UNIVERZITA MATEJA BELA V BANSKEJ BYSTRICI

FAKULTA PRÍRODNÝCH VIED

OPTIMALIZÁCIA ŠTRUKTÚR CHEMICKÝCH KLASTROV POMOCOU SOFT COMPUTING METÓD

Diplomová práca

10eba573-16e1-4889-84fc-cb6a986af6a4

Študijný program:	Aplikovaná informatika
Študijný odbor:	Informatika
Školiace pracovisko:	Katedra informatiky
Vedúci bakalárskej práce:	RNDr. Alžbeta Michalíková, PhD.

Banská Bystrica, 2025 **Bc. Michal Mojžiš**

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Michal Mojžiš
Študijný program: aplikovaná informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: Magisterská záverečná práca
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Optimalizácia štruktúr chemických klastrov pomocou Soft Computing metód

Anotácia: Klastre atómov, molekúl a mikrosolvatovaných iónov majú dôležitý význam v mnohých oblastiach chémie, ako je katalýza, separácia, materiálové vedy a atmosférická chémia. Určiť štruktúru klastrov experimentálne je náročné, ale veľmi dôležité pre pochopenie a predpovedanie ich chemických vlastností. Na skúmanie štruktúry klastrov je možné použiť výpočtové metódy, nájsť však najstabilnejšiu štruktúru nie je jednoduché, najmä pri zväčšovaní počtu častíc tvoriacich klaster. Kým na hľadanie lokálnych miním štruktúr chemických klastrov existujú efektívne metódy, hľadanie tých najstabilnejších štruktúr s najnižšou energiou (hľadanie globálneho minima) stále predstavuje problém.

Opíšte vlastnosti a štruktúru údajov chemických klastrov, ktoré je potrebné optimalizovať. Analyzujte súčasný stav hľadania globálneho minima vo vybraných chemických klastroch. Uveďte rozdiely v prístupoch pri hľadaním lokálneho a globálneho minima. Na základe vstupných údajov navrhnite metódu, ktorá bude slúžiť na nájdenie ich globálneho minima. Danú metódu implementujte ako program, ktorý dokáže minimalizovať energiu rozličných chemických klastrov. Výsledky vami navrhnutého programu porovnajte s výsledkami iných známych metód, ako aj s výsledkami už vyriešených klastrov.

Vedúci: RNDr. Alžbeta Michalíková, PhD.
Katedra: KIN FPV - Katedra informatiky
Vedúci katedry: doc. Ing. Jarmila Škrinárová, PhD.

Dátum zadania: 10.04.2024

Dátum schválenia smerovej katedry: 07.06.2024 prof. Dr. Ing. Miroslav Svítek, Dr.
garant študijného programu

ČESTNÉ VYHLÁSENIE

Vyhlasujem, že som diplomovú prácu vypracoval samostatne pod odborným vedením školiteľky RNDr. Alžbety Michalíkovej, PhD. a doc. RNDr. Šimona Budzáka, PhD. s použitím literatúry, ktorú uvádzam v zozname použitej literatúry v záverečnej časti práce a na základe vedomostí nadobudnutých počas štúdia.

V Banskej Bystrici, 26. apríl 2025

.....

Podpis študenta

POĎAKOVANIE

Chcel by som sa poďakovať pani RNDr. Alžbete Michalíkovej, PhD a pánovi doc. RNDr. Šimonovi Budzákovi, PhD. za ich odborné vedenie diplomovej práce, pravidelné konzultácie a pomoc pri pochopení základných princípov kvantovej chémie, ktoré mi pomáhali pri vypracovaní tejto práce. Zároveň by som sa chcel poďakovať aj mojej rodine za ich pomoc a podporu.

ABSTRAKT

MOJŽIŠ, Michal: Optimalizácia štruktúr chemických klastrov pomocou Soft Computing metód [Diplomová práca]. Univerzita Mateja Bela v Banskej Bystrici. Fakulta prírodných vied, Katedra informatiky. Vedúci diplomovej práce: RNDr. Alžbeta Michalíková, PhD. Stupeň odbornej kvalifikácie: Magister. Banská Bystrica, 2025. Počet strán: 83.

Táto práca sa zaoberá globálnou optimalizáciou štruktúr chemických klastrov pomocou genetických algoritmov. V práci sú v prvých dvoch kapitolách vysvetlené základné definície a postupy optimalizačných problémov a genetických algoritmov. Tretia kapitola sa zaoberá problematikou optimalizácie chemických klastrov a chemických výpočtových modelov. Výstupom tejto práce je program, v ktorom je implementovaný genetický algoritmus, ktorý dokáže hľadať globálne energetické minimum chemických klastrov. Štvrtá kapitola obsahuje návrh tohto algoritmu a piata kapitola opisuje finálnu implementáciu. Výsledky vytvoreného programu sú zobrazené a porovnané v šiestej kapitole.

Kľúčové slová: Globálna optimalizácia štruktúr chemických klastrov, Genetické algoritmy, Optimalizačné problémy, Soft Computing metódy

ABSTRACT

MOJŽIŠ, Michal: Optimization of chemical cluster structures using Soft Computing methods [Master's thesis]. Matej Bel University in Banská Bystrica. Faculty of Natural Sciences, Department of Informatics. Supervisor: RNDr. Alžbeta Michalíková, PhD. Degree of Qualification: Master. Banská Bystrica, 2025. Number of pages: 83.

This thesis is concerned with the global optimization of chemical cluster structures using genetic algorithms. The first two chapters of the thesis explain the basic definitions and procedures of optimization problems and genetic algorithms. The third chapter addresses the issue of optimization of chemical clusters and chemical computational models. The result of this thesis is a program in which a genetic algorithm is implemented, capable of searching for the global energy minimum of chemical clusters. The fourth chapter contains the design of this algorithm, and the fifth chapter describes the final implementation. The results of the created program are presented and compared in the sixth chapter.

Keywords: Global optimization of chemical cluster structures, Genetic algorithms, Optimization problems, Soft Computing methods

OBSAH

ZOZNAM OBRÁZKOV, TABULIEK A GRAFOV

ZOZNAM SKRATIEK A ZNAČIEK

PES – Potential Energy Surface

QM – Quantum Mechanics

MM – Molecular Mechanics

SD – Steepest Descent (method)

CG – Conjugate Gradient (method)

DFT – Density Functional Theory

xTB – Extended Tight Binding

ASE – Atomic Simulation Environment

RMSD – Root Mean Square Deviation

ÚVOD

Chemické klastre zohrávajú dôležitú rolu v dnešnom výskume, najmä v oblastiach, ako sú nanotechnológie alebo materiálová veda. Každý chemický klaster má svoju špecifickú štruktúru, ktorá definuje jeho jedinečné fyzikálno-chemické vlastnosti. Optimalizáciou týchto štruktúr je možné lepšie porozumieť ich stabilite, reaktivite a ďalším dôležitým vlastnostiam. Hľadanie najstabilnejších (t. j. optimálnych) štruktúr je ale výpočtovo náročný proces, ktorý si vyžaduje efektívne algoritmy prehľadávania priestoru riešení a presné chemické výpočtové modely. Cieľom tejto práce je navrhnúť a vytvoriť takýto algoritmus, ktorý bude slúžiť na nájdenie globálneho energetického minima daného vstupného chemického klastra.

Práca sa v prvej kapitole zaoberá prehľadom vývoja matematickej disciplíny optimalizácie, všeobecnou definíciou optimalizačného problému, klasifikáciou a metódami riešenia optimalizačných problémov. Druhá kapitola je zameraná na oblasť evolučných algoritmov so zameraním na genetické algoritmy. V tejto kapitole sú vysvetlené základné princípy genetických algoritmov a uvedené stratégie na hľadanie optimálnych parametrov týchto algoritmov. V tretej kapitole je priblížený chemický aspekt riešeného problému, pričom sa berú do úvahy primárne veci dôležité a potrebné pre vykonávanie globálnej optimalizácie. Sú tu uvedené základne definície chemických klastrov, chemických výpočtových modelov a optimalizačných algoritmov.

Štvrtá kapitola obsahuje návrh riešenia vytváraného programu. Je tu uvedený dôvod výberu genetických algoritmov, štruktúra vytváraného programu a definícia funkcionálnych a nefunkcionálnych požiadaviek, ktoré musí vytvorený program spĺňať. V piatej kapitole je opísaná implementácia vytvoreného programu a jeho jednotlivých častí. Šiesta kapitola prezentuje dosiahnuté výsledky na testovacích úlohách a porovnanie výsledkov a výkonnosti s iným algoritmom globálnej optimalizácie.

1. OPTIMALIZAČNÉ PROBLÉMY

Optimalizačné problémy predstavujú typ problému, s ktorým sa môžeme stretnúť skoro každodenne, aj v bežnom živote. Patria sem problémy, ako napríklad výber najkratšej alebo najrýchlejšej cesty do práce alebo minimalizovanie nákladov pri rozličných výrobných procesoch.

Optimalizačnými problémami sa zaoberali už matematici v antickej dobe. V rozmedzí rokov 300 – 100 pred Kr. sa grécki matematici, medzi ktorých patrili Euklides a Hérón, venovali riešeniu rôznych geometricky motivovaných optimalizačných úloh. V 17. a 18. storočí sa matematici, ako Johannes Kepler a Pierre de Fermat, zaoberali rozličnými optimalizačnými problémami. Pierre de Fermat skúmal, ako sa svetlo šíri v priestore medzi dvoma bodmi, pričom objavil princíp najmenšieho času, ktorý hovorí, že zo všetkých možných dráh, ktorými sa dá dostať z jedného bodu do druhého si svetlo vyberá takú, ktorú prejde za najkratší čas. Johannes Kepler, okrem jeho astronomických objavov, pri hľadaní (optimálnej) novej manželky sformuloval základy problému, ktorý dnes patrí do formalizovanej oblasti problémov optimálneho zastavenia. Neskôr vedci Isaac Newton a Gottfried W. Leibniz definovali základy diferenciálneho a integrálneho počtu, ktorý neskôr formalizovali Leonhard Euler a Joseph-Louis Lagrange. Pomocou formalizovaného variačného počtu bolo riešenie optimalizačných úloh (s funkciami) výrazne uľahčené. Ďalší vývoj optimalizácie v histórii je charakterizovaný tvorbou optimalizačných algoritmov, medzi ktoré patrí napríklad gradientová metóda, ktorej základné myšlienky definoval Augustin-Louis Cauchy a aplikovaním optimalizácie v rôznych oblastiach, ako sú napríklad ekonomika alebo operačná analýza. S nástupom počítačov, ktoré dokázali riešiť komplexné optimalizačné úlohy, sa začalo vyvíjať veľké množstvo optimalizačných algoritmov, ako napríklad genetické alebo rojové algoritmy [1].

Disciplína matematiky, ktorá sa zaoberá riešením optimalizačných problémov, sa nazýva optimalizácia. Cieľom optimalizácie je nájsť globálne minimum resp. maximum jednej alebo viacerých účelových funkcií pri dodržaní obmedzení špecifických pre daný problém. V dnešnej dobe sa optimalizácia využíva na hľadanie optimálnych riešení v rôznych odvetviach, ako sú napríklad ekonomika, medicína alebo umelá inteligencia. Optimalizačný problém pre minimalizáciu je možne matematicky vyjadriť nasledovne [2]:

Nájdite vektor $\mathbb{X} = (x_1, x_2, \dots, x_n)$, ktorý minimalizuje funkciu $f_i(\mathbb{X})$, pre $i = 1, 2, \dots, n_o$ pričom musí platiť:

$$g_j(\mathbb{X}) \leq 0, j = 1, 2, \dots, n_g$$

$$h_k(\mathbb{X}) = 0, k = 1, 2, \dots, n_e$$

$$x_m^l \leq x_m \leq x_m^u, m = 1, 2, \dots, n_s$$

Vektor $\mathbb{X}$ predstavuje n premenných účelovej funkcie, $f_i(\mathbb{X})$ predstavuje účelovú funkciu, ktorá sa má minimalizovať. $g_j(\mathbb{X})$ resp. $h_k(\mathbb{X})$ sú obmedzenia nerovností resp. rovností, ktoré ohraničujú množinu riešení vzhľadom na charakter problému alebo požiadavky systému. Hodnoty x_m^l a x_m^u predstavujú dolné a horné ohraničenie hodnôt, ktoré môže dané $x_m \in \mathbb{X}$ dosahovať. Hodnoty n_o, n_g, n_e, n_s vyjadrujú počet účelových funkcií, počet obmedzení nerovností, počet obmedzení rovností a počet ohraničení optimalizovaných premenných. Pomocou tohto zápisu je možné formálne definovať optimalizačné úlohy [2].

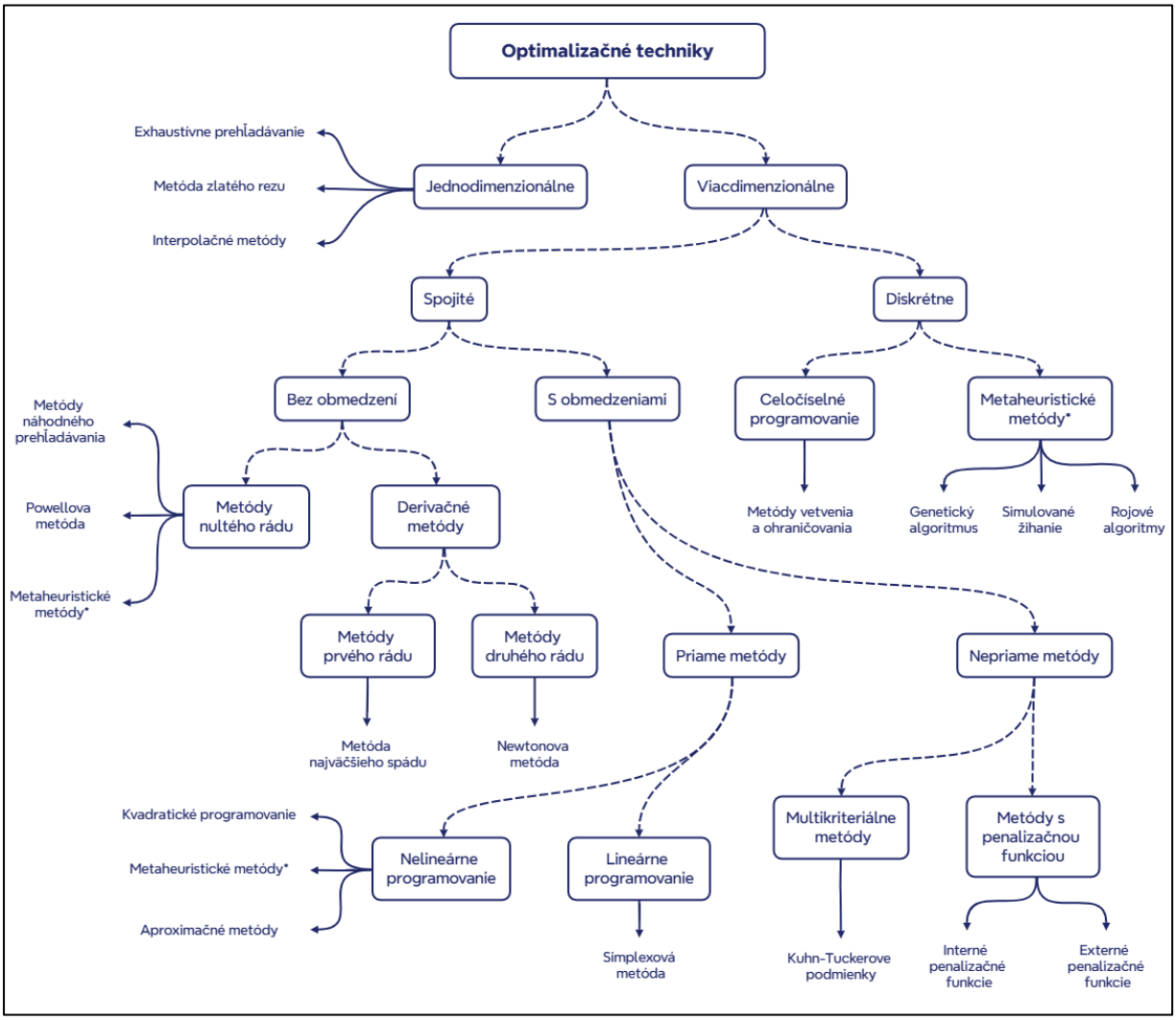
1.1. Klasifikácia optimalizačných problémov

Existuje viacero prístupov k tomu, ako klasifikovať optimalizačné problémy. Jedna z možností je rozdelenie do dvoch skupín na základe typu problému. Prvá skupina sa označuje ako funkčná optimalizácia, kde skúmaný problém je vyjadrený jednou alebo viacerými neznámymi funkciami. Cieľom optimalizácie v tomto prípade je nájdenie optimálnych funkčných hodnôt funkcií. Využíva sa tu napríklad diferenciálny kalkulus alebo variačný počet. Druhá skupina problémov sa nazýva parametrová optimalizácia, kde sa namiesto hľadania optimálnych hodnôt na spojitých funkciách hľadajú optimálne hodnoty pre premenné vstupného vektora $\boldsymbol{X}$. Na riešenie týchto problémov sa využíva napríklad matematické programovanie alebo metaheuristické metódy [2]. Ďalšie klasifikácie optimalizačných problémov sú uvedené v *Tabuľke č.1*.

Na základe klasifikácie optimalizačných problémov je možné k nim priradiť optimalizačné algoritmy a metódy, ktoré ich dokážu efektívne riešiť. Na *Obrázku č.1* sú zobrazené vybrané príklady techník riešenia rozličných typov optimalizačných problémov. Napríklad pre malé jednodimenzionálne problémy je možné prehľadávať celý priestor hrubou silou alebo pre väčšie problémy je možné využiť interpolačné metódy. Metaheuristické metódy sa v tejto schéme vyskytujú viackrát, pričom sú vyznačené * a vyjadrujú rovnaké metódy, ktoré sú uvedené len raz pri diskrétnych úlohách.

Tabuľka 1: Klasifikácia optimalizačných problémov, zdroj: [2]

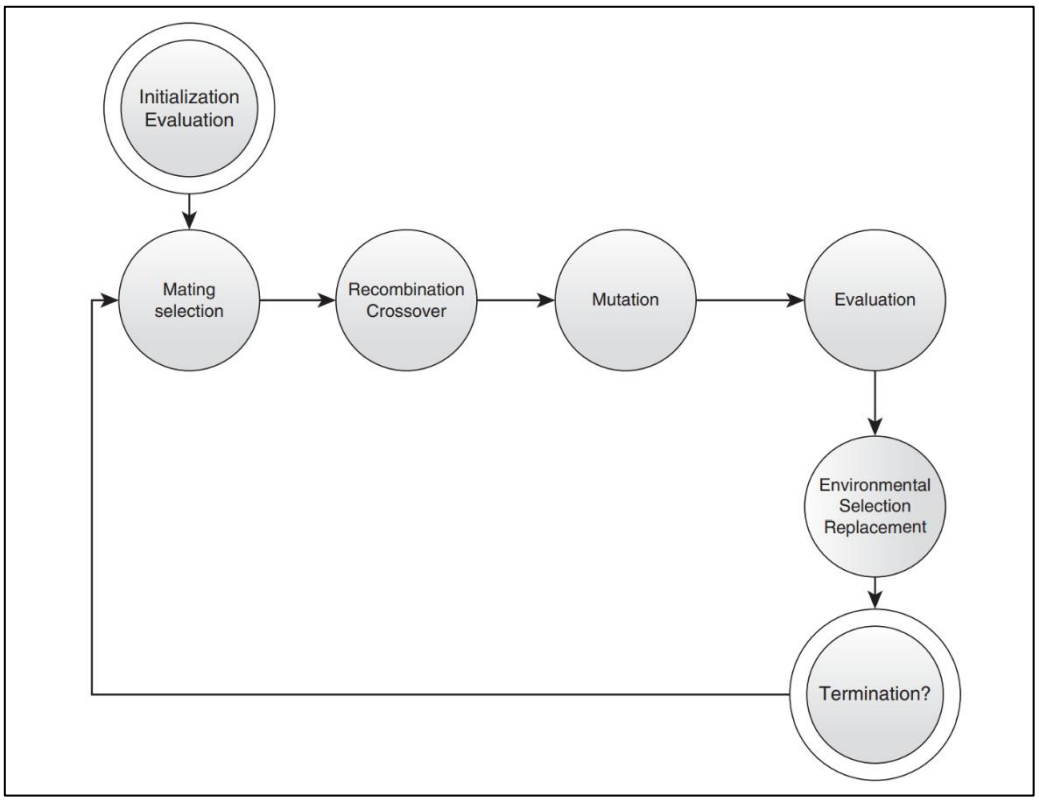
Typ klasifikácie	Kategória	Popis
Počet optimalizovaných premenných	Jedna premenná	Problém obsahuje iba jednu premennú na optimalizovanie.
	Viacero premenných	Problém obsahuje viacero premenných na optimalizovanie.
Počet účelových funkcií	Jednocieľový	Obsahuje jeden cieľ, vyjadrený pomocou jednej účelovej funkcie.
	Viaccieľový	Obsahuje viacero účelových funkcií, ktoré sú optimalizované súčasne.
Obmedzenia	Bez obmedzení	Neobsahuje žiadne ohraničenia účelovej funkcie.
	S obmedzeniami	Obsahuje ohraničenia, ktoré definujú množinu prípustných riešení.
Vlastnosti obmedzení a účelových funkcií	Lineárne programovanie	Účelové funkcie a obmedzenia sú lineárne.
	Nelineárne programovanie	Niektoré účelové funkcie a obmedzenia sú nelineárne. Patrí sem napríklad kvadratické programovanie.
Povaha optimalizovaných premenných	Statické	Premenné sú nezávislé - nie sú funkcie iných parametrov.
	Dynamické	Premenné sú funkcie iných parametrov - napríklad času.
Typ optimalizovaných premenných	Diskrétne	Premenné môžu obsahovať iba celé čísla alebo diskrétne hodnoty.
	Spojité	Premenné môžu obsahovať ľubovolné reálne číslo.
	Zmiešané	Niektoré premenné sú diskrétne a niektoré spojité.
Povaha optimalizovaných premenných a vstupných dát	Deterministické	Všetky premenné a parametre sú deterministické.
	Pravdepodobnostné	Všetky alebo niektoré optimalizované premenné a parametre sú opisané náhodnými alebo pravdepodobnostnými premennými
Povaha účelových funkcií a obmedzení	Ostré	Účelové funkcie a obmedzenia sú vyjadrené deterministickými výrazmi.
	Fuzzy	Účelové funkcie a obmedzenia sú vyjadrené fuzzy výrazmi.



