{R}^2$ která je projekcí V do roviny a $\sigma_e : E \to T$. Naše navržené genetické operátory vypadají následovně:

- **inicializace**: Do V přidáme všechny kotvy z úrovně. Náhodně vybereme materiál $t \in T$, vrchol $v_1 \in V$, úhel φ a délku $0 < l < l_{max}$. Vytvoříme nový vrchol v_2 a vložíme jej do V. Upravíme σ_v tak, že v_2 se promítne na bod ve vzdálenosti l a pod úhlem φ od $\sigma_v(v_1)$. Vytvoříme novou hranu (v_1, v_2) a přidáme jí do E a zároveň upravíme σ_e tak, že $\sigma_e((v_1, v_2)) = t$. Opakujeme dokud nevytvoříme n nových vrcholů. Následně náhodně volíme dva vrcholy $v_1, v_2 \in V$. Pokud $||\sigma_v(v_1) - \sigma_v(v_2)|| < l_{max}$ vytvoříme novou hranu (v_1, v_2) a přidáme do E. Upravíme $\sigma_e((v_1, v_2)) = t$ pro náhodně zvolené $t \in T$. Opakujeme $2n$-krát.

- **mutace**: Budeme rozlišovat mutaci pro vrcholy a mutaci pro hrany. Mutace pro vrcholy upraví σ_v tak, že projekci vrcholu $v \in V$ přemístí na náhodně zvolený bod z $\{x \in \mathbb{R}^2 | \forall v_2 \in Adj(v_1), ||x - \sigma_v(v_2)|| < l_{max}\}$ kde $Adj(v_1) = \{v \in V | (v_1, v) \in E\}$. Jinými slovy náhodně posumeme vrchol v tak, aby nebyl příliš daleko od žádného vrcholu s nímž byl v spojen hranou. Mutace pro hrany může hranu přidat, odebrat jí nebo změnit σ_e náhodné hrany na jiný typ materiálu.

- **křížení**: Dva jedince můžeme skřížit následovně. Nechť V_1, E_1 je množina všech vrcholů a hran prvního z rodičů a V_2, E_2 druhého. Nechť σ_{v_p} je spojení σ_v funkcí obou rodičů a σ_{e_p} spojení σ_e funkcí obou rodičů. Zvolíme náhodně hranici $min\{\sigma_{v_p}(v)_x | v \in V_1 \cup V_2\} < p_x < max\{\sigma_{v_p}(v)_x | v \in V_1 \cup V_2\}$. Nechť $L = \{v | v \in V_1, \sigma_{v_p}(v)_1 < p_x\}$ a $R = \{v | v \in V_2, \sigma_{v_p}(v)_1 > p_x\}$. Vrcholy genu potomka pak budou z $V_p = R + L$ a hrany $E_p = (E_1 + E_2) \cap V_p \times V_p$, kterým navíc přidáme všechny $(v_1, v_2), v_1 \in R, v_2 \in L, ||\sigma_{v_p}(v_1) - \sigma_{v_p}(v_2)|| < l_{max}$ s pravděpodobností α. σ_v potomka bude σ_{v_p} a stejně tak pro σ_e. Příklad takové mutace můžeme vidět na obrázku 3.7



Obrázek 3.7 Příklad křížení dvou rodičů. Na 1. a 2. obrázku zleva jsou rodiče. Hranice pro náhodné křížení je vyznačena červenou čarou. Na 2. obrázku zprava jsou skřížené částí mosty. Na pravém obrázku překřížení s vystužením.

Jako selekci použijeme turnajovou selekci.

3.4.6 Lepší inicializace

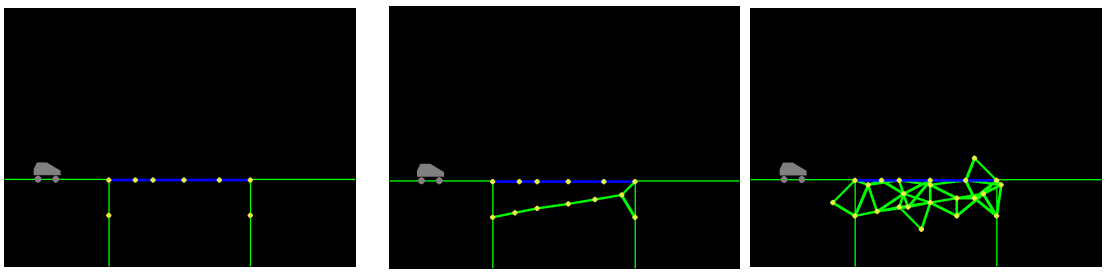
Do našeho algoritmu můžeme ještě zahrnout jednu z nejsilnějších technik pro evoluční algoritmy a to použití *domain-specific* znalostí [17]. Přestože ještě nevíme, jak bude optimální most vypadat, dokážeme obecným způsobem navrhnout most, který sice nebude optimální, ale bude lepší, než náhodně umístěný materiál. Z tohoto důvodu jsme implementovali postup podobný tomu, který navrhl Hugo Lispector ve své diplomové práci [10]. Tento postup se skládá ze tří kroků.

1. **Vytvoření vozovky**: Nejprve pro vozidlo vytvoříme vozovku a to tím způsobem, že spojíme levý a pravý břeh vozovkami o náhodné délce.

2. **Vytvoření opor**: Následně pro každou ještě nevyužitou kotvu vybereme jeden spojový kloub z předchozího kroku a spojíme je dřevěnými díly o náhodné délce.

3. **Zpevnění**: Nakonec pro každý přidaný materiál náhodně vybereme nový bod tak, abychom jej mohli spojit jedním dílem dřeva se začátkem a koncem tohoto materiálu a navíc bod spojíme se všemi ostatními body v blízkém okolí s pravděpodobností ω.

Vizualizaci těchto tří kroků můžeme vidět na obrázku 3.8



Obrázek 3.8 Výtváření jedince pomocí lepší inicializace. Krok **Vytvoření vozovky** vlevo, krok **Vytoření opor** uprostřed a krok **Zpevění** s $\omega = 1$ vpravo.

Použijeme selekci, křížení a mutaci z přechozího návrhu.

