ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta stavební
Katedra betonových a zděných konstrukcí

Optimalizace betonové konstrukce pomocí genetického algoritmu

Diplomová práce

Praha, Květen 2018 Bc. Stanislav Zažirej

Vedoucí práce:
Ing. Martin Petřík, Ph.D.
České vysoké učení technické v Praze
Fakulta stavební
Katedra betonových a zděných konstrukcí
Thákurova 7
166 29 Praha 6
Česká republika

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Zažirej** Jméno: **Stanislav** Osobní číslo: **410726**

Fakulta/ústav: **Fakulta stavební**

Zadávající katedra/ústav: **Katedra betonových a zděných konstrukcí**

Studijní program: **Stavební inženýrství**

Studijní obor: **Konstrukce pozemních staveb**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Optimalizace betonové konstrukce pomocí genetického algoritmu

Název diplomové práce anglicky:

Optimization of concrete structure using genetic algorithm

Pokyny pro vypracování:

Seznam doporučené literatury:

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Martin Petřík, Ph.D., katedra betonových a zděných konstrukcí FSv

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **22.02.2018** Termín odevzdání diplomové práce: **20.05.2018**

Platnost zadání diplomové práce: _____

Ing. Martin Petřík, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Jiří Máca, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

. _____
Datum převzetí zadání

_____
Podpis studenta

Čestné prohlášení

Prohlašuji, že jsem diplomovou práci na téma Optimalizace betonové konstrukce pomocí genetického algoritmu zpracoval samostatně za použití uvedené literatury a pramenů. Dále prohlašuji, že odevzdaná elektronická forma této práce je shodná s listinnou formou.

V Praze, Květen 2018

.....
Bc. Stanislav Zažirej

Poděkování

Chtěl bych poděkovat vedoucímu práce Ing. Martinu Petříkovi, Ph.D. za odborné vedení, ochotu, předané znalosti a zkušenosti, cenné rady a za věnovaný čas. Dále děkuji rodičům za trpělivost a neustálou podporu během celého mého studia.

Abstrakt

Tato diplomová práce se zabývá optimalizací betonové konstrukce pomocí genetického algoritmu. Cílem je implementace algoritmu v jazyce Python a jeho následná aplikace při optimalizaci konkrétních konstrukcí. V první část jsou popsány vybrané metody optimalizačních technik. V další je vysvětlen princip činnosti genetických algoritmů. V poslední části je názorně ukázán způsob implementace algoritmu v Pythonu a ověření jeho funkčnosti při optimalizaci vybraných konstrukcí.

Klíčová slova:
genetický algoritmus, optimalizace, hodnocení, křížení, populace, Python, objektově orientované programování, metoda konečných prvků

Abstract

This diploma thesis deals with an optimization of concrete structure using genetic algorithm. The aim of the thesis is to implement genetic algorithm in Python language and use it for specific structures optimization. In the first part are described chosen methods of optimization. The second explains the principle of operation of genetic algorithms. The last one illustrates the implementation of genetic algorithm in Python and verification of its functionality for specific structure optimization.

Keywords:
genetic algorithm, optimization, fitness, crossover, population, Python, object oriented programming, finite element method

Obsah

Seznam obrázků **xiii**

Seznam tabulek **xv**

Seznam symbolů a zkratek **xvii**

1 Úvod **1**

2 Metody optimalizace **3**
2.1 Principy činnosti výbraných algoritmů 6

3 Genetické algoritmy **13**
3.1 Principy činnosti genetických algoritmů 14
3.2 Genetické operátory 15
3.2.1 Ohodnocení 16
3.2.2 Výběr 17
3.2.3 Křížení 19
3.2.4 Mutace 21
3.2.5 Doplnění 21

4 Implementace v jazyce Python **23**
4.1 Metoda konečných prvků 23
4.2 Generátory stěn 29
4.2.1 Objektově orientované programování 32
4.3 Celkové nastavení GA a operátorů 33
4.3.1 Nastavení ohodnocení 34
4.3.2 Nastavení křížení 36
4.3.3 Nastavení mutace 36
4.4 Příklad 1- symetricky podepřená konstrukce 37
4.4.1 Generátor linií 37
4.4.2 Voronoi generátor 39
4.5 Příklad 2 - nesymetricky podepřená konstrukce 40
4.5.1 Generátor linií 40
4.5.2 Voronoi generátor 41
4.6 Výpočet vybrané varianty v programu RFEM 42

5 Závěr **45**

Literatura **47**

Příloha A **49**
A.1 Příklad 1 49
A.1.1 Generátor linií 49
A.1.2 Voronoi generátor 50
A.2 Příklad 2 51
A.2.1 Generátor linií 51
A.2.2 Voronoi generátor 52

Seznam obrázků

1.1 Ukázka moderní architektury a příkladu použití optimalizce. 1

2.1 Dělení optimalizačních metod dle Vesterstrøma [1]. 4
2.2 Dělení optimalizačních metod dle Zelinky[1]. 4
2.3 Princip horolezeckého algoritmu [2]. 7
2.4 Simulované žíhání. 8
2.5 Princip ACO algoritmu. 9

3.1 Charles Robert Darwin (a), Anaximandros z Milétu (b). 13
3.2 Alan Mathison Turing (a) a John Henry Holland (b). 14
3.3 Dělení evolučních algoritmů. 14
3.4 Obecný cyklus evolučního algoritmu GA [4]. 16
3.5 Ruletový výběr. 18
3.6 Stochastický výběr[5]. 19

4.1 Bázové funkce pro bilineární čtyřúhelníkový prvek. 24
4.2 Lokalizace prvků. 26
4.3 Globální pásová matice tuhosti. 26
4.4 Izoparametrický prvek. 27
4.5 Italský pavilon Expo 2015. 29
4.6 Tvar stěny vytvořený pomocí generátoru linií. 29
4.7 Příklady výskytu Voroného diagramu v architektuře a přírodě. 30
4.8 Tvar stěny vytvořený pomocí Voronoi generátoru. 31
4.9 Definice tvaru stěny pomocí matice. 31
4.10 Lokalizace prvků ve stěně. 31
4.11 Ukázka použití OOP v Pythonu. 32
4.12 Schéma vytvořeného algoritmu. 33
4.13 Cyklus GA v Pythonu. 33
4.14 Graf penalizačních funkcí. 35
4.15 Křížení v Pythonu. 36
4.16 Mutace v Pythonu. 36
4.17 Symetrické podepření - okrajové podmínky pro generátor linií. 37
4.18 Tvar nejlepšího jedince v 9. generaci. 38
4.19 Porovnání nejlepšího jedince v první a poslední generaci. 38
4.20 Vývoj hodnoty fitness pro generátor linií - symetrické podepření. 38
4.21 Symetrické podepření - okrajové podmínky pro Voronoi generátor. 39
4.22 Vývoj nejlepšího jedince populace. 39
4.23 Nesymetrické podepření - okrajové podmínky pro generátor linií. 40
4.24 Vývoj nejlepšího jedince populace. 40
4.25 Vývoj hodnoty fitness pro generátor linií - nesymetrické podepření. 41
4.26 Nesymetrické podepření - okrajové podmínky pro Voronoi generátor. 41

4.27 Vývoj nejlepšího jedince populace. 41
4.28 Vybraná konstrukce pro ověření správnosti výpočtu. 42
4.29 Definice zatížení a tvaru stěny pro výpočet v RFEMu. 42
4.30 Porovnání průhybů. 42
4.31 Porovnání napětí. 43
4.32 Vizualizace možné aplikace stěny v praxi. 43

A.1 Přehled nejlepších jedinců populace ve vybraných generacích. 49
A.2 Přehled nejlepších jedinců populace ve vybraných generacích. 50
A.3 Přehled nejlepších jedinců populace ve vybraných generacích. 51
A.4 Přehled nejlepších jedinců populace ve vybraných generacích. 52

Seznam tabulek

2.1 Analogie procesu žíhání a kombinatorické optimalizace 7

3.1 Binární kódování jedinců. 16
3.2 Ohodnocení jedinců. 16
3.3 Pravděpodobnost výběru jedinců. 18
3.4 Turnajový výběr jedinců. 19
3.5 Jednobodové křížení. 20
3.6 Dvoubodové křížení. 20
3.7 Uniformní křížení. 20
3.8 Binarní mutace genů. 21

4.1 Gaussovy integrační body a váhy. 27

Seznam symbolů a zkratek

E Youngův modul
N Bázová funkce
γ_{xy} Smyková deformace ve rovině xy
ν Poissonův součinitel
σ_x Normálové napětí ve směru x
σ_y Normálové napětí ve směru y
$\sigma_{1,2}$ Hlavní napětí
τ_{xy} Smykové napětí ve rovině xy
ε_x Normálová deformace ve směru x
ε_y Normálová deformace ve směru y
k Lokální matice tuhosti

ACO Ant Colony Optimization

BIM Building Information Model neboli Informační model budovy

EVT Evoluční výpočetní techniky

GA Genetický algoritmus

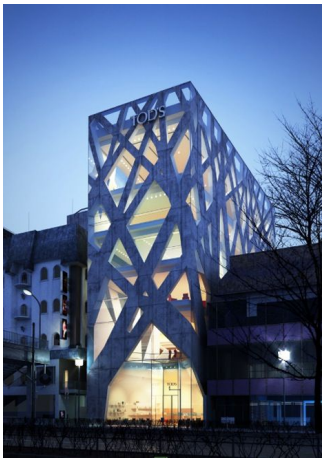
MKP Metoda konečných prvků

OOP Objektově orientované programování

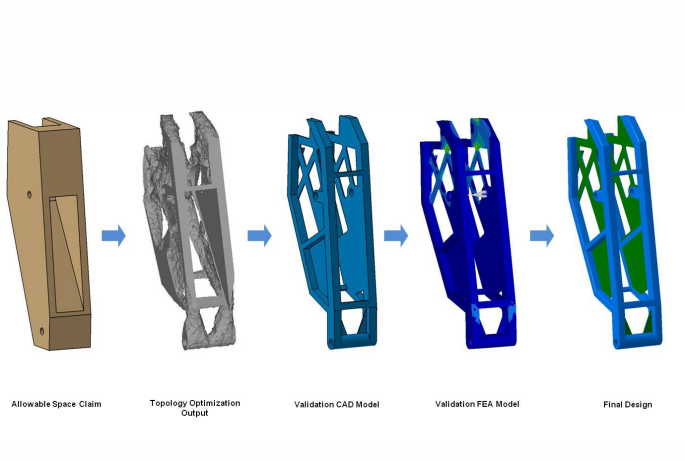
Kapitola 1

Úvod

Nové trendy v architektuře a nástup technologií BIM a 3D tisku mění přístup k návrhu staveb. To klade nové požadavky na projektanty a statiky například v podobě snížení objemu použitého materiálu nebo nákladů na provoz budovy. Motivací pro danou práci byl zájem naučit se používat zajímavou optimalizační metodu, která by vedla k vyřešení daných problému.



(a) Budova obchodu TOD's v Tokiu.



(b) Optimalizace konstrukčního prvku.

Obrázek 1.1: Ukázka moderní architektury a příkladu použití optimalizce.

Cílem této práce je vytvoření genetického algoritmu v jazyce Python a jeho aplikace při optimalizaci betonové konstrukce. Jako předmět optimalizace byla vybrána nosná stěna. Aby algoritmus pracoval efektivně, je nutné do něj zahrnout nástroj pro tvorbu náhodných tvarů stěn a nástroj pro statický výpočet konstrukcí.

Kapitola 2

Metody optimalizace

Optimalizační algoritmy jsou výkonným nástrojem pro řešení celé řady problémů inženýrské praxe a obvykle se používají v případech, kdy řešení daného problému analytickou cestou je nevhodné či nereálné. Většina těchto problémů může být definována jako optimalizační úloha převedena na matematickou úlohu, která je dána vhodným funkčním předpisem. Optimalizace tohoto předpisu vede k nalezení argumentů účelové funkce.[1] Pro řešení takových problémů byla vyvinuta třída výkonných algoritmů, které umožňují řešit velmi efektivně složité problémy. Algoritmy této třídy nesou název „evoluční algoritmy".

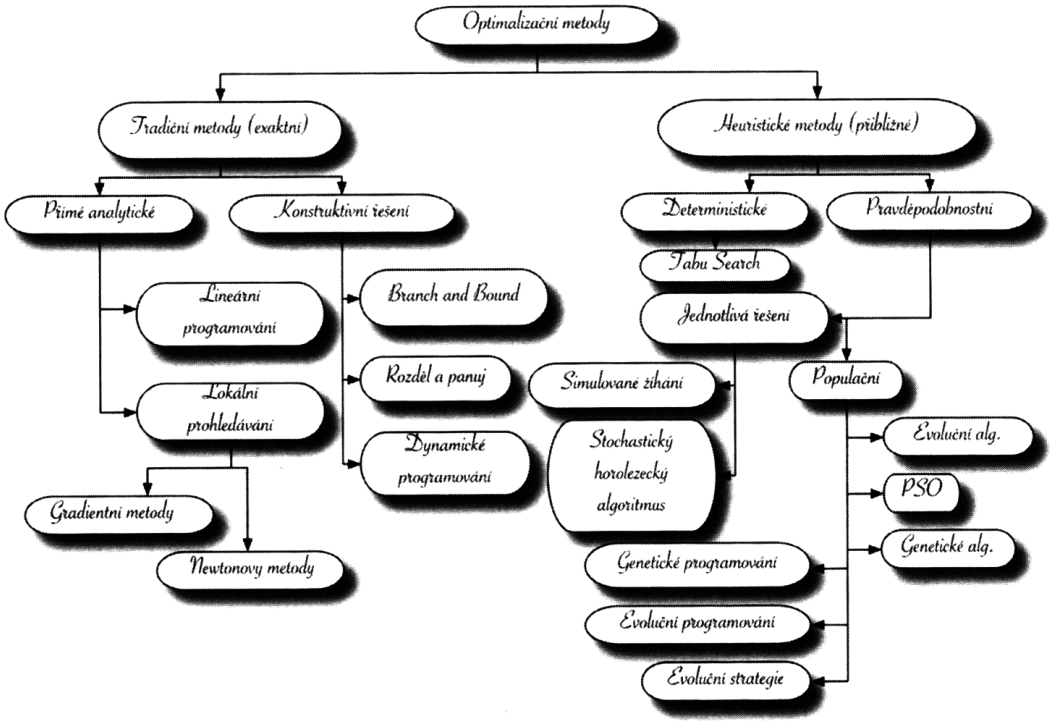
Evoluční algoritmy patří k heuristickým algoritmům,které můžeme rozdělit na deterministické a stochastické (viz. Obr. 2.1). Algoritmy druhé skupiny se liší v tom, že některé jejich kroky využívají náhodné operace. To znamená, že výsledky řešení, které získáme, se při každém spuštění programu mohou lišit. Proto má smysl spustit program vícekrát a vybrat nejlepší získané řešení.

Stochastické heuristické metody poskytují pouze obecný rámec a jednotlivé operace algoritmu je třeba definovat(např. operaci křížení a mutace u genetických algoritmů, operaci sousedství u simulovaného žíhaní, atd.) v závislosti na konkrétním problému. Jelikož se tyto metody často inspirují přírodními procesy, označují se jako evoluční algoritmy. Podle strategie je lze rozdělit do dvou tříd:

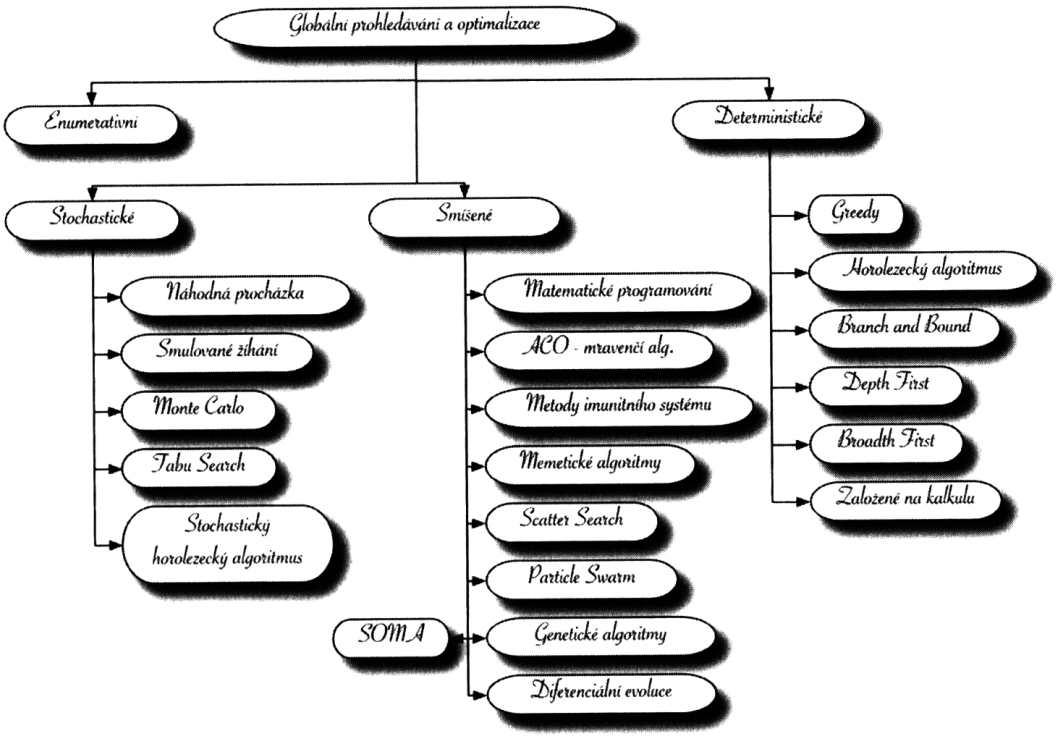
- Metody založené na bodové strategii, např. simulované žíhání, horolezecký algoritmus a zakázané prohledávání. Základem těchto operací je operace sousedství aktuálního řešení, v němž hledáme řešení lepší.
- Metody založené na strategii populace. Sem patří genetické algoritmy.