Obrázok 1: Príklady optimalizačných techník, zdroj: [2]

2. GENETICKÉ ALGORITMY

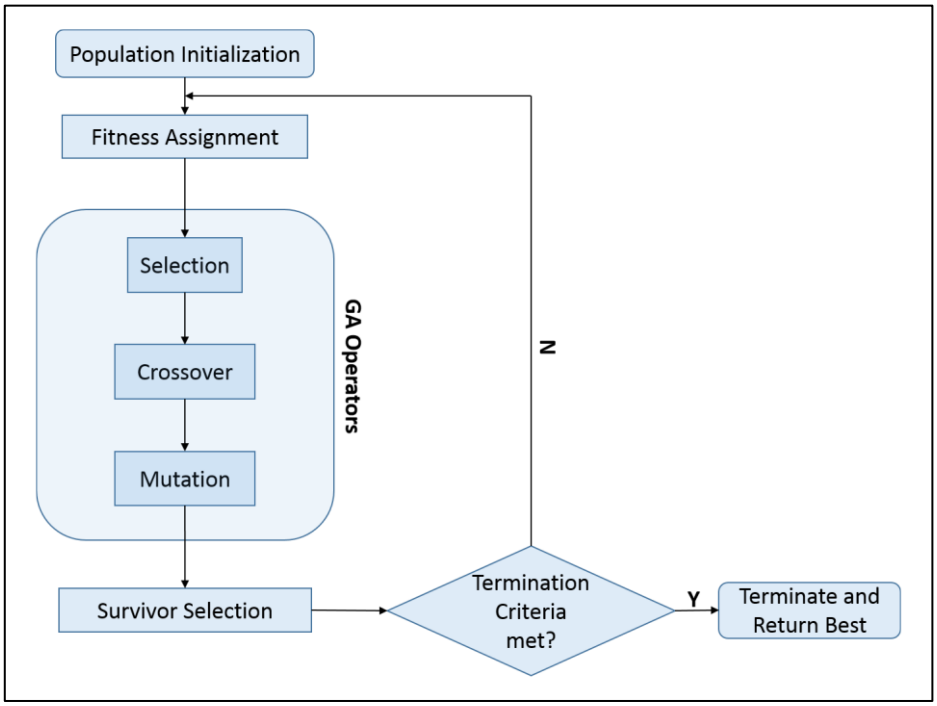
Genetický algoritmus patrí do skupiny evolučných algoritmov. Evolučné algoritmy predstavujú skupinové pomenovanie pre vyhľadávacie stochastické algoritmy, ktoré využívajú princípy populácie a prirodzenej selekcie podloženej teóriou evolúcie. Vývoj prvých evolučných algoritmov začal v druhej polovici 20. storočia. Medzi prvé metódy patrili *genetické algoritmy*, ktoré vznikli za cieľom riešenia kombinatorických problémov pomocou binárnych reťazcov inšpirovaných genetickým kódovaním v prírode. Ďalšie metódy boli *evolučné programovanie* a *evolučné stratégie*, ktoré vznikli na optimalizovanie konečných automatov a inžinierskych problémov. Neskôr, okolo roku 1990, vzniklo *genetické programovanie*, ktorého cieľom bolo optimalizovať počítačové programy. Tieto štyri metódy sú často uvádzané ako jedny zo základných metód evolučných algoritmov, pričom vývoj nových metód napredoval a dnes sa využívajú napríklad algoritmy založené na neurónových sieťach. Na *Obrázku č.2* je znázornená všeobecná schéma, ktorá zobrazuje postupy evolučných algoritmov, pričom každá metóda môže mať inú implementáciu daného kroku [3].



Obrázok 2: Všeobecná schéma evolučných algoritmov, zdroj: [3]

2.1. Štruktúra genetického algoritmu

Ako bolo naznačené v úvode kapitoly, genetický algoritmus má obdobnú štruktúru ako ostatné evolučné algoritmy. Na začiatku sa vygeneruje počiatočná populácia, ktorá reprezentuje jednotlivé riešenia, inak nazývané aj jedinci populácie. Každý jedinec má určitú fitness hodnotu, ktorú im priraďuje fitness funkcia. Cieľom genetického algoritmu je nájsť optimum tejto funkcie za pomoci iteratívneho aplikovania genetických operátorov [4]. Štruktúra štandardného genetického algoritmu je zobrazená na *Obrázku č.3* a nasledujúce podkapitoly vysvetľujú jednotlivé kroky genetického algoritmu v poradí na základe tejto štruktúry.



Obrázok 3: Štruktúra štandardného genetického algoritmu, zdroj: [4]

2.1.1. Populácia a kódovanie jedincov

Genetický algoritmus patrí medzi algoritmy, ktoré využívajú populáciu, skladajúcu sa z jedincov. Jedinec predstavuje jedno konkrétne potenciálne riešenie. Základná terminológia používaná v genetike, ktorou je tento algoritmus inšpirovaný, označuje zakódovaného jedinca ako chromozóm. Chromozóm sa skladá z genotypu, ktorý sa následne skladá z reťazcov génov. Genotyp je možné dekódovať na fenotyp a pomocou fitness funkcie určiť jeho fitness hodnotu. Na začiatku každého návrhu genetického algoritmu je potrebné zvoliť, ako zakódovať jedincov. Existuje veľa spôsobov, akým je možné jedinca zakódovať, patria sem napríklad binárne alebo permutačné kódovanie a je veľmi dôležité zvoliť vhodný typ, aby zakódovaný

jedinec zároveň dokázal vhodne reprezentovať reálne riešenie v pôvodnej forme a zároveň bolo možné naňho aplikovať genetické operátory [5].

Existuje viacero podmienok, ktoré definujú, kedy je zvolený typ kódovania vhodný a aké kritéria by mal spĺňať. V problémoch reálneho sveta sa často stáva, že nie je možné navrhnúť kódovanie, ktoré by spĺňalo všetky kritéria a je potrebné nájsť vhodné kompromisy. Medzi niektoré kritéria patria napríklad nasledovné body [5]:

- Vhodne opisuje základné stavebné bloky dôležité pre daný typ problému.
- Je schopný tieto základné stavebné bloky propagovať z rodičovských genotypov na potomkov za pomoci genetických operátorov.
- Minimalizuje epistázu, ktorá vyjadruje, do akej miery sú navzájom jednotlivé gény previazané. Pri nízkej hodnote každý gén samostatne ovplyvňuje výsledné riešenie.
- Podporuje jednoduché mapovanie genotypu na fenotyp pre rýchly výpočet fitness hodnoty.
- Obsahuje iba prípustné riešenia a v prípade, že nie, musia existovať buď penalizačné funkcie alebo vhodný opravný algoritmus.
- Obmedzuje izomorfizmus genotypov, čo predstavuje situáciu, kedy viacero rozličných genotypov reprezentuje rovnaký fenotyp.

Po zadefinovaní vhodného kódovania je možné prejsť na tvorbu populácie. Populácia predstavuje charakteristickú stratégiu genetických algoritmov, pomocou ktorej sa súčasne uchováva viacero riešení daného problému v každej iterácii. Iterácie sa preto taktiež označujú ako generácie. Jedna z výhod použitia populácie je napríklad to, že ak najlepší jedinec v danej generácii uviazol v lokálnom minime, je možné ho pomocou ostatných jedincov a genetických operátorov uvoľniť a zároveň preskúmať okolie daného minima [6].

V prvom kroku každého genetického algoritmu je potrebné vytvoriť počiatočnú generáciu. Pri štandardnom prístupe sa nastaví hodnota, ktorá reprezentuje počet jedincov v každej generácii, ktorý je nemenný. Následne sa vygenerujú náhodní jedinci tak, aby ich diverzita bola čo najväčšia. To znamená, že jedinci sú generovaní tak, aby uniformne pokryli celý priestor prípustných riešení. Na dosiahnutie tohto výsledku je možné využiť napríklad Gaussovu normálnu distribúciu. Existujú taktiež modifikované genetické algoritmy, kde je

veľkosť populácie dynamická a mení sa na základe behu programu. Táto modifikácia vznikla z dôvodu, že ak je počet jedincov veľmi nízky, tak šanca nájsť globálne minimum resp. maximum sa znižuje, ale naopak, ak je veľmi vysoký, tak aj výpočtová náročnosť sa zvyšuje. Táto situácia poukazuje na to, že je potrebné vhodne zvoliť veľkosť populácie tak, aby jedinci dokázali vhodne pokryť všetky riešenia, ale zároveň, aby algoritmus nebol extrémne výpočtovo náročný [6], [7].

2.1.2. Fitness funkcia

Fitness funkcia je účelová funkcia špecifická pre doménu daného problému. Jej cieľom je vyhodnocovať vhodnosť potenciálnych riešení genetického algoritmu. Keďže priraďovanie fitness hodnôt je jedna z najčastejšie vykonávaných operácií v genetických algoritmoch, tak správny návrh tejto funkcie je veľmi dôležitý. Kvalita fitness funkcie priamo vplýva na výkon a kvalitu výsledkov genetického algoritmu. V prípade zlého návrhu môžu nastať situácie, ako napríklad, že funkcia je schopná rýchlo vyhodnotiť fitness hodnotu jedinca, ale priradené hodnoty obsahujú výrazný šum a na konci algoritmu nájdené optimálne riešenie nezodpovedá hľadanému globálnemu minimu resp. maximu. Iná situácia, ktorá môže nastať, je napríklad, že fitness funkcia síce dáva presné výsledky, ale jej výpočet je extrémne výpočtovo náročný a nie je možné priestor riešení prehľadávať v reálnom čase [8].

Fitness funkcie genetických algoritmov môžu mať ľubovoľnú podobu, pričom jedinou podmienkou je, že na vstupe má potenciálne riešenie, ktorému priradí jednu konkrétnu fitness hodnotu. Medzi bežné fitness funkcie patria napríklad funkcie s jedným argumentom a jedným alebo viacerými výrazmi alebo vektorové fitness funkcie, ktoré dokážu súčasne vyhodnotiť viacero vstupov v jednom volaní. V špecifických typoch problémov, kde nie je možné inak vyjadriť účelovú funkciu, sa využívajú napríklad simulácie. Genetické algoritmy, ktoré optimalizujú zlepšovanie kvality obrázkov, môžu ako fitness funkciu využívať priemernú absolútnu chybu (MAE). Pri rekonštrukcii a segmentácii obrázkov je bežné napríklad použitie strednej kvadratickej chyby (MSE) alebo cenovej funkcie ako účelovej funkcie [8].

2.1.3. Genetické operátory

Väčšina evolučných algoritmov využíva kombináciu troch evolučných operátorov a to selekciu, kríženie a mutáciu. V genetickom algoritme existujú rôzne implementácie, pričom každá má inú stratégiu, výhody a nevýhody. Genetické operátory sú aplikované na každú generáciu genetického algoritmu na simulovanie evolúcie s cieľom zlepšovania génov

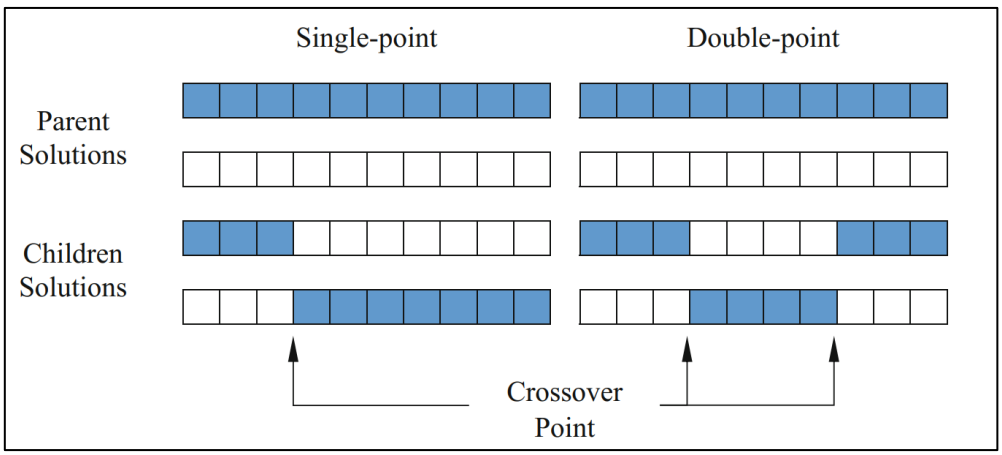
(riešení). V tejto kapitole sú uvedené základné informácie o štandardných genetických operátoroch s príkladmi ich implementácii [6].

Selekcia predstavuje genetický operátor, ktorý sa vykonáva prvý v poradí a simuluje princíp prirodzeného výberu v prírode. Jeho úlohou je vybrať jedincov z aktuálnej populácie do nasledovnej generácie. Na vybratých jedincoch sú následne aplikované ďalšie genetické operátory a na základe tohto procesu sú ich gény prenesené do ďalšej generácie. V štandardných implementáciách selekcie je výber vždy podložený fitness hodnotou jedinca, čo znamená, že čím má jedinec vyššiu fitness hodnotu, tým má vyššiu šancu byť vybratý operátorom selekcie. Ďalšou dôležitou charakteristikou je, že každý jedinec, aj ten s najnižšou fitness hodnotou, má priradenú šancu byť vybratý. V prípade, že by sa napríklad vždy vyberali len jedinci s najvyššou fitness hodnotou, diverzita populácie by rýchlo zanikla a algoritmus by vo väčšine prípadov skonvergoval k lokálnemu minimu resp. maximu. Jedinci s nižšou resp. rozdielnou fitness hodnotou dokážu pomocou ďalších genetických operátorov upraviť najlepších jedincov, čím ich uvoľnia z uviaznutia v lokálnych extrémoch [6].

Ako klasickú implementáciu operátora selekcie sa zvyčajne v literatúre uvádza ruletová selekcia. Táto implementácia je založená na tom, že fitness hodnoty všetkých jedincov aktuálnej generácie sa normalizujú a priradí sa im interval proporčne veľký ich fitness hodnote, vzhľadom na ostatných jedincov. Následne sa vygeneruje n náhodných hodnôt z normalizovaného intervalu (čo predstavuje „točenie rulety“) a ak hodnota patrí do intervalu konkrétneho jedinca, tak daný jedinec je vybraný. Ruletová selekcia dodržiava spomenuté požiadavky pre operátory selekcie a to na základe toho, že každý jedinec má priradenú šancu byť vybraný, ale zároveň jedinec s najvyššou fitness hodnotu má najväčší interval a teda aj najvyššiu šancu, že vygenerované číslo bude patriť práve do jeho intervalu. Jednou z hlavných nevýhod ruletovej selekcie je to, že jeden konkrétny jedinec môže byť vybraný viackrát, čo má negatívny vplyv na diverzitu. Okrem tejto implementácie existuje aj množstvo ďalších, ktoré majú svoje vlastné výhody a nevýhody. Patria sem napríklad turnajová selekcia, ktorá vyberie náhodnú podmnožinu jedincov z populácie a následne určí víťaza (vybraného jedinca) na základe fitness hodnoty alebo selekcia na základe poradia, ktorá zoraďuje jedincov podľa fitness hodnoty a na základe rovnice im priradí šancu byť vybraný [6].

Druhý genetický operátor v poradí sa nazýva kríženie. Cieľom tohto operátora je simulovať prenos (najlepších) génov zo starej do novej generácie na základe princípu evolúcie. Každá dvojica jedincov, vybraná operátorom selekcie, má definovanú šancu na vykonanie operátora kríženia. Táto hodnota sa nazýva šanca kríženia a zvyčajne býva nastavená ako konštantná, pričom existujú aj genetické algoritmy, v ktorých sa šance na vykonanie jednotlivých operátorov dynamicky menia. V bežnej implementácii sa na začiatku operátora vyberie náhodná hodnota z jednotkového intervalu pre každú dvojicu jedincov a ak je vyššia ako šanca kríženia, tak operátor sa následne vykoná. Existuje veľa implementácií, ale princíp tohto operátora zostáva v zásade rovnaký. Na vstupe sú dvaja jedinci, ktorí sa označujú aj ako rodičia a cieľom je vytvoriť dvoch potomkov na základe kombinácie rodičovských génov. V ideálnom prípade, ak sa na účelovej funkcii rodič 1 nachádza v bode X a rodič 2 v bode Y, tak potomkovia sa budú nachádzať v okolí rodičovských bodov X a Y, pričom ich presná poloha závisí od typu operátora. Pomocou tohto procesu je možné efektívne prehľadávať okolie účelovej funkcie a zároveň sa predchádza uviaznutiam v lokálnych extrémoch [6].

Ako klasickú implementáciu operátora kríženia je možné uviesť jednobodové a dvojbodové kríženie. Princíp týchto metód je znázornený na *Obrázku č.4*. V prípade jednobodového kríženia sa vygeneruje jeden index a v prípade dvojbodového dva indexy, ktoré ohraničujú podmnožiny génov, pomocou ktorých budú zostavení potomkovia. Pre rôzne typy problémov existujú rôzne implementácie operátorov kríženia. Napríklad, pre triedu permutačných problémov by jednobodové kríženie nebolo vhodné z dôvodu, že tieto typy problémov vyžadujú, aby každý gén jedinca bol unikátny a jednobodové kríženie nedokáže toto obmedzenie zaručiť. Existujú preto varianty s opravnými mechanizmami alebo metódy, ktoré priamo dokážu zabezpečiť, že potomok nebude obsahovať duplicitné gény [6], [9].



Obrázok 4: Jednobodové a dvojbodové kríženie, zdroj: [6]

Posledným genetickým operátorom je operátor mutácie. Tento operátor simuluje mutácie, ktoré zohrávajú významnú rolu pri evolúcii. Mutácia predstavuje situáciu, kedy sa náhodne zmenia gény jedinca. Táto zmena môže mať pozitívny vplyv na fitness hodnotu daného jedinca, čo zvyšuje jeho šancu na selekciu a prenos tejto mutácie do ďalšej generácie. V opačnom prípade môže mať mutácia výrazne negatívny vplyv, čo by viedlo k významnému zníženiu jeho fitness hodnoty a následnému zániku. Obdobné procesy, na základe ktorých je tento operátor inšpirovaný, sú bežné v prírode. Kým operátor kríženia slúži na skúmanie okolia rodičovských jedincov, operátor mutácie dokáže náhodne preskúmavať rôzne body na fitness funkcii. Každý vybratý jedinec má definovanú šancu na mutáciu, tak ako pri operátore kríženia. Táto šanca sa zvyčajne nastavuje výrazne nižšie ako šanca kríženia z dôvodu, že ak by bola nastavená na 100%, genetický algoritmus by sa stal len algoritmom prehľadávania náhodných bodov (random search) [6].