4 Experimenty

V následující sekci ukážeme několik experimentů, které jsme provedli s pomocí naší implementace. Experimenty budou vždy porovnávat několik hyper parametrů evolučního algoritmu a graficky ukáží fitness nejlepšího nalezeného jedince v závislosti na počtu vyhodnocení fitness funkce. Na obrázcích s výsledky vždy ukážeme nalevo minimální vzdálenost od cíle a napravo cenu mostu (kromě experimentu s batohem). Jako prostředí simulace jsme zvolili vždy pouze 1. úroveň. Pro každý experiment a každou zmíněnou kombinaci hypermarametrů jsme provedli 5 běhů a jejich výsledek zprůměrovali. Na obrázcích jsme tak znázornili také 95% konfidenční interval. Všechny experimenty jsme provedli na procesoru `AMD Ryzen 5 4500U with Radeon Graphics`. Vyhodnocování fitness jsme paralelizovali na všech 6 jádrech procesoru.

4.1 Jednoduchý příklad s batohem

Experiment demonstruje výsledky běhu jednoduchého evolučního algoritmu na známém problému batohu, jak bylo představeno v sekci 1.3.1 Problém batohu. Výsledky tohoto experimentu jsou prezentovány na obrázku 4.1. Celková doba běhu jednoho experimentu byla přibližně 1 minuta.

Z výsledků můžeme vidět, že běh poměrně rychle zkonvegroval. Už po 50 generacích nelze pozorovat téměř žádné zlepšení. Zda jsme našli skutečně optimální řešení bohužel nemužeme ověřit.

4.2 Naivní přístup

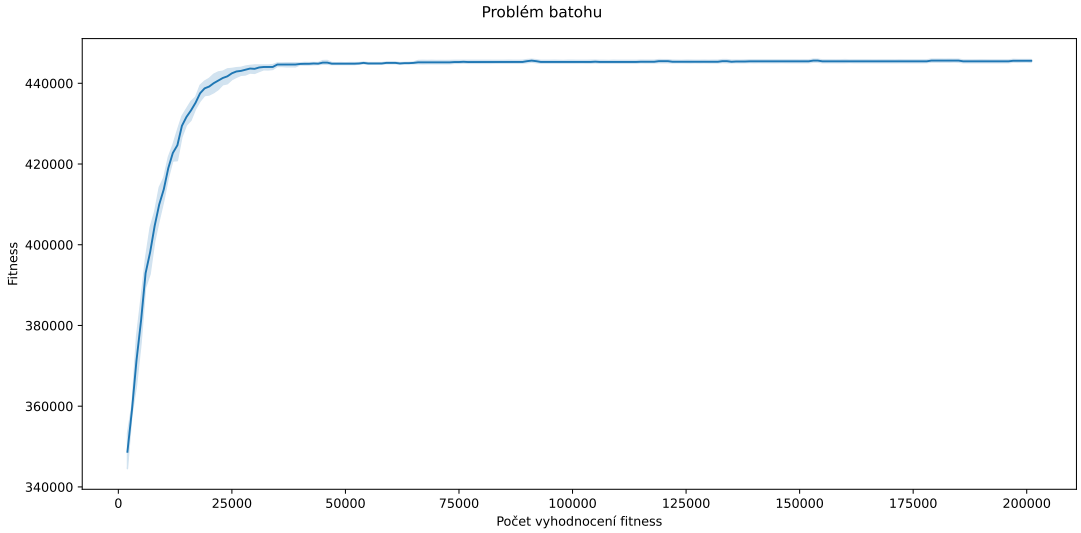
V tomto experimentu chceme ukázat, jakých výsledků lze dosáhnout pomocí jednoduchého kódování jedince. Z podstaty experimentu jsme se rozhodli vyzkoušet pouze jednu sadu hyperparametrů. Výsledek můžeme vidět na obrázku 4.2. Doba jednoho běhu byla přibližně 11 minut.

I přes jednoduchost našeho kódování se nám podařilo najít obstojný most. Tento postup bohužel vždy příliš brzy zkonverguje a tak se nám nedaří úroveň vyřešit.

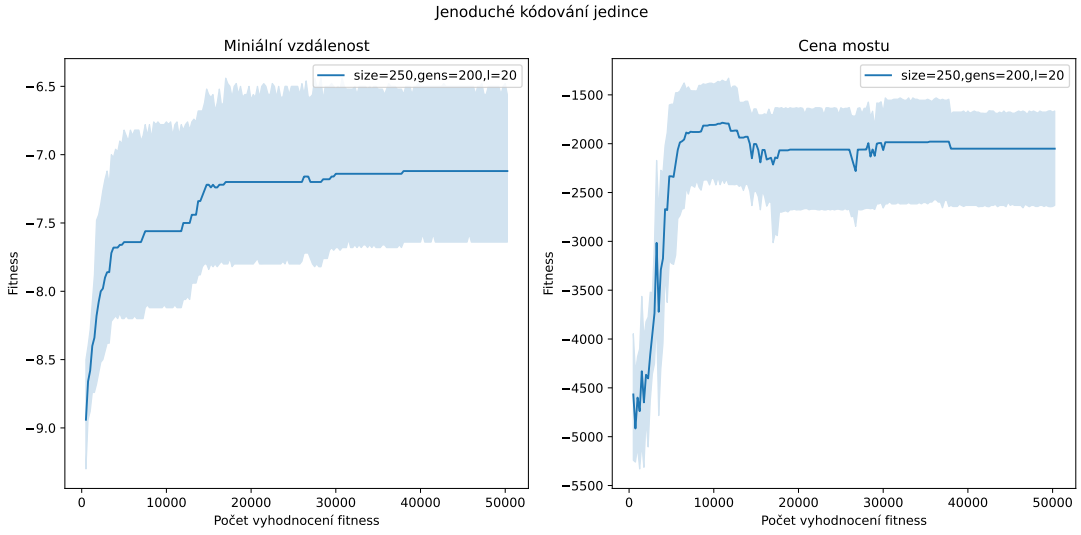
4.3 Různé velikosti populace

Zkoumali jsme vliv různých kombinací velikosti populace a počtu generací na výkon algoritmu. Pro kódování jedinců jsme použili polární kódování. Výsledky jsou zobrazeny na obrázku 4.3. Doba trvání jednoho běhu byla přibližně 11 minut.

Z výsledků lze odvodit, že lepší fitness můžeme dosáhnout tím, že zvýšíme počet generací na úkor velikosti populace.



Obrázek 4.1 Fitness nejlepšího jedince v závislosti na počtu vyhodnocení fitness funkce. Průměr 5 běhů s 95% konfidenčním intervalem. Velikost populace = 1000, počet generací = 200, p. mutace = $1/n$, p. křížení = 1.