Tyto metody se liší od klasických gradientních metod tím, že připouštějí přijetí horšího řečení do další iterace. Tím se snaží vyhnout uváznutí v lokálním optimu. Lze najít celou řadu heuristických metod, např. neuronové sítě, Lagrangeova relaxační metoda.

[1] veličina, jejíž minimalizací nebo maximalizací hledáme optimální řešení a která je rozhodující pro posouzení řešení daného problému.



Obrázek 2.1: Dělení optimalizačních metod dle Vesterstrøma [1].



Obrázek 2.2: Dělení optimalizačních metod dle Zelinky[1].

Optimalizační algoritmy slouží k nalezení minima zadané funkce tak, že hledají optimální numerickou kombinaci jejich argumentů. Tyto algoritmy lze rozdělit na základě principů jejich činnosti, podle složitosti apod. Tato rozdělení však nejsou jediná možná mohou se mezi sebou lišit. Různé pohledy na klasifikaci evolučních algoritmů demonstruje Obr. 2.1 a Obr. 2.2. Jednotlivé třídy algoritmu představují obecně způsoby řešení daného problému metodami s různým stupněm efektivity a složitosti. Podle jejich vlastnosti dělíme algoritmy do těchto kategorií [1]:

- **Enumerativní**
  Algoritmus provede výpočet všech možných kombinací daného problému. Tento přístup je vhodný pro problémy, u nichž jsou argumenty účelové funkce diskrétního charakteru a nabývají malého množství hodnot. Pokud by byl použit obecně, je možné, že pro úspěšné ukončení by potřeboval čas, který je delší, než existence našeho vesmíru.

- **Deterministické**
  Tato skupina algoritmů je postavená pouze na rigorózních metodách klasické matematiky. Algoritmy tohoto charakteru obvykle vyžadují omezující předpoklady, které těmto metodám umožňují podávat efektivní výsledky. Tyto předpoklady obvykle jsou:

  – problém je lineární, konvexní
  – prohledávaný prostor možných řešení je malý a souvislý
  – účelová funkce, pokud možno, má pouze jeden extrém
  – mezi parametry účelové funkce nejsou nelineární interakce
  – problém je definován v analytickém tvaru

  Výsledkem deterministického algoritmu je pak jediné řešení

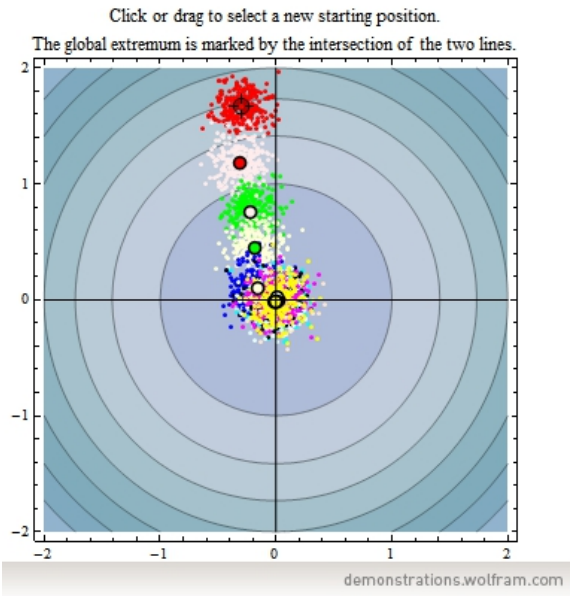
- **Stochastické**
  Algoritmy tohoto typu jsou založeny na využití náhody. Jde v podstatě o čitě náhodné hledání hodnot argumentů účelové funkce s tím, že výsledkem je vždy to nejlepší řešení, jenž bylo nalezeno během celého náhodného hledání. Algoritmy tohoto typu jsou obvykle:

  – pomalé
  – vhodné jen pro malé prohledávané prostory možných řešení
  – vhodné pro hrubý odhad

- **Smíšené**
  Algoritmy této třídy představují směs metod deterministických a stochastických, které ve vzájemné spolupráci dosahují překvapivě dobrých výsledků. Poměrně silnou podmnožinou těchto algoritmů jsou již zmíněné evoluční algoritmy. Algoritmy smíšeného charakteru jsou:
  - robustní, což znamená, že nezávisle na počátečních podmínkách velmi často najdou kvalitní řešení, jenž je reprezentováno obvykle jedním či více globálními extrémy
  - efektivní a výkonné, což znamená, že jsou schopny nalézt kvalitní řešení během relativně malého počtu ohodnocení účelové funkce
  - jsou odlišné od čistě stochastických metod
  - mají minimální nebo žádné požadavky na předběžné informace
  - jsou schopné pracovat s problémy typu „černá skříňka“, tzn. nepotřebují ke své činnosti analytický popis problému
  - jsou schopny nalézt více řešení během jednoho spuštění

2.1 Principy činnosti výbraných algoritmů

1. **Stochastický horolezecký algoritmus** (Stochastic Hill-Climbing)
   Je to verze tzv. horolezeckého algoritmu, která obohacená o stochastickou složku. Patří mezi gradientní metody. To znamená, že prohledává prostor možných řešení ve směru největšího spádu. Funguje následujícím způsobem. Vždy se vychází z náhodného bodu v prostoru možných řešení. Pro momentálně navržené řešení se navrhne určité okolí a daná funkce se minimalizuje jen v tomto okolí. Získané lokální řešení se pak použije jako střed pro výpočet nového okolí. Celý proces se iterativně opakuje. Během procesu se zaznamenává nejlepší nalezené řešení. Po ukončení slouží toto řešení jako nalezené optimum. Nevýhodou tohoto algoritmu je to, že v něm může za určitých podmínek dojít k zacyklení a řešení tak uvízne v lokálním extrému.
   Na Obr. 2.3 je vidět, že terč v horní části obrázku je startem algoritmu. Pro něj je vygenerována množina červených řešení. Nejlepší řešení je označeno tučným červeným bodem, který slouží jako nová pozice pro vygenerování nové (bílé) množiny řešení. Nejlepší řešení je označeno jako bílý tučný bod a ten je pak použit pro vygenerování zelené množiny řešení



Obrázek 2.3: Princip horolezeckého algoritmu [2].

2. **Simulované žíhání**

   Simulované žíhání patří mezi algoritmy, které připouští kroky, po nichž dojde ke zhoršení hodnoty účelové funkce. Inspirace k jeho formulaci byla nalezena ve statistické mechanice v popisu fyzikálního procesu *žíhání tuhého tělesa*. Při žíhání kovů s nestabilní krystalovou mřížkou dochází ke stabilizaci volných částic k optimálnímu stavu, tj. k vytvoření stabilní krystalové mřížky. Těleso se nejdříve zahřeje na vysokou teplotu. Tím se jeho atomům umožní dostat z lokálních minim vnitřní energie mezi stavy s vyšší energií (do rovnovážných poloh) a postupným snižováním teploty se polohy atomů fixují. To znamená, že při konečné teplotě žíhání, která je podstatně nižší než byla počáteční, jsou všechny atomy v rovnovážných polohách a těleso neobsahuje žádné vnitřní defekty ani pnutí. Takový kov pak má mnohem lepší vlastnosti. Simulované žíhání je nejrozšířenější metoda pro registraci MRI obrazů.

   Tabulka 2.1: Analogie procesu žíhání a kombinatorické optimalizace

žíhání	**kombinatorická optimalizace**
stav systému	přípustné řešení
energie stavu systému	hodnota účelové funkce
změna stavu systému	přechod k sousednímu řešení
teplota	řídící parametr
ustálený stav	heuristické řešení

   Hledání globálního minima funkce může být realizováno podobným způsobem. Vnitřní energii nahrazuje účelová funkce a teplotu řídící parametr (viz. Tab. 2.1). Simulované žíhání postupně „ochlazuje“ průběžný stav. V každém kroku najde náhodný bod v okolí současného stavu, porovná jeho „energii“ se současným stavem a s určitou pravděpodobností závislou na tomto rozdílu a na „teplotě“ se do

tohoto nového stavu přesune.Tato pravděpodobnost není nulová v případě, že nový stav je horší než původní, proto tato metoda při dostatečné teplotě neuvázne v lokálním minimu.

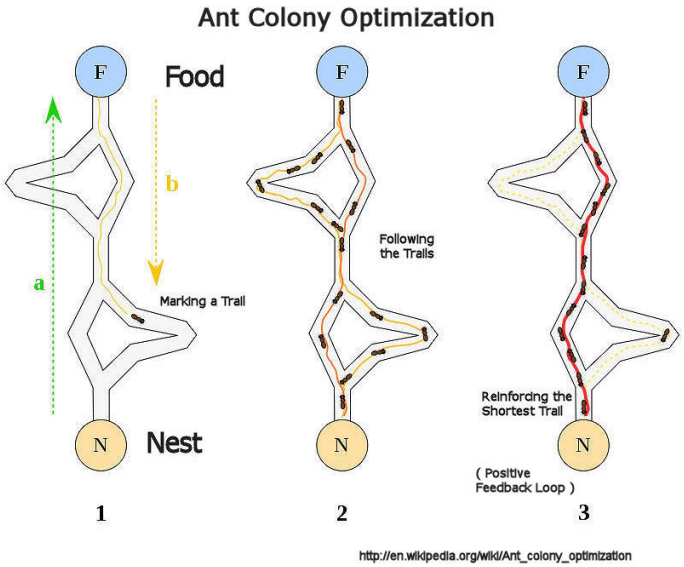
Obrázek 2.4: Simulované žíhání.

3. **Zakázané prohledávání** (Tabu Search)
   Je to vylepšená verze horolezeckého algoritmu, do které byla zavedena tzv. krátkodobá paměť. Jejím úkolem je pamatovat si ty transformace, pomocí kterých byl vypočítán aktuální střed. To má za následek, že nedochází k zacyklení díky zakázanému použití těchto transformací (odtud název „zakázané prohledávání“). Tato metoda byla vylepšena ještě o tzv. dlouhodobou paměť, která obsahuje transformace, které nejsou v paměti krátkodobé, ale byly často použity. Jejich použití je pak penalizováno. To pak snižuje četnost jejich použití. Na rozdíl od horolezeckého algoritmu nedochází tak často k uvíznutí v lokálních extrémech.

4. **Evoluční strategie** (Evolutionary strategy)
   Tento algoritmus patří mezi první úspěšné stochastické algoritmy v historii. Byl navržen počátkem šedesátých let. Vychází z principu přirozeného výběru podobně jako genetické algoritmy. Na rozdíl od jiných stochastických algoritmů pracuje evoluční strategie přímo s reálnými hodnotami. Jejím jádrem je práce s řešením ve formě vektoru x, které je mutováno pomocí vektoru náhodných čísel. Problém akceptace nového řešení je striktně deterministický.

5. **Optimalizace mravenčí kolonií** (Ant Colony Optimization (ACO))
   Je to algoritmus, jehož činnost napodobuje chování mravenců v kolonii. Je založen na následujícím principu. Je definován zdroj mravenců (mraveniště) a cíl jejich snažení (potrava). Když jsou vypuštěni, tak po nějaké době dojde k tomu, že všichni mravenci se pohybují po kratší (optimální) cestě mezi zdrojem a cílem (viz. Obr. 2.5). To je dáno tím, že si svou cestu značkují feromonem. Pokud dorazí první mravenec k rozcestí dvou cest, které vedou ke stejnému cíli, pak je jeho rozhodnutí, po které cestě se vydá, náhodné. Ti, kteří najdou potravu, začnou cestu značkovat a při návratu jsou díky těmto značkám při rozhodování ovlivnění ve prospěch této cesty. Při návratu ji označují podruhé, což opět zvyšuje pravděpodobnost rozhodnutí dalších mravenců v jejich prospěch. Tyto principy jsou použity v ACO algo-

ritmu. Feromon je zde zastoupen váhou, která je přiřazena dané cestě vedoucí k cíli. Tato váha je aditivní, což umožňuje přidávat další „feromony“ od dalších „mravenců“. V ACO algoritmu je zohledněn i fakt vypařování feromonů tak, že váhy u jednotlivých spojů s časem slábnou. To zvyšuje robustnost algoritmu z hlediska nalezení globálního extrému. „aco“ byl použit s úspěchem na optimalizační problémy, jako je problém obchodního cestujícího či návrh telekomunikačních sítí.[1]



Obrázek 2.5: Princip ACO algoritmu.

6. **Metoda imunitního systému** (Immunology System Method)
   Tento algoritmus je založen na principech fungování imunitního systému v živých organizmech. Na imunitní systém je nahlíženo jako na multiagentní systém, kde jednotliví agenti mají svůj specifický úkol. Tito agenti mají různé pravomoci a schopnosti komunikovat s jinými agenty. Na základě této komunikace a určité „svobody“ v rozhodování jednotlivých agentů vzniká hierarchická struktura schopná řešit komplikované problémy. Jako příklad použití metody lze uvést antivirovou ochranu u velkých a rozlehlých počítačových systémů.

7. **Memetické algoritmy** (Memetic Algorithms)
   Významnou charakteristikou těchto algoritmů je použití různých aproximačních algoritmů, technik lokálního vyhledávání, speciálních rekombinačních operátorů apod. Memetické algoritmy mohou být charakterizovány jako strategie soutěže a spolupráce, která projevuje atributy synergetiky. Jako příklad lze uvést hybridní kombinaci genetických algoritmů a simulovaného žíhání nebo paralelní lokální prohledávání. Memetické algoritmy byly použity např. pří řešení problému obchodního cestujícího, učení neuronové vícevrstvé sítě, plánování údržby, nelineární celočíselné programování a další.

8. **Rozptýlené prohledávání** (Scatter Search)
   Tento optimalizační algoritmus se svou podstatou liší od standardních evolučních

algoritmů a je dost podobný algoritmu Tabu Search. Je to vektorově orientovaný algoritmus, který má za úkol generovat nové vektory (řešení) na základě pomocných heuristických technik. Při startu vychází z řešení získaných pomocí vhodné heuristické techniky. Poté jsou generována nová řešení na základě podmnožiny nejlepších řešení ze startu. Z těchto nově nalezených řešení se opět vybere množina těch nejlepších a celý proces se opakuje. Tento algoritmus byl použit k řešení problému, jako je řízení dopravy, učení neuronové sítě, optimalizace bez omezení a mnohé další.

9. **Rojení částic** (Particle Swarn)
   Algoritmus je založen na práci s populací jedinců. Jejich pozice v prostoru možných řešení je měněna pomocí tzv. rychlostního vektoru. V základní verzi nedochází mezi jedinci k vzájemnému ovlivňování. To je odstraněno ve verzi s tzv. sousedstvím. V rámci tohoto sousedství pak dochází k vzájemnému ovlivňování tak, že jedinci patřící do jednoho sousedství putují k nejhlubšímu extrému, který byl v tomto sousedství nalezen[1].

10. **Neuronové sítě**
    Tyto algoritmy patří mezi nejstarší části umělé inteligence. Vznik lze datovat do druhé poloviny dvacátého století. Lze je najít v řízení, identifikaci, modelování, predikci apod. Jejich velkou nevýhodou je, že neexistuje rigorózní metoda, která by dokázala jednoznačně interpretovat informace o systému obsažené ve vahách neučené sítě.

11. **Fuzzy logika**
    Na rozdíl od neuronové sítě má fuzzy logika tu výhodu, že veškeré kroky jsou rigorózně matematizovány a podpořeny mnoha teorémy a důkazy. Lze je najít prakticky všude, kde se používají neuronové sítě. Podmínkou však je, že některé její části musí být nastaveny expertem v dané problematice, nebo pomocí speciálních algoritmů, které vycházejí z naměřených dat systému.

12. **Evoluční algoritmy**
    Tato část umělé inteligence tvoří most mezi soběstačnými algoritmy typu neuronové sítě a těmi, které potřebují ke spuštění či běhu lidského operátora. Lze je použít např. pro naučení neuronové sítě či optimalizaci její struktury, nastavení fuzzy modelu nebo fuzzy regulátoru apod. Pomocí evolučních algoritmů lze řešit prakticky jakýkoliv problém, pokud je definován jako optimalizační úloha. Zajímavými algoritmy z této oblasti jsou tzv. genetické programování a gramatická evoluce. Tyto speciální algoritmy neslouží k nalezení parametrů regresní funkce[2], ale k nalezení regresní funkce samotné (včetně parametrů) tak, aby prokládala příslušná data. Oba

[2]Na regresní funkci je založen konečný odhad regresní analýzy, což je statistická metoda, která umožňuje prozkoumat vztah mezi dvěma proměnnými.

algoritmy byly s úspěchem použity na identifikaci struktury systému na základě naměřené přechodové charakteristiky, na návrh složitých elektronických obvodů atd. Z jejich podstaty plyne, že jsou ideální při identifikaci a modelování komplexních systému, u nichž je matematická nebo fyzikální analýza obtížná.