Implementácie operátora mutácie sú závislé od typu a kódovania problému. Pri binárnych reťazcoch je napríklad možné obrátiť jednu hodnotu (gén) z daného reťazca alebo v permutačných problémoch je možné zameniť poradie dvoch rôznych znakov (génov) vybraného jedinca. Je dôležité zabezpečiť, aby mutácie neporušovali obmedzenia daného problému. Napríklad, ak je jedinec reprezentovaný poľom prirodzených čísel a mutácia náhodne upraví niektoré hodnoty, je potrebné zabezpečiť, aby žiadne číslo nebolo záporné. Vplyv mutácie na jedinca je závislý od typu mutácie. Mutácie, ktoré upravujú len jeden gén jedinca, by ho pri správnom kódovaní nemali výrazne ovplyvniť. Naopak existujú aj mutácie, ktoré upravujú súčasne viacero génov a majú výraznejší vplyv [6].

Kombinácia všetkých troch operátorov umožňuje efektívne prehľadávať priestor riešení daného problému. Selekcia vyberá najlepšie riešenia a zároveň udržiava diverzitu. Kríženie umožňuje prehľadávať okolité body a mutácie náhodne preskúmavajú potenciálne riešenia. V každej novej generácii sú jedinci z predchádzajúcej generácie iteratívne vylepšovaní aplikovaním genetických operátorov, až kým nie je splnená jedna z definovaných ukončovacích podmienok. Okrem týchto troch štandardných operátorov existujú aj iné operátory, ktoré sú využívané v genetických algoritmoch a pomáhajú efektívnejšie nájsť optimálne riešenie [6].

2.1.4. Ďalšie operátory

Okrem troch základných genetických operátov sa v dnešných genetických algoritmoch využívajú aj ďalšie operátory. Cieľom týchto operátorov môže byť napríklad zlepšovanie kvality výsledkov, uchovávanie diverzity alebo adresovanie problémov špecifických pre daný typ úlohy.

Elitný operátor funguje podobne ako operátor selekcie s tým rozdielom, že jedincov nevyberá náhodne. Jeho úlohou je vybrať určitý počet jedincov s najvyššou fitness hodnotou a preniesť ich do ďalšej generácie bez vykonania iných genetických operátorov. Tento proces sa zvykne označovať ako elitizmus a slúži na zachovanie najlepších dosiahnutých riešení. V prípade, že algoritmus nevyužíva elitizmus, sa môže napríklad stať, že najlepší jedinec prejde mutáciou, ktorá mu výrazne zníži fitness hodnotu a zanikne, pričom ostatní jedinci nemali dostatok času preskúmať okolie tohto najlepšieho riešenia. Hlavnou výhodou použitia tohto operátora je lepšia kvalita výsledkov, ale zároveň môže viesť aj k predčasnej konvergencii genetického algoritmu [6].

Ďalší operátor je založený na tom, že pridáva jedincom vek. Na začiatku má každý jedinec vek nula a po každej generácii je zvýšený o jedna. Na základe veku je možné sledovať, ako dlho sa jedinci nachádzajú v populácii a pomocou tejto metriky je možné vykonávať dodatočné operácie na „starých“ jedincoch. Jednou z možností je nastavenie maximálneho veku jedincov, po ktorom budú odstránení a nahradení novými jedincami. Ďalšia možnosť je dynamické upravovanie šance na vykonanie jednotlivých genetických operátorov na základe veku jedinca [10].

V distribuovaných genetických algoritmoch sa využíva operátor migrácie. Distribuované genetické algoritmy predstavujú paralelnú implementáciu genetického algoritmu, v ktorom existuje viacero menších a čiastočne izolovaných populácií. Každá populácia sa vyvíja nezávisle od ostatných súčasným aplikovaním genetických operátorov. Raz za určitý počet generácií sa vykonáva operátor migrácie, ktorý presúva jedincov z jednej populácie do druhej. Na základe tohto procesu je možné zdieľať informácie medzi rôznymi populáciami a predchádzať strate diverzity. Medzi parametre tohto operátora patrí nastavenie topológie, ako napríklad mriežka, kruh alebo náhodné rozdelenie. Definovaná topológia následne určuje „susedné“ populácie, medzi ktorými môže prebiehať výmena jedincov. Ďalšie parametre určujú stratégiu pre výber jedincov a počet vymenených jedincov. Okrem statickej

verzie tohto operátora existujú aj dynamické, ktoré nastavujú šancu migrácie na základe aktuálnej diverzity všetkých populácii [11].

2.1.5. Ukončovacie podmienky

Na konci každej iterácie genetického algoritmu sa kontrolujú ukončovacie podmienky. Vhodné nastavenie tejto časti genetického algoritmu je kľúčové pre efektívny beh a výsledky algoritmu. V prípade, že by genetický algoritmus nebol konečný, tak by ho nebolo možné považovať za algoritmus. Medzi základné ukončovacie podmienky, ktoré zabezpečujú konečnosť genetického algoritmu, je ukončenie na základe dosiahnutia maximálneho počtu generácií (iterácií). Hlavný problém spočíva v tom, že nie je dopredu známe, aký je minimálny počet generácií na dosiahnutie optimálneho riešenia. Táto hodnota sa môže taktiež meniť kvôli stochastickým procesom v genetických algoritmoch. Z tohto dôvodu boli vytvorené ďalšie ukončovacie podmienky, ktoré sa môžu využívať v kombinácii. Patrí sem napríklad ukončenie v prípade, ak algoritmus skonvergoval skôr, než sa dosiahol maximálny počet generácií, čím sa môže čas behu algoritmu výrazne skrátiť. Cieľom je definovať ukončovacie podmienky tak, aby bol algoritmus konečný, skončil čo najrýchlejšie a zároveň aby našiel optimálne riešenie. Medzi ďalšie príklady využívaných ukončovacích podmienok patria [12]:

- Ukončenie na základe stagnácie fitness – genetický algoritmus ukončí, ak sa fitness hodnota najlepšieho jedinca nezlepšila za posledných n generácií.
- Ukončenie na základe času - algoritmus ukončí po uplynutí definovaného času, ako napríklad 5 minút.
- Ukončenie na základe počtu vyhodnotených fitness hodnôt – ukončuje algoritmus po x výpočtoch fitness funkcie.
- Ukončenie na základe fitness hranice – ukončí algoritmus, ak fitness hodnota najlepšieho jedinca dosiahne rovnakú alebo vyššiu hodnotu, než je definovaná hraničná hodnota.
- Ukončenie na základe diverzity – ukončí algoritmus, keď diverzita jedincov v populácii klesne pod definovanú hodnotu.

2.2. Nastavovanie parametrov genetických algoritmov

Genetický algoritmus obsahuje množstvo parametrov a hyperparametrov, ktoré je potrebné vhodne nastaviť pre efektívny beh algoritmu. Patria sem parametre ako veľkosť populácie, šance vykonania jednotlivých genetických operátorov alebo hodnoty ukončovacích podmienok. Problémom efektívneho nastavenia parametrov evolučných algoritmov sa zaoberalo množstvo výskumníkov a cieľom tejto kapitoly je uviesť ich poznatky a použité stratégie [13], [14].

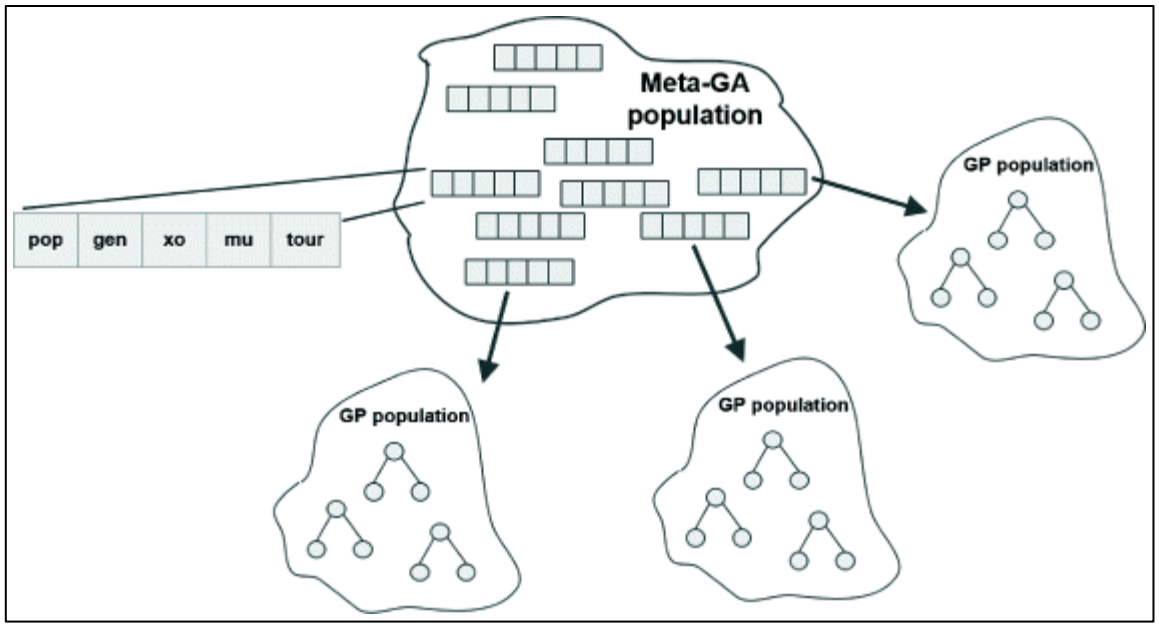
V praxi je bežné, že parametre týchto algoritmov sa nastavujú na základe konvencií, náhodným výberom alebo na základe výsledkov limitovaných experimentov. Problém s týmto prístupom je, že nemusí byť vhodný pre každý typ problému. Kým konvenčné nastavenia parametrov v jednom algoritme môžu viesť k rýchlym a efektívnym riešeniam, v iných algoritmoch môžu výrazne zhoršiť rýchlosť a kvalitu výsledkov daného algoritmu. Zhrnutím poznatkov z vybraných článkov je, že optimálne nastavenie parametrov vedie k robustnejším a kvalitnejším výsledkom týchto algoritmov. Medzi hlavné problémy ale patrí zvolenie vhodnej stratégie, keďže priestor riešení optimálneho nastavenia parametrov je veľmi veľký. Ďalší získaný poznatok je, že neexistuje len jedno optimálne nastavenie (a v prípade, že áno bolo by veľmi zložité ho nájsť), ale množiny dostatočne efektívnych nastavení parametrov. Na dosiahnutie dostatočnej efektívnosti genetického algoritmu by všeobecne malo stačiť sa aspoň priblížiť k jednému efektívnemu nastaveniu parametrov [13].

Hľadanie optimálnych parametrov je možné kategorizovať do dvoch hlavných kategórií na základe prístupu. Prvá kategória sa označuje ako ladenie parametrov a je charakteristická tým, že parametre sa určujú predtým, než je algoritmus zapnutý. Cieľom je nájsť vhodné hodnoty parametrov, ktoré sa nemenia počas behu algoritmu. Tieto hodnoty je možné identifikovať na základe heuristických metód alebo experimentovaním. Tento statický spôsob nastavovania parametrov je vhodný najmä pre problémy, kde sa fitness funkcie a ďalšie procesy algoritmu dynamicky nemenia počas behu programu. Naopak druhá kategória nazývaná riadenie parametrov, dynamicky upravuje parametre počas behu programu. Existuje viacero prístupov k tomu, ako dynamicky riadiť parametre. Patrí sem napríklad deterministický prístup, ktorý parametre upravuje na základe preddefinovaných pravidiel bez ohľadu na aktuálny stav algoritmu. Druhý prístup sa nazýva adaptívny a v tomto prípade sú pravidlá dodatočne aktualizované na základe stavu, v akom sa optimalizačný proces nachádza. Posledný hlavný

prístup sa nazýva seba-adaptívny prístup. Pri tomto prístupe sú parametre nastavené priamo v jedincoch a sú súčasťou evolučného vývoja spolu s jednotlivými riešeniami [13], [14].

2.2.1. Metódy na hľadanie optimálnych parametrov

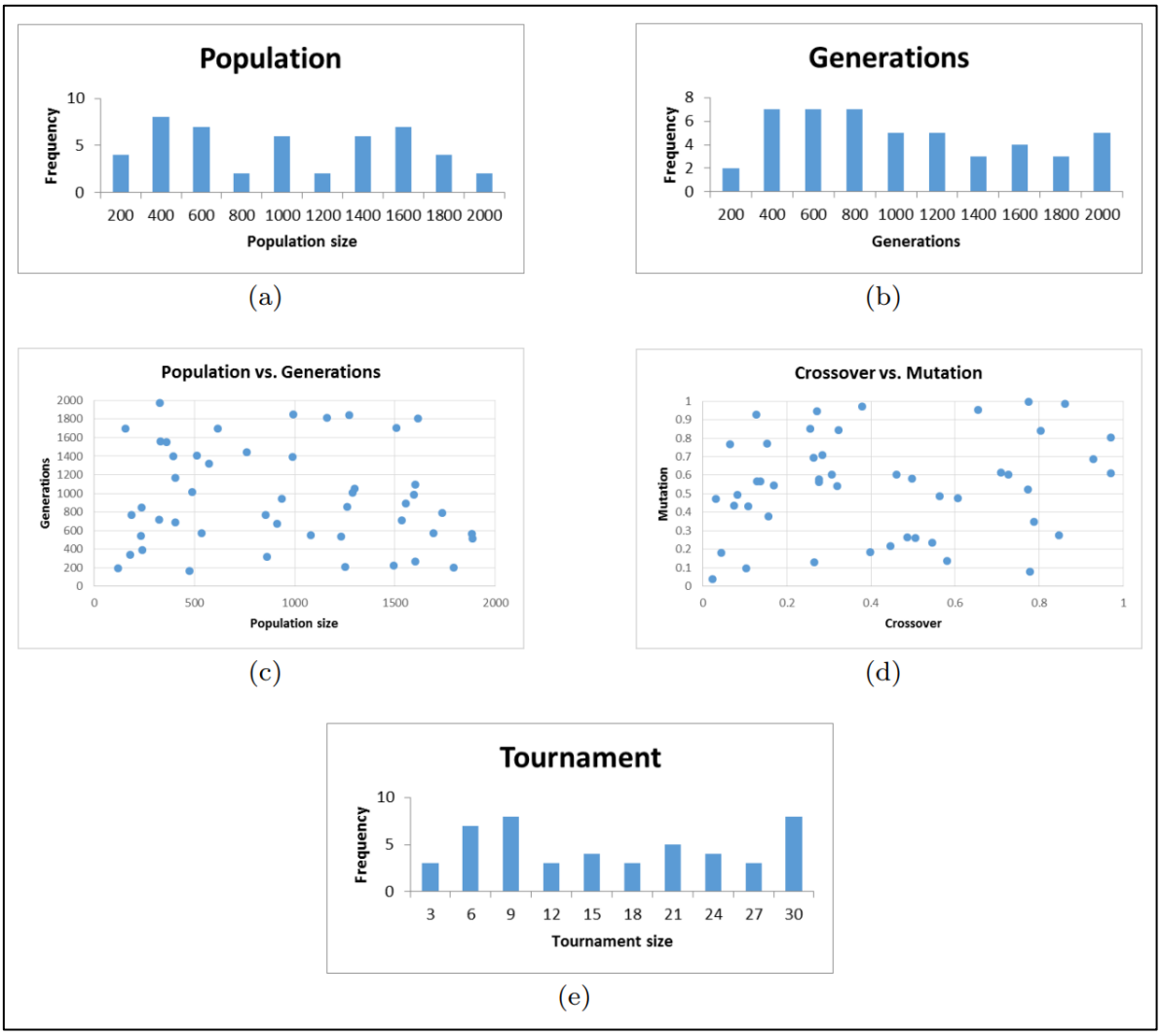
Existuje množstvo metód, pomocou ktorých je možné hľadať optimálne nastavenia evolučných algoritmov. Jednou z metód na optimalizáciu parametrov genetických algoritmov sú tzv. meta-genetické algoritmy. Štruktúra tohto hľadania je definovaná hierarchicky, kde genetický algoritmus vyššej vrstvy optimalizuje parametre genetického algoritmu nižšej vrstvy. Jedinci vyššej vrstvy predstavujú zakódované parametre genetického algoritmu nižšej vrstvy. Týmto jedincom sú následne priraďované fitness hodnoty na základe definovaných výsledkov genetického algoritmu nižšej vrstvy. Konkrétny príklad schémy meta-genetického algoritmu je zobrazený na *Obrázku č.5*. V tomto prípade sa jedinec skladá z reťazca piatich



Obrázok 5: Schéma meta-genetického algoritmu, zdroj: [13]

hodnôt, ktoré predstavujú veľkosť populácie, maximálny počet generácii, pravdepodobnosť vykonania operátora kríženia, pravdepodobnosť vykonania operátora mutácie a veľkosť turnaja pri turnajovej selekcii. Meta-genetický algoritmus funguje na rovnakom princípe ako genetický algoritmus a na začiatku je vygenerovaná náhodná populácia a každému jedincovi je priradená fitness hodnota na základe dosiahnutých výsledkov skúmaného genetického algoritmu. Pomocou vhodného definovania metrík na meranie efektívnosti genetického algoritmu je možné na základe tejto metódy identifikovať vhodné možnosti nastavenia parametrov [13].

Na základe poznatkov, že pre jeden genetický algoritmus existuje množstvo riešení, ktoré sú vzájomne podobné v kvalite výsledkov, je možné využiť metódu náhodného prehľadávania s cieľom identifikovať aspoň niektoré množiny vhodných konfigurácií. Algoritmus je v tomto prípade celkom jednoduchý a pozostáva len z vytvorenia náhodných konfigurácií, vyhodnotenia ich výsledkov a následnej analýzy najlepších riešení. Na *Obrázku č.6* sú zobrazené výsledky testovania náhodného prehľadávania pre 10 rôznych problémov, riešených pomocou genetického algoritmu implementovaného knižnicou M4GP.



Obrázok 6: Výsledky náhodného hľadania parametrov genetického algoritmu M4GP, zdroj: [13]

V grafoch sú zobrazené len konfigurácie s najlepšími dosiahnutými výsledkami. Porovnávajú sa tu frekvencie počtu populácií, generácii a veľkostí turnajov. Ďalej sú vzájomne porovnávané šance operátorov kríženia a mutácie a veľkosť populácie vzhľadom na počet maximálnych generácií. Na základe týchto výsledkov je možné vidieť, že neexistuje žiadny zrejmý trend, ktorý by napríklad vyjadroval, ako vzájomne nastaviť šance operátorov. Zároveň je ale možné

vidieť, že existuje viacero možností, ako vhodne nastaviť parametre genetického algoritmu. Tieto konfigurácie sú skutočne prítomné na celom definičnom obore optimalizovaných parametrov a nie len zhlukované okolo jedného bodu [13].

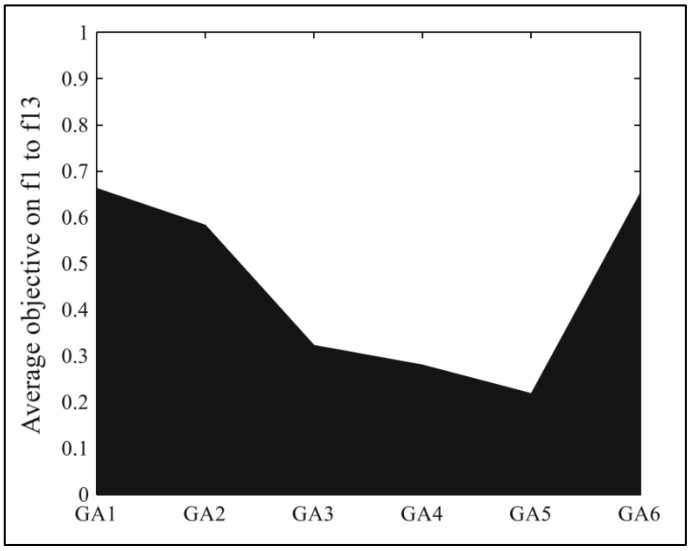
Na základe skúmania 25 rôznych problémov autori došli k záveru, že neexistuje žiadny jasný trend nastavenia parametrov a pre dané skúmané problémy a ich vhodné konfigurácie platia nasledovné výroky [13]:

- Veľkosť populácie nemusí byť maximálna.
- Maximálny počet generácií nemusí byť maximálny.
- Veľkosť populácie a maximálny počet generácií by súčasne nemali byť minimálne.
- Veľkosť turnaja nemusí byť v bežne používanom intervale $[3,\ 7]$.
- Šanca kríženia môže byť nízka aj vysoká.
- Šanca mutácie môže byť nízka aj vysoká.
- Kombinácie šancí kríženia a mutácie neagregujú okolo žiadneho bodu, pričom súčasne by ale mala byť aspoň jedna z nich vyššia.