Obrázek 4.2 Fitness nejlepšího jedince s jednoduchým kódováním v závislosti na počtu vyhodnocení fitness funkce. Nalevo minimální vzdálenost, napravo cena mostu. Průměr 5 běhů s 95% konfidenčním intervalem. Velikost populace = 250, počet generací = 200, p. mutace = $1/n$, p. křížení = 1, délka jedince = 20.

4.4 Různé délky jedince

Experimentovali jsme s různými délkami genomu pro jedince s polárním kódováním. Výsledky tohoto experimentu můžeme vidět na obrázku 4.4. Délka běhu se lišila v závislosti na délce genomu, ale nepřesáhla 15 minut.

Výsledky tohoto experimentu poukazují na důležitost výběru délky genomu jedince. Příliš krátký genom má za následek nedostatečnou schopnost jedince konstruovat kvalitní řešení, zatímco nadměrná délka vede k nadměrné hmotnosti a zhroucení celého mostu.

4.5 Vylepšená fitness funkce

Cílem tohoto experimentu je prozkoumat, zda vylepšená fitness funkce zmíněná v kapitole 3.4.3 může zlepšit celkový výsledek algoritmu. Zkusili jsme několik různých vah α a β pro penalizace ve fitness funkci pro jedince s polárním kódováním. Výsledek tohoto experimentu můžeme vidět na obrázku 4.5. Doba jednoho běhu byla přibližně 11 minut.

Bylo překvapivé, že náš návrh fitness funkce s penalizacemi nepomohl nalezení lepšího mostu. Z výsledků experimentů se zdá, že algoritmus místo toho, aby optimalizoval strukturu mostu, přestane úplně most stavit. Docílí se tak minimální penalizace a tudíž i lepší fitness a předčasně zkonvergujeme do lokálního minima.

4.6 Populace s elitismem

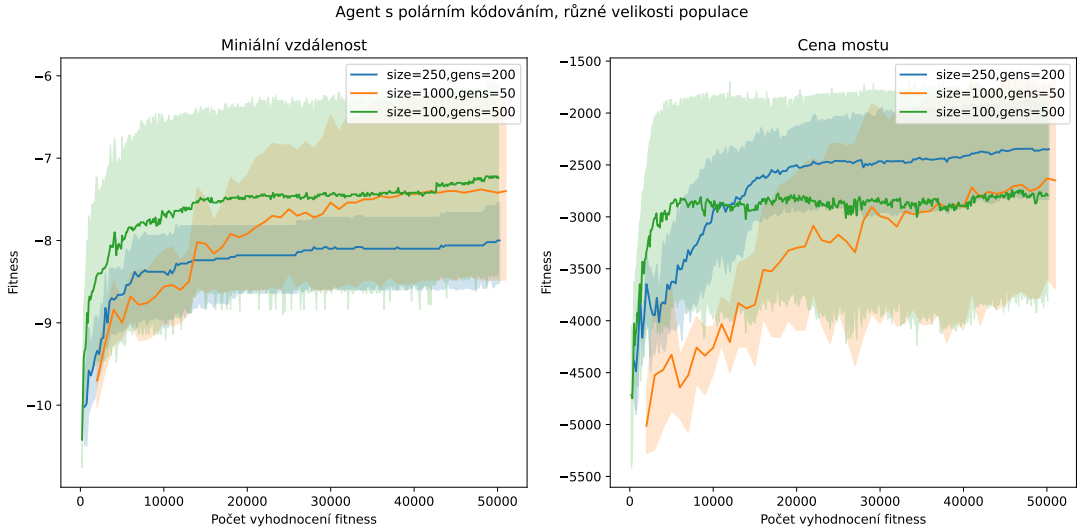
V tomto experimentu jsme vyzkoušeli, jaký vliv má elitismus na běh algoritmu. Zkusili jsme několik úrovní elitismu pro jedince s polárním kódováním. Jejich rozdíly můžeme vidět na obrázku 4.6. Doba jednoho běhu byla přibližně 11 minut.

V tomto experimentu si můžeme všimnout, že elitismus tolik nepomáhá. Zajímavé je si všimnout i toho, že pokud nastavíme sílu elitismu vysoko, můžeme dosáhnout ještě lepších výsledků než bez něj. Tím se nám znovu potvrzuje náš závěr z experimentu 4.3, že můžeme dosáhnout lepších výsledků, pokud budeme upřednostňovat exploataci.

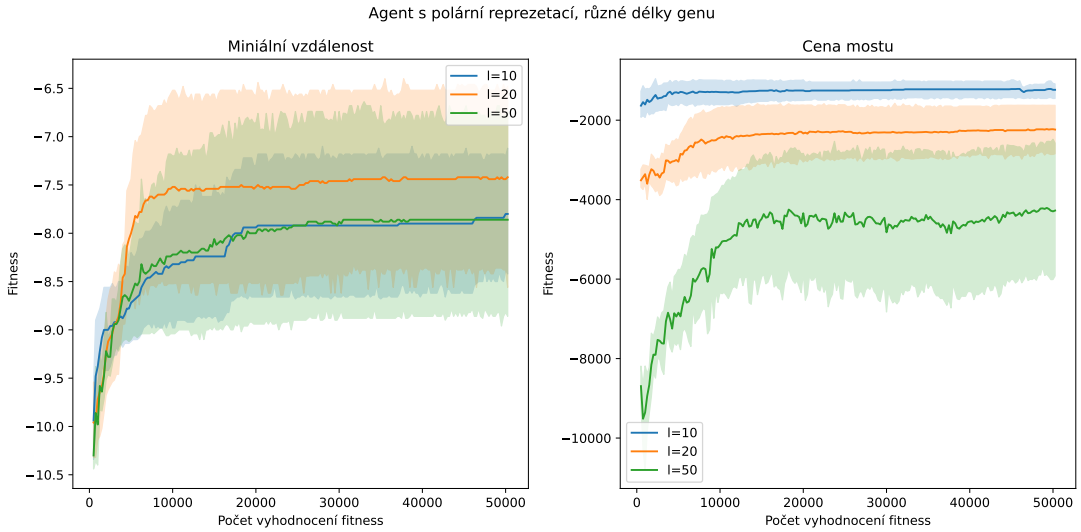
4.7 Ztěžující se fitness

V následujícím experimentu vyzkoušíme, zda pomáhá postupné zvyšování obtížnosti problému. Počáteční vzdálenost břehů je nastavena na 10% původní vzdálenosti. Vyzkoušeli jsme různé hranice průměrné fitness populace nutné na zvýšení obtížnosti. Také jsme zkusili různé velikosti kroků ztěžování. Použili jsme polární kódování jedince. Výsledek tohoto experimentu můžeme vidět na obrázku 4.7. Doba jednoho běhu byla přibližně 11 minut.