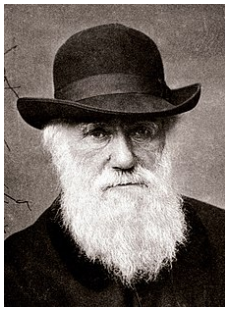
13. **Synergetika**
    Tento obor je složen z teorií, jako je teorie katastrof, teorie chaosu nebo nerovnovážné termodynamiky. Má velkou budoucnost v kybernetice a chemii. Již dnes existuje tzv. řízení chaosu, kde se využívá teorie chaosu a teorie katastrof pro řízení speciální třídy systému nebo k návrhu řízení, které se má vyhnout chaotickým režimům. Její potenciál leží rovněž v biochemických technologiích, kde se rýsuje možnost syntetizace složitých biochemických látek využitím principů kybernetiky a nerovnovážné termodynamiky. Také ji lze využít pro vyšetřování strukturální stability či existence režimů, při nichž by daný systém vykazoval chaotické či katastrofické chování.

Pro optimalizační algoritmy existuje jistá statisticky významná skutečnost, která se v odborné terminologii nazývá „No Free Lunch Teorém“. Je to toerém, ve kterém se tvrdí, že neexistuje algoritmus, který by dokázal řešit všechny problémy lépe než jiné algoritmy, neboli existuje podmnožina problému, pro které je algoritmus A lepší než algoritmus B a naopak. Neexistuje tedy „Bůh“ mezi algoritmy [1].

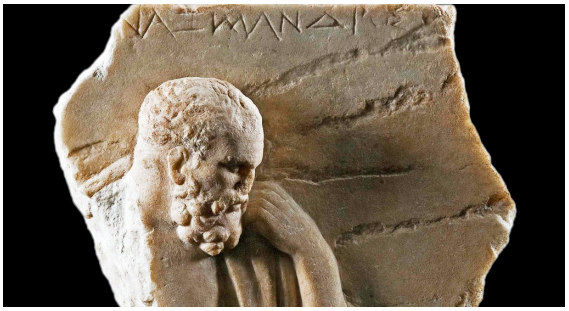
Kapitola 3

Genetické algoritmy

Evoluční výpočetní techniky (EVT) jsou numerické algoritmy, které vycházejí ze základních principů Darwinovy a Mendelovy teorie evoluce, jejíž hlavní ideou je předávání rodičovského genomu novým potomkům a následné uvolněné životního prostoru [1]. Avšak ani Darwin nebyl první, kdo přišel s takovou myšlenkou. Výrazným myslitelem, který již před Darwinem propagoval myšlenku evoluce, byl Anaximandros z Milétu, jehož názory jsou shrnuty v jeho nedochovaném filosofickém spise ”O přírodě”(tento název získal až později). Podle něj původním principem světa a příčinou všeho bytí je tzv. „neomezeno“ (řecky apeiron), z něhož se pak vyděluje studené a teplé a suché a vlhké. Tento princip si lze představit jako neomezenou a nedefinovanou vlhkost, ze které pak postupně vznikají další přírodní látky i jednotlivé druhy živých bytostí.



(a)



(b)

Obrázek 3.1: Charles Robert Darwin (a), Anaximandros z Milétu (b).

Technologie EVT závisí na existencí tzv. evolučních algoritmů. Začátek jejich historie se obvykle datuje do poloviny 70. let, kdy se poprvé objevily genetické algoritmy (GA), případně do poloviny 60. let, kdy byly poprvé s úspěchem použity tzv. evoluční strategie. Duchovními otci evolučních algoritmů jsou osobnosti, jako byl matematik A. M. Turing, N. A. Barricelli a další. V této době byly formulovány a definovány principy, které zcela jasně popisují principy evolučních algoritmů. To, že nebyly programátorsky realizovány, bylo dáno nedostatkem výkonné výpočetní techniky. Pojem GA jako první formuloval v roce 1975 John Holland. Ten na základě svého výzkumu navrhl genetický algoritmus, který je abstrakcí příslušných biologických procesů a pracuje s populací jedinců, přičemž

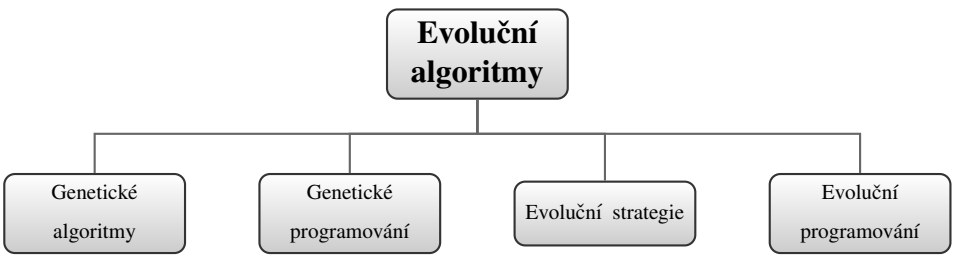
každý z jedinců reprezentuje vhodným způsobem zakódované řešení daného problému. Definoval také operátor *křížení* (crossover), který je považován za hlavní rozlišovací znak.

(a) (b)

Obrázek 3.2: Alan Mathison Turing (a) a John Henry Holland (b).

3.1 Principy činnosti genetických algoritmů

Genetické algoritmy spolu s genetickým programováním, evoluční strategií a evolučním programováním patří do skupiny evolučních algoritmů (viz. Obr. 3.3). Jak již bylo uvedeno, jedná se o vyhledávací algoritmus, který je založen na darwinovském principu evoluce. Hledání optimálního (nebo dostatečně vyhovujícího) řešení probíhá formou soutěže v rámci populace. Úspěšnost jedince je charakterizována schopnosti přežít, rozmnožit se a předat svůj genom do další populace. Tato schopnost musí být kvalifikovatelná. Každý jedinec je posuzován *hodnoticí funkcí*, jejíž velikost je vyjádřena hodnotou *fitness*. Jedinci s lepším ohodnocením mají vyšší pravděpodobnost přežít a podílet se na vytváření následující generace. Po použití rozmanitých technik křížení a reprodukce vznikne nová generace, ve které jsou vlastnosti jedinců částečně zděděny a částečně ovlivněny náhodnými mutacemi. Opakuje-li se tento evoluční cyklus mnohokrát, obvykle vznikne populace s jedinci, kteří mají vysoké ohodnocení a mohou představovat dostatečné nebo dokonce optimální řešení. Protože tento evoluční proces v sobě zahrnuje značný díl náhodnosti, je zřejmé, že každý běh příslušného algoritmu se bude odvíjet odlišným způsobem. Z téhož důvodu se v některých případech poměrně snadno může stát i to , že celá populace v procesu zdegenerujea nejlepší jedinec bude reprezentovat pouze lokální optimum [3].



Obrázek 3.3: Dělení evolučních algoritmů.

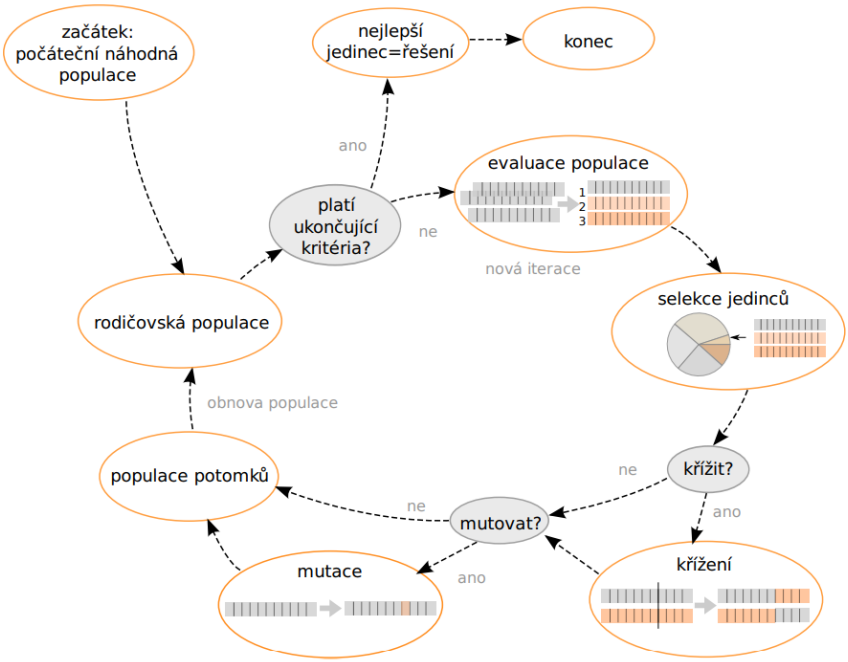
Z Obr. 3.4 vyplývá, že populace v GA je složena z jedinců, kde každý jedinec představuje jedno konkrétní řešení problému o více proměnných, které jsou zakódovány v genech. Chromozom je pak soubor všech genů jedince. Při křížení získá nový jedinec část chromozomu od každého rodiče a postupně se prosazuje genom, který reprezentuje nejlepší řešení daného problému. Aby nedocházelo ke kombinování již existujícího genofondu populace, dochází při reprodukci k náhodné mutaci genomu, která zajišťuje to, že do populace přibývají nová řešení a zvyšuje se diverzita populace. Selekce, křížení a mutace se označují jako genetické operátory.

Lze nalézt různé definice GA, které se liší zejména ve způsobu vytváření nové populace, obecné schéma je však patrné z Obr. 3.4 a je následující:

1. **Inicializace.** Náhodné vygenerování počáteční populace.
2. **Ohodnocení.** Výpočet účelové funkce a přiřazení fitness každému jedinci v populaci
3. **Selekce.** Náhodný výběr dvojice jedinců z populace a vytvoření potomků.
4. **Křížení** (crossover).Výměna části genetické informace (chromozomu).
5. **Mutace.** Náhodná změna genů v chromozomu.
6. **Nová populace.** Vytvoření nové populace z potomků.
7. **Opakování/ukončení.** Proces se opakuje od bodu 2 dokud není splněna ukončující podmínka (nejlepší jedinec dosáhl požadovaných parametrů nebo počítadlo dosáhlo zadaného počtu generací)

3.2 Genetické operátory

Způsob, kterým jsou jedinci geneticky popsána (zakódována), je velmi důležitý pro úspěch nebo neúspěch genetického algoritmu, který je použit pro konkrétní úlohu. Pro lepší názornou ukázku fungování jednotlivých operátorů bude v této kapitole uvažováno binární kódování. To znamená, že každý gen jedince může nabývat pouze dvou hodnot (0 nebo 1) a chromozomy jsou pak tvořeny řetězcem těchto hodnot o dané délce. Binární kódování patří mezi nejstarší a nejpoužívanější druhy kódování. Má však svá omezení. Může docházet ke zkreslování, a to zejména při křížení a mutaci, kdy malá změna genotypu způsobí rozsáhlou změnu fenotypu a naopak. Tento jev se nazývá Hammingova bariéra a dá se řešit použitím operátoru inverze (nebo inverzní matice) nebo pomocí Grayova kódování. V tomto ilustrativním příkladě je počáteční populace složena ze čtyř náhodně vygenerovaných jedinců. Každý má délkou chromozomu 8 (viz.Obr. 3.1). V dalším kroku by mělo následovat ohodnocení, selekce, křížení, mutace a vytvoření nové populace z potomků.



Obrázek 3.4: Obecný cyklus evolučního algoritmu GA [4].

Tabulka 3.1: Binární kódování jedinců.

Populace	
Jedinec č.	*Chromozom*
1	(1, 1, 1, 0, 0, 1, 0, 0)
2	(0, 1, 0, 1, 0, 1, 1, 1)
3	(0, 0, 1, 0, 0, 0, 1, 0)
4	(1, 1, 0, 1, 0, 0, 0, 1)

3.2.1 Ohodnocení

Jako hodnotící funkce může být použita přímo účelová funkce, na které hledáme minimum či maximum, nebo ji mlžeme sestavit jiným způsobem. Zásadní ovšem je, že hodnotící funkce musí být vytvořena pro konkrétní problém, který se pokoušíme pomocí GA vyřešit. V našem případě je hodnocen počet jedniček v chromozomu jedince. Čím vyšší počet jedniček, tím vyšší je ohodnocení.

Tabulka 3.2: Ohodnocení jedinců.

Populace		
Jedinec č.	*Chromozom*	*Fitness*
1	(1, 1, 1, 0, 0, 1, 0, 0)	4
2	(0, 1, 0, 1, 0, 1, 1, 1)	6
3	(0, 0, 1, 0, 0, 0, 1, 0)	2
4	(1, 1, 0, 1, 0, 0, 0, 1)	4

3.2.2 Výběr

Operátor selekce slouží k výběru jedinců vhodných k další reprodukci a měl by napodobovat proces přirozeného výběru. Selekce by měla probíhat tak, aby potomstvo bylo kvalitnější než rodiče a zároveň aby byla zachována rozmanitost. Při tom by selekce neměla být příliš volná, jelikož by evoluční proces mohl postupovat pomalu a jakýkoliv pokrok by byl málo zřetelný. Existuje několik metod selekce:

1. **Ruletový výběr**
   Jedná se zřejmě o nejrozšířenější selekční metodu. Na rozdíl od klasické rulety, kde každé z čísel může být vybráno se stejnou pravděpodobnosti, jsou v tomto případě favorizováni jedinci s vyšším ohodnocením. Těmto jedincům je na ruletovém kole přiřazena větší výseč a existuje tedy vyšší pravděpodobnost, že při hodu budou vybráni právě oni. Je několik možností, jak takové ruletové kolo sestrojit, a liší především ve způsobu určování velikosti výsečí. Jedna z nich je ruletový výběr s pravděpodobnosti výběru přímo úměrnou ohodnocení. K vytvoření takového kola je potřeba sečíst hodnoty *fitness* všech jedinců v populaci a pak každému přiřadit proporcionálně odpovídající kruhovou výseč. Pro populaci s počtem jedinců N je velikost takové výseče dána vztahem:

$$p_i = \frac{f_i}{\sum_{i=1}^{N} f_i} ; \qquad i \in \{0,...,N\} \tag{3.1}$$

   Po vytvoření ruletového kola pak stačí při každém výběru vygenerovat náhodné číslo $R \in < 0,1 >$ a vybrat i-tého jedince, právě když platí následující vztah:

$$\overline{f_{i-1}} < R \leq \overline{f_i} ; \qquad i \in \{0,...,N\} \tag{3.2}$$

   Kde $\overline{f_i}$ je kumulativní hodnocení, které je dáno vztahem:

$$\overline{f_i} = \sum_{j=1}^{i} p_j = \sum_{j=1}^{i} \frac{f_i}{\sum_{k=1}^{N} f_k} ; \qquad i \in \{0,...,N\} \tag{3.3}$$

   Pro náš ilustrativní příklad to znamená, že se ruletové kolo rozdělí na čtyři výseče, které svou plochou odpovídají hodnocení jedinců, jak je ukázáno v Tab. 3.3. Je zřejmé, že např. jedinec č. 2 má větší šanci být vybrán než jedinec č. 3 apod.

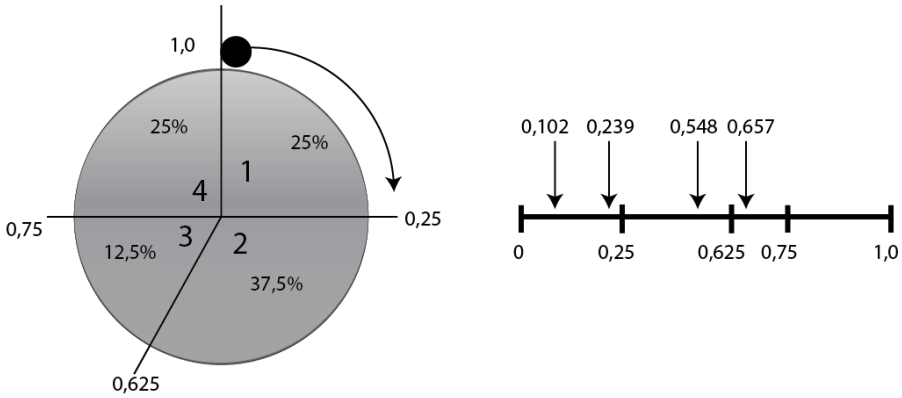
   Celý proces výběru je pak znázorněn na Obr. 3.5. Nejdříve jsou náhodně vygenerována čtyři čísla: $0.102, 0.548, 0.239, 0.657$. A na základě porovnání s kumulativním hodnocením dle vztahu 3.2 jsou vybráni jedinci číslo: $1, 2, 1, 3$. Ti následně budou vystavení působení dalších genetických operátorů.

Tabulka 3.3: Pravděpodobnost výběru jedinců.