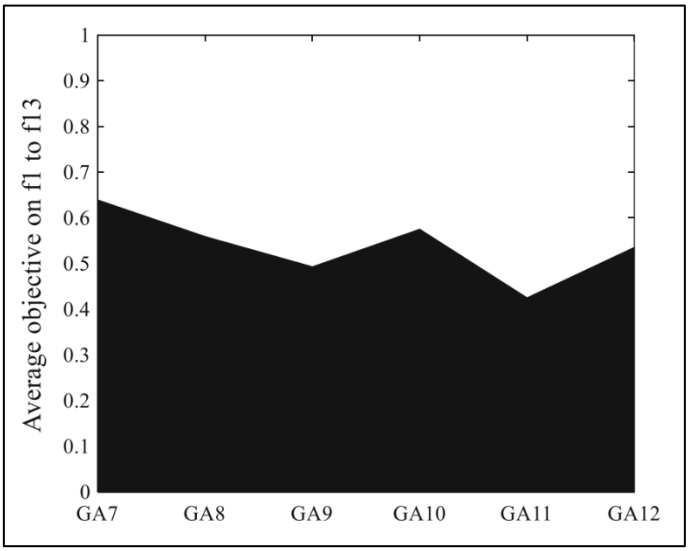
Okrem spomínaných metód existujú aj ďalšie, ako napríklad meta-Lamarckové učenie. V týchto algoritmoch sa využíva lokálna optimalizácia a pomocou meta-Lamarckového učenia sa počas behu algoritmu vyberá vhodná stratégia lokálnej optimalizácie. Ďalší algoritmus sa nazýva REVAC (Relevance Estimation and Value Calibration) a patrí do kategórie meta-evolučných algoritmov. Na rozdiel od meta-genetických algoritmov a evolúcie, aproximuje pravdepodobnostné distribúcie parametrov, ktoré iteratívne vylepšuje. Medzi dynamické metódy patrí napríklad metóda pravidla jednej pätiny, ktorá upravuje parametre algoritmu počas behu, na základe šance zlepšenia fitness v ďalšej generácii. Ak je šanca zlepšenia vyššia ako 1/5, tak sledovaný parameter sa zníži o preddefinovaný faktor a naopak, ak je šanca zlepšenia nižšia ako 1/5, sledovaný parameter sa zvýši o preddefinovaný faktor [13], [14].

Okrem špecializovaných metód je možné využiť aj jednoduché experimenty, kde sa parametre nastavia, napríklad pomocou definovania kroku a porovnajú na základe dosiahnutých výsledkov. V *Grafe č.1* a *Grafe č.2* sú zobrazené výsledky experimentov 30 behov genetického algoritmu s meniacimi sa hodnotami pravdepodobnosti mutácie resp.

kríženia. GA1 (0), GA2 (0.1), GA3 (0.2), GA4 (0.4), GA5 (0.6) a GA6 (0.8) zobrazujú jednotlivé genetické algoritmy s rôzne definovanými šancami mutácie. Z grafu je vidieť, že pre tento konkrétny problém dosahuje najlepšie výsledky GA5 so šancou mutácie 0.6, pričom menšie a vyššie hodnoty majú negatívny vplyv na kvalitu výsledkov. V druhom grafe, kde sa skúma ako zmena pravdepodobnosti kríženia ovplyvňuje výsledné hodnoty, sú definované GA7 (0.5), GA8 (0.6), GA9 (0.7), GA10 (0.8), GA11 (0.9) a GA12 (1). V grafe je možné vidieť, že zmena tejto hodnoty nemá až taký výrazný vplyv, ako zmena pri mutácii, pričom najlepšia hodnota pre tento problém je 0.9 [6].



Graf 1: Porovnanie zmeny šance mutácie a kvality výsledkov, zdroj: [6]



Graf 2: Porovnanie zmeny šance kríženia a kvality výsledkov, zdroj: [6]

3. HĽADANIE GLOBÁLNEHO MINIMA CHEMICKÉHO KLASTRA

Cieľom tejto kapitoly je vysvetliť základné pojmy spojené s chemickými klastrami a uviesť čitateľa do problematiky hľadania globálneho minima chemických klastrov. Jednotlivé príklady a vysvetlenia sú ohraničené pre potreby správneho pochopenia princípu riešeného problému a tvorby programu.

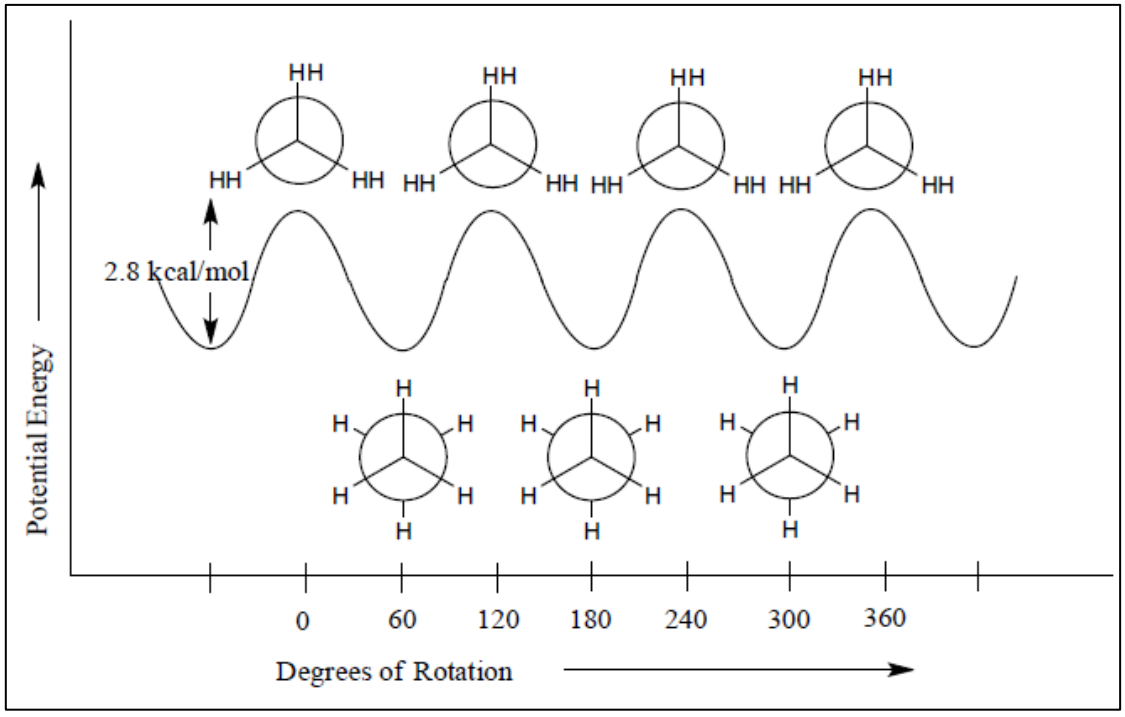
Všeobecne je chemický klaster definovaný ako množina štrukturálnych jednotiek. Každá štrukturálna jednotka sa môže skladať z rôznych atómov alebo molekúl. Klastre sa skladajú z niekoľko štrukturálnych jednotiek, pričom ich počet môže byť rádovo aj v miliónoch. Chemické klastre majú svoju vlastnú unikátnu geometriu a elektrónovú štruktúru, ktorá definuje ich fyzikálne a chemické vlastnosti [15].

Existuje množstvo spôsobov, ako usporiadať jednotlivé štrukturálne jednotky v chemickom klastri a počet možných usporiadaní sa proporčne zvyšuje s počtom štrukturálnych jednotiek. Z tohto dôvodu existuje veľmi veľký počet lokálnych miním chemického klastra, ktoré reprezentujú (semi-)stabilné geometrie, pričom globálne minimum predstavuje najstabilnejšiu štruktúru, ktorá má pri daných externých podmienkach najvyššiu šancu sa vyskytovať v reálnom svete. Identifikácia týchto najstabilnejších štruktúr je pre chemické klastre výrazne náročná a jediný spôsob, ako zistiť alebo odhadnúť tieto štruktúry, je na základe experimentálnych meraní. Z tohto dôvodu vznikli výpočtové metódy na odhadovanie štruktúr bez potreby vykonávania experimentov. Na základe identifikovaných stabilných štruktúr je možné určiť chemické vlastnosti, ktoré sú ďalej aplikovateľné na výskum v chémii. Patria sem napríklad oblasti, ako výskum nových materiálov a liečiv, priebeh reakcií, katalýza alebo atmosférická chémia [15], [16].

3.1. Povrch potenciálnej energie (PES)

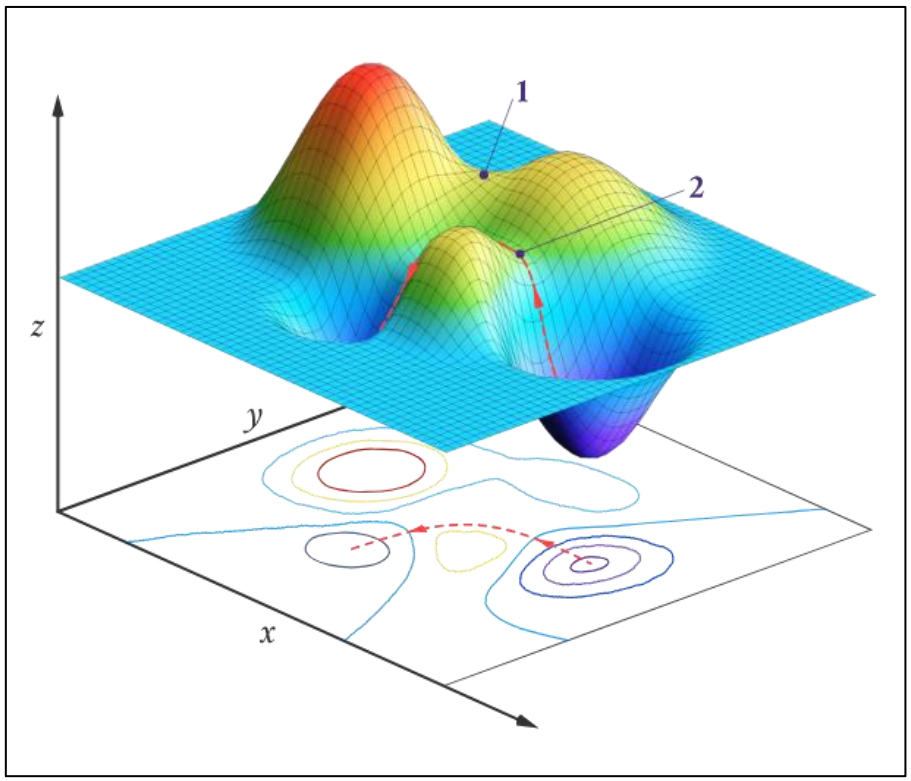
Stabilita geometrie chemického klastra je odvodená od jeho energie. Čím nižšiu energiu klaster má, tým viacej energie by bolo potrebné vynaložiť na zmenu pozícií jednotlivých štrukturálnych jednotiek, z ktorých sa skladá. Lokálne minimá na PES sú oddelené tzv. energetickými bariérami, ktoré reprezentujú prechodové stavy. Prechodový stav, zvyčajne charakterizovaný lokálnym maximom na PES, predstavuje energeticky nestabilnú štruktúru, ktorá vo väčšine prípadov konverguje k najbližšiemu (stabilnému) lokálnemu minimu. Príklad PES pre jednu molekulu etánu (C_2H_6), vzhľadom na rotáciu okolo uhlíkovej väzby ($C-C$), je

zobrazený na *Obrázku č.7*. Je možné si všimnúť, že daná funkcia je periodická a otočením uhlíkovej väzby o 120° sa vždy dosiahne energeticky ekvivalentná štruktúra [16].



Obrázok 7: Povrch potenciálnej energie etánu vzhľadom na torziu väzby uhlík-uhlík, zdroj: [39]

Povrch potenciálnej energie predstavuje účelovú funkciu pre algoritmy na hľadanie globálneho minima. Predošlý príklad PES etánu predstavuje zjednodušený príklad, pričom v reálnych skúmaných chemických klastroch môže existovať množstvo štrukturálnych jednotiek a väzieb, z ktorých sa všetky môžu voľne upravovať. Dimenzionalita funkcie povrchu potenciálnej energie rastie lineárne so zvyšujúcim sa počtom atómov. V prípade, že by sa v *Obrázku č.7* nebral do úvahy iba jeden parameter (uhol rotácie), ale všetky možné parametre, PES etánu by mala 18 dimenzií, ktoré by nebolo možné (rozumne) vizualizovať. Na *Obrázku č.8* je zobrazený príklad 3D PES, ktorý zobrazuje hypotetickú (endotermickú) reakciu, kde energia z je závislá od dvoch parametrov x a y. Priebeh reakcie je vyznačený červenou líniou, pričom body 1 a 2 predstavujú tranzitné stavy na PES [17], [18].



Obrázok 8: Príklad 3D PES, zdroj: [18]

3.2. Chemické výpočtové modely

V teoretickej chémii existuje množstvo výpočtových modelov, pomocou ktorých je možné vypočítať energiu. Jeden z dôležitých faktorov pri hľadaní globálneho minima je výber vhodného modelu pre daný skúmaný systém. Zvolený model má výrazný vplyv na dĺžku výpočtov a kvalitu dosiahnutých výsledkov. Motiváciou tejto kapitoly je objasnenie dôvodu existencie veľkého počtu modelov a ich primárnych charakteristík. Na výpočet presnej energie atómu, molekuly alebo klastra je potrebné vyriešiť Schrödingerovu rovnicu. Jej všeobecný tvar je [16]:

$$\hat{H} \cdot \Psi = E \cdot \Psi$$

Ψ predstavuje vlnovú funkciu viacerých elektrónov, ktorá opisuje ich pohyby, $\hat{H}$ predstavuje Hamiltonov operátor, ktorý vyjadruje celkovú kinetickú a potenciálnu energiu systému a E predstavuje energiu stavu opísaného vlnovou funkciou. Jediný systém, pre ktorý sa podarilo (exaktne) vyriešiť túto rovnicu, je jeden atóm vodíka, ktorý sa skladá z jedného atómového jadra a jedného elektrónu (2 častíc). Pokusy o vyriešenie rovnice pre jeden atóm hélia (3 častice) neboli úspešné a okrem atómu vodíka nie je možné túto rovnicu vyriešiť exaktne pre

žiadne iné systémy. Formulácia tejto rovnice je podobná známemu neriešiteľnému problému troch telies[1]. So zvyšujúcim sa počtom častíc sa komplexnosť tohto problému len ďalej zvyšuje. Z tohto dôvodu vznikli rôzne aproximácie, ktoré problém zjednodušujú na základe rôznych predpokladov. Je ale dôležité poznamenať, že energia vypočítaná na základe použitých aproximácií nezodpovedá exaktnému riešeniu Schrödingerovej rovnice [16].

Ako základná aproximácia sa v praxi využíva Bornová-Oppenheimerova aproximácia, ktorá uvažuje o atómových jadrách ako o statických nehybných časticiach. V skutočnosti sa jadra pohybujú, ale ich pohyb je minimálny vzhľadom na elektróny, ktoré sú omnoho ľahšie a pohybujú sa teda rádovo rýchlejšie. Táto aproximácia upravuje tvar Schrödingerovej rovnice na tzv. elektronický tvar. Hartreeho-Fockova aproximácia ďalej zjednodušuje túto rovnicu na základe predpokladu, že elektróny sa pohybujú nezávisle od seba, pričom v skutočnosti sa elektróny pohybujú v priestore molekulárnych orbitálov, ktoré sa navzájom ovplyvňujú. Výsledkom tejto aproximácie je množina diferenciálnych rovníc, z ktorých každá reprezentuje jeden konkrétny elektrón. Ďalšia aproximácia sa nazýva LCAO (Linear Combination of Atomic Orbitals) a zavádza molekulárne orbitály ako lineárne kombinácie funkcií bázovej množiny. Túto bázu tvoria funkcie vypočítané pre atóm vodíka. Metódy postavené na základe LCAO aproximácie teda požadujú na výpočet dostatočnú množinu bázových funkcií. Kombinácia Hartreeho-Fockovej aproximácie a LCAO aproximácie vedie k množine algebrických Roothaanovských-Hallovských rovníc, ktorých riešenie je výrazne ľahšie ako riešenie diferenciálnych rovníc pomocou numerických metód. Predstavenie týchto základných aproximácií je dôležité pre pochopenie nasledujúcich výpočtových metód, pomocou ktorých je možné počítať energiu v programoch hľadajúcich globálne minimum. Využívanie týchto aproximácií taktiež vysvetľuje dôvod existencie veľkého množstva modelov. Každá aproximácia zjednodušuje výpočty a teda vo všeobecnosti vedie k rýchlejšiemu získaniu výsledkov za cenu presnosti vypočítanej energie [16].

[1] Úloha v oblasti mechaniky, kde sa určujú pohyby troch telies, ktoré sa navzájom ovplyvňujú

Chemické výpočtové modely je možné zaradiť do jednej z troch hlavných tried na základe použitých aproximácií. Tieto triedy sú následne delené na základe ďalších spoločných postupov a dodatočných aproximácií. Patria sem napríklad [16], [19]:

- Metódy molekulárnej mechaniky (MM)
    - Metódy silového poľa
    - Neurónové siete predikujúce energiu
- Kvantovo-mechanické metódy (QM)
    - Semiempirické metódy
    - Metódy funkcionálov elektrónovej hustoty (DFT)
    - Ab initio[2] metódy
        - Hartreeho-Fockova metóda
        - Metódy zahŕňajúce korelačnú energiu
- Hybridné metódy
    - QM/MM[3] metódy

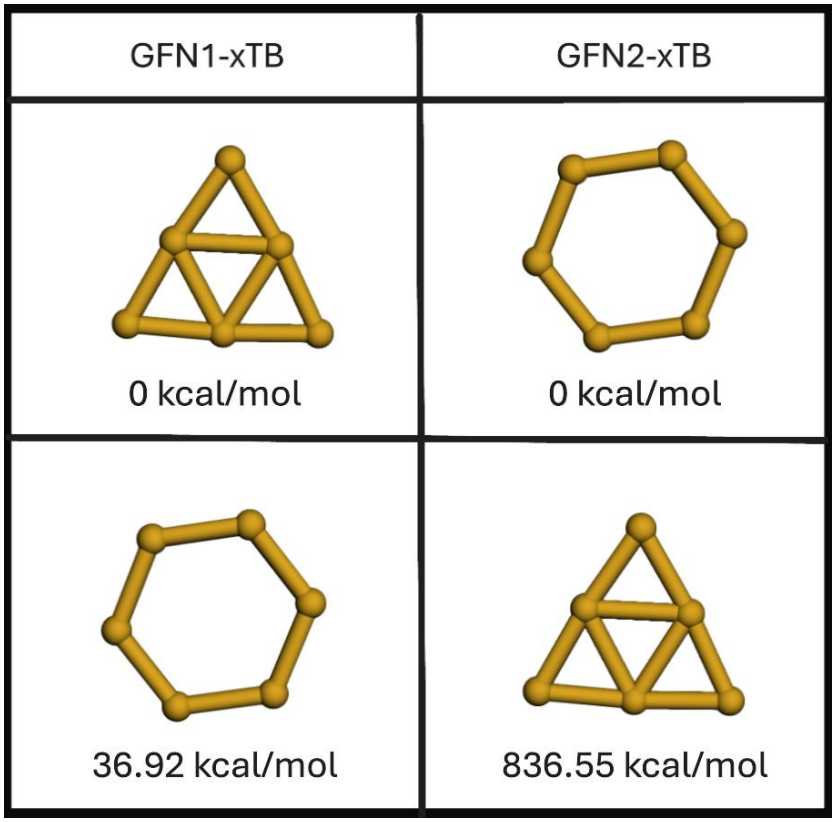
Metódy molekulárnej mechaniky a silových polí využívajú princípy a postupy klasickej fyziky na výpočet molekulárnej energie. Tieto metódy sú jedny z najrýchlejších, ale zvyčajne sú menej presné než kvantovo-mechanické metódy, pretože nepracujú priamo so Schrödingerovou rovnicou a explicitne nezahŕňajú kvantové efekty vo výpočtoch. Energia sa počíta na základe silového poľa, ktoré definuje interakcie medzi atómami. Silové polia využívajú parametre, ktoré je potrebné získať buď pomocou experimentálnych dát alebo prostredníctvom kvantovo–mechanických výpočtov na referenčných molekulách. Tieto metódy sú najlepšie využiteľné pre systémy, kde sú dané parametre silového poľa vhodne definované [16], [19].