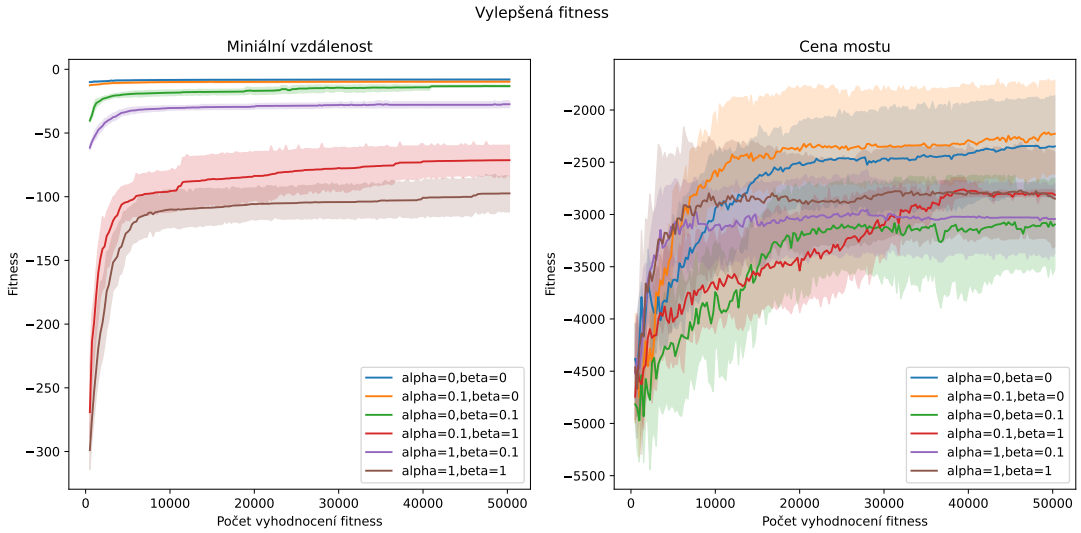
Bohužel se nám touto metodou nepodařilo dosáhnout požadovaných výsledků. Z podstaty metody však můžeme odvodit, že algoritmus nestihl zkonvergovat k suboptimálnímu řešení, tudíž je možné, že kdybychom měli k dispozici více výpočetního času, mohli bychom dosáhnout mnohem lepších řešení.



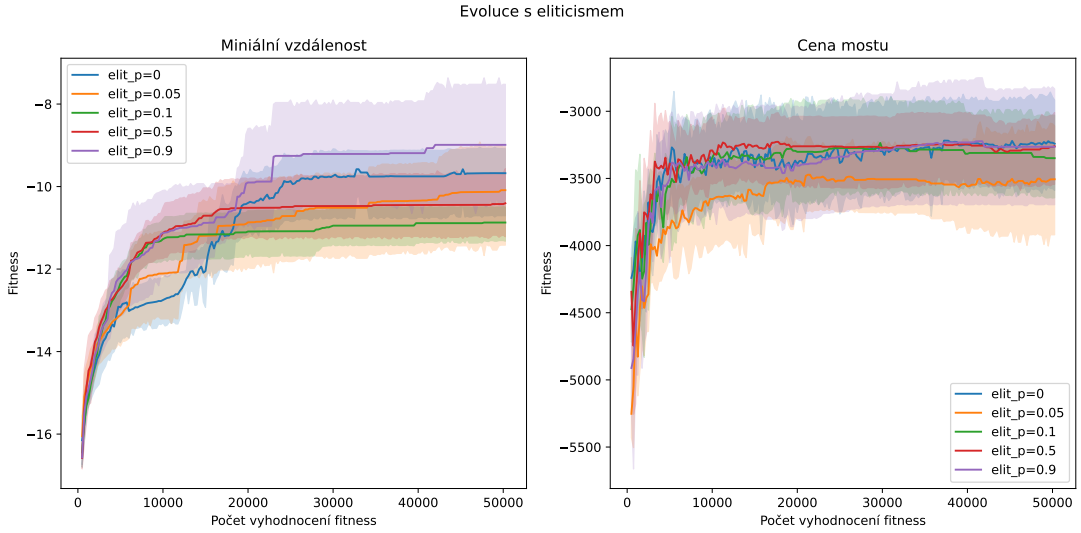
Obrázek 4.3 Fitness nejlepšího jedince s polárním kódováním v závislosti na počtu vyhodnocení fitness funkce. Nalevo minimální vzdálenost, napravo cena mostu. Průměr 5 běhů s 95% konfidenčním intervalem. Velikost populace značená *size*, počet generací značný *gens*, p. mutace = $1/n$, p. křížení = 1.



Obrázek 4.4 Fitness nejlepšího jedince s polárním kódováním v závislosti na počtu vyhodnocení fitness funkce. Nalevo minimální vzdálenost, napravo cena mostu. Průměr 5 běhů s 95% konfidenčním intervalem. Velikost populace = 250, počet generací = 200, p. mutace = $1/n$, p. křížení = 1, délku jedince značí l.



Obrázek 4.5 Fitness nejlepšího jedince s polárním kódováním a vylepšenou fitness funkcí v závislosti na počtu vyhodnocení fitness funkce. Nalevo minimální vzdálenost, napravo cena mostu. Průměr 5 běhů s 95% konfidenčním intervalem. Velikost populace = 250, počet generací = 200, p. mutace = $1/n$, p. křížení = 1, délka jedince = 20, parametry α, β značí *alpha, beta*.



Obrázek 4.6 Fitness nejlepšího jedince s polárním kódováním v závislosti na počtu vyhodnocení fitness funkce. Nalevo minimální vzdálenost, napravo cena mostu. Průměr 5 běhů s 95% konfidenčním intervalem. Velikost populace = 250, počet generací = 200, p. mutace = $1/n$, p. křížení = 1, sílu elitismu značí *elit*.

4.8 Grafové kódování

Několik následující experimentů budou zkoumat efektivitu algoritmu s jedinci, kteří používají grafové kódování. Nejdříve jsme zkoumali, jaký vliv má počet vrcholů na kvalitu jedinců. Výsledky experimentu jsou zobrazeny na obrázku 4.8. Doba běhu se lišila v závislosti na délce jedince, ale nebyla delší než 21 minut.