Populace

Jedinec č.	*Chromozom*	*Fitness*	*% z celkového fitness*	*Kumulované fitness*
1	(1, 1, 1, 0, 0, 1, 0, 0)	4	25,0%	0,250
2	(0, 1, 0, 1, 0, 1, 1, 1)	6	37,5%	0,625
3	(0, 0, 1, 0, 0, 0, 1, 0)	2	12,5%	0,750
4	(1, 1, 0, 1, 0, 0, 0, 1)	4	25,0%	1,000

Nevýhodou ruletového výběru je, že příliš favorizuje jedince s nejlepším ohodnocením. To pak může vést ke snížení rozmanitosti v populaci a výsledek může konvergovat k lokálnímu extrému.



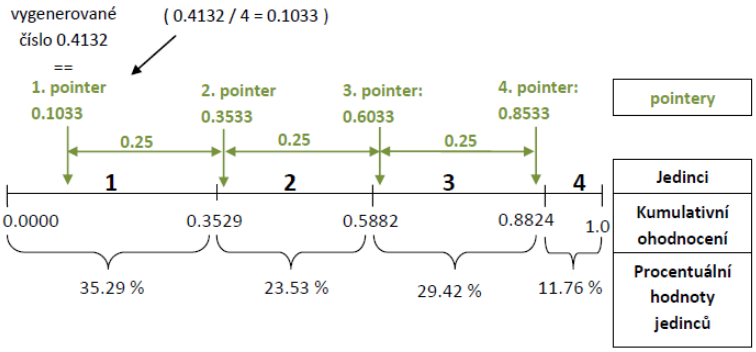
Obrázek 3.5: Ruletový výběr.

2. **Stochastický univerzální výběr**

   Tato selekční metoda je podobná rulétovému výběru s jedním rozdílem. Interval $< 0; 1 >$ je rozdělen na stejné části, ale pro výběr např. čtyř jedinců ($k = 4$) stačí náhodně vygenerovat pouze jedno číslo. To se následně vydělí tímto počtem jedinců k. Výsledné číslo nám určuje prvního vybraného jedince a na tuto pozici se umístí tzv. pointer [5]. Dále jsou na osu přidány další pointery, přičemž vzdálenost mezi nimi se rovná $\frac{1}{k}$. Schématicky tuto metodu znázorňuje Obr. 3.6. Tento způsob výběru umožňuje ponechat rozmanitost v populaci, ale neřeší tzv. problém předčasné konvergence, který je charakteristický pro selekční mechanismy, kde pravděpodobnost výběru je přímo úměrná ohodnocení.

3. **Seříznutý výběr**

   Na rozdíl od předchozích metod je tato metoda umělá. Používá se pro populace s velkým počtem jedinců, kteří jsou seřazení dle svého ohodnocení a vybráni jsou jen nejlepší. Parametrem pro výběr je tzv. řezací práh, který určuje, jaký podíl populace bude vybrán. Obvykle se tento parametr pohybuje v rozmezí $10\% - 50\%$.



Obrázek 3.6: Stochastický výběr[5].

4. **Turnajový výběr**
   V této metodě se z populace vybere k jedinců (nejčastěji $k = 2$), kteří se setkají v simulovaném souboji. Vítězí jedinec s vyšší hodnotou fitness a postupuje dál. Výhodou tohoto selekčního mechanizmu je vyšší šance výběru jedinců s horním ohodnocením, a tím zachování větší diverzity v populaci. Pro výběr jedinců do turnaje se používá buď náhodný index nebo roletový výběr. Metoda je znázorněna v Tab. 3.4:

Tabulka 3.4: Turnajový výběr jedinců.

Populace		**Turnaj**	
Jedinec č.	*Fitness*	*Jedinci vybraní do souboje*	*Vítěz souboje*
1	4	(4,1)	1
2	6	(3,2)	2
3	4	(1,3)	1
4	2	(3,1)	3

5. **Pořadový výběr**
   Při použití této metody jsou nejdříve jedinci vzestupně seřazeni dle hodnoty fitness. Dále jim je přiřazená kruhová výseč, jejíž velikost již není závislá na fitness, ale na pořadí jedince. Lépe umístění jedinci mají tedy vetší šanci být vybráni. Tento způsob selekce řeší problém např. ruletového výběru, kde výrazně silnější jedinci byli opakovaně vybíráni do nové generace, bránili výběru slabších jedinců a snižovali tím různorodost populace.

3.2.3 Křížení

Operátor křížení (*crossover*) slouží k vytvoření potomků vybraných rodičů, rozšiřuje prohledávaný prostor tím, že skládá nová řešení z již existujících, a snižuje riziko uváznutí v lokální extrému. Existuje celá řada technik křížení, přičemž jejich společná vlastnost je ta, že jde vždy o vzájemnou výměnu chromozomů. Níže jsou vysvětleny principy několika základních metod křížení.

1. **Jednobodové křížení**

   Jedná se o nejjednodušší a nejznámnější operátor binárního křížení. Nejprve je náhodně zvolen jeden gen v chromozomu, od kterého počínaje jsou vyměněny zbylé části chromozomu mezi rodiči. Vznikou tím dva noví jedinci, kteří mají část genů po prvním a část po druhém z obou rodičů [3].Pro náš ilustrativní příklad, kde do prvního páru byli vybráni jedinci č. 1 a 2, předpokládejme, že byl vybrán 3. gen jako bod křížení. Výsledek celé operace je znázoněn v Tab. 3.5. Je zřejmé, že křížením vznikl jedinec, jehož ohodnocení 7, a je tedy lepší než oba jeho rodiče. Bod křížení by měl být přirozeným číslem z intervalu $K \in <1; L-1>; K \in \mathbb{N}$, kde L je délka chromozomu. Nemá smysl volit jako bod křížení poslední gen, jelikož by nedošlo k žádné genetické výměně a tato operace by byla zbytečná.

Tabulka 3.5: Jednobodové křížení.

Jedinec č.	*Chromozom*		*Nový chromozom*
1	(**1, 1, 1** | **0, 0, 1, 0, 0**)	→	(**1, 1, 1** | 1, 0, 1, 1, 1)
2	(0, 1, 0 | 1, 0, 1, 1, 1)	→	(0, 1, 0 | **0, 0, 1, 0, 0**)

2. **Dvou a vícebodové křížení**

   Tato metoda je podobná jednobodovému křížení s tím rozdílem, že lichý zvolený bod křížení mi udává začátek výměny genů a sudý konec výměny. U dvoubodového křížení tedy mezi prvním a druhým bodem k výměně nedochází. Předpokládejme, že pro náš případ byly zvoleny jako body křížení geny číslo dva a šest. Výsledek je ukázán v Tab. 3.6.

Tabulka 3.6: Dvoubodové křížení.

Jedinec č.	*Chromozom*		*Nový chromozom*
1	(**1, 1** | **1, 0, 0, 1** | **0, 0**)	→	(**1, 1** | 0, 1, 0, 1 | **0, 0**)
2	(0, 1 | 0, 1, 0, 1 | 1, 1)	→	(0, 1 | **1, 0, 0, 1** | 1, 1)

3. **Uniformní křížení**

   Tento typ křížení zobecňuje jednobodové a vícebodové křížení tak, že každý gen může být bodem výměny. K tomu jsou použity tzv. křížící masky, které jsou stejně dlouhé jako chromozomy jedinců a na jednotlivých pozicích náhodně nabývají hodnot 0 nebo 1. V případě, kdy na dané pozici v masce je hodnota 0, převezme potomek gen od prvního rodiče. Naopak když je tato hodnota rovna 1, převezme gen od druhého rodiče. Příklad použití této metody je zobrazen v Tab. 3.7.

Tabulka 3.7: Uniformní křížení.

Jedinec č.	*Chromozom*	*Maska*	*Nový chromozom*
1	(**1, 1, 1, 0, 0, 1, 0, 0**)	`[1,1,0,0,1,1,1,0]`	(0, 1, **1, 0**, 0, 1, 1, **0**)
2	(0, 1, 0, 1, 0, 1, 1, 1)	`[0,1,1,1,0,1,1,1]`	(**1**, 1, 0, 1, **0**, 1, 1, 1)

4. **Aritmetické křížení**
   Tato metoda se používá u křížení s reálnými hodnotami chromozomů. Potomek (O) je kombinací obou rodičů (P_1, P_2), jak je ukázano v rovnici 3.4, kde a je náhodně vygenerované číslo v intervalu $a \in <0;1>; a \in \mathbb{R}$.

$$O = a \cdot P_1 + (1-a) \cdot P_2 \tag{3.4}$$

3.2.4 Mutace

Tento operátor, kterému jsou vystavení nově vytvoření jedinci, rozšiřuje prohledávaný prostor o další řešení, která nebylo možné získat křížením. Mutace s velmi malou pravděpodobností (obvykle od 0.1% do 5%) náhodně mění hodnoty jednotlivých genů. Příklad použití mutace je znázorněn v Tab. 3.8. U stagnujících generací můžeme zvýšit pravděpodobnost mutace nebo zavést dynamickou mutaci, která bude automaticky reagovat na vývoj populace.

Tabulka 3.8: Binarní mutace genů.

Jedinec č.	*Chromozom před mutací*		*Chromozom po mutaci*
1	(1, **1**, 1, 0, 0, 1, 0, 0)	→	(1, **0**, 1, 0, 0, 1, 0, 0)
2	(0, 1, 0, 1, **0**, 1, 1, 1)	→	(0, 1, 0, 1, **1**, 1, 1, 1)
3	(0, 0, 1, 0, 0, 0, **1**, 0)	→	(0, 0, 1, 0, 0, 0, **0**, 0)
4	(1, 1, **0**, 1, 0, 0, 0, 1)	→	(1, 1, **1**, 1, 0, 0, 0, 1)

3.2.5 Doplnění

V případě, že počet nově vygenerovaných jedinců je menší než v předchozí populaci, je třeba zbývající jedince doplnit. Naopak pokud je tento počet vyšší, musí se vybrat jedinci, kteří se dostanou do nové populace a kteří budou zapomenuti. Jedny z možných metod doplnění jsou:

1. **Ryzí doplnění**
   Během křížení byl vytvořen stejný počet jedinců jako v populaci rodičů. Nová populace se tedy skládá pouze z těchto potomků.

2. **Uniformní doplnění**
   Byl vytvořen menší počet jedinců než v předešlé populaci. Nová populace se doplní náhodně vybranými jedinci z populace rodičů.

3. **Elitní doplnění**
   Byl vytvořen menší počet jedinců než v předešlé populaci. Nová generace se doplní nejlepšími jedinci z předešlé generace.

Kapitola 4

Implementace v jazyce Python

Tato kapitola je věnována vytvoření GA v jazyce Python, který je následně použit k optimalizaci železobetonové nosné stěny. V jednotlivých krocích jsou představeny principy výpočtu, způsoby generování stěn a nastavení jednotlivých operátorů algoritmu. V závěru kapitoly jsou demonstrovány příklady aplikace vytvořeného GA na konkrétních konstrukcích a porovnání výpočtu s komerčním softwarem.

4.1 Metoda konečných prvků

Metoda konečných prvků (MKP) představuje přibližné numerické řešení parciálních diferenciálních rovnic s příslušnými okrajovými podmínkami. MKP nachází uplatnění v mnoha oborech - ve strojním, automobilovém, leteckém, elektrotechnickém a stavebním. Kromě problémů statiky a dynamiky pevných a poddajných těles se běžně využívá pro modelování proudění tekutin, vedení tepla, k analýze elektromagnetických polí apod.

Pro úspěšné sestavení GA je nutné opakovaně spočítat několik variant konstrukcí v jedné populaci. V běžných komerčních softwarech určených pro statickou analýzu konstrukcí je tento požadavek obtížně proveditelný. Z tohoto důvodů byl vytvořen balík pro výpočet konstrukcí pomocí MKP , ve kterém byly uvažovány následující předpoklady:

1. **Základní rovnice**
   Výpočet vychází ze základní rovnice deformační metody:

   $$\mathbf{K} \cdot \mathbf{r} = \mathbf{F} \tag{4.1}$$

   kde K je matice tuhosti konstrukce, r je vektor posunutí a f je vektor zatížení.

2. **Slabé řešení diferenciální rovnice a jeho diskretizace**
   Základem MKP se stala Galerkinova metoda. Používá se při řešení soustavy parciálních diferenciálních rovnic a její princip spočívá v nahrazení původní rovnice (tzv. silné řešení) její integrální formou (tzv. slabé řešení) a následnou diskretizací (převedení na úlohu s konečným počtem parametrů). Řečeno jinými slovy, můžeme spojitou

funkci aproximovát pomocí diskrétního modelů, který se skládá z jednoho nebo několika aproximačních polynomů a spojitá funkce je rozdělena na konečné elementy (prvky). Každý prvek je definován pomocí interpolační funkce, která popisuje jeho chování mezi koncovými body, které se nazývají uzly [6]. Aproximativní řešení budeme hledat jako lineární kombinaci bázových funkcí:

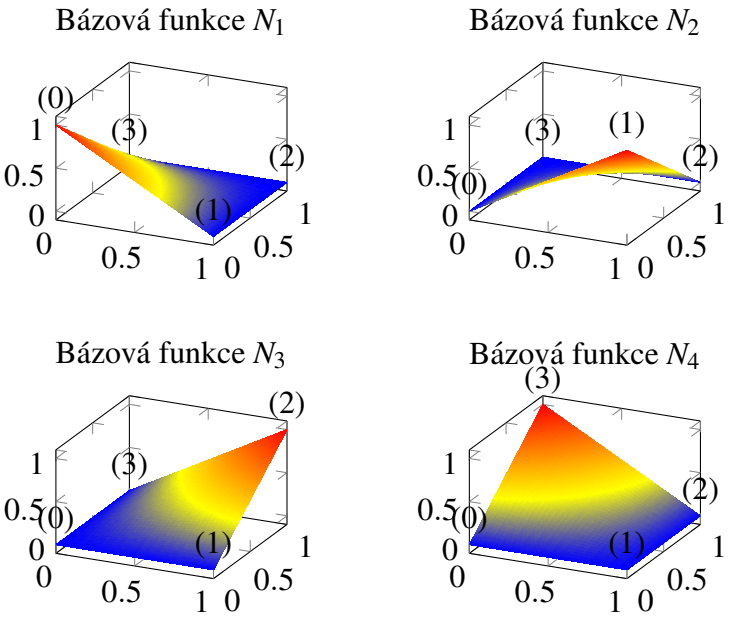
$$u = \sum_{i=1}^{n} u_i N_i(x) \tag{4.2}$$

Pokud bychom dosadili do řešené rovnice tuto aproximaci, nebude řešená rovnice splněna přesně. To snažíme vyřešit tím, že k rovnici přičteme zbytkovou funkci (*reziduum*). Abychom zbytkovou funkci minimalizovali (tzn. dostali co nejpřesnější aproximaci), násobíme zbytkovou funkci *funkci váhovou*, integrujeme přes celou oblast řešení a výsledek položíme rovno nule. Tento postup nazýváme metodou *vážených reziduí*.

3. **Bázové funkce**

   Bázové funkce se obvykle značí písmenem N. Tyto funkce jsou předepsány pro každý uzel prvku a využívá se vlastnosti, kterou má Kroneckerovo delta. To znamená, že pro daný uzel pvku nabývá bázová funkce hodnoty 1 a ve všech ostatních uzlech je rovna nule. Názorně je to zobrazeno na Obr. 4.1. Při výpočtu byly použity *bilineární čtvercové* prvky s následujícími bázovými funkcemi:

$$\begin{aligned} N_1 &= \frac{1}{4}(1-x)(1-y) \qquad N_2 = \frac{1}{4}(1+x)(1-y) \\ N_3 &= \frac{1}{4}(1+x)(1+y) \qquad N_4 = \frac{1}{4}(1-x)(1+y) \end{aligned} \tag{4.3}$$



Obrázek 4.1: Bázové funkce pro bilineární čtyřúhelníkový prvek.