Kvantovo-mechanické metódy sa snažia aproximovať riešenie Schrödingerovej rovnice. Vo všeobecnosti dosahujú presnejšie výsledky, ale ich výpočet je značne časovo aj výpočtovo náročnejší než pri metódach molekulárnej mechaniky. Semiempirické metódy sú najrýchlejšie kvantovo-mechanické metódy a na dosiahnutie daného zrýchlenia využívajú aproximáciu Hamiltonovského operátora. Výpočtovo náročné prvky Hamiltoniánu sú

[2] Z lat. „Od základov"
[3] Z angl. „Quantum Mechanics/Molecular Mechanics" – kombinácia metód kvantovej a molekulárnej mechaniky

nahradené parametrami, ktoré sú odvodené na základe experimentálnych údajov. DFT metódy riešia Schrödingerovu rovnicu aproximáciou vlnovej funkcie elektrónovou hustotou. Tieto metódy poskytujú rýchlejšie výsledky v porovnaní s náročnejšími ab initio metódami, pričom ich presnosť je zvyčajne vyššia než pri semiempirických metódach. Ab initio metódy sú charakteristické tým, že vo svojich výpočtoch nepoužívajú žiadne empirické (experimentálne) údaje. Patria sem metódy založené na Hartreeho-Fockovej aproximácii a jej ďalších rozšíreniach, ktoré napríklad spätne zahŕňajú koreláciu medzi elektrónmi. Tieto metódy sa zvyčajne radia medzi výpočtovo najnáročnejšie, ale zároveň najpresnejšie kvantovo-mechanické výpočtové metódy [16], [19].

Výber vhodnej výpočtovej metódy je pre algoritmy hľadania globálneho minima chemického klastra veľmi dôležitý. V procese hľadania sa často počíta energia rôznych štruktúr a je potrebné zvoliť dostatočne rýchlu metódu, aby sa priestor riešení mohol efektívne prehľadávať v rámci dostupných výpočtových zdrojov. Zároveň je však dôležité, aby zvolená metóda dokázala vhodne aproximovať energiu, inak získané výsledné najstabilnejšie štruktúry nemusia zodpovedať experimentálnym údajom. Na *Obrázku č.9* je zobrazený príklad dvoch semiempirických metód GFN1-xTB a GFN2-xTB [20] pri globálnej optimalizácii klastra zloženého zo šiestich atómov zlata - Au_6. Pre ľahšie porovnanie jednotlivých energií sa v chémii energia nájdeného globálneho minima určí ako $0\ kcal/mol$ a energia ostatných štruktúr ako odchýlka od globálneho minima. Skutočné globálne minimum tohto systému predstavuje trojuholníková štruktúra, ktorú správne identifikovala metóda GFN1-xTB. V prípade použitia GFN2-xTB by nájdené globálne minimum predstavovalo nesprávnu štruktúru a skutočné globálne minimum by bolo vyjadrené len ako jedno z lokálnych miním. Vzájomným porovnávaním výsledkov semiempirických metód sa nedá určiť, ktoré minimum je skutočne globálne. Hierarchia metód však hovorí, že ak je trojuholníková štruktúra systematicky identifikovaná ako globálne minimum náročnejšími metódami (Hartreeho-Focková, DFT alebo vyššie ab initio), bude ním zrejme aj vo fyzikálnej realite.



Obrázok 9: Porovnanie výsledkov metód GFNn-xTB pre Au_6, zdroj: Autor

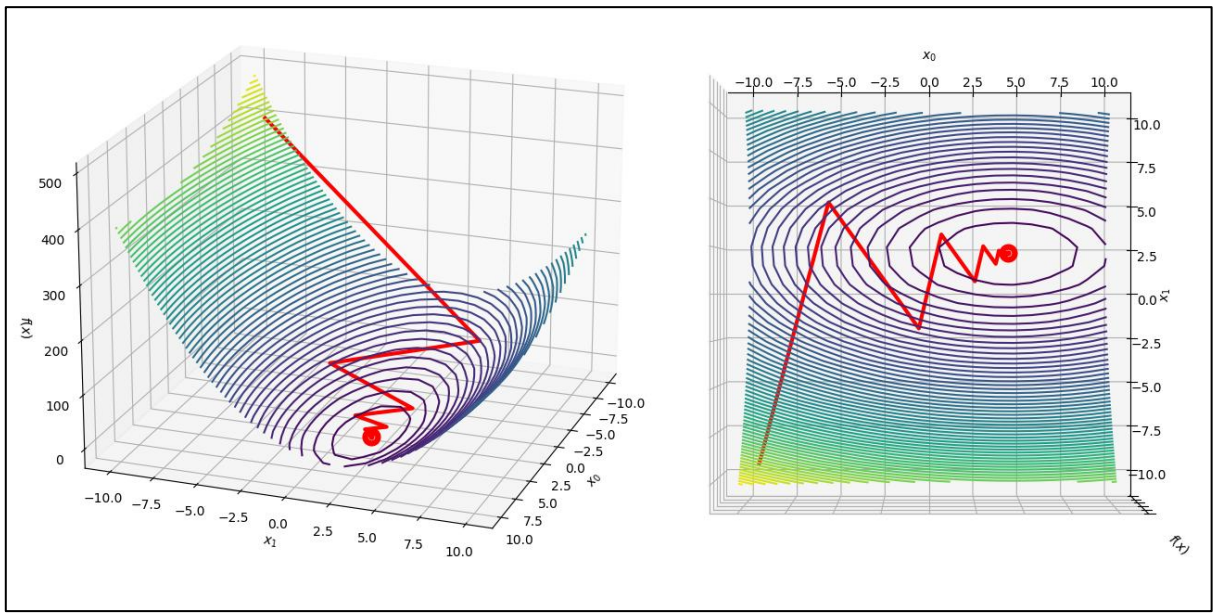
3.3. Hľadanie lokálneho minima

Hľadanie lokálneho minima, v kontexte chemických klastrov, predstavuje nájdenie najbližšej stabilnej štruktúry na danom povrchu potenciálnej energie (PES). Lokálna optimalizácia (minimalizácia) je dôležitým komponentom vybraných algoritmov, ktoré hľadajú globálne energetické minimum chemických klastrov. Evolučné algoritmy, ktoré využívajú lokálnu optimalizáciu, sú tiež známe aj ako Lamarckové varianty daných algoritmov. Vzhľadom na charakter riešeného problému, kde energeticky najnižšie štruktúry predstavujú výstupy týchto algoritmov, je lokálna optimalizácia zakomponovaná vo väčšine globálne zameraných algoritmov pre efektívnejšie preskúmavanie danej PES. Cieľom tejto kapitoly je uviesť základné stratégie lokálnej optimalizácie chemických klastrov a príklady niektorých konkrétnych algoritmov, s ktorými je možné sa stretnúť v praxi. Je dôležité uviesť, že tieto algoritmy sa využívajú aj na iné problémy než je minimalizácia chemických klastrov, ako napríklad pri strojovom učení. Vo všeobecnosti je možné tieto algoritmy použiť na ľubovoľnú funkciu, pričom niektoré vyžadujú, aby funkcia mala špecifické vlastnosti, ako napríklad diferencovateľnosť alebo konvexnosť [15].

Bežne využívané optimalizačné metódy vo výpočtovej chémii je možné zaradiť do jednej z dvoch kategórií. Prvá kategória predstavuje metódy prvého rádu, ktoré sú založené na počítaní gradientu funkcie pomocou prvej derivácie. Druhá kategória metód sa označuje ako metódy druhého rádu, ktoré využívajú druhé derivácie na zostavenie Hessovej matice [19].

3.3.1. Metódy najstrmšieho zostupu a konjugovaného gradientu

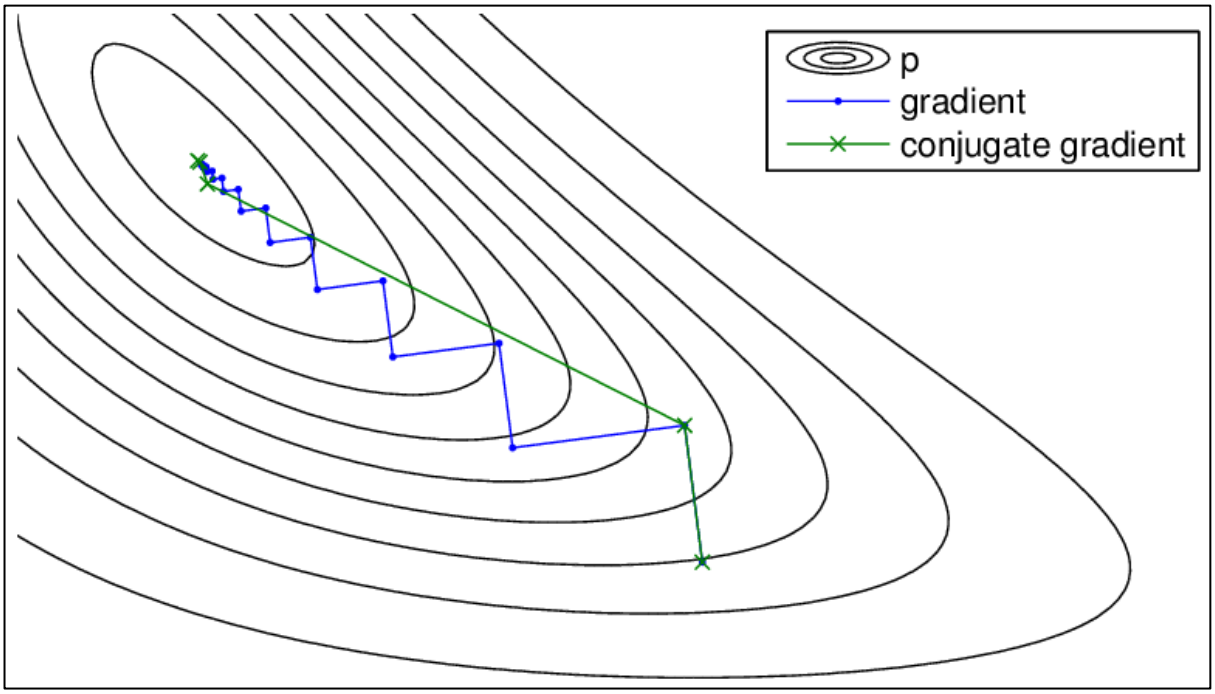
Medzi najznámejšie príklady metód prvého rádu patrí metóda najstrmšieho zostupu (SD[4]). V tejto metóde sa na začiatku (náhodne) vyberie počiatočný bod a pomocou prvej derivácie sa určí vektor gradientu funkcie v tomto bode. Na základe získaného vektora sa určí smer, v ktorom sa potenciálna energia systému najviac znižuje. Požadovaný smer zodpovedá opačnému smeru vektora gradientu. V prípade minimalizácie chemického klastra sa tento smer určuje numericky individuálnym pohybom každého atómu okolo troch súradníc a prepočítaním zmeny výslednej energie na základe vykonaného pohybu. Po identifikovaní najlepšieho smeru sa navrhne nová geometria súčasným pohybom všetkých atómov, pričom čím je príspevok ku stabilite (gradient) väčší, tým je vzdialenosť vykonaného posunu väčšia. Tento postup sa iteratívne opakuje, až kým nie je dosiahnuté minimum, t. j. napríklad kým nie je dosiahnutá konštantná energia. Táto metóda je najviac vhodná pre geometrie, ktoré sú vzdialenejšie od hľadaného minima z dôvodu, že rýchlosť konvergencie tejto metódy sa výrazne znižuje v blízkosti minima. Príklad tejto metódy je zobrazený na *Obrázku č.10* [19].



Obrázok 10: Príklad metódy najstrmšieho zostupu, zdroj: [40]

[4] z angl. Steepest Descent

Medzi ďalšie metódy prvého rádu patrí napríklad metóda konjugovaného gradientu (CG[5]) a jej dva varianty – Fletcherova-Reevesova (FR) resp. Polakova-Ribiereho (PR) metóda. Táto metóda si v porovnaní s metódou najstrmšieho zostupu uchováva informácie o funkcii z predošlej iterácie a využíva ich na prevenciu zhoršovania konvergencie v ďalších krokoch. Nevýhodou tejto metódy je vyššia výpočtová náročnosť oproti metóde najstrmšieho zostupu, pričom ale dosahuje rýchlejšiu konvergenciu a môže byť výhodné ju použiť pre veľké systémy. Na *Obrázku č.11* je zobrazené porovnanie metód najstrmšieho zostupu a konjugovaného gradientu pre znázornenú funkciu p [19].



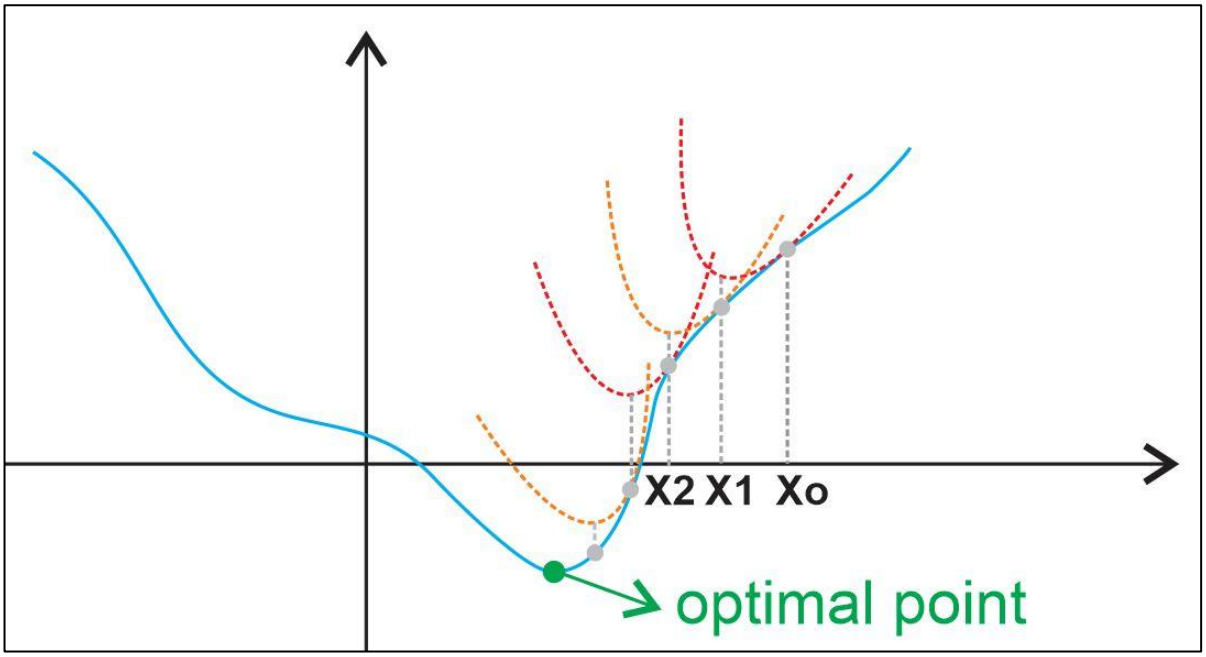
Obrázok 11: Porovnanie metód najstrmšieho zostupu a konjugovaného gradientu, zdroj: [42]

3.3.2. Newtonova-Raphsonova metóda

Metódy druhého rádu využívajú Hessovu maticu (maticu druhých derivácií) na zrýchlenie konvergencie za cenu vyššej výpočtovej náročnosti počítania druhých derivácií. Medzi tieto metódy patrí Newtonova-Raphsonova metóda, kde sa pomocou prvej derivácie (gradientu) určuje sklon a pomocou druhých derivácií rýchlosť zmeny sklonu. Táto metóda dokáže priamo určiť minimum kvadratickej funkcie a v prípade komplexnejších funkcií dokáže minimum nájsť v menšom počte iterácií (v ideálnych prípadoch až tri krát rýchlejšie) než metódy prvého rádu. Podmienkou pre túto metódu je, že Hessova matica musí byť kladne

[5] z angl. Conjugate Gradient

definitná[6], inak sa minimalizácia stáva nestabilnou a minimum nemusí byť nájdené. Hlavnou nevýhodou a dôvodom, prečo sa táto metóda nepoužíva pri každom systéme, je jej vysoká výpočtová náročnosť[7] (hlavne pre veľké systémy). Na *Obrázku č.12* je znázornený priebeh tejto metódy. Minimalizácia začína v počiatočnom bode x_o aproximáciou pomocou kvadratickej funkcie vytvorenej na základe gradientu a Hessovej matice. Následne sa vyberie minimum tejto vytvorenej kvadratickej funkcie ako ďalší bod x_1, od ktorého sa ďalej hľadá minimum skúmanej funkcie. Tento postup sa iteratívne opakuje, až kým nie je nájdené minimum (nebola dosiahnutá ukončovacia podmienka) [19].



Obrázok 12: Príklad Newtonovej-Rhapsonovej metódy, zdroj: [43]

3.3.3. Metódy BFGS a L-BFGS

Broydenova-Fletcherova-Goldfrabova-Shannova (BFGS) metóda patrí medzi jedny z najviac využívaných optimalizačných metód v oblasti výpočtovej chémie. BFGS patrí do skupiny kvázi-Newtonovských metód. Tieto metódy počítajú len prvú deriváciu a na základe vypočítaných gradientov aproximujú (inverznú) Hessovu maticu. Pomocou tohto prístupu je možné zrýchliť konvergenciu, podobne ako pri metódach druhého rádu, pričom nie je potrebné počítať časovo náročné druhé derivácie na zostavenie (skutočnej) Hessovej matice. Špecifickým problémom BFGS je, že na základe aproximovanej inverznej Hessovej matice je

[6] Matica je kladne definitná, ak funkcia v jej okolí rastie vo všetkých smeroch (má konvexný tvar)
[7] Jedným z krokov metódy je vytvorenie inverznej Hessovej matice, ktorý má časovú zložitosť $O(n^3)$

možné určiť iba smer pohybu, ale už nie je možné určiť veľkosť kroku (z minima kvadratickej funkcie). Na riešenie tohto problému sa využívajú tzv. line search algoritmy, ktoré vypočítajú optimálnu dĺžku kroku na základe rôznych kritérií špecifických pre daný algoritmus [21], [22].

Jedným z problémov BFGS metódy pri veľkých systémoch je vysoká pamäťová zložitosť spojená s uchovávaním Hessovej matice. Táto matica má tvar $n \times n$ a priestorová zložitosť na uchovanie tejto matice je $O(n^2)$. V prípade veľmi veľkých systémov toto predstavuje obmedzenie, ktoré vyžaduje buď dostatočné výpočtové zdroje alebo použitie inej pamäťovo efektívnejšej metódy. Na riešenie tohto problému vznikla modifikácia pôvodnej metódy, ktorá sa nazýva L-BFGS. Táto metóda je priestorovo ohraničená pomocou parametra m, ktorý určuje maximálny počet uložených vektorov z posledných iterácií[8]. Hlavnou zmenou v tejto metóde je, že Hessova matica sa neukladá do počítačovej pamäte. Pri metóde BFGS je v každej iterácii Hessova matica aktualizovaná pomocou posledného získaného gradientu a veľkosti vykonaného kroku. Týmto iteratívnym prepočítavaním je aproximácia Hessovej matice implicitne zostavená zo všetkých predošlých gradientových vektorov. Pri metóde L-BFGS sa ukladá len m posledných dvojíc krokových vektorov a rozdielov gradientov. Na základe týchto vektorov sa v každej iterácii získa aproximácia Hessovej matice, čo vedie k celkovej priestorovej zložitosti $O(m \cdot n)$. Táto metóda vychádza z predpokladu, že vektory z posledných m iterácií obsahujú dostatok informácií pre vhodnú aproximáciu Hessovej matice. V praxi sa pri veľkých systémoch často používa L-BFGS namiesto BFGS, pričom dosahuje porovnateľnú kvalitu výsledkov s výrazne nižšími nárokmi na pamäť [21], [22].