Je těžké z výsledků tohoto experimentu něco odvodit. Počet hran v genomu jedince se může mutacemi měnit, tím pádem na počátečním počtu vrcholů (tudíž i hran) příliš nezáleží.

4.9 Grafové kódování a ztěžující se fitness

Tento experiment rozšiřuje předchozí testování grafového kódování o postupné ztěžování problému během evoluce. Cílem bylo zjistit, zda kombinace grafového kódování a postupného zvyšování obtížnosti může vést k robustnější adaptaci jedinců na složitější úkoly. Výsledek tohoto experimentu můžeme vidět na obrázku 4.9. Doba jednoho běhu byla přibližně 20 minut.

Z tohoto experimentu můžeme odvodit podobné závěry jako z experimentu 4.7. Z výsledků se nejeví, že by algoritmus zkonvergoval, tudíž je možné, že bychom fitness mohli nadále vylepšovat, kdybychom měli k dispozici více výpočetního času.

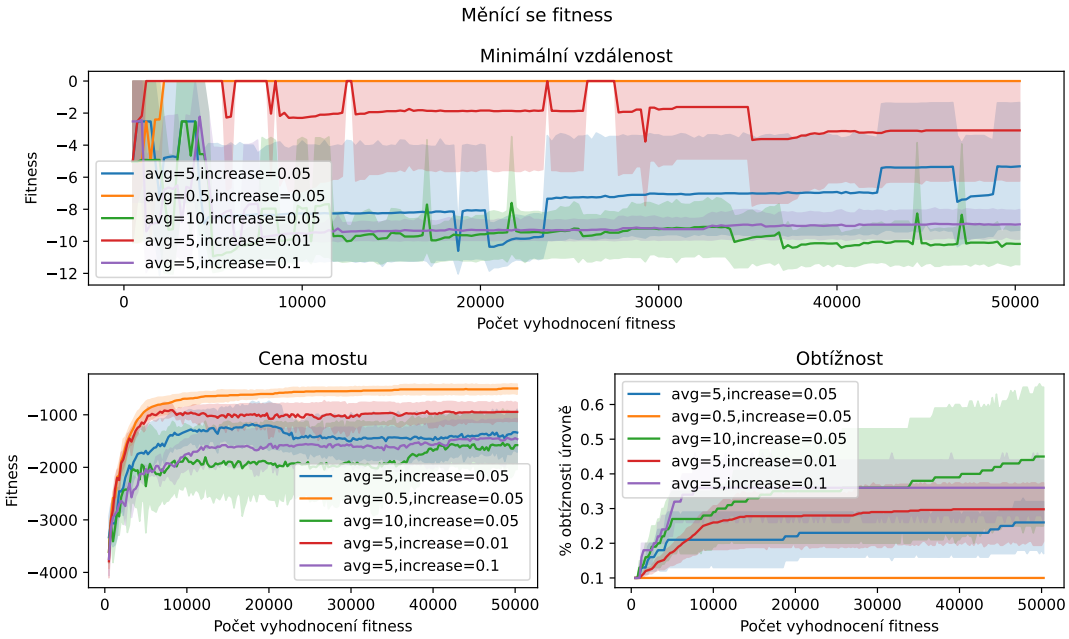
4.10 Lepší inicializace jedince

V tomto exeperimentu ukážeme jaký má vliv vylepšené inicializace jedinců s grafovým kódováním zmíným v kapitole 3.4.6. Vyzkoušeli jsme různé hodnoty ω. Výsledky jsou ilustrovány na obrázku 4.10. Doba běhu se mírně lišila v závislosti na metodě inicializace, ale průměrně trvala 10 minut.

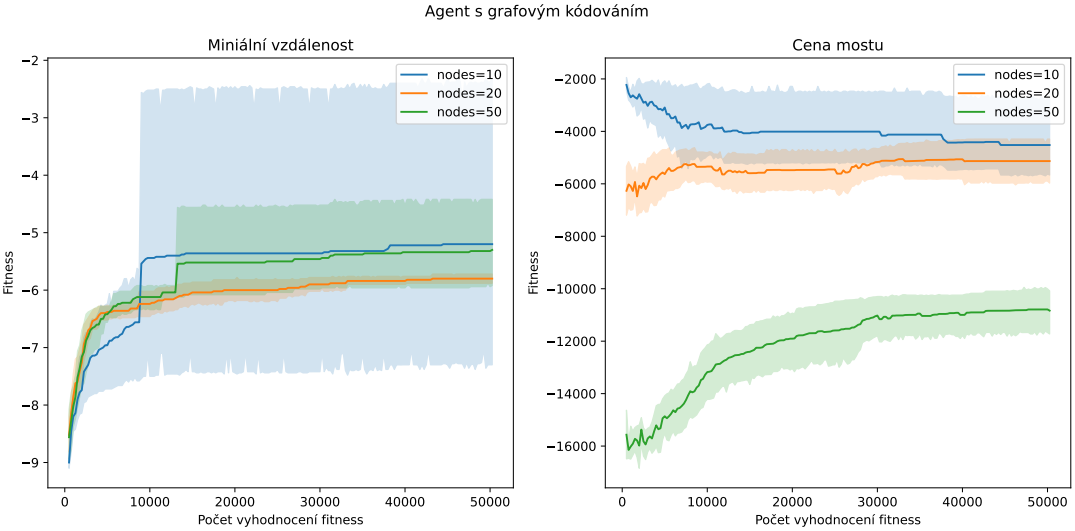
Tento experiment ukazuje, že tímto způsobem inicializace těžkou část problému, tedy dostat vozidlo z jedné strany na druhou, vyřešíme neoptimálním návrhem. Algoritmu pak zbývá pouze tento návrh lehce modifikovat mutacemi a křížením, což se jeví jako vhodnější postup, než se nejprve snažit najít optimální strukturu mostu.