4. **Lokální matice tuhosti**

   Lokální matice tuhosti k je dána vztahem:

$$[k] = \int_A [B]^T [D][B]t \; dxdy \tag{4.4}$$

   kde t je tloušťka prvků, B je matice derivací bázových funkcí a D je matice tuhosti materiálu.

$$[B] = \begin{bmatrix} \frac{\partial N_1}{\partial x} & 0 & \frac{\partial N_2}{\partial x} & 0 & \frac{\partial N_3}{\partial x} & 0 & \frac{\partial N_1}{\partial x} & 0 \\ 0 & \frac{\partial N_1}{\partial y} & 0 & \frac{\partial N_2}{\partial y} & 0 & \frac{\partial N_3}{\partial y} & 0 & \frac{\partial N_1}{\partial y} \\ \frac{\partial N_1}{\partial y} & \frac{\partial N_1}{\partial x} & \frac{\partial N_2}{\partial y} & \frac{\partial N_2}{\partial x} & \frac{\partial N_3}{\partial y} & \frac{\partial N_3}{\partial x} & \frac{\partial N_4}{\partial y} & \frac{\partial N_4}{\partial x} \end{bmatrix} \tag{4.5}$$

$$[D] = \frac{E}{1-\nu^2} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{bmatrix} \tag{4.6}$$

   Lokální matice tuhosti prvku má velikost 8x8 (dáno dvěma neznámými posuny v každém ze čtyř uzlů), je symetrická a obecně se dá zapsat takto :

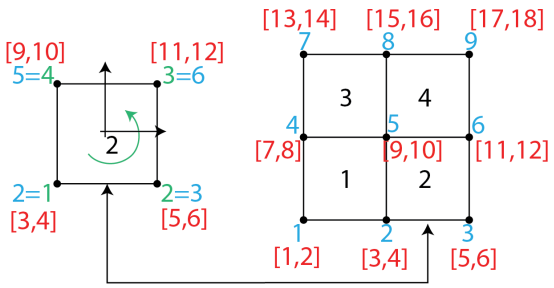
$$[k] = \begin{bmatrix} k_{11} & & & & & & & \\ k_{21} & k_{22} & & & & & & \\ k_{31} & k_{32} & k_{33} & & & & & \\ k_{41} & k_{42} & k_{43} & k_{44} & & & & \\ k_{51} & k_{52} & k_{53} & k_{54} & k_{55} & & & \\ k_{61} & k_{62} & k_{63} & k_{64} & k_{65} & k_{66} & & \\ k_{71} & k_{72} & k_{73} & k_{74} & k_{75} & k_{76} & k_{77} & \\ k_{81} & k_{82} & k_{83} & k_{84} & k_{85} & k_{86} & k_{87} & k_{88} \end{bmatrix} \tag{4.7}$$

5. **Kódová čísla a lokalizace**

   Po vytvoření lokální matice tuhosti je nutné ji správně lokalizovat do matice globální. K tomu se využívají tzv. kódová čísla, které přestavují jednotlivé neznámé posuny uzlů (na Obr. 4.2 označeny červeně). Nejdříve jsou očíslovány všechny prvky a uzly v konstrukci. Každý prvek má svoje lokální čísla uzlů (označeny zeleně), kterým odpovídají globální čísla uzlů (označeny modře) v závislosti na pozici daného prvku. Na základě toho převodu, jsou jednotlivé prvky lokální matice tuhosti přičteny na příslušné pozice globální matice.

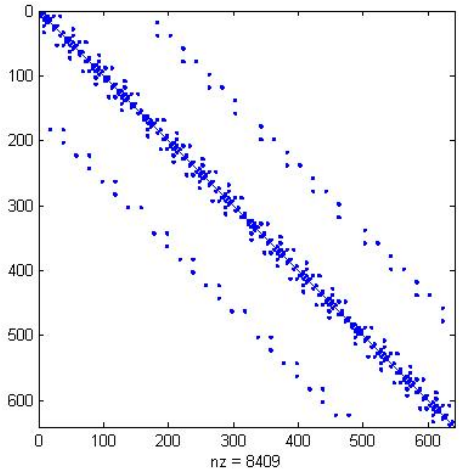
6. **Globální matice tuhosti**

   Při dodržení číslování kódových čísel dle Obr. 4.2, je globální matice řídká (většina prvků v matici je nulová) a pásová (Obr. 4.3). Tyto vlastnosti měly zásadní vliv na výpočet a na to, jakým způsobem byla MKP implementována v jazyce Python. Při výpočtu konstrukcí s jemnější sítí (a tedy s větším počtem neznámých a větší ma-



Obrázek 4.2: Lokalizace prvků.

ticí tuhostí) docházelo k rychlému zaplnění RAM pamětí a výpočet byl zastaven. Způsobovalo to obrovské množství dat v podobě nul, které se do matice ukládalo. Tento problém se podařilo vyřešit pomocí přídavného balíku `scipy`. Konkrétně se jednalo o funkci `scipy.sparse`, která ukládala matici tuhosti v podobě souřadnic a příslušných hodnot na těchto pozicích (lze vyjádřit jako K[i,j]=data). Nulové prvky matice byly tedy zcela eliminovány. Tento přístup umožnil provádět výpočet konstrukcí s jemnější sítí prvků a taky zrychlení výpočtu.



Obrázek 4.3: Globální pásová matice tuhosti.

7. **Numerická integrace**

Výpočet matice tuhosti vyžaduje integraci součinů bázových funkcí a tuhosti. Pro složitější případy však tuto integraci nelze provádět analyticky. Místo toho se využívá numerická integrace. Existuje několik metod, ale pro polynomy je zvláště vhodná Gaussova numerická integrace. Princip spočívá v tom, že se snažíme stanovit hodnoty vah w_i a souřadnic integračních bodů ξ_i tak, abychom integrovali přesně polynom co nejvyššího řádu. Hledaný integrál pak může být vyjádřen následovně:

$$\int_a^b f(x)\,dx \approx \sum_{i=1}^{n} w_i f(\xi_i) \tag{4.8}$$

V Tab. 4.1 je přehled integračních bodů a vah. Při volbě n bodů bude tato metoda

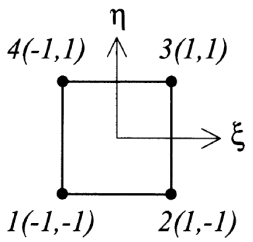
přesná pro polynom $(2n-1)$ stupně. Při výpočtu byla uvažována dvoubodová integrace s hodnotou vah 1.

Tabulka 4.1: Gaussovy integrační body a váhy.

n	*Integrační body* ξ_i	*Váhy* w_i
1	0	2
2	$\pm 1/\sqrt{3}$	1
3	0	$8/9 \approx 0.888889$
	$\pm\sqrt{3/5} \approx 0.774597$	$5/9 \approx 0.555556$
4	$\pm\sqrt{\left(3-2\sqrt{6/5}\right)/7} \approx 0.339981$	$(18+\sqrt{30})/36 \approx 0.652145$
	$\pm\sqrt{\left(3+2\sqrt{6/5}\right)/7} \approx 0.861136$	$(18+\sqrt{30})/36 \approx 0.347855$

8. **Izoparametrické prvky**

Numerická integrace se zpravidla provádí v přirozených souřadnicích ξ, η na intervalu $< -1; 1 >$. Geometrií prvků je proto třeba do těchto souřadnic transformovat. Tzv. izoparametrické prvky používají pro aproximaci souřadnic stejné funkce jako pro aproximaci uzlových posunů. Tyto funkce mají stejný počet parametrů, a proto se takové prvky označují jako izoparametrické.



Obrázek 4.4: Izoparametrický prvek.

Výpočet lokální matice tuhosti je pak dán vztahem:

$$[k] = \int_{-1}^{1}\int_{-1}^{1} [B]^T [D][B]t \; d\xi d\eta = \sum_{i=1}^{n}\sum_{j=1}^{n} [B]^T [D][B]t \; w_i w_j \; |detJ| \tag{4.9}$$

kde J je Jakobián transformace, který se dá vyjádřit jako [6]:

$$J = \begin{bmatrix} \sum \frac{\partial N_i}{\partial \xi} \\ \sum \frac{\partial N_i}{\partial \eta} \end{bmatrix} \begin{bmatrix} x_i & y_i \end{bmatrix} \tag{4.10}$$

$$J = \frac{1}{4}\begin{bmatrix} -(1-\eta) & (1-\eta) & (1+\eta) & -(1+\eta) \\ -(1-\xi) & -(1+\xi) & (1+\eta) & (1-\xi) \end{bmatrix} \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ x_4 & y_4 \end{bmatrix} \tag{4.11}$$

9. **Vztahy teorie pružnosti pro rovinný problém**

   V každém uzlu konečného prvků, budou dva neznáme posuny (u, v). Na celý prvek tedy bude celkem osm neznámých:

$$\begin{bmatrix} u_1, v_1, u_2, v_2, u_3, v_3, u_4, v_4 \end{bmatrix}^T \tag{4.12}$$

   Pro řešení problému rovinné napjatosti byly použity následující geometrické rovnice v maticovém tvaru:

$$\begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial x} & 0 \\ 0 & \frac{\partial}{\partial y} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} \tag{4.13}$$

   Fyzikální rovnice byly uvažovány v tomto maticovém tvaru:

$$\begin{bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{bmatrix} = \frac{E}{1-\nu^2} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{bmatrix} \begin{bmatrix} \varepsilon_x \\ \varepsilon_y \\ \tau_{xy} \end{bmatrix} \tag{4.14}$$

   Výpočet hlavních napětí konstrukce, které pak byly využity pro ohodnocení jedinců, byl proveden dle vztahu:

$$\sigma_{1,2} = \frac{\sigma_x + \sigma_y}{2} \pm \sqrt{\left(\frac{\sigma_x - \sigma_y}{2}\right)^2 + \tau_{xy}^2} \tag{4.15}$$

4.2 Generátory stěn

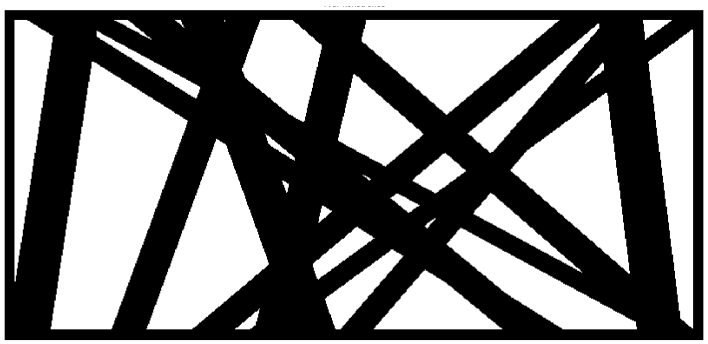
Jak již bylo zmíněno, předmětem optimalizace je železobetonová betonová nosná stěna. Aby bylo možné provést výpočeta a následnou optimalizace, byly vytvořeny dva generátory stěn:

1. **Generátor linií**
   Tento generátor byl inspirován budovou Italského pavilonu na Expu 2015. Její fasáda je vyrobena z panelů z *biodynamického* cementu. Jeho bio složka je dána fotokatalytickými vlastnostmi materiálu, při kontaktu se slunečním světlem materiál zachycuje některé znečisťující látky z ovzduší a proměňuje je v inertní sůl. Dynamická složka je dána tekutostí materiálu a umožňuje vytvářet složité tvary [7]. Strukturu pak tvoří změť čar různých tlouštěk, které se kříží pod rozličnými úhly. Tento tvar má připomínat zkamenělý les a přírodu.

(a) Pohled na budovu (b) Detail fasády

Obrázek 4.5: Italský pavilon Expo 2015.

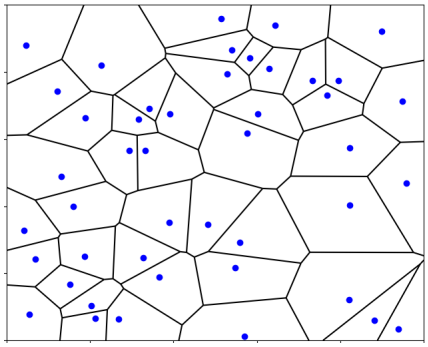
Na podobném principu je založen vytvořený generátor. Nejdříve je na základě vstupních hodnot (výška a délka stěny, maximální a minimální hodnota tloušťky linií a jejich počet) vytvořena matice, která obsahuje náhodně vygenerované souřadnice koncových bodů a tloušťky linií. Na základě této matice jsou linie vykresleny do zadaného prostoru (Obr. 4.6).



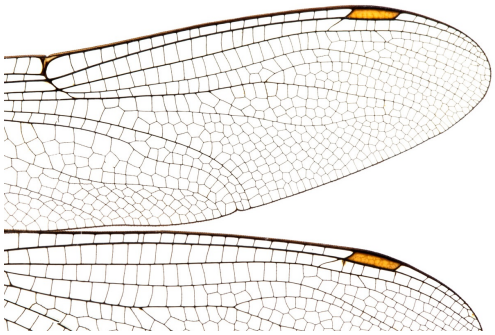
Obrázek 4.6: Tvar stěny vytvořený pomocí generátoru linií.

2. **Voronoi generátor**

   Tento generátor byl inspirován tzv. Voroného diagramem (Obr. 4.7(a)), což je způsob rozdělení prostoru, který je určený vzdálenostmi k množině bodů v tomto prostoru. Existuje několik algoritmů pro sestrojení daného diagramu (např.inkrementální, rozděl a panuj, Fortunova metoda, z Delaunayovy triangulace apod.), ale nejjednodušším případem je rozdělení roviny podle množiny bodů M. Voroného diagram každému bodu b z množiny M přiřadí buňku V tak, že všechny body v buňce V jsou blíže k bodu b než k jakémukoliv bodu z množiny M.
   Voroného diagram se využívá např. v počítačové grafice (mozaiky), geografii (analýza sídel), chemii (3D modelování buněk a prvků) a robotice (plánování cesty robotů). Tento vzor lze najít i v přírodě např. na křídlech vážky nebo na srsti žiraf.



(a) Voroného diagram



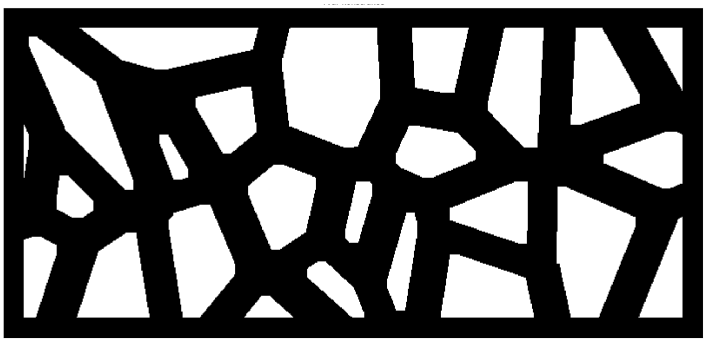
(b) Křídla vážky

(c) Fasáda budovy

(d) Vzory na srsti žiraf

Obrázek 4.7: Příklady výskytu Voroného diagramu v architektuře a přírodě.

Stejně jako u generátoru linií je i v tomto případě nejdříve vytvořena matice obsahující náhodně vygenerované souřadnice bodů, na základě kterých se vykreslí Voroného diagram (Obr. 4.7).



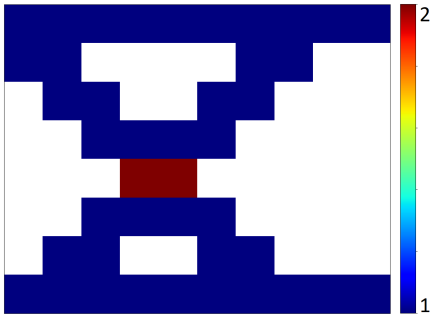
Obrázek 4.8: Tvar stěny vytvořený pomocí Voronoi generátoru.

Aby bylo možné u takto vygenerovaných stěn při tvorbě sítě MKP lokalizovat jednotlivé prvky v místech, kde je materiál, a následně tyto stěny spočítat, byl zvolen následující způsob, jak definovat tvar stěny. Nejdříve je vytvořená nulová matice W o rozměrech:

$$W = \left(\frac{H}{a} \times \frac{L}{a}\right) \tag{4.16}$$

kde H je výška stěny, L délka stěny a a je délka hrany konečných prvků. Do takto připravené matice byl na základě již dříve vygenerovaných souřadnic bodů přenesen tvar konstrukce. K tomu byl využit přídavný balík `skimage.draw`, který do matice W přičetl 1 na místa, kde je materiál. Názorně je tento proces ukázán na Obr. 4.9.

$$\begin{vmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{vmatrix}$$



Obrázek 4.9: Definice tvaru stěny pomocí matice.

Na Obr. 4.10 je demonstrován algoritmus, který vyhledá nenulové prvky matice W, a zaznamená jejich polohu, která se následně využije k přičtení lokální matice tuhosti do globální na základě kódových čísel.

Obrázek 4.10: Lokalizace prvků ve stěně.

4.2.1 Objektově orientované programování

Hlavní důvodem pro zvolení Pythonu jako programovacího jazyka, je možnost využití tzv. objektově orientovaného programování (OOP). Jedná se o způsob programování, ve kterém základní jednotkou je objekt, který odpovídá nějakému objektu z reálného světa (např. člověk, auto, nosná stěna nebo databáze). Každý objekt má své :

- **vlastnosti** - data, která uchovává (např. věk, značka auta, tvar stěny)
- **schopnosti** - funkce, které umí vykonávat (např. jdi do školy, natankuj, zapiš hodnotu do databáze)

Rozlišujeme dva typy objektů:

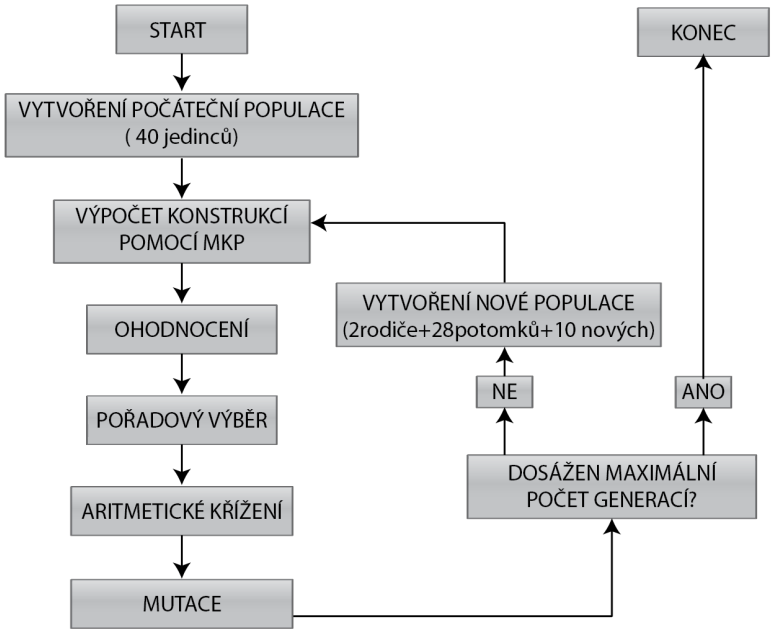
- **vestavěné typy** (např. seznam)
- **třídy** (class)

Třídy jsou „šablony“, podle kterých se tvoří jednotlivé objekty. Říkáme pak, že objekt je instancí dané třídy. Na Obr. 4.11 já znázorněna definice třídy *individual*, podle které se generují jednotlivé stěny v populaci. Zároveň je vidět přiřazení vlastnosti *fitness* každé instanci této třídy.