3.3.4. Metóda FIRE

Jedna z novších metód je FIRE (Fast Inertial Relaxation Engine), ktorá bola vytvorená v roku 2006. Táto metóda patrí medzi metódy prvého rádu a využíva prvky molekulárnej dynamiky. Metódy molekulárnej dynamiky špecializované na lokálnu optimalizáciu boli pred navrhnutím tejto metódy zvyčajne uvádzané len ako príklady vďaka ich ľahkej implementácii, pričom ich výsledky nedosahovali rovnakú kvalitu iných metód. Metóda FIRE rozšírila existujúce metódy molekulárnej dynamiky a dosahuje kvalitnejšie výsledky, než metódy konjugovaných gradientov a porovnateľné výsledky ako kvázi-Newtonovské metódy bez potreby počítania (aproximovania) Hessovej matice [23].

[8] Odporúčajú sa malé hodnoty, napríklad $m \in \langle 3,\ 20 \rangle$

Na rozdiel od metódy gradientového zostupu, kde sa pohyb a jeho veľkosť vždy vykonáva len na základe aktuálneho gradientu, sa pri metóde FIRE využívajú princípy dynamiky, ako sú rýchlosť a sila. Tieto parametre sú dynamicky upravované v každej iterácii algoritmu na základe aktuálnej situácie a pomáhajú viesť k rýchlejšej konvergencii. Vektory rýchlosti a sily sú počítané pomocou jedného z integrátorov (napríklad Velocity Verlet), ktoré sa využívajú v molekulárnej dynamike. Optimalizácia je inicializovaná v počiatočnom bode s určitou rýchlosťou a v každej iterácii sa kontroluje, či tieto dva vektory rýchlosti a sily majú rovnaký smer. V prípade, že ich skalárny súčin nie je záporný, veľkosť vykonaných krokov je zvýšená a vektor rýchlosti je kombinovaný s normalizovanou silou pre rýchlejšiu konvergenciu. Ak tieto dva vektory nemajú rovnaký smer, metóda vykoná korekčnú akciu. V prípade, že ich skalárny súčin je záporný (čo predstavuje pohyb zlým smerom), je rýchlosť nastavená na nulu, čím sa optimalizačný proces zresetuje. Veľkosť vykonaných krokov sa zníži a smer sa upraví tak, aby znova smeroval k minimu. Na základe tejto korekcie sa predchádza situáciám, ako napríklad preskočenie minima alebo divergencii. Pomocou tohto dynamického upravovania jednotlivých zložiek molekulárnej dynamiky dosahuje metóda FIRE porovnateľnú kvalitu ako kvázi-Newtonovské metódy [23].

V roku 2020 bola navrhnutá upravená verzia s názvom FIRE v2.0 (skrátene FIRE2). V tejto metóde sa autori zamerali na zlepšenie efektívnosti a robustnosti pôvodnej metódy. Jedným zo záverov bolo, že výber vhodného integrátora má výrazný vplyv na efektívnosť optimalizácie, pričom v pôvodnej metóde nie je výber integrátora molekulárnej mechaniky nejako obmedzený. Ďalšou navrhnutou zmenou je dodatočné vykonanie spätného kroku v prípade, keď skalárny súčin vektorov je záporný. Pomocou týchto zmien a ďalších úprav, ako napríklad pridanie dodatočných ukončovacích podmienok, sa im podarilo zrýchliť pôvodnú metódu. Na základe výsledkov ich testovania odporúčajú využívať ich nový rýchlejší variant FIRE2 namiesto pôvodnej metódy FIRE [24].

3.4. Súčasný stav riešenia problému globálnej optimalizácie

Ako bolo už spomenuté v predošlých kapitolách, hľadanie globálneho (energetického) minima chemického klastra predstavuje komplexný problém. Povrch potenciálnej energie (PES), ktorý definuje účelovú funkciu globálnych prehľadávacích algoritmov, môže obsahovať veľký počet lokálnych miním. Vo všeobecnosti platí, že čím chemický klaster obsahuje viacej štrukturálnych jednotiek, tým sa počet lokálnych miním PES zvyšuje. S rastúcou veľkosťou chemického klastra sa taktiež zvyšuje aj dimenzionalita skúmanej PES

a výpočtová náročnosť na výpočet energie chemického klastra a jeho lokálnej optimalizácie taktiež úmerne vzrastá. Z dôvodu, že experimentálne identifikovanie najstabilnejších štruktúr je veľmi náročné a často aj fyzicky nerealizovateľné, vznikli algoritmy na hľadanie globálneho minima a energeticky najnižších (najbližších) lokálnych miním [15].

Vývoj algoritmov na globálne prehľadávanie PES je prepojený s aktuálnym vývojom výpočtových zdrojov. Pred rokom 2005 boli bežné len algoritmy, ktoré dokázali hľadať globálne minimum malých klastrov a väčšinou iba s použitím rýchlych empirických potenciálov. Neskôr, s vyššou dostupnosťou rýchlejších výpočtových prostriedkov, sa začala realizovať globálna optimalizácia na komplexnejších klastroch a aj za pomoci využitia ab inito metód. Od roku 2016 existuje aspoň 50 000 internetových výsledkov pre články, ktoré sa zaoberajú globálnou optimalizáciou chemických klastrov. Dnešná globálna optimalizácia je realizovateľná aj na komplexných chemických klastroch, ktoré sa skladajú z veľkého množstva rozličných štrukturálnych jednotiek. Vďaka tomuto vývoju je možné aplikovať výsledky týchto algoritmov na ešte väčšie množstvo reálnych problémov v chémii [15].

Medzi populárne využívané algoritmy patria napríklad metaheuristické algoritmy alebo basin hopping algoritmus [15]. Tieto algoritmy dokážu efektívne preskúmavať aj viacdimenzionálne účelové funkcie, vďaka čomu dosahujú kvalitné výsledky. Okrem týchto algoritmov neustále vzniká množstvo ďalších. V roku 2023 vznikol algoritmus, ktorý pomocou hlbokých neurónových sietí dokáže hľadať globálne minimum platinového klastra [25]. Ďalší program s názvom CSearch, vytvorený v roku 2024, slúži na hľadanie globálneho minima organických klastrov pomocou grafových neurónových sietí (GNN) [26]. Existujúce algoritmy, ako napríklad metaheuristické metódy, sú taktiež periodicky inovované a vznikajú nové a efektívnejšie implementácie alebo špecializované implementácie pre konkrétne skupiny klastrov [27], [28]. Vývoj nových optimalizačných algoritmov je pre globálnu optimalizáciu chemických klastrov veľmi aktívny a požadovaný vzhľadom na potenciálne aplikácie pri chemickom výskume.

3.4.1. Algoritmus ORCA GOAT

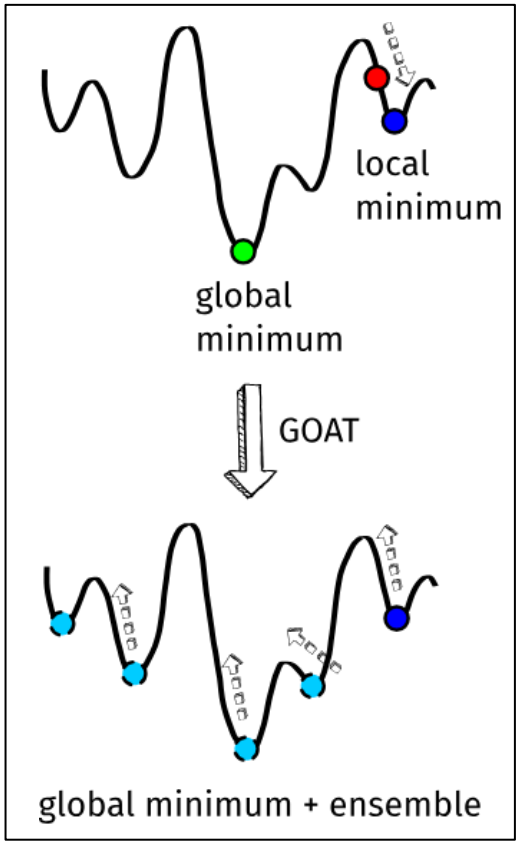
V populárnom kvantovo-chemickom programe s názvom ORCA [29] je implementovaný podprogram na globálnu optimalizáciu chemických klastrov. Tento algoritmus sa nazýva GOAT (Global Optimizer Algorithm) a bol navrhnutý na základe inšpirácie z algoritmov basin hopping, minima hopping, simulovaného žíhania a tabu prehľadávania [30].

Základnou stratégiou prehľadávania PES pomocou GOAT je kombinácia lokálnej minimalizácie na získavanie miním a pohybu smerom hore[9] na prekonávanie energetických bariér medzi jednotlivými minimami. Na začiatku je definovaná počiatočná konfigurácia (bod na PES), ktorá je minimalizovaná do najbližšieho lokálneho minima. Následne sa začne vykonávať pohyb smerom hore po susednej energetickej bariére. Po dosiahnutí maxima sa môže znova vykonať lokálna minimalizácia na danej konfigurácií, čím sa získa ďalšie susedné lokálne minimum. Tento proces je súčasne vykonávaný viacerými pomocnými procesmi pomocou paralelizácie. Každý proces súčasne preskúmava okolité minimá a počet týchto procesov je možné upraviť na základe dostupných výpočtových zdrojov a komplexnosti optimalizovaného chemického klastra. Každá iterácia GOAT je definovaná ako sekvencia lokálnej minimalizácie, prekonanie energetickej bariéry a následnej lokálnej minimalizácie. Algoritmus je ukončený, ak v posledných dvoch iteráciách nebolo nájdené nové globálne minimum. Porovnanie klasickej lokálnej optimalizácie a GOAT s viacerými paralelizovanými procesmi je zobrazený na *Obrázku č.13*. V hornej časti obrázku je vyznačená klasická lokálna optimalizácia, kde počiatočný bod je vyznačený červenou farbou a jeho lokálne minimum tmavomodrou farbou. Globálne minimum, vyznačené zelenou farbou, v tomto prípade nebolo nájdené. V dolnej časti je zobrazené fungovanie metódy GOAT. Jednotlivé tyrkysové body predstavujú jeden z aktuálnych bodov algoritmu a tmavomodré farby vyjadrujú už navštívené body [30].

Výhodou algoritmu GOAT je, že vďaka použitej metóde prehľadávania dokáže efektívne preskúmavať všetky okolité lokálne minimá a na výstupe poskytnúť všetky energeticky blízke konfigurácie nájdeného globálneho minima (nazývané aj ako *ensemble*). Hranica, ktorá definuje, kedy je daná konfigurácia blízko globálneho minima, je definovaná používateľom ako vzdialenosť od globálneho minima v jednotkách $kcal/mol$. Algoritmus GOAT je vhodný pre rôzne výpočtové modely, ako napríklad rýchle semiempirické alebo presnejšie DFT metódy. Vo všeobecnosti dokáže pracovať s ľubovoľnou metódou, ktorá je implementovaná v programe ORCA, pričom náročnejšie metódy, ako ab initio, by mali veľmi veľké nároky na dostupné výpočtové zdroje a ich použitie sa v bežných prípadoch neodporúča. Jednou z nevýhod GOAT je vysoký počet vykonávaných lokálnych optimalizácií, ktorých celkový počet je rádovo okolo $N_{ATOM} \cdot 100$, kde N_{ATOM} vyjadruje celkový počet atómov klastra. Tento

[9] Na vykonávanie týchto pohybov sú pri vybraných variantoch GOAT niektoré prvky skúmaného klastra, ako napríklad väzby, zamrazené, aby nedošlo k ich porušeniu

problém je adresovaný pomocou paralelizácie a v prípade dostatočného počtu výpočtových jadier je možné efektívne vykonávať globálnu optimalizáciu [30].



Obrázok 13: Porovnanie GOAT a lokálnej minimalizácie, zdroj: [30]

GOAT obsahuje viacero variant, ktoré upravujú, ako algoritmus funguje a definujú, čo sa očakáva od výstupu. *GOAT-ENTROPY* slúži na dôkladné prehľadanie okolia globálneho minima, pričom je pridaná dodatočná ukončovacia podmienka na základe entropie. *GOAT-EXPLORE* slúži na hľadanie globálneho minima od náhodnej aglomerácie atómov, kde iniciálna konfigurácia nehrá dôležitú úlohu a pri hľadaní môžu byť vytvárané nové resp. trhané existujúce väzby. *GOAT-REACT* dokáže preskúmavať možné reakcie a ich výstupné produkty na základe vstupnej konfigurácie atómov, ktoré predstavujú reaktanty. Každý z týchto variantov upravuje existujúce parametre. Napríklad na vykonávanie pohybu smerom nahor je možné pri *GOAT-ENTROPY* použiť metódu GFN-FF [31], ktorá je založená na metóde silového poľa a nie je dostupná pri *GOAT-EXPLORE* alebo *GOAT-REACT* [30].

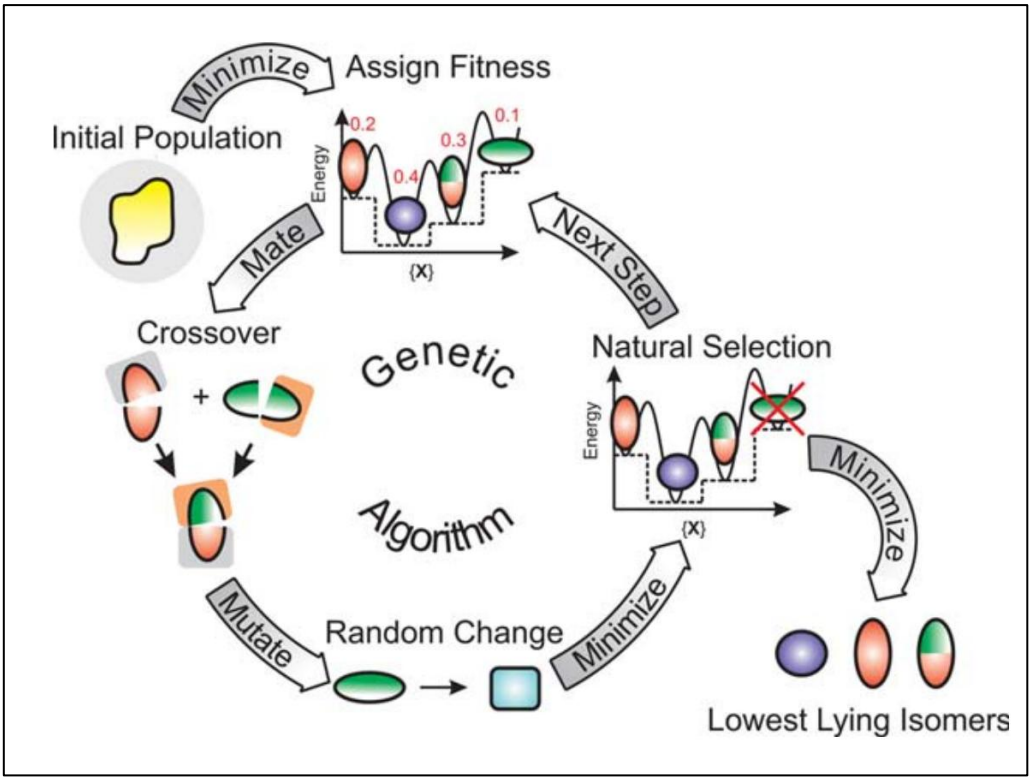
4. NÁVRH RIEŠENIA

Táto kapitola popisuje návrh riešenia praktickej časti diplomovej práce. Na základe konzultácií a analýzy problému a jeho častí definovaných v predošlých kapitolách, bol vytvorený návrh aplikácie na hľadanie globálneho minima chemických klastrov. Návrh je rozdelený do dvoch celkov – štruktúra programu a špecifikácia požiadaviek. V prvej podkapitole je opísaná štruktúra, na základe ktorej by mal byť vytvorený program. Druhá podkapitola obsahuje špecifikáciu požiadaviek, ktorá definuje funkcionálne a nefunkcionálne požiadavky na vytvorený program.

Na základe analýzy jednotlivých algoritmov, ktoré slúžia na globálnu optimalizáciu, boli zvolené genetické algoritmy. Problém optimalizácie chemických klastrov predstavuje špecifický problém, ktorý si vyžaduje buď priamu implementáciu chemických výpočtových modelov alebo využitie externého programu, ktorý má tieto modely už naprogramované a je možné ho využiť vo vytváranom programe. Štruktúru základného genetického algoritmu je možné jednoducho rozšíriť, aby dokázala riešiť problémy špecifické pre globálnu optimalizáciu chemických klastrov. Toto predstavuje jeden z hlavných dôvodov výberu tohto algoritmu.

4.1. Štruktúra programu

Ako základná inšpirácia pre vytváraný genetický algoritmus boli použité Lamarckové genetické algoritmy. Princíp fungovania týchto algoritmov pri optimalizácií chemických klastrov je zobrazený na *Obrázku č.14*. Na začiatku je vygenerovaná náhodná populácia, na ktorej sa vykoná lokálna minimalizácia. Minimalizovaným jedincom (konfiguráciám) sú priradené fitness hodnoty na základe ich potenciálnych energií. Následne sa vykoná operátor selekcie a kríženia. Operátor kríženia, znázornený na *Obrázku č.14*, zoberie jednu polovicu z každého rodičovského jedinca a ich spojením vzniká potomok, ktorý predstavuje novú konfiguráciu. Tento operátor predstavuje jednu z vhodných implementácií operátora kríženia, špecifických pre chemické klastre. Následne sa vykoná operátor mutácie, ktorého implementácie môžu napríklad aplikovať rôzne perturbácie na jednotlivé atómy klastra. Ďalej nasleduje lokálna minimalizácia jedincov na identifikovanie lokálnych miním, ktoré predstavujú finálnu populáciu jednej iterácie. Tento postup je iteratívne opakovaný, až kým sa nesplní jedna z ukončovacích podmienok.



Obrázok 14: Príklad Lamarckového genetického algoritmu, zdroj: [17]

4.2. Špecifikácia požiadaviek na softvér

Špecifikácia požiadaviek, pomocou funkcionálnych a nefunkcionálnych požiadaviek, definuje požadované správanie vytváraného softvéru. Hlavným cieľom tejto špecifikácie je jasne definovať, čo by vytvorený program mal byť schopný vykonávať a slúži aj na systematické overenie výsledného programu. Nasledovné požiadavky boli vytvorené a rozširované na základe konzultácií počas tvorby programu.

4.2.1. Funkcionálne požiadavky

Funkcionálne požiadavky opisujú požadované správanie a funkcionalitu vytváraného softvéru. Medzi bežné príklady funkcionálnych požiadaviek patria napríklad definície operácií systému alebo požadovaná štruktúra výstupov. Funkcionálne požiadavky pre vytváraný genetický algoritmus sú:

- *FGC-01*: Program obsahuje vhodné kódovanie klastrov, ktoré negatívne neovplyvňuje výpočet ich potenciálnej energie.
- *FGC-02*: Štruktúra vstupu je vyjadrená ako molekula v *.xyz* formáte.

- *FGC-03*: Konfiguráciu, parametre a operátory algoritmu je možné nastaviť na jednom mieste.
- *FGC-04*: Je možné optimalizovať všetky validné vstupné kombinácie molekúl.
- *FGC-05*: Je možné generovať náhodné konfigurácie klastrov na vytvorenie iniciálnej populácie.
- *FGC-06*: Je možné pridať vlastné odhady konfigurácií klastrov do iniciálnej populácie.
- *FGC-07*: Je možné vypočítať potenciálnu energiu klastra pomocou semiempirických, DFT alebo ab initio metód.
- *FGC-08*: Je možné vykresliť geometrie nájdených konfigurácií.
- *FGC-09*: Je možné vypočítať RMSD dvoch rôznych konfigurácií.
- *FGC-10*: Je možné vykonávať lokálnu optimalizáciu nájdených konfigurácií.
- *FGC-11*: Je možný výber použitého optimalizačného algoritmu pri lokálnej optimalizácii.
- *FGC-12*: Program obsahuje iba genetické operátory, ktoré dokážu vykonávať operácie na klastroch bez porušenia ich vstupnej štruktúry.
- *FGC-13*: Program obsahuje korekčné mechanizmy na opravenie porušenia štruktúr.
- *FGC-14*: Používateľ má možnosť meniť implementácie genetických operátorov a ich parametre.
- *FGC-15*: Používateľ má na výber viacero konfigurovateľných ukončovacích podmienok.
- *FGC-16*: Používateľ má možnosť využitia paralelizácie na zrýchlenie výpočtov programu.
- *FGC-17*: Výstup obsahuje nájdené globálne energetické minimum.
- *FGC-18*: Nájdené konfigurácie obsahujú vo výstupe ich súradnice v *.xyz* formáte spolu s ich potenciálnou energiou (a prípadne gradientmi).