4.11 Přenesení do Poly Bridge

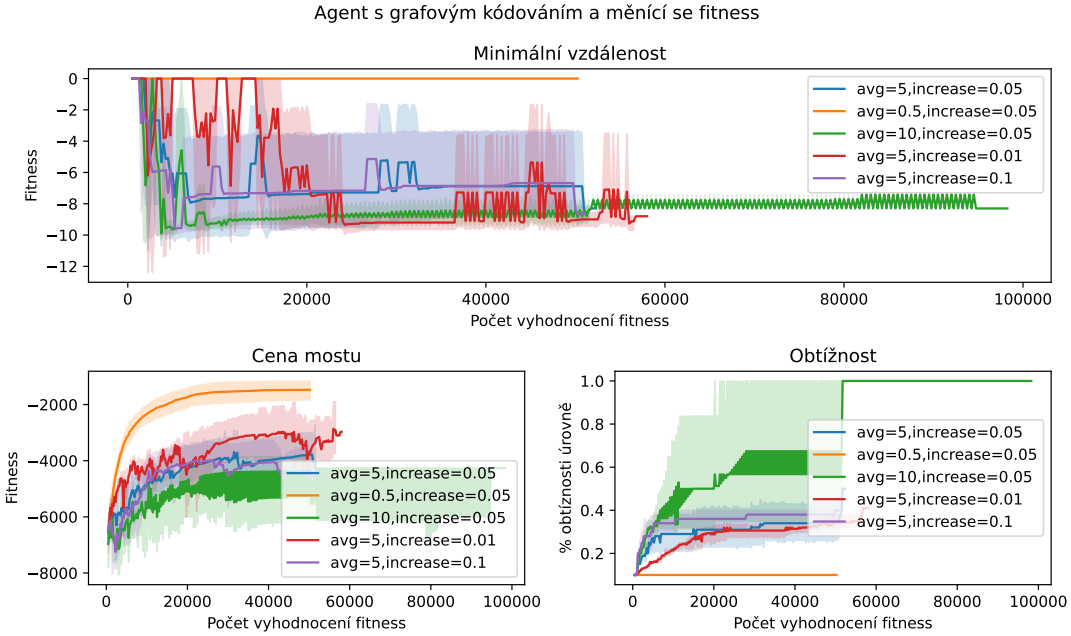
Poslední experiment ukazuje, jak nejlepší nalezená řešení pomocí evolučních algoritmů fungují ve hře. Na obrázcích 4.12, 4.13, 4.14 a 4.15 můžeme vidět porovnání nalezených řešení v simulaci, ve hře a řešení navržených člověkem. Tabulka 4.1 zobrazuje cenu navrženého mostu a zda se nám úroveň podařilo vyřešit evolučním algoritmem. Pro hledání řešení jsme pro všechny úrovně zvolili grafové kódování jedince s vylepšenou inicializací ($\omega = 1$), velikostí populace $size = 500$, počtem generací $gens = 1000$ s nízkým elitismem (0.02). Takto nastavený algorimtus jsme pro každou úroveň zopakovali pětkrát a vybrali nejlepší řešení. Doba jednoho běhu byla zhruba 25 minut. Evolučním algoritmem se nám podařilo vyřešit 3 ze 4 úrovní. U dvou z nich algoritmus navrhl levnější řešení než člověk. Kvůli naší nepřesné simulaci nebylo možné použít všechna řešení.



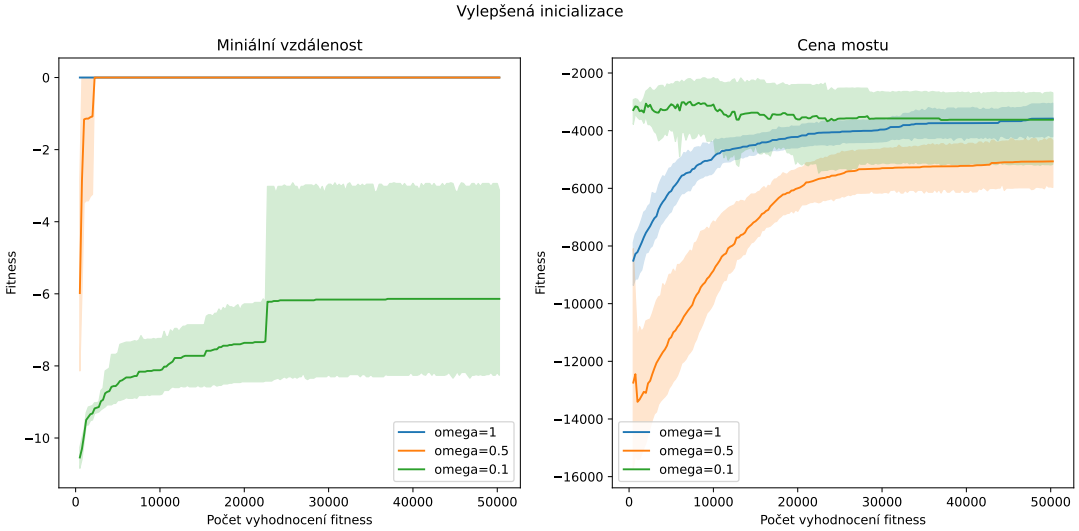
Obrázek 4.7 Fitness nejlepšího jedince s polárním kódováním v závislosti na počtu vyhodnocení fitness funkce. Nahoře minimální vzdálenost, nalevo cena mostu, napravo procento původní obtížnosti. Průměr 5 běhů s 95% konfidenčním intervalem. Velikost populace = 250, počet generací = 200, p. mutace = $1/n$, p. křížení = 1, hranice pro zvýšení obtížnosti značí *avg*, velikost kroku *increase*.



Obrázek 4.8 Fitness nejlepšího jedince s grafovým kódování v závislosti na počtu vyhodnocení fitness funkce. Nalevo minimální vzdálenost, napravo cena mostu. Průměr 5 běhů s 95% konfidenčním intervalem. Velikost populace = 250, počet generací = 200, p. mutace vrcholu = $1/n$, p. mutace hrany = $1/n$, p. křížení = 1, *nodes* značí různé počty použitých vrcholů.