Obrázek 4.11: Ukázka použití OOP v Pythonu.

4.3 Celkové nastavení GA a operátorů

Vytvořený algoritmus začíná náhodným vygenerováním počáteční populace o velikosti 40 jedinců. To zahrnuje vytvoření matice souřadnic bodů a matice definující tvar stěny pro každého jedince (viz. Kapitola 4.2). Následně je celá populace stěn spočítána pomocí MKP a výsledky jsou uloženy jako vlastnost daného jedince. Dále jsou jednotlivé konstrukce ohodnoceny a seřazeny. Na základě pořadového výběru jsou aritmeticky zkříženy matice souřadnic bodů, kterým jsou stěny definovány. V dalším kroku probíhá náhodná mutace, která nepatrně mění souřadnice bodů. Pokud není dosažen maximální zadaný počet generací, je vytvořena nová populace, která se skládá z 28 potomků, 10 nově náhodně vygenerovaných jedinců a dvou kopií nejlepšího jedinec z minulé populace. Operátor mutace je aplikován pouze na jedné z těchto kopií. To zaručuje, že potenciální nejlepší řešení nebude v průběhu zničeno. Při dosažení maximálního počtu generací je proces ukončen. Názorně to ilustruje Obr. 4.12.



Obrázek 4.12: Schéma vytvořeného algoritmu.

Pro každý krok algoritmu byl vytvořený samostatný balík - pro vytvoření populace, výpočet konstrukcí, ohodnocení, křížení, mutaci a uložení výsledků v podobě obrázků a *.txt* dokumentu. Tyto balíky jsou naimportovány do centrálního souborů, ze kterého probíhá spuštění celého algoritmu (viz. Obr. 4.13).

Obrázek 4.13: Cyklus GA v Pythonu.

4.3.1 Nastavení ohodnocení

Do hodnotící funkce vstupuje několik parametrů:

- **maximální deformace** - v podobě poměru maximální deformace jedince k maximální deformaci z celé populace :

$$def = \frac{max\ def(ind)}{max\ def(pop)} \tag{4.17}$$

  Tento poměr u jedince s největší deformací tedy bude roven 1. Naopak u jedince s nejmenší deformací bude tento poměr nejmenší z populace.

- **vylehčení** - v podobě poměru počtu použitých prvků (počet nenulových prvků v matici, kterou je definován tvar stěny (4.16)), k maximálnímu počtu použitých prvků v populaci :

$$elements = \frac{elements\ used(ind)}{max\ elements\ used(pop)} \tag{4.18}$$

- **průměrná deformace** - v podobě poměru průměrné deformace jedince k maximální průměrné deformaci v populaci. Průměrná deformace u_{prum} je vyjádřena jako součet absolutních hodnot deformací na konstrukci k počtu použitých prvků.

$$u_{prum} = \frac{\sum |u|}{elements\ used} \tag{4.19}$$

$$prum\ def = \frac{u_{prum}(ind)}{max\ u_{prum}(pop)} \tag{4.20}$$

Kromě ohodnocení konstrukcí dochází i k jejich penalizaci, která je založena na těchto parametrech:

- **penalizace deformace** - penalizace je spočítána na základě následující funkce, ve které jako proměnná vystupuje maximální deformace konstrukce:

$$pen\ def = \frac{1}{1 + 100e^{-0.4 \cdot u\max}} \tag{4.21}$$

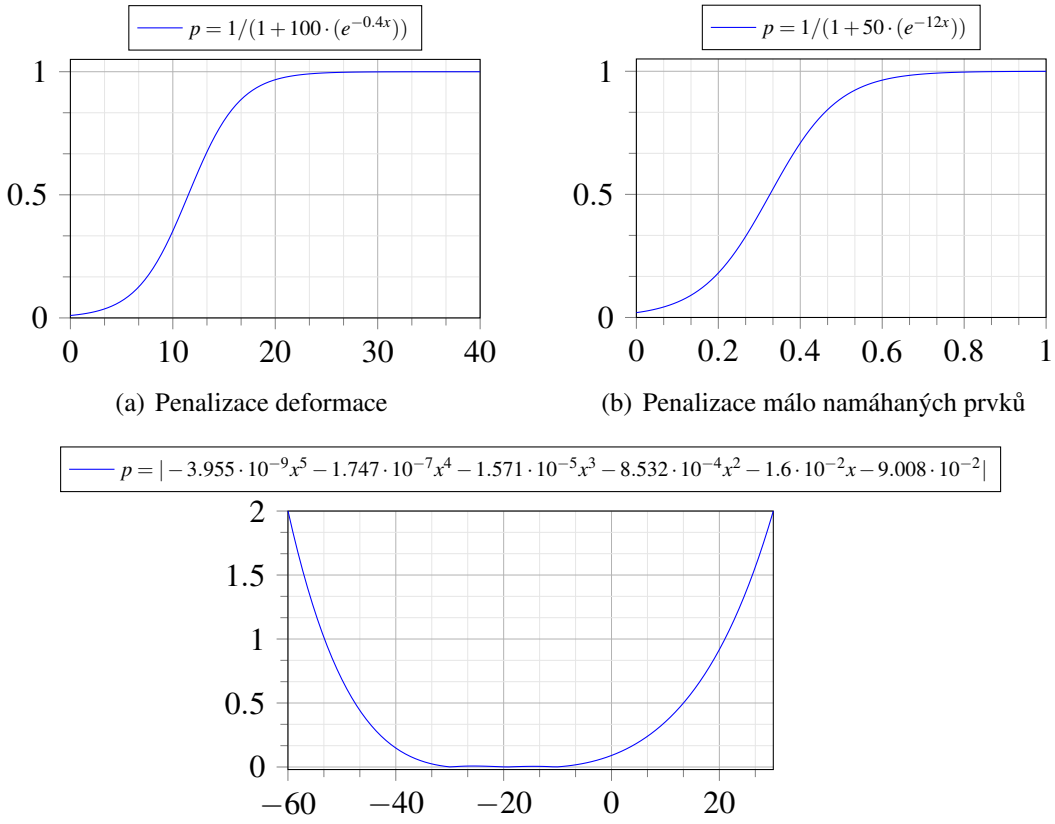
- **penalizace využití** - penalizace je spočítána na základě funkce, ve které jako proměnná vystupuje poměr počtu málo využitých prvků k celkovému počtu prvků. Prvek je považován za málo využitý, pokud jeho napětí leží v intervalu $< -5; 0.5 > MPa$. Penalizace je spočítána pro σ_1 i σ_2 a následně zprůměrována.

$$usage = \frac{elem_{\sigma \in <-5,0.5>}}{elements\ used} \tag{4.22}$$

$$pen\ usage = \frac{1}{1+50e^{-12 \cdot usage}} \tag{4.23}$$

- **penalizace napětí** - penalizace je spočítána na základě následující funkce, ve které jako proměnná vystupuje 1% a 99% percentil hlavního napětí. Nejsou tedy uvažovány maximální hodnoty napětí, jelikož konstrukce není zcela hladká (dáno zadáváním tvaru pomocí matice) a v některých místech by mohlo napětí dosahovat hodnot, které v reálné konstrukci nebudou. Penalizace je spočítána pro obě hlavní napětí následovně:

$$\begin{aligned} pen\ stress = |&-3.955 \cdot 10^{-9}\sigma^5 - 1.747 \cdot 10^{-7}\sigma^4 - 1.571 \cdot 10^{-5}\sigma^3 \\ &-8.532 \cdot 10^{-4}\sigma^2 - 1.6 \cdot 10^{-2}\sigma - 9.008 \cdot 10^{-2}| \end{aligned} \tag{4.24}$$



(a) Penalizace deformace

(b) Penalizace málo namáhaných prvků

(c) Penalizace napětí

Obrázek 4.14: Graf penalizačních funkcí.

Celá hodnotící funkce se dá vyjádřit následovně:

$$fitness = \frac{def + 3 \cdot elements + prum\ def}{5} + pen\ def + pen\ usage + pen\ stress \tag{4.25}$$

4.3.2 Nastavení křížení

Pro křížení matic obsahující souřadnice bodů bylo použito aritmetické křížení (4.26). Nejdříve je opakovaně vygenerováno náhodné číslo b a na jeho základě jsou pořadovým výběrem určeni rodiče pro křížení.

$$new\ matrix = a \cdot matrix_1 + (1-a) \cdot matrix_2 \tag{4.26}$$

Konkrétní definice křížení v Pythonu je ukázána na Obr. 4.15.

Obrázek 4.15: Křížení v Pythonu.

4.3.3 Nastavení mutace

Pro spuštění operátoru mutace je nutné, aby náhodně vygenerované číslo bylo v intervalu $< 0; 0.3 >$. Během mutace jsou pak jednotlivé souřadnice linií nebo bodů Voroného diagramu násobeny číslem c. To je náhodně vygenerované v intervalu $< 0.7, 1.3 >$. Může tedy souřadnici náhodně posunou dál nebo blíž k počátku souřadného systému.

Konkrétní definice mutace v Pythonu je ukázána na Obr. 4.15.

Obrázek 4.16: Mutace v Pythonu.

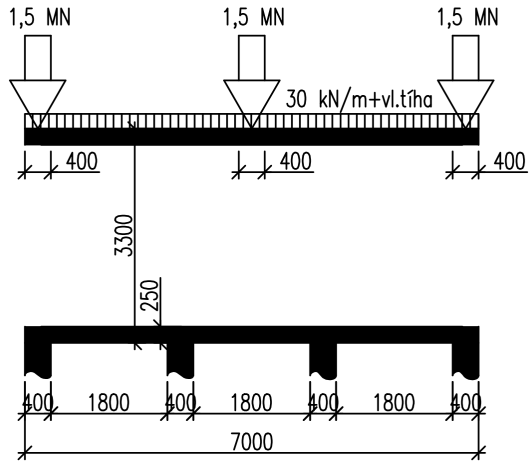
4.4 Příklad 1- symetricky podepřená konstrukce

Pro ověření funkčnosti sestaveného algoritmu byly pro oba generátory stěn vytvořený dvě varianty konstrukcí s rozdílným umístěním podpor.

4.4.1 Generátor linií

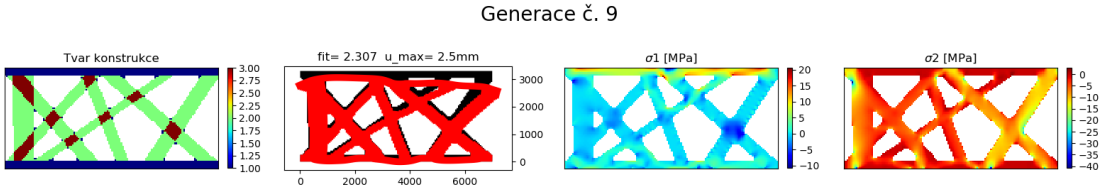
Schéma konstrukce je ukázáno na Obr. 4.17. Stěna je nahoře a dole ohraničena pásem o tloušťce 250mm, který reprezentuje desku, ale tuhost v kolmém rovině nebyla uvažována. Dole je stěna podepřena podepřena čtyřmi sloupy a nahoře je zatížena liniovým zatížením a třemi sloupy.
Dále byly uvažovány následující parametry:

- $E = 33GPa$
- $\nu = 0.2$
- výška stěny: $H = 3300mm$
- délka stěny: $L = 7000mm$
- tloušťka stěny: $t = 300mm$
- tloušťka „desky“: $d = 250mm$
- rozměr prvků sítě MKP: $a = 50mm$
- šířky podpor: $P = 400mm$
- zatížení od sloupů: $F = 1500kN$
- šířky sloupů: $S = 400mm$
- liniové zatížení: $f = 30kN/m$
- vlastní tíha: $q = 25kN/m^3$



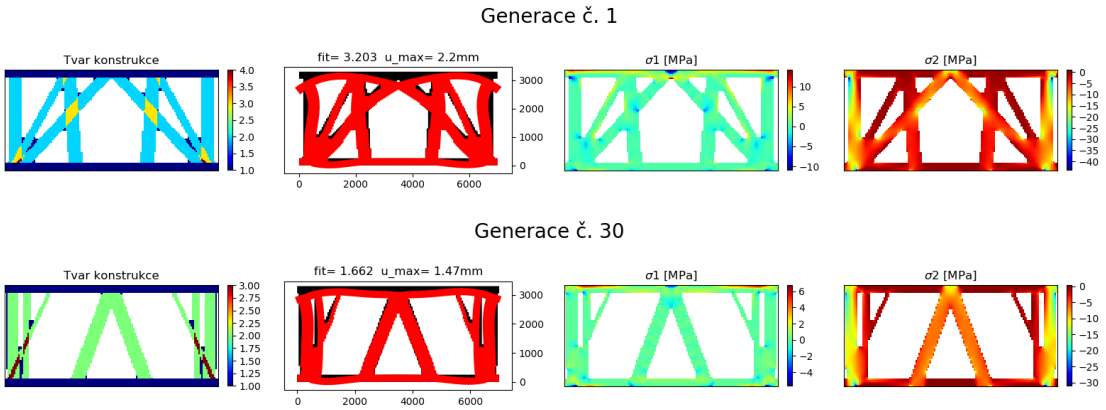
Obrázek 4.17: Symetrické podepření - okrajové podmínky pro generátor linií.

Pomocí generátoru linií byla vytvořena počáteční populace a algoritmus pokračoval dle schématu na Obr. 4.12. Na Obr. 4.18 je ukázka tvaru stěny nejlepšího jedince spolu s vykreslenými deformacemi a napětím.



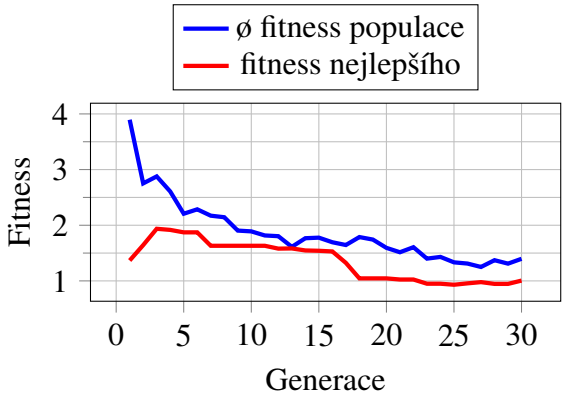
Obrázek 4.18: Tvar nejlepšího jedince v 9. generaci.

Pro symetrické podepření se však jevilo lepší generátor poupravit a konstrukci zrcadli kolem svislé osy. Na Obr. 4.24 je ukázka výsledku úpravy generátoru.



Obrázek 4.19: Porovnání nejlepšího jedince v první a poslední generaci.

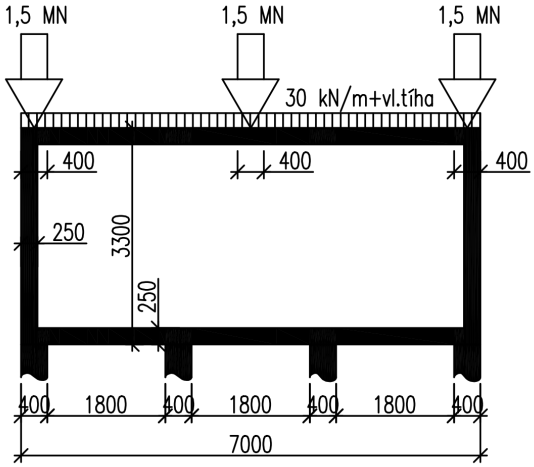
Aby bylo možné posoudit funkčnost vytvořeného generátoru, byly zaznamenány hodnoty *fitness* nejlepšího jedince populace a průměrná hodnota *fitness* celé populace. Na Obr. 4.20 je ukázan jejich vývoj, ze kterého vyplývá, že hodnota *fitness* postupně klesala. Během křížení a mutace tedy vznikali jedinci, kteří byli lepší než jejich rodiče a projevilo na to snížení deformací a napětí.
Přehled nejlepších jedinců z vybraných generaci je umístěný v příloze A.



Obrázek 4.20: Vývoj hodnoty fitness pro generátor linií - symetrické podepření.