- *FGC-19*: Výstup obsahuje používateľom definovaný počet energeticky najnižších konfigurácií okrem nájdeného globálneho minima.
- *FGC-20*: V prípade neočakávaného ukončenia program dokáže obnoviť výpočty od posledného uloženého bodu.

4.2.2. Nefunkcionálne požiadavky

Nefunkcionálne požiadavky priamo nešpecifikujú konkrétne funkcionality vytváraného softvéru, ale definujú jeho vlastnosti a obmedzenia. Patria sem napríklad požiadavky na kompatibilitu s externými softvérmi alebo rôzne iné operačné požiadavky. Navrhnuté nefunkcionálne požiadavky pre vytváraný genetický algoritmus sú:

- *NGC-01*: Vytvorený program musí byť kompatibilný a spustiteľný na operačnom systéme Linux.
- *NGC-02*: Výstupy programu sú ukladané do súborov, ktorých adresár je definovaný používateľom.
- *NGC-03*: Program obsahuje integráciu na externý chemicko-kvantový výpočtový program ORCA.
- *NGC-04*: Program obsahuje integráciu na externý chemicko-kvantový výpočtový program XTB.
- *NGC-05*: Program musí byť spustiteľný a funkčný na systémoch vysokovýkonnej výpočtovej techniky (napr. pomocou SLURM).
- *NGC-06*: Program by mal minimalizovať potrebný výpočtový čas bez zhoršenia kvality dosahovaných výsledkov.
- *NGC-07*: Program poskytuje informácie na štandardnom výstupe o priebehu výpočtov pre účely ladenia a kontroly.
- *NGC-08*: Kód programu a štruktúra genetického algoritmu musia byť modulárne a rozšíriteľné.
- *NGC-09*: Dokumentácia musí byť dostupná pre všetky konfigurácie, ktoré nastavuje používateľ.

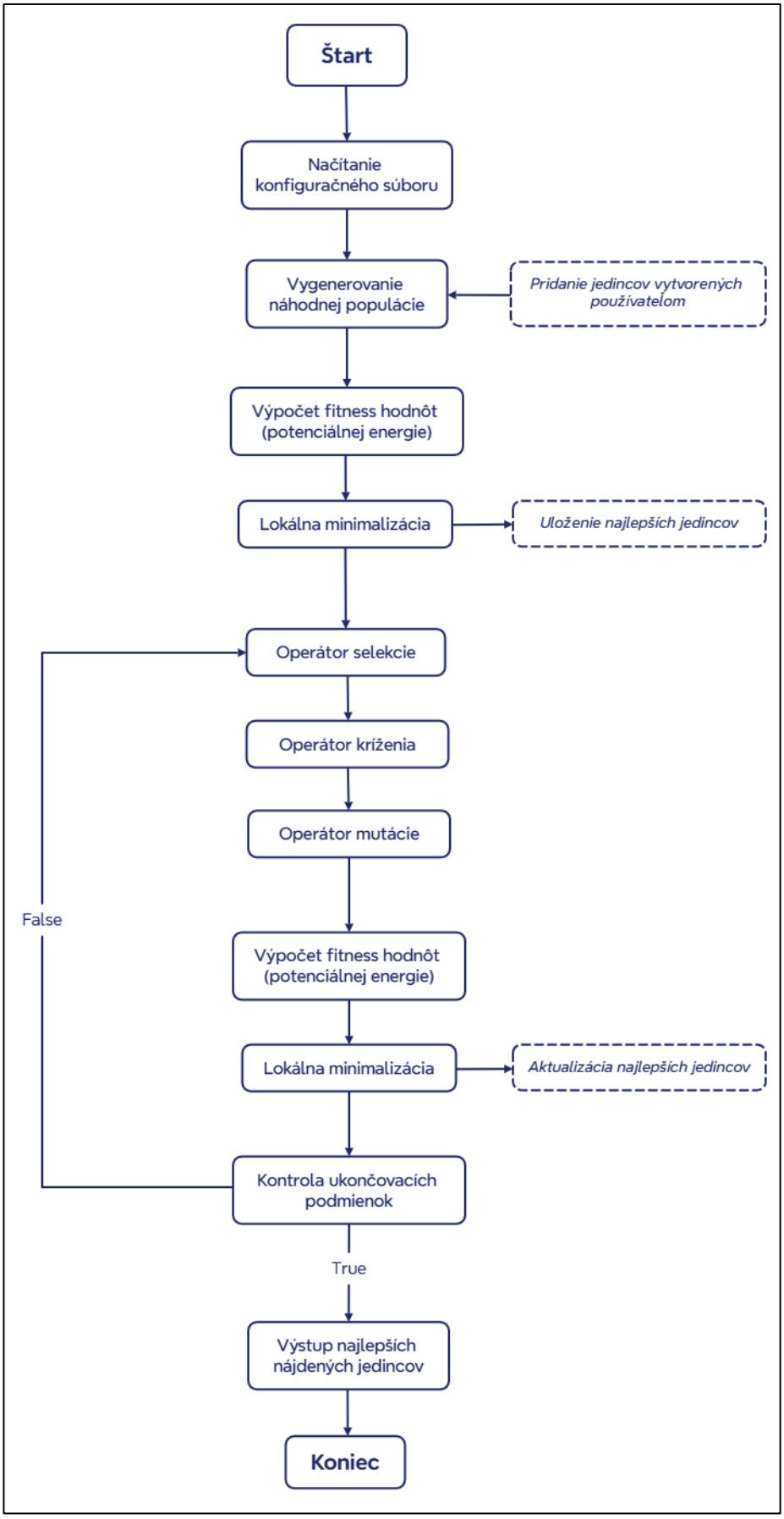
5. IMPLEMENTÁCIA

Cieľom tejto kapitoly je predstaviť vytvorený softvér a uviesť implementáciu jeho jednotlivých komponentov. Na základe návrhu z predošlej kapitoly bol vytvorený program, ktorý pomocou genetického algoritmu hľadá globálne minimum vstupného chemického klastra. Program bol vytvorený pomocou programovacieho jazyka Python a vybraných knižníc, ktoré zabezpečujú jednotlivé chemické aspekty programu. Medzi najdôležitejšie knižnice patria *MLatom* [32] a *ASE* (Atomic Simulation Environment) [33].

Knižnica *MLatom* obsahuje rozhrania pre rôzne chemické výpočtové programy, medzi ktoré patria aj programy *ORCA* a *XTB*. Pomocou týchto rozhraní je realizovaná komunikácia medzi vytvoreným optimalizačným programom s názvom *GenCluster* a externými programami, pomocou ktorých sa počíta potenciálna energia chemických klastrov. Ďalším dôležitým komponentom tejto knižnice je trieda *molecule*, ktorá bola rozšírená a na jej základoch bola vytvorená reprezentácia jedinca v aplikácii.

Knižnica *ASE* slúži v aplikácii primárne na lokálnu optimalizáciu. V tejto knižnici je implementované množstvo lokálnych optimalizačných algoritmov, ako napríklad L-BFGS alebo FIRE. Knižnica *ASE,* v spolupráci s knižnicou *Mlatom,* posúva jednotlivé atómy na základe zvoleného optimalizačného algoritmu a prepočítava ich energie pomocou rozhraní pre externé programy.

Štruktúra vytvoreného programu je znázornená na *Obrázku č.15*. Na vytvorenie genetického algoritmu nebola použitá žiadna knižnica a predstavuje vlastnú implementáciu. Tento spôsob bol zvolený hlavne z dôvodu, aby bolo možné algoritmus jednoducho rozšíriť a zároveň môcť správne zapracovať špecifické operácie spojené s chemickými klastrami. Jednotlivé kroky algoritmu a ich implementácie sú opísané v nasledujúcich podkapitolách.



Obrázok 15: Štruktúra vytvoreného programu, zdroj: Autor

5.1. Inicializácia algoritmu a populácie

Program začína načítaním *.yaml* konfiguračného súboru[10], ktorý obsahuje všetky potrebné vstupné údaje pre optimalizáciu, genetické operátory a ich parametre a vybraný spôsob generovania iniciálnej generácie. Na základe tohto súboru, ktorý predstavuje jediný povinný argument pre spúšťanie algoritmu, sa vytvoria všetky požadované triedy potrebné pre beh algoritmu. V prípade chýbajúcich alebo nesprávnych parametrov vo vstupnej konfigurácií je používateľ informovaný o detegovanej chybe a algoritmus je ukončený.

Jednou z konfigurácií je výber generátora, ktorý generuje náhodných jedincov. Tento generátor sa využíva primárne pri generovaní jedincov vstupnej populácie a v programe existuje implementácia s názvom *RandomGeometryGenerator*, ktorá generuje jedincov náhodným usporiadaním vstupných jednotiek (atómov alebo molekúl) na základe definovaných veľkostí. Táto implementácia vyžaduje dva argumenty, ktoré definujú minimálne a maximálne vzdialenosti medzi ťažiskami vstupných molekúl v Angströmoch[11] (Å).

Používateľ má taktiež možnosť vo vstupnom konfiguračnom súbore definovať cestu k súboru, ktorý obsahuje klastre v *.xyz* formáte. Tieto klastre sa pridajú do iniciálnej populácie namiesto vygenerovaných jedincov. Táto funkcionalita je určená hlavne pre skúsených používateľov, ktorí dokážu vytvoriť vhodný odhad geometrie globálneho minima a pomocou tohto odhadu urýchliť konvergenciu algoritmu.

Po vytvorení iniciálnej populácie je pre každého jedinca vypočítaná jeho fitness hodnota. Používateľ ďalej definuje, či sa má vykonať lokálna minimalizácia na tejto vygenerovanej populácií. Následne sú jedinci zoradení na základe ich fitness hodnôt a tí najlepší sú uložení do pamäte, pričom aspoň jeden z nich predstavuje počiatočné nájdené globálne minimum. Maximálny počet uložených jedincov je definovaný používateľom a na konci každej iterácie algoritmu sa kontroluje, či nebolo nájdené lepšie riešenie. Ak má nájdený jedinec lepšiu fitness hodnotu, tak na základe RMSD sa kontroluje, či uložený jedinec odpovedá rovnakému lokálnemu minimu a v prípade, že áno, daného jedinca nahradí v uložených výsledkoch. V opačnom prípade daného jedinca posunie o jednu pozíciu nižšie a na jeho pôvodnú pozíciu je uložený nový a lepší jedinec (slúži na predchádzanie duplicitným klastrom vo výsledkoch).

[10] Príklad konfiguračného súboru sa nachádza v prílohe Používateľská príručka
[11] Jeden Å je rovný 10^{-10} metrov

Každý jedinec začína s vekom nula a po každej iterácii genetického algoritmu, v ktorej nedošlo k jeho zmene, sa jeho vek zvýši o jedna. Tento mechanizmus zabezpečuje, že jedinci, ktorí zostávajú nezmenení počas viacerých iterácií, sú následne odstránení z populácie, čo prispieva k efektívnejšiemu prehľadávaniu účelovej funkcie. Používateľ má možnosť nastaviť maximálny vek jedincov alebo tento mechanizmus neaplikovať.

5.2. Výpočet energie a lokálna minimalizácia

Výpočet potenciálnej energie klastra v rámci genetického algoritmu je definovaný ako výpočet fitness hodnoty. Ako bolo spomenuté v tretej kapitole, výpočet tejto energie predstavuje výpočtovo náročný proces. Existuje množstvo metód, pomocou ktorých je možné túto energiu vypočítať. Z tohto dôvodu boli pridané rozhrania pre dva externé a voľne dostupné programy *ORCA* a *XTB*. Tieto programy obsahujú efektívne implementácie množstva metód a poskytujú používateľovi dostatočne široký výber pre potreby optimalizácie chemického klastra. Program taktiež obsahuje rozhranie pre knižnicu *PySCF* [34], ktorá je priamo implementovaná v Pythone a obsahuje implementácie rôznych výpočtových metód. Výber programu a použitej metódy pre výpočet energie je definovaný vo vstupnom konfiguračnom súbore.

Ďalší proces, ktorý je úzko prepojený s výpočtom energie, je lokálna minimalizácia a vykonáva sa hneď po výpočte energie. V konfiguračnom súbore je možné ju vypnúť, určiť maximálny počet krokov optimalizácie, vybrať program a metódu alebo nastaviť frekvenciu jej vykonávania vzhľadom na iterácie. Lokálna minimalizácia predstavuje výpočtovo najnáročnejší proces iterácie genetického algoritmu. Z tohto dôvodu je možné nastaviť, aby sa minimalizácia vykonávala len na definovanom počte najlepších jedincov s odlišnou RMSD hodnotou. Používateľ má možnosť si vybrať z viacero optimalizačných knižníc, ako napríklad *ASE* a ich rôznych implementácií optimalizačných algoritmov.

Tieto dve časti genetického algoritmu sú paralelizované a v konfiguračnom súbore je možné nastaviť počet procesov, ktoré budú v každej iterácii súčasne počítať energiu resp. minimalizovať jedincov. Tieto dva procesy predstavujú tzv. *bottleneck* vytvoreného genetického algoritmu a vo všeobecnosti aj všetkých algoritmov, ktoré hľadajú globálne minimum chemických klastrov. Väčšina bežných programov, ktoré implementujú chemické výpočtové metódy majú podporu pre paralelizáciu daných metód. Paralelizácia je v programe implementovaná ako počet procesov, ktoré súčasne počítajú energiu a počet jadier, ktorý každý proces využíva. Pre náročnejšie (väčšie) klastre je odporúčané spúšťať globálnu optimalizáciu,

len ak je dostupný dostatočný počet výpočtových zdrojov. Tieto dva procesy algoritmu zároveň predstavujú jednu z najväčších prekážok na vykonanie komplexnej analýzy vplyvov jednotlivých parametrov genetického algoritmu na kvalitu výsledkov. V prípade veľmi veľkých klastrov môže pridanie jedného jedinca do populácie znamenať dodatočnú hodinu počítania v každej iterácii algoritmu.

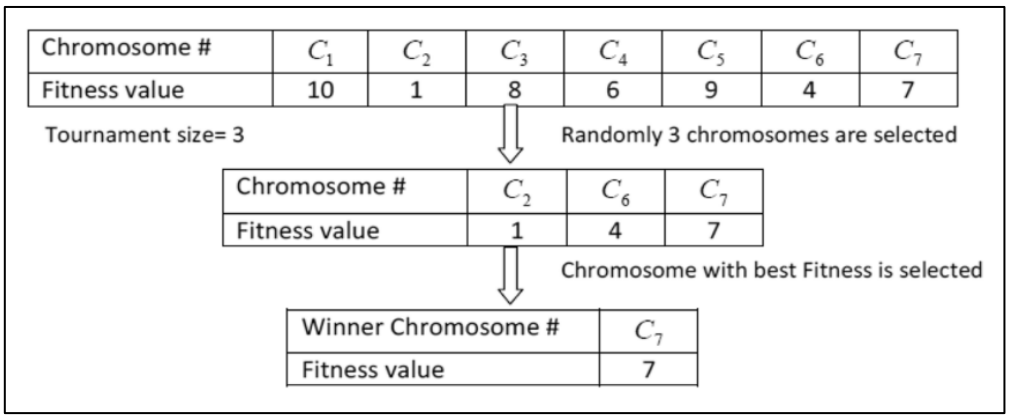
5.3. Genetické operátory

Genetické operátory predstavujú kľúčovú súčasť globálnej optimalizácie vytvoreného genetického algoritmu, keďže zabezpečujú efektívne prehľadávanie skúmaného povrchu potenciálnej energie. Medzi hlavné problémy pri tvorbe operátorov kríženia a mutácie pre optimalizáciu chemických klastrov patrí vytvorenie implementácií, ktoré správne aproximujú princípy a ciele daných operátorov. Cieľom operátora kríženia je nájsť klastre, ktoré sa nachádzajú v okolí rodičovských klastrov. Úlohou operátora mutácie je objaviť alternatívnu konfiguráciu prostredníctvom náhodnej úpravy klastra zodpovedajúcej inej polohe na skúmanom PES. Správne implementácie týchto operátorov predstavujú základnú podmienku pre korektné a efektívne fungovanie každého genetického algoritmu. Vlastné implementácie navrhnutých operátorov vo vytvorenom programe sú opísané v nasledujúcich podkapitolách.

5.3.1. Selekcia

Operátor selekcie vyberá dvoch jedincov na základe ich fitness hodnôt. Na týchto dvoch jedincoch môžu byť ďalej vykonané operátory mutácie a kríženia. Tento proces je opakovaný, až kým nie je vytvorená nová generácia.

Vo vytvorenom programe je implementovaná trieda *TournamentSelector*. Tento operátor selekcie predstavuje štandardnú implementáciu turnajovej selekcie. Používateľ má možnosť nastaviť veľkosť turnaja, ktorá vyjadruje veľkosť podmnožiny jedincov, z ktorých



Obrázok 16: Princíp fungovania turnajovej selekcie, zdroj: [41]

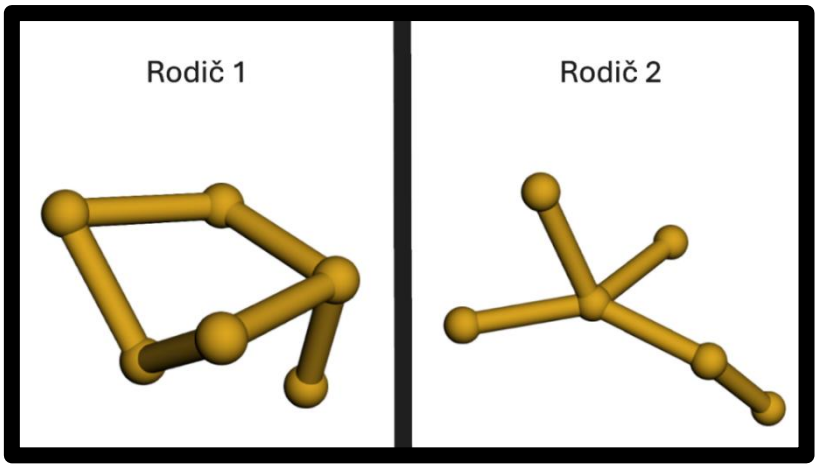
bude vybraný jedinec s najlepšou fitness hodnotou. Tento proces sa následne zopakuje pre druhého jedinca, pričom prvý vybraný jedinec je odstránený z pôvodnej populácie, aby nedošlo k duplicitnému výberu rovnakého jedinca. Princíp fungovania tohto operátora je znázornený na *Obrázku č.16*.

Vzhľadom na charakteristiku riešeného problému a využívania lokálnej minimalizácie je bežné, že sa v populácii súčasne vyskytuje viacero klastrov, ktoré po vykonaní lokálnej minimalizácie konvergujú k rovnakému lokálnemu minimu. Na riešenie tohto problému bol navrhnutý a vytvorený variant turnajovej selekcie, ktorý je implementovaný v triede *RmsdTournamentSelector*. Tento operátor selekcie funguje na rovnakom princípe ako *TournamentSelector* s tým rozdielom, že po výbere prvého jedinca sa aplikuje RMSD[12] penalizačná funkcia na podmnožinu potenciálnych druhých jedincov. To znamená, že čím bližšie je RMSD k nule, vzhľadom na prvého vybraného jedinca, tým výraznejšie je pri selekcii znížená fitness hodnota uvažovaného jedinca. Toto riešenie pomáha udržovať diverzitu v novej generácii a vedie k efektívnejšiemu prehľadávaniu skúmaného povrchu potenciálnej energie.

V programe je taktiež implementovaný elitný selektor *EliteSelector*, ktorý vyberá jedincov s najlepšou fitness hodnotou. Títo jedinci sú prenesení do ďalšej generácie bez vykonania ostatných genetických operátorov. Používateľ má možnosť nastaviť počet prenesených jedincov a zároveň aj možnosť vypnúť používanie tohto operátora.