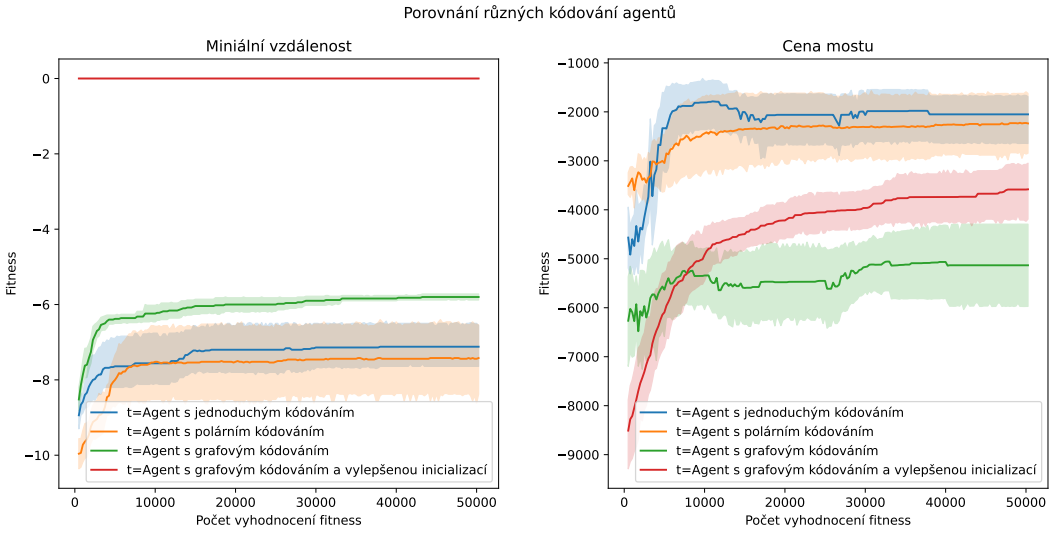
Obrázek 4.9 Fitness nejlepšího jedince s grafovým kódování v závislosti na počtu vyhodnocení fitness funkce. Nahoře minimální vzdálenost, nalevo cena mostu, napravo procento původní obtížnosti. Průměr 5 běhů s 95% konfidenčním intervalem. Velikost populace = 250, počet generací = 200, p. mutace = $1/n$, p. křížení = 1, hranice pro zvýšení obtížnosti značí *avg*, velikost kroku *increase*.



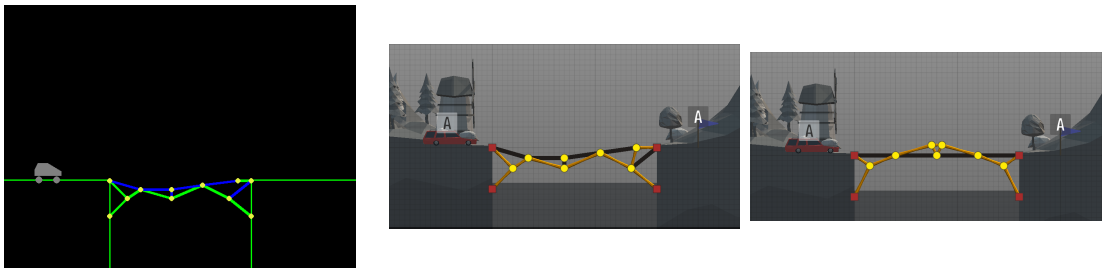
Obrázek 4.10 Fitness nejlepšího jedince s grafovým kódování a vylepšenou inicializací v závislosti na počtu vyhodnocení fitness funkce. Nalevo minimální vzdálenost, napravo cena mostu. Průměr 5 běhů s 95% konfidenčním intervalem. Velikost populace = 250, počet generací = 200, p. mutace vrcholu = $1/n$, p. mutace hrany = $1/n$, p. křížení = 1, *omega* značí různé hodnoty ω.

Úroveň	Vyřešeno algoritmem	Cena $	Cena lidského řešení $
Úroveň 1	Ano	4133	3954
Úroveň 2	Ano	3262	5752
Úroveň 3	Ano	5544	5736
Úroveň 4	Ne	3788	4821

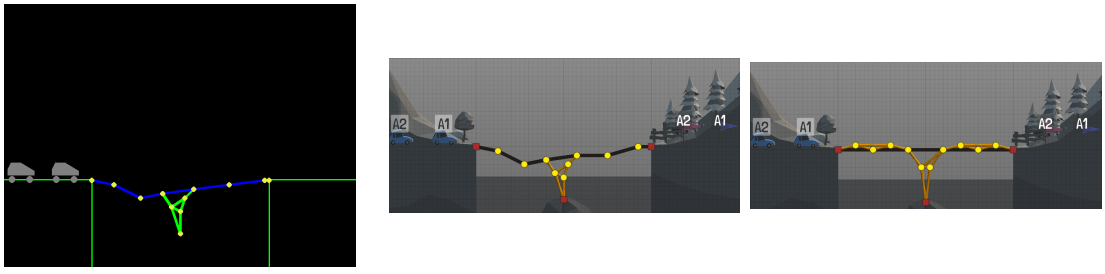
Tabulka 4.1 Ceny mostů navržených evolučním algoritmem v porovnání s lidským řešením.



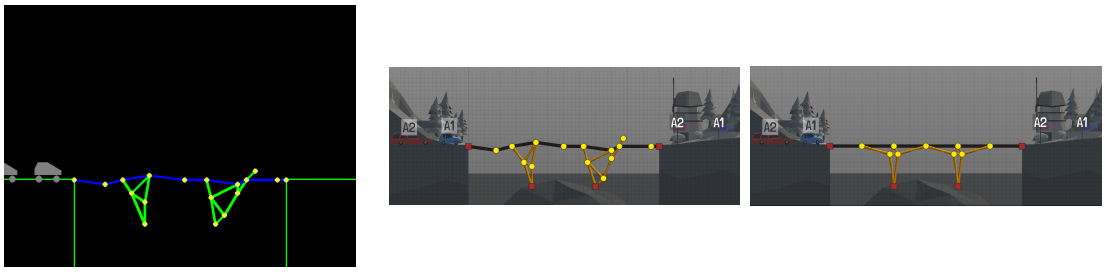
Obrázek 4.11 Porovnání různých kódování jedinců. *t* značí typ jedince.



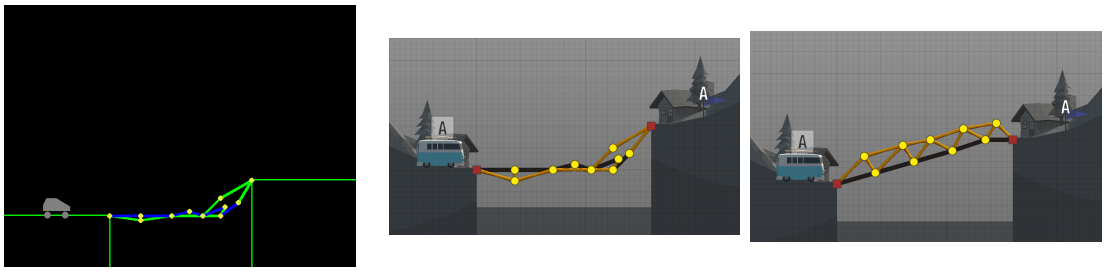
Obrázek 4.12 Řešení první úrovně navržené evolučním algoritmem. Nalevo je řešení zobrazené v simulaci, uprostřed ve hře, napravo řešení navržené člověkem.