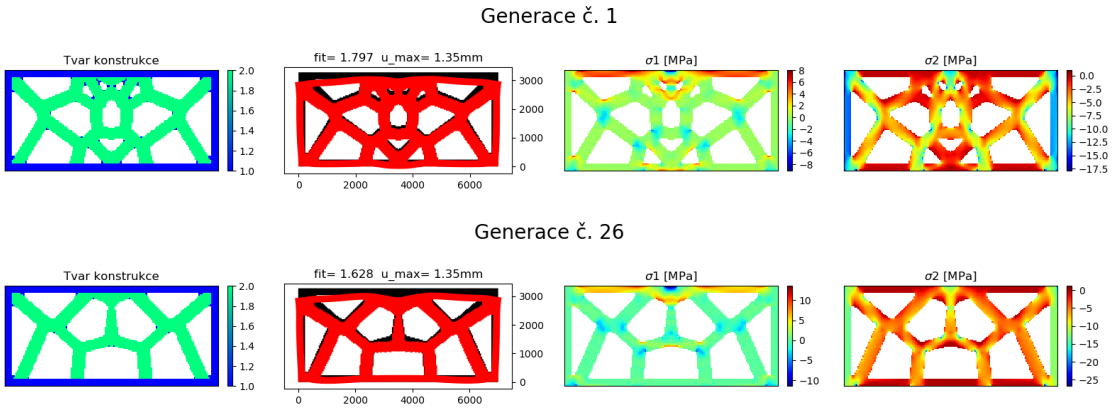
4.4.2 Voronoi generátor

V tomto případě byly generátor rovněž upraven tak, aby výsledné konstrukce byly symetrické. Kvůli způsobu, jakým jsou tvořeny stěny pomocí daného generátoru, byly na okraje přidány dva svislé pruhy o šířce 250mm. Další parametry výpočtu zůstaly stejné jako v předchozím případě. Schéma je ukázáno na Obr. 4.21.



Obrázek 4.21: Symetrické podepření - okrajové podmínky pro Voronoi generátor.

Na Obr. 4.27 je ukázka výsledku optimalizace. Přehled nejlepších jedinců z vybraných generaci je umístěný v příloze A.

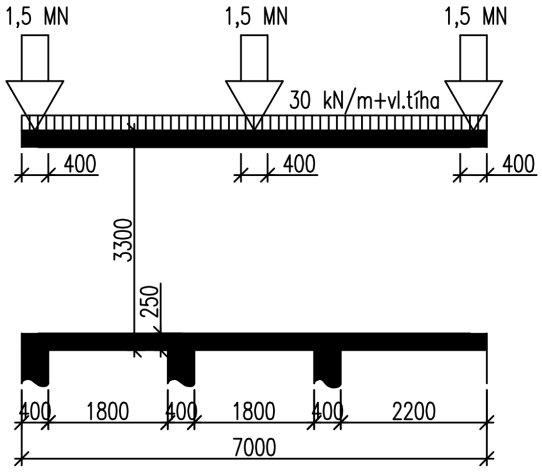


Obrázek 4.22: Vývoj nejlepšího jedince populace.

4.5 Příklad 2 - nesymetricky podepřená konstrukce

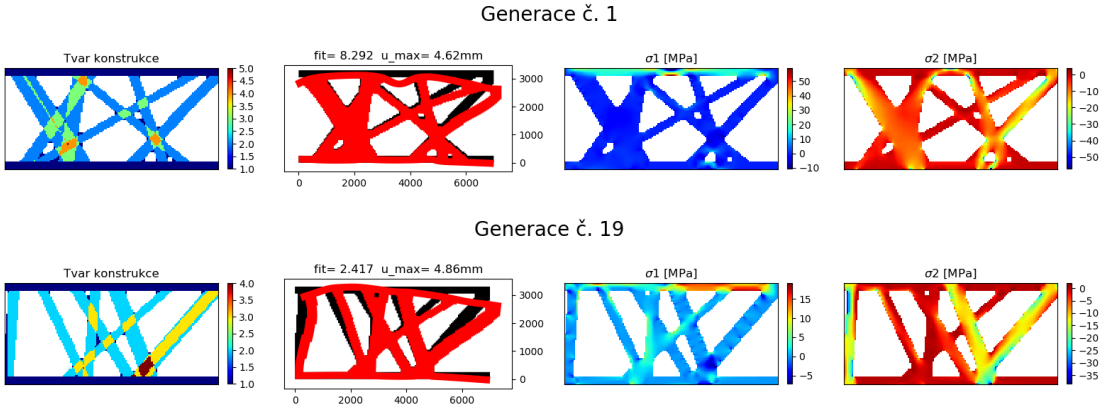
V tomto případě byla odebrána pravá krajní podpora. Schéma je zobrazeno na Obr. 4.23. Zbylé parametry výpočtu zůstaly stejné.

4.5.1 Generátor linií



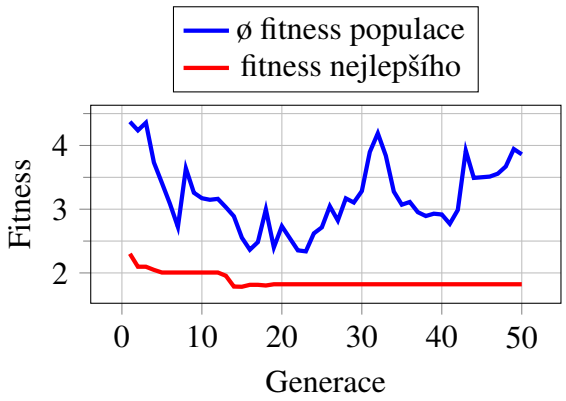
Obrázek 4.23: Nesymetrické podepření - okrajové podmínky pro generátor linií.

Výsledek optimalizace je na Obr. 4.24. Maximální průhyb zůstal prakticky stejný, ale došlo k výraznému zmenšení napětí.



Obrázek 4.24: Vývoj nejlepšího jedince populace.

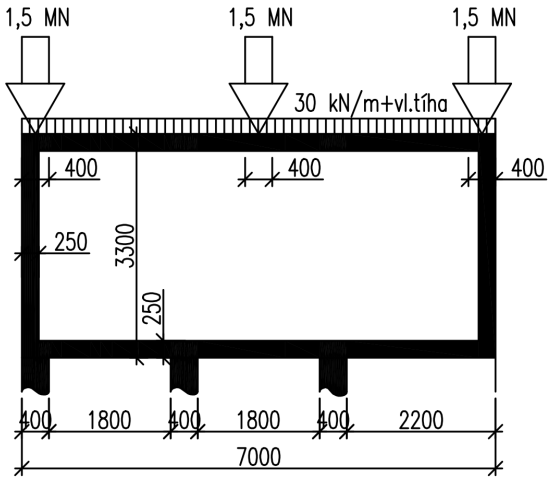
V grafu na Obr. 4.25 je ukázán vývoj hodnoty *fitness*. Zatímco *fitness* nejlepšího jedince postupně klesala a od 19. generace se neměnila, průměrná *fitness* populace se v průběhu výrazně měnila. Stoupání této hodnoty zřejmě měli za následek nově vygenerovaní jedinci, kteří byli výrazně horší než zbytek populace.



Obrázek 4.25: Vývoj hodnoty fitness pro generátor linií - nesymetrické podepření.

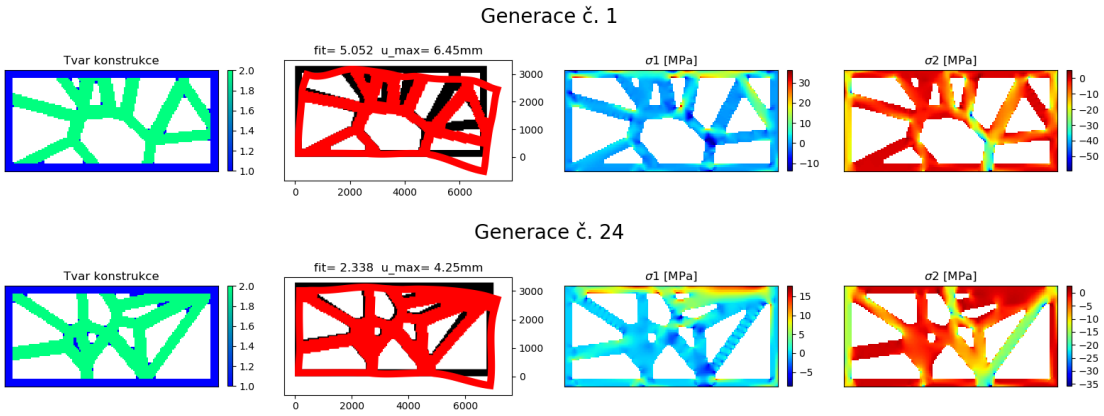
4.5.2 Voronoi generátor

Stejně jako v předchozím případě, byla i zde odebrána pravá krajní podpora. Schéma je zobrazeno na Obr. 4.26. Zbylé parametry výpočtu rovněž zůstaly nezměněny.



Obrázek 4.26: Nesymetrické podepření - okrajové podmínky pro Voronoi generátor.

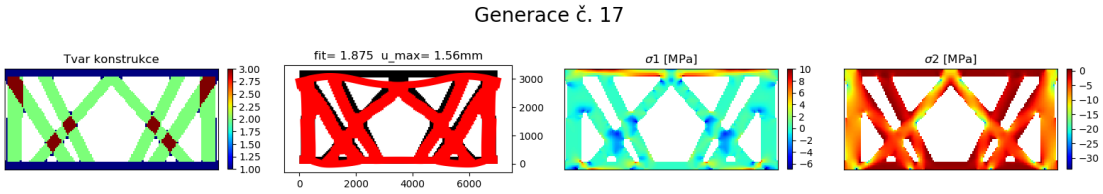
Výsledek optimalizace je ukázan na Obr. 4.27



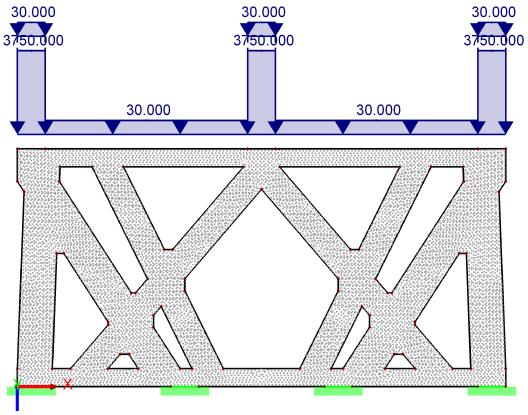
Obrázek 4.27: Vývoj nejlepšího jedince populace.

4.6 Výpočet vybrané varianty v programu RFEM

Pro ověření správnosti výpočtů pomocí MKP byl proveden výpočet v komerčním programu RFEM od společnosti Dlubal. Pro výpočet byla vybrána symetricky podepřená stěna vytvořená pomocí generátoru linií. Nejlepší jedinec v poslední generaci má sice lepší statické vlastnosti než jeho předchůdci, ale z architektonického hlediska byla vybrána nejlepší konstrukce 17. generace. Tvar konstrukce byl přenesen do programu a následně

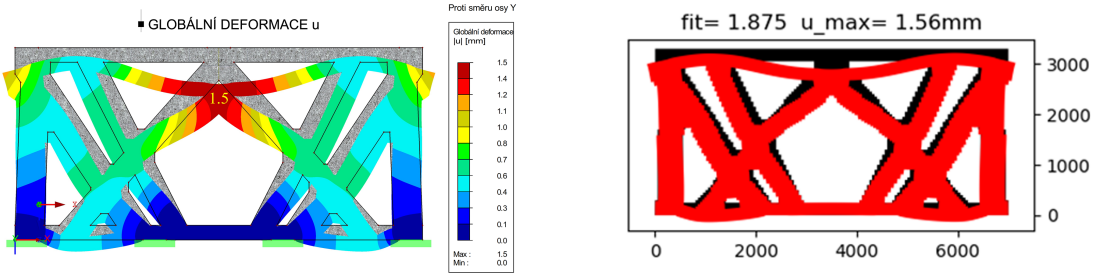


Obrázek 4.28: Vybraná konstrukce pro ověření správnosti výpočtu.



Obrázek 4.29: Definice zatížení a tvaru stěny pro výpočet v RFEMu.

podepřen a zatížen. Veškeré vstupní parametry výpočtu byly zachovány. Z Obr. 4.30 a Obr. 4.31 vyplývá, že výpočet byl proveden správně. Hodnoty a průběhy deformací a napětí jsou si velice podobné.

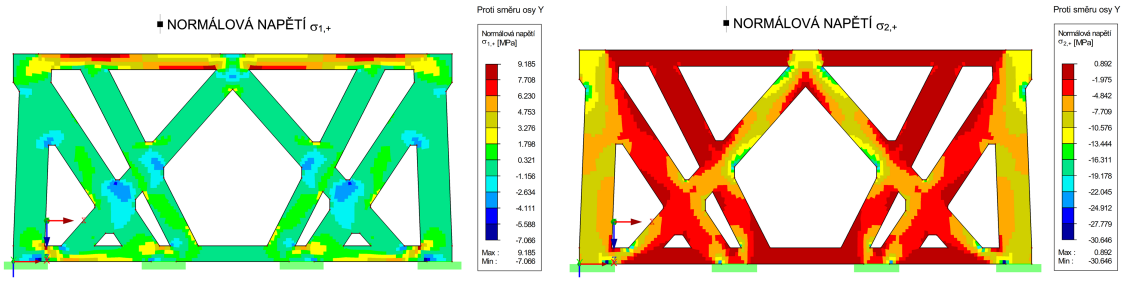


(a) Výpočet v RFEMu

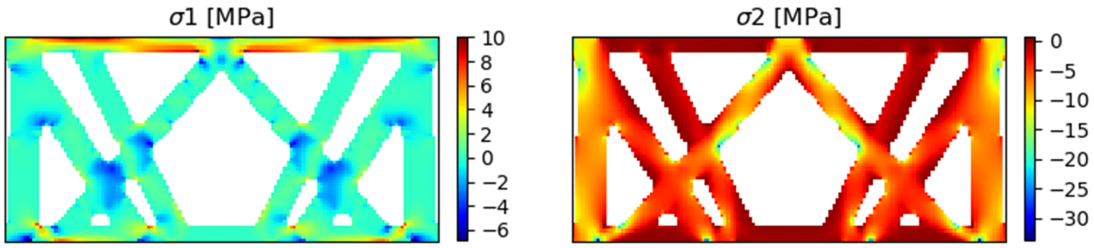
(b) Výpočet v rámci GA

Obrázek 4.30: Porovnání průhybů.

Na Obr. 4.32 je ukázka možného použití optimalizované konstrukce v praxi s naznačeným možným vyztužením.

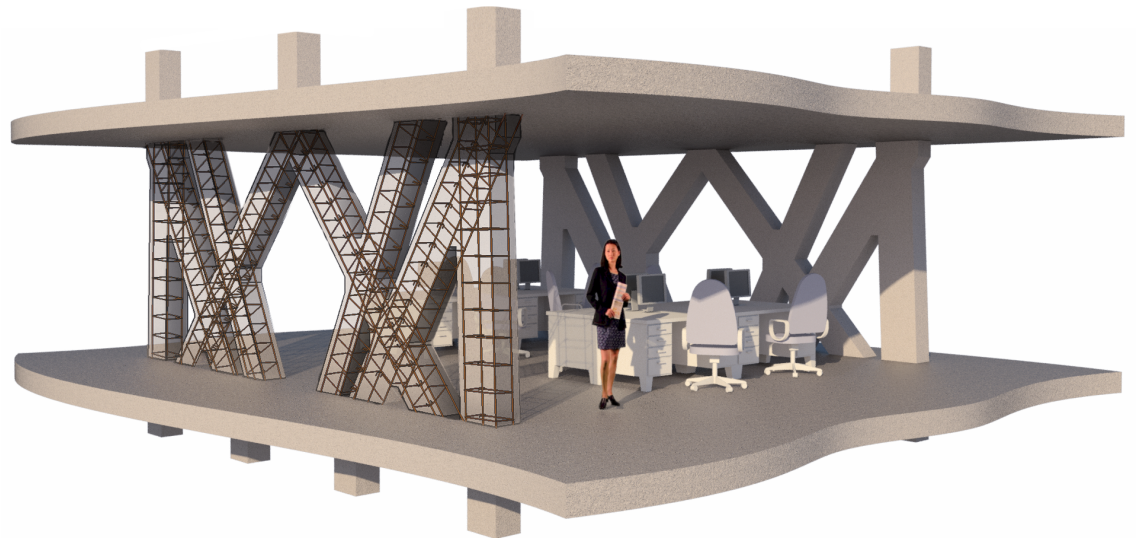


(a) Výpočet v RFEMu



(b) Výpočet v rámci GA

Obrázek 4.31: Porovnání napětí.



Obrázek 4.32: Vizualizace možné aplikace stěny v praxi.

Kapitola 5

Závěr

Cílem této práce bylo vytvoření genetického algoritmu v jazyce Python a jeho aplikace při optimalizaci betonové konstrukce.

Tento cíl se podařilo splnit. Byly vytvořeny dva nástroje pro náhodné generování tvarů stěn a nástroj pro výpočet konstrukce pomocí MKP. Funkčnost algoritmu byla ověřena na konkrétních případech optimalizace a správnost výpočtu byla ověřena při porovnání výsledků získaných z komerčního programu.