Obrázek 4.13 Řešení druhé úrovně navržené evolučním algoritmem. Nalevo je řešení zobrazené v simulaci, uprostřed ve hře, napravo řešení navržené člověkem.



Obrázek 4.14 Řešení třetí úrovně navržené evolučním algoritmem. Nalevo je řešení zobrazené v simulaci, uprostřed ve hře, napravo řešení navržené člověkem.



Obrázek 4.15 Řešení čtvrté úrovně navržené evolučním algoritmem. Nalevo je řešení zobrazené v simulaci, uprostřed ve hře, napravo řešení navržené člověkem.

5 Závěr

V této bakalářské práci jsme zkoumali možnosti využití evolučních algoritmů pro konstrukci mostů ve hře Poly Bridge. Práce ukázala teoretický základ evolučních algoritmů a aplikovala tyto principy na specifické problémy hry. Bylo vytvořeno fyzikální prostředí schopné simulovat chování mostů. Navrhli jsme a implementovali různé genetické operátory, které byly následně testovány pro tvorbu a optimalizaci mostových struktur.

Experimentální výsledky ukázaly, že přístup založený na evolučních algoritmech je schopen efektivně generovat funkční mostní konstrukce, které nejenže splňují požadavky dané úrovně hry, ale často jsou nákladově efektivnější ve srovnání s konstrukcemi vytvořenými člověkem. Bohužel, kvůli naší nepřesné simulaci fyzikálního prostředí nebylo možné použít všechna řešení i ve hře.

Existuje několik možností, jak bychom mohli vylepšit celkové výsledky našich experimentů. Jednou z cest je zdokonalení fyzikální simulace, kterou využíváme. Toho bychom mohli dosáhnout například rozšířením a zpřesněním sady testů nebo experimentováním s různými komponentami enginu Box2D.

Také by bylo užitečné prozkoumat další metody umělé inteligence, jako je zpětnovazební učení, neuronové sítě nebo techniku zvanou *novelty search* [18], které by mohly přinést nové perspektivy a vylepšení našeho výzkumu.

Literatura

1. Mnih, Volodymyr; Kavukcuoglu, Koray; Silver, David; Graves, Alex; Antonoglou, Ioannis; Wierstra, Daan; Riedmiller, Martin. *Playing Atari with Deep Reinforcement Learning*. 2013. Dostupné z arXiv: 1312.5602 [cs.LG].
2. Vinyals, Oriol; Babuschkin, Igor. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*. 2019, roč. 575, č. 7782, s. 350–354. ISSN 1476-4687. Dostupné z DOI: 10.1038/s41586-019-1724-z.
3. Eiben, A. E.; Smith, J. E. *Introduction to Evolutionary Computing*. 2. vyd. Springer-Verlag Berlin Heidelberg, 2015. Natural Computing Series. ISBN 978-3-662-44873-1. Dostupné z DOI: 10.1007/978-3-662-44874-8.
4. Cactus, Dry. *Poly Bridge - Webová stránka*. 2024. Dostupné také z: http://polybridge.drycactus.com/. Přistupováno 28. dubna 2024.
5. Turing, Alan. Intelligent Machinery (1948). In: 1948. ISBN 9780198250791. Dostupné z DOI: 10.1093/oso/9780198250791.003.0016.
6. Holland, John H. Genetic Algorithms. *Scientific American*. 1992, roč. 267, č. 1, s. 66–73.
7. Koza, John R. Genetic programming as a means for programming computers by natural selection. *Statistics and Computing*. 1994, roč. 4, č. 2, s. 87–112. ISSN 1573-1375. Dostupné z DOI: 10.1007/BF00175355.
8. Jones, Terry C. Crossover, Macromutation, and Population-based Search. *ResearchGate*. 1995. Dostupné také z: https://www.researchgate.net/publication/2611224_Crossover_Macromutation_and_Population-based_Search.
9. Krňák, Václav. *Github repozitář pro bakalářskou práci*. 2024. Dostupné také z: https://github.com/NejsemTonda/Bachelor-thesis-/tree/main/code. Přistupováno 28. dubna 2024.
10. Lispector, Hugo. *Truss Bridges Creation and Structural Mass Optimization with Evolutionary Algorithm*. Recife, Brazil, 2022. Dipl. pr. Universidade Federal de Pernambuco. Under the supervision of Paulo Salgado Gomes de Mattos Neto.
11. AstroSam. *Evolving Genetic Neural Network Optimizes Poly Bridge Problems*. 2024. Dostupné také z: https://www.youtube.com/watch?v=qnFCescnO1Q. Přistupováno 28. dubna 2024.
12. Catto, Erin. *Box2D - Webová stránka*. 2024. Dostupné také z: https://box2d.org/. Přistupováno 28. dubna 2024.
13. thread, Polybridge reddit. *How do the physics work?* 2015. Dostupné také z: https://www.reddit.com/r/PolyBridge/comments/3f0naz/how_do_the_physics_work/. Přistupováno 28. dubna 2024.
14. Catto, Erin. *dokumentace Box2D*. 2024. Dostupné také z: https://box2d.org/documentation/. Přistupováno 28. dubna 2024.

15. Bergstra, James; Bengio, Yoshua. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.* 2012, roč. 13, s. 281–305. ISSN 1532-4435.

16. Mansour, Nashaat; Awad, Mohamad; Fakih, khaled. Incremental Genetic Algorithm. *International Arab Journal of Information Technology*. 2006, roč. 3, s. 42–47.

17. Passone, S.; Chung, P.W.H.; Nassehi, V. Incorporating domain-specific knowledge into a genetic algorithm to implement case-based reasoning adaptation. *Knowledge-Based Systems*. 2006, roč. 19, č. 3, s. 192–201. ISSN 0950-7051. Dostupné z DOI: https://doi.org/10.1016/j.knosys.2005.07.007.

18. Lehman, Joel; Stanley, Kenneth O. Abandoning Objectives: Evolution through the Search for Novelty Alone. *Evolutionary Computation*. 2011, roč. 19, č. 2, s. 189–223.

Seznam obrázků

Seznam tabulek

Seznam použitých zkratek

.1 Zdrojový kód

Veškerý kód a výsledky experimentů pro tuto práci lze nalézt v repozitáři https://github.com/NejsemTonda/Bachelor-thesis-