Literatura

[1] I. Zelinka, Z. Oplatková, P. Ošmera, M. Šeda a F. Včelař, *Evoluční vỳpočetní techniky*. BEN, 2008.

[2] I. Zelinka, *Hill-Climbing Algorithm (Wolfram Demonstrations Project)*, [online; datum citace 02.05.2018], 2009. WWW: `http://demonstrations.wolfram.com/HillClimbingAlgorithm/HTMLImages/index.en/popup_3.jpg`.

[3] J. Hynek, *Genetické algoritmy a genetické programování*. Grada Publishing as, 2008.

[4] P. Pospíchal, „Akcelerace genetického algoritmu s využitím GPU“, dipl, VUT v Brně, Fakulta informačních technologií, 2009.

[5] V. Studnička, „Genetické algoritmy - multi-core CPU implementace“, dipl, VUT v Brně, Fakulta strojního inženýrství, 2010.

[6] G. R. Buchanan, *Theory and problems of finite element analysis. Schaum's outline series*, 1995.

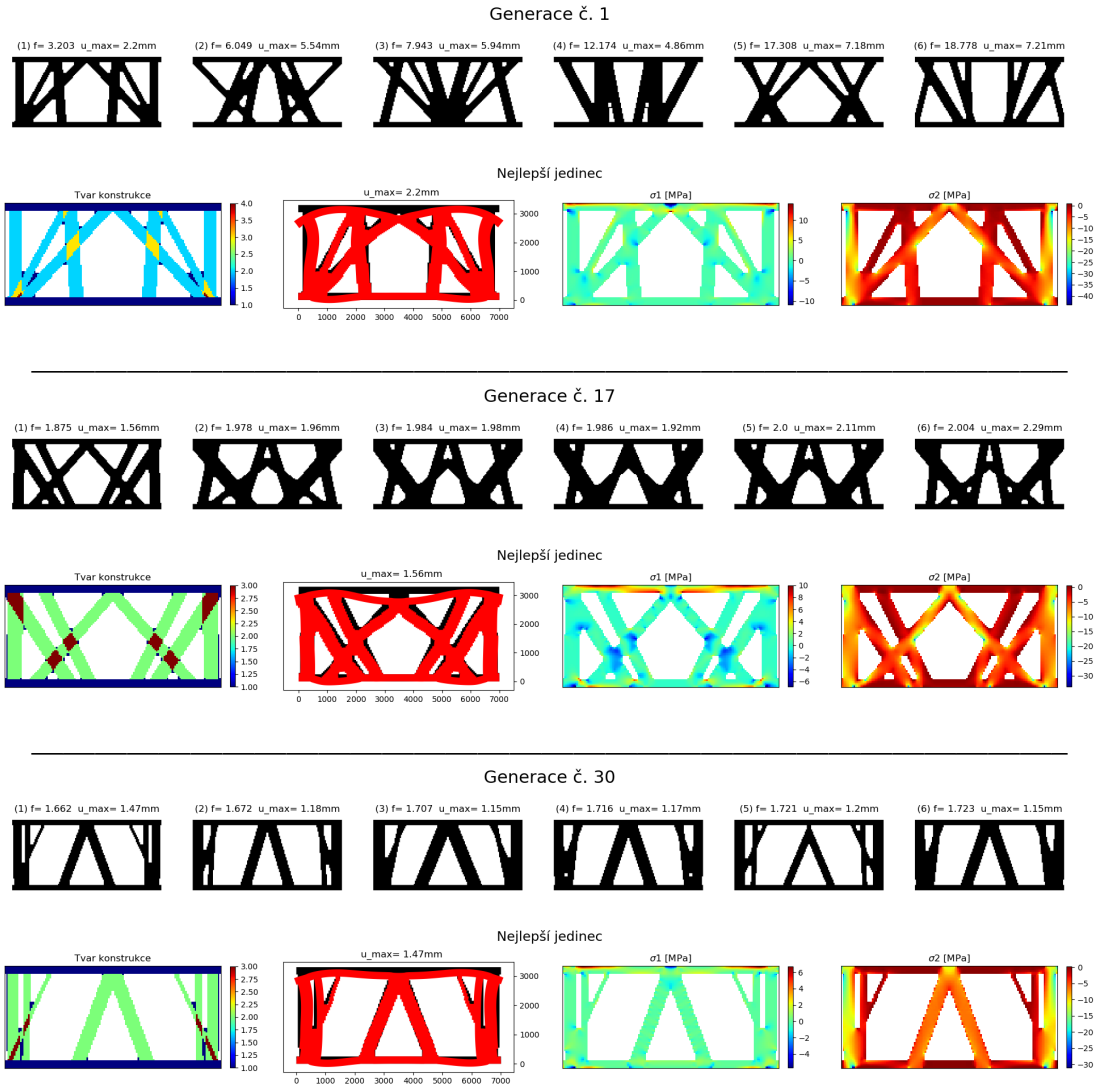
[7] I. Třosková, *Biodynamický cement, revoluční materiál vyvinutý v Itálii*, [online; datum citace 19.05.2018], 2014. WWW: `http://www.businessinfo.cz/cs/clanky/biodynamicky-cement-revolucni-material-vyvinuty-v-49115.html`.

Příloha A

V této příloze je ukázan přehled nejlepších jedinců vybraných generací pro jednotlivé příklady z Kapitoly 4.

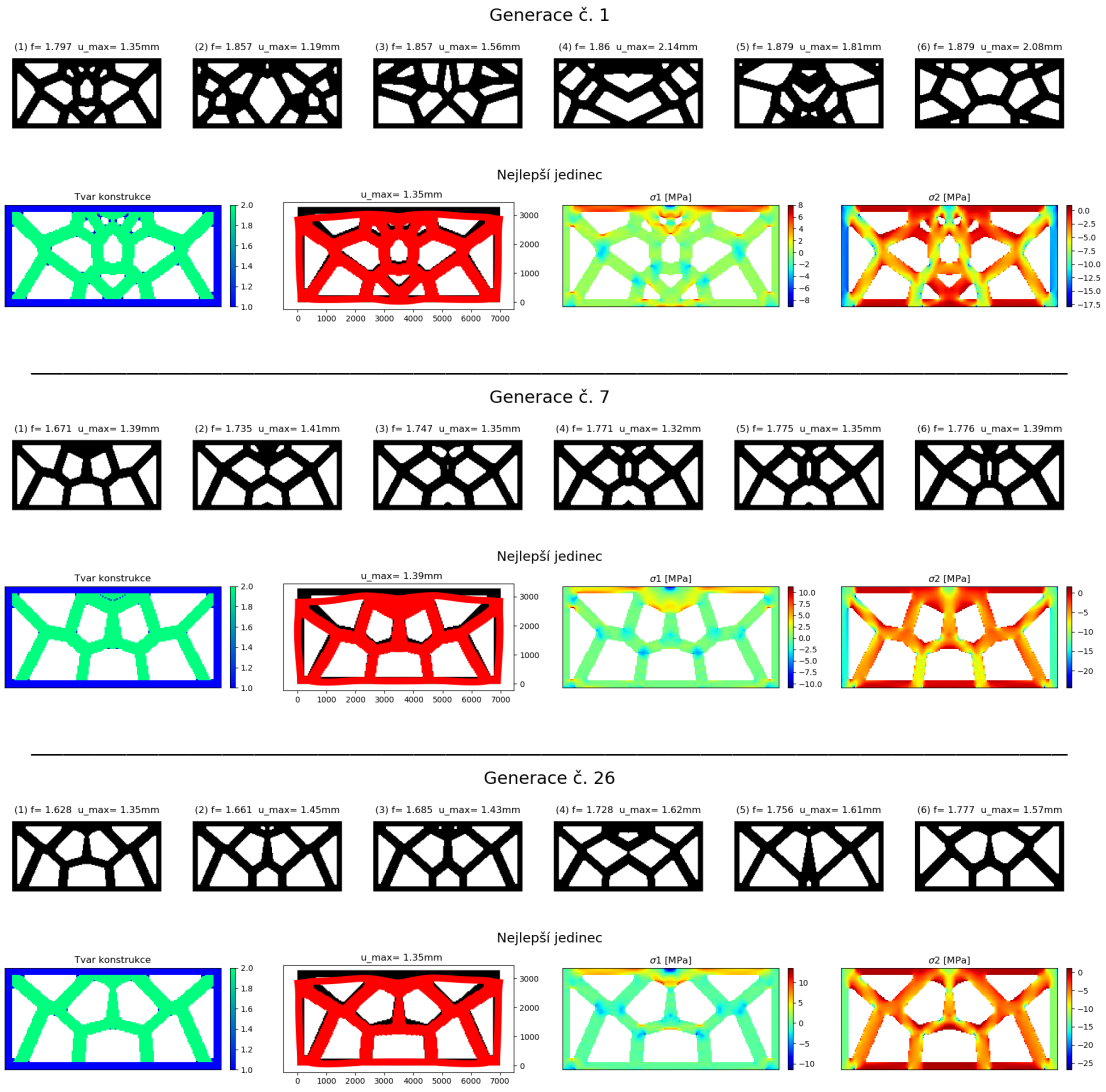
A.1 Příklad 1

A.1.1 Generátor linií



Obrázek A.1: Přehled nejlepších jedinců populace ve vybraných generacích.

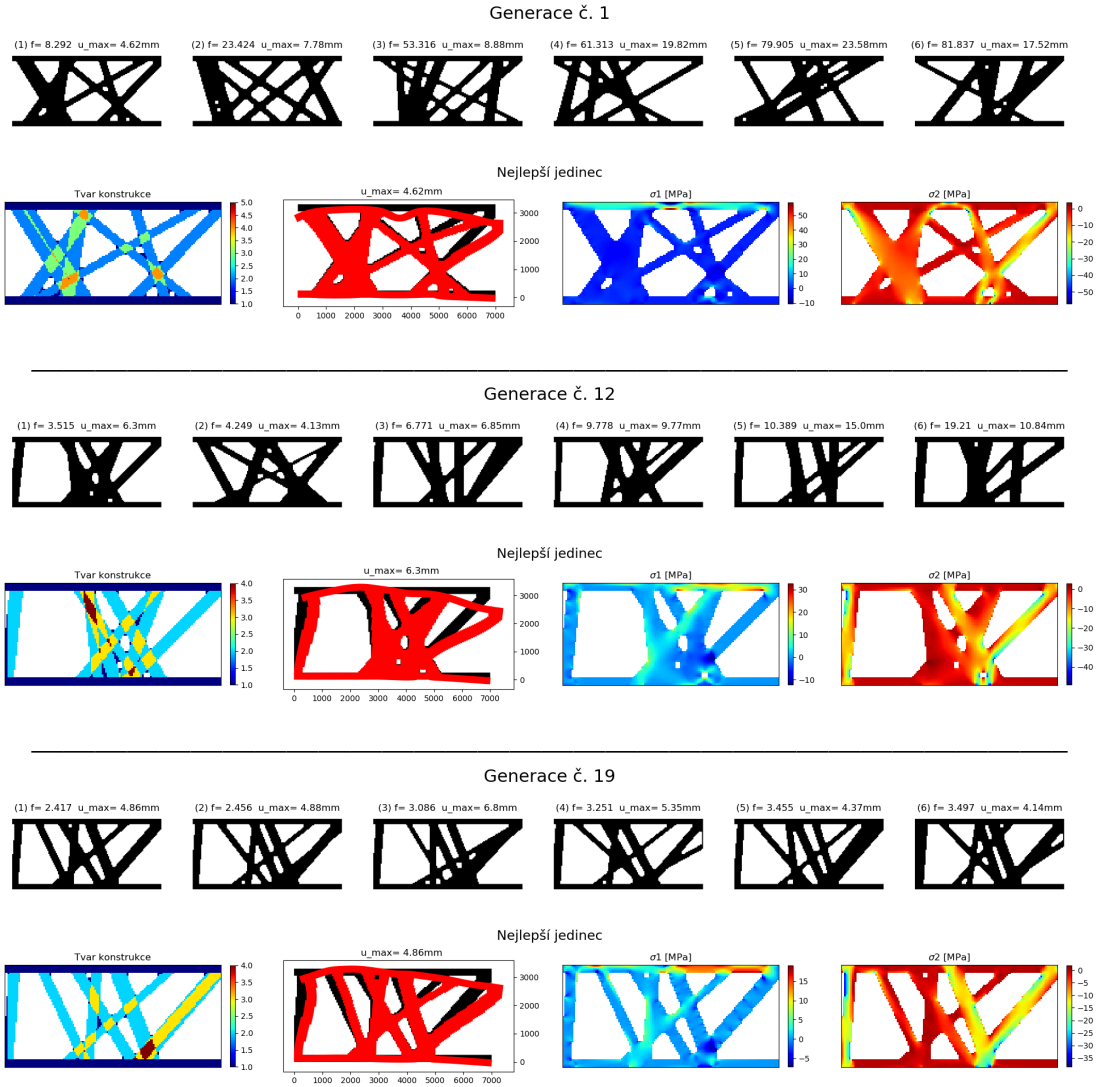
A.1.2 Voronoi generátor



Obrázek A.2: Přehled nejlepších jedinců populace ve vybraných generacích.

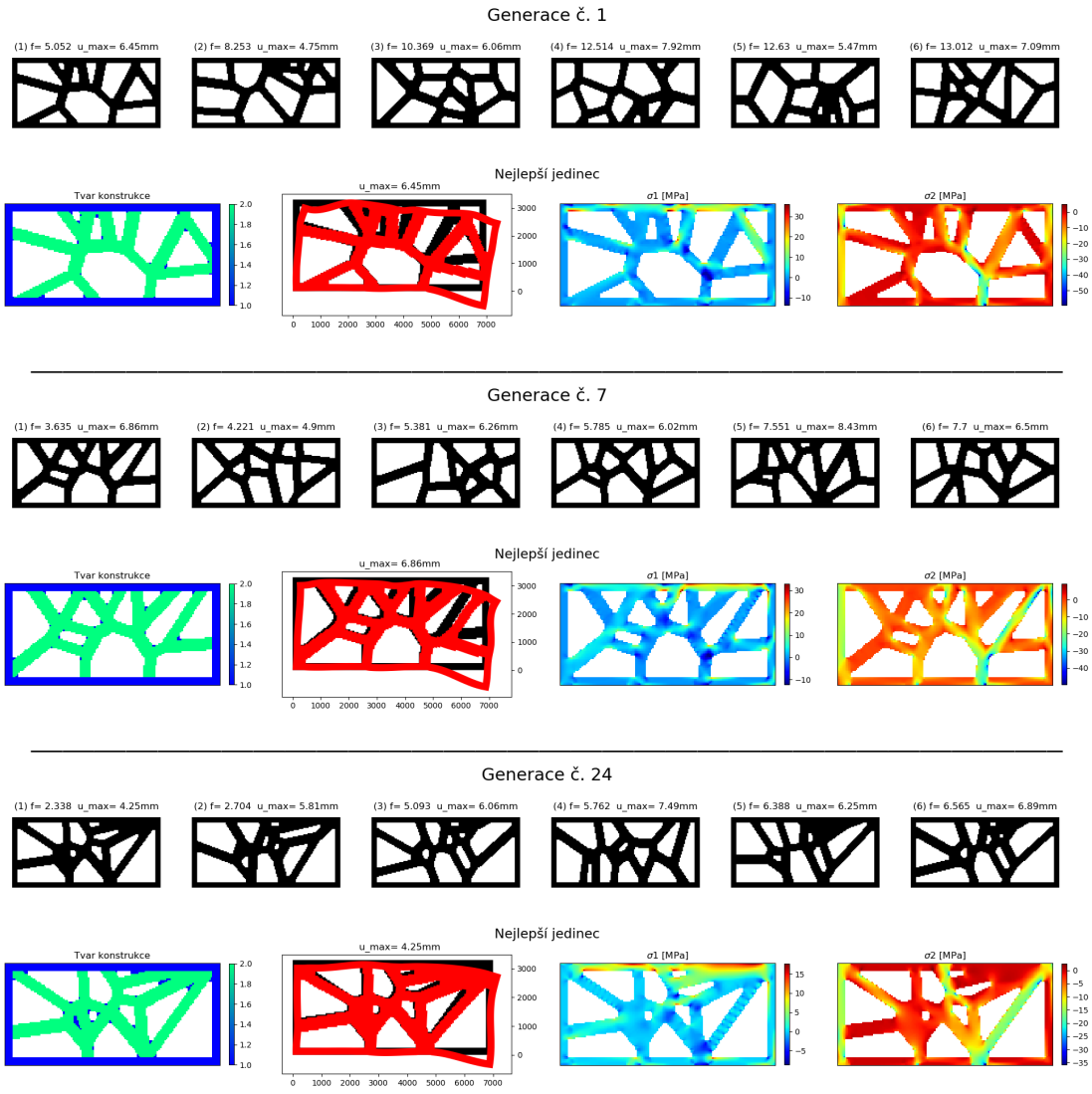
A.2 Příklad 2

A.2.1 Generátor linií



Obrázek A.3: Přehled nejlepších jedinců populace ve vybraných generacích.

A.2.2 Voronoi generátor



Obrázek A.4: Přehled nejlepších jedinců populace ve vybraných generacích.