5.3.2. Kríženie

Cieľom operátora kríženia je zabezpečiť vhodnú výmenu informácii medzi dvoma jedincami. V programe je implementovaný operátor kríženia *SpliceCrossover*. Tento operátor

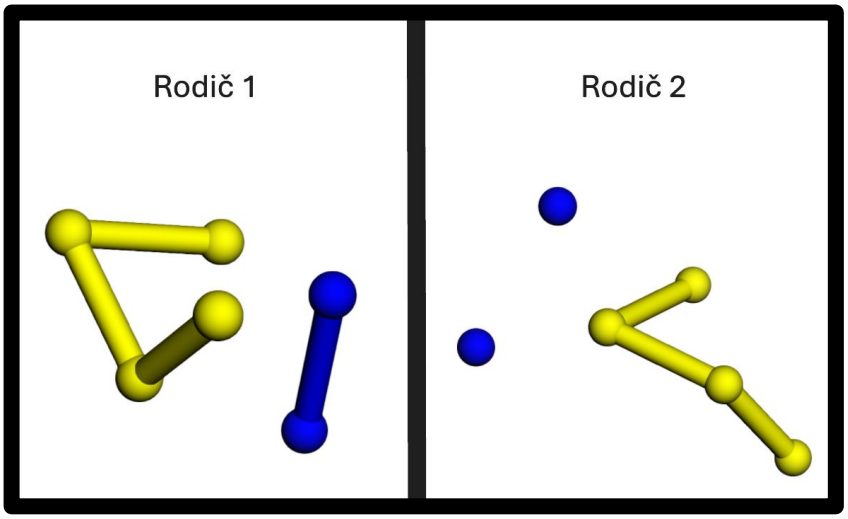


Obrázok 17: Rodičovskí jedinci pred operátorom kríženia, zdroj: Autor

[12] z angl. Root Mean Square Deviation

vykonáva kríženie dvoch jedincov pomocou náhodných rezov, ktoré rozdelia pôvodných jedincov na dve skupiny atómov. Tieto skupiny sú následne spojené s odpovedajúcimi skupinami druhého jedinca. Priebeh tohto operátora je znázornený na nasledovných obrázkoch. Operátor začína tým, že má na vstupe dvoch jedincov, ktorí sú zobrazení na *Obrázku č.17*.

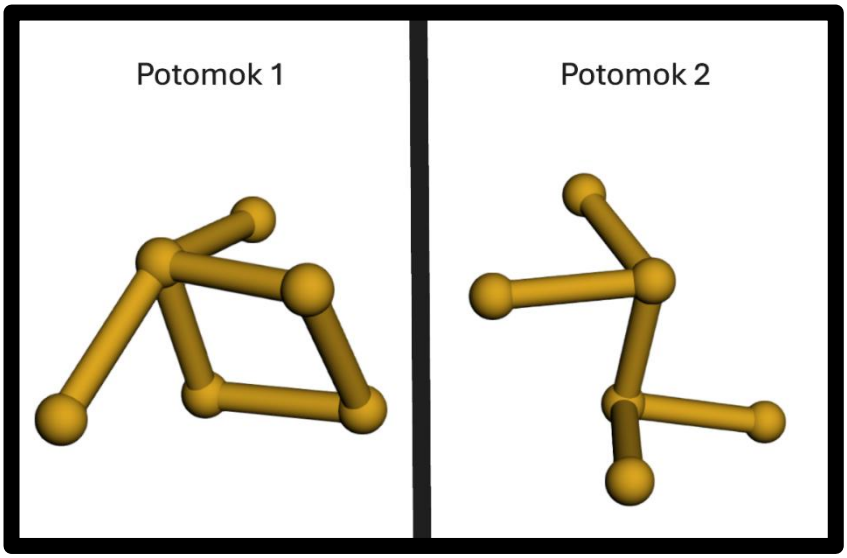
Následne sa na prvom rodičovi vykoná náhodný rez, ktorý vytvorí dve skupiny atómov. Druhý rodič je rozdelený tak, aby vznikli skupiny atómov, ktoré sú kompatibilné so skupinami prvého rodiča. To znamená, že ak je prvý rodič rozdelený na skupiny troch a piatich atómov, tak aj druhý rodič bude rozdelený na skupiny rovnakej veľkosti. V prípade, že sa optimalizácia vykonáva na molekulách, ktoré sa skladajú z viacerých atómov, berú sa do úvahy celé molekuly tak, aby nebola porušená ich štruktúra[13]. Výsledok tejto operácie je zobrazený na *Obrázku č.18*. Vzniknuté skupiny atómov sú zobrazené modrou a žltou farbou.



Obrázok 18: Rozdelenie rodičovských jedincov na dve skupiny, zdroj: Autor

Ďalej nasleduje spojenie odpovedajúcich skupín jednotlivých rodičov, čím sa získajú finálni potomkovia tohto operátora. Atómy prvého rodiča, vyznačené modrou farbou, sú spojené s atómami druhého rodiča, ktoré sú vyznačené žltou farbou. Rovnaký postup sa využíva aj pri rekombinácii modrých atómov druhého rodiča. V prípade prekryvov sa využíva korekčný mechanizmus, ktorý dané atómy posunie tak, aby vznikol korektný finálny klaster. Výslední potomkovia sú zobrazení na *Obrázku č.19*.

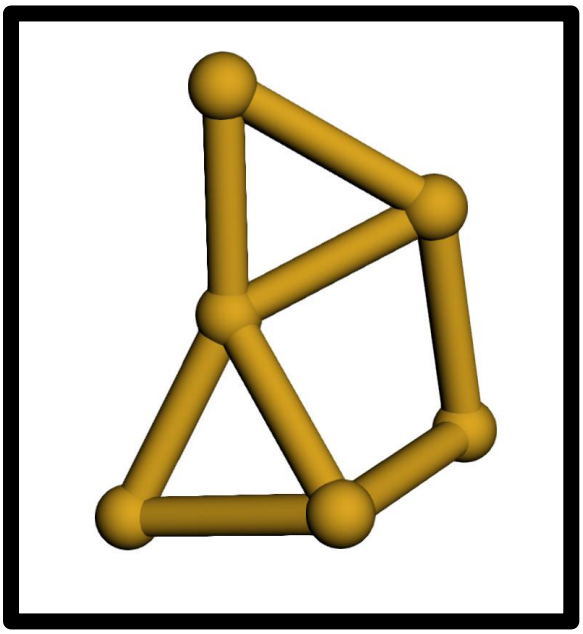
[13] Situáciu pri klastri skladajúceho sa z viacerých molekúl je možné si predstaviť pomocou *Obrázku č.18*, kde každý odrezaný modrý bod nepredstavuje atóm ale molekulu skladajúcu sa z viacerých atómov. Molekuly sú následne vymenené a nahradené ich zodpovedajúcimi atómami.



Obrázok 19: Potomkovia po vykonaní operátora kríženia, zdroj: Autor

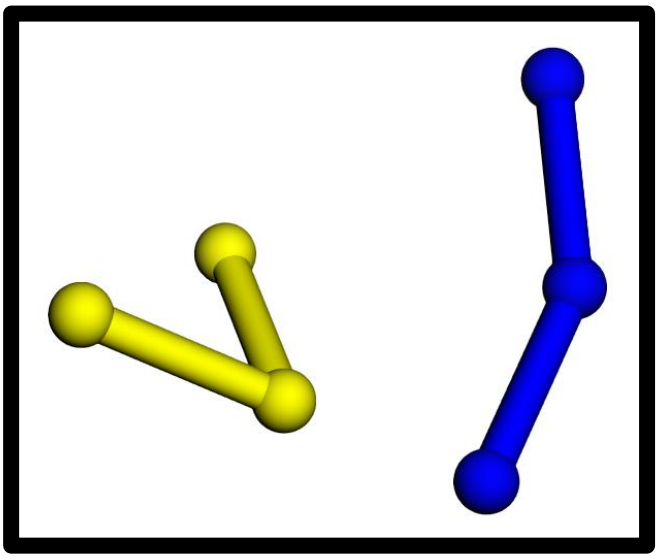
5.3.3. Mutácia

Úlohou operátora mutácie je náhodne zmeniť jedinca tak, aby sa presunul do iného priestoru skúmanej PES. Na docielenie tohto výsledku bol v programe vytvorený operátor mutácie s názvom *TwinningMutator*. Tento operátor používa podobné operácie ako operátor *SpliceCrossover*. Na vstupe je jeden jedinec, ktorý je rozdelený na dve skupiny atómov. Následne je na jednej z týchto skupín vykonaná náhodná rotácia a tieto dve výsledné skupiny sú spojené pre vytvorenie zmutovaného jedinca. Postup tohto operátora je znázornený na nasledovných obrázkoch. Vstupný jedinec je vizualizovaný na *Obrázku č.20*.

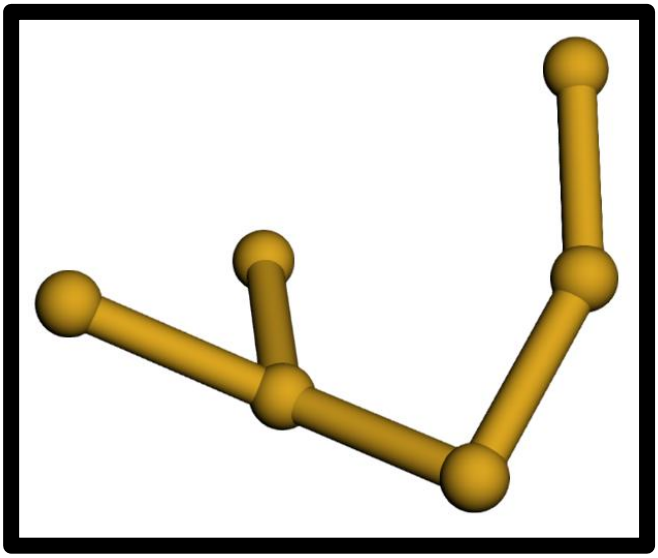


Obrázok 20: Jedinec pred operátorom mutácie, zdroj: Autor

Tento jedinec je rozdelený na dve skupiny atómov pomocou vykonania náhodného rezu. Výsledok tejto operácie je zobrazený na *Obrázku č.21*. Následne sa vykoná náhodná rotácia jednej zo skupín a výsledný jedinec sa získa spätným spojením týchto dvoch skupín. V prípade výskytu prekryvu sa aj v tomto operátore vykonáva korekčný mechanizmus, ktorý zabezpečuje, že výsledný klaster má realistickú štruktúru. Výsledný jedinec je zobrazený na *Obrázku č.22*.



Obrázok 21: Rozdelenie jedinca na dve skupiny, zdroj: Autor



Obrázok 22: Jedinec po vykonaní operátora mutácie, zdroj: Autor

5.4. Ukončovacie podmienky a výstupy

Vo vytvorenom programe sú implementované štyri ukončovacie podmienky, pričom používateľ má možnosť vo vstupnom konfiguračnom súbore definovať ľubovoľnú kombináciu týchto podmienok. Správna konfigurácia ukončovacích podmienok je kľúčová pre dosiahnutie kvalitných výsledkov. Ako bolo spomenuté v predošlých kapitolách, povrch potenciálnej energie obsahuje množstvo lokálnych miním a preto je potrebné ukončovacie podmienky nastaviť tak, aby algoritmus neskonvergoval predčasne. Dostupné ukončovacie podmienky v programe sú:

- *MaxIterationsTerminator*, ktorý algoritmus ukončuje po dosiahnutí maximálneho počtu iterácií.
- *StagnationTerminator*, ktorý algoritmus ukončuje po dosiahnutí maximálneho počtu iterácií, kedy sa fitness hodnota najlepšieho jedinca nezlepšila.
- *DurationTerminator*, ktorý algoritmus ukončuje po vypršaní používateľom definovaného času.
- *ConvergenceTerminator*, ktorý algoritmus ukončuje, ak sa za posledných x a y iterácií fitness hodnota najlepších jedincov nezlepšila o hodnotu delta, ktorá je definovaná používateľom.

Po ukončení algoritmu sú všetky nájdené energeticky najnižšie klastre vypísané na štandardný výstup. Klastre sú zoradené vo vzostupnom poradí na základe ich energií a obsahujú informácie o vypočítanej potenciálnej energii, štruktúre klastra v *.xyz* formáte a v prípade, že daný klaster má vypočítané gradienty, tak výstup obsahuje dodatočne aj informáciu o norme gradientu daného klastra. Ak prostredie, v ktorom beží program, podporuje *Ipykernel*, tak výsledné klastre sú na štandardnom výstupe aj vizualizované.

Výsledné štruktúry klastrov sú zároveň uložené do súboru v adresári, ktorý bol definovaný používateľom. V prípade, že bola zapnutá možnosť *checkpoint*, ktorá ukladá priebežné výsledky po každej iterácii algoritmu, výstup obsahuje dodatočný súbor, ktorý odpovedá poslednému uloženému priebežnému výsledku.

5.5. Výpočet RMSD a translácia jedincov

Jedným z dôležitých procesov na rozlišovanie jedincov vo vytvorenom programe je výpočet RMSD. Na základe tejto hodnoty sa v programe určuje miera podobnosti dvoch jedincov. Tradičný spôsob zahŕňa centrovanie oboch molekúl podľa ich ťažiska, nasledované rotáciou atómov tak, aby vzdialenosti medzi zodpovedajúcimi atómami boli čo najmenšie. Tento proces je závislý od usporiadania atómov a existuje viacero algoritmov, ktorých cieľom je atómy usporiadať tak, aby RMSD bola minimálna [35].

Medzi usporiadacie algoritmy, ktoré sú implementované vo vytvorenom programe, patria maďarský algoritmus a brute force algoritmus. Cieľom maďarského algoritmu je usporiadať atómy tak, aby súčet vzdialeností medzi priradenými dvojicami atómov bol minimálny. Brute force algoritmus preskúmava všetky možné kombinácie a vyberá z nich tú najlepšiu. Tento algoritmus sa neodporúča používať pri klastroch skladajúcich sa z viacej ako 6 atómov kvôli jeho faktoriálovej časovej zložitosti. Na usporiadaných atómoch sa nakoniec vykoná translácia a vypočíta sa výsledná RMSD. Usporiadacie algoritmy boli implementované s využitím knižníc a transláciu s výpočtom RMSD vykonáva knižnica *rdkit*. Presnosť týchto metód ale nie je dostatočná pre potreby vytvoreného programu a z tohto dôvodu bola implementovaná ďalšia metóda výpočtu RMSD [35].

Ako inšpirácia bola zvolená metóda výpočtu RMSD pomocou vlastných čísel matíc vzdialeností, ktorá sa využíva v algoritme GOAT-EXPLORE. Pri použití tejto metódy nemá poradie atómov ani translácia žiadny vplyv na výslednú RMSD hodnotu. Medzi jednu z nevýhod tejto metódy patrí nesprávne určenie RMSD chirálnych izomérov[14]. Metóda bola implementovaná v programe nasledovne [30]:

1. Určenie matíc internukleárnych vzdialeností obidvoch molekúl D_1 a D_2:

$$D = \begin{pmatrix} d_{11} & \cdots & d_{1N} \\ \vdots & \ddots & \vdots \\ d_{N1} & \cdots & d_{NN} \end{pmatrix},$$

   kde d_{ij} predstavuje euklidovskú vzdialenosť medzi atómami s indexmi i a j, pričom $i, j = 1, 2, \dots, N$, kde N predstavuje celkový počet atómov.

2. Výpočet vlastných čísel matíc:

$$D_1 \vec{v}_i = \lambda_i \vec{v}_i$$
$$D_2 \vec{w}_i = \mu_i \vec{w}_i,$$

   kde λ_i predstavuje vlastné čísla matice D_1 a μ_i predstavuje vlastné čísla matice D_2. Následne dostávame vzostupne zoradené vlastné čísla matíc:

$$D_1\colon\ \lambda_1, \lambda_2, \dots, \lambda_N$$
$$D_2\colon\ \mu_1, \mu_2, \dots, \mu_N$$

3. Výpočet RMSD na základe rozdielov vlastných čísel pomocou vzťahu:

$$RMSD(D_1,\ D_2) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\lambda_i - \mu_i)^2}$$

[14] Chirálne izoméry predstavujú dve štruktúry, ktoré sú vzájomným zrkadlovým obrazom, ale nie je možné vykonať transláciu a rotáciu tak, aby sa kompletne prekrývali.

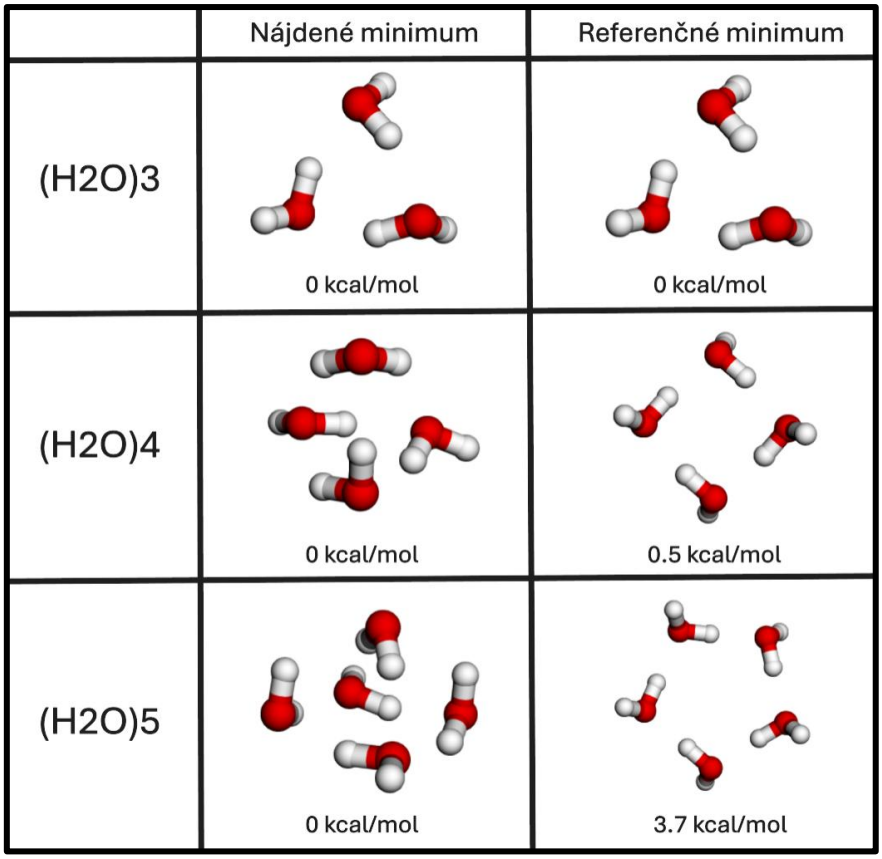
6. POROVNANIE VÝSLEDKOV

Na overenie správnosti a kvality vytvoreného programu bolo vykonaných niekoľko testovacích úloh. Testy boli vykonávané na klastroch vody $(H_2O)_n$, pre $n = 3, 4, 5$ a na klastre uhlíka C_{60}. Posledný test predstavoval porovnanie s algoritmom *ORCA GOAT-EXPLORE* pri globálnej optimalizácií vybraných klastrov zlata. Výsledky týchto testov sú opísané v nasledovných podkapitolách.

6.1. Globálna optimalizácia $(H_2O)_n$

Prvý test sa vykonával na klastroch vody, s cieľom nájsť globálne minimum každého uvažovaného klastra. Na overenie dosiahnutých výsledkov boli použité vypočítané globálne minimá klastrov z vedeckého článku ako validačné referencie [36].

Výsledky globálnej optimalizácie sú zobrazené a porovnané na *Obrázku č.23.* Program dokázal nájsť globálne minimum všetkých troch klastrov s použitím metódy *GFN1-xTB*. Výsledné globálne minimá nezodpovedajú identifikovaným globálnym minimám pre $n = 4, 5$ z článku [36]. Je to z dôvodu, že metóda *GFN1-xTB* používa zjednodušený model vodíkových väzieb a elektrónových interakcií, ktoré majú vplyv na výslednú energiu klastrov vody, kým autori [36] použili na optimalizáciu metódu *MP2/6-31G(d)*. Ako príklad je možné uviesť klaster vody $(H_2O)_4$, kde vytvorený program identifikoval globálne minimum ako jedno z lokálnych miním z literatúry, ktoré má odchýlku od skutočného globálneho minima 4.19 $kcal/mol$. Globálne minimá z [36] boli lokálne optimalizované pomocou metódy *GFN1-xTB* a porovnané s nájdenými globálnymi minimami. Energia nami zisteného globálneho minima bola vždy nižšia než výsledky z [36], čo naznačuje kvalitu našich výsledkov. Zároveň je možné vidieť, ako sa so zvyšujúcim počtom atómov rozdiel medzi našim výsledkom a prácou [36] zvyšuje. Toto pozorovanie však neznamená, že by dané výsledky [36] boli nedostatočné, je za neho zodpovedná metodologická odlišnosť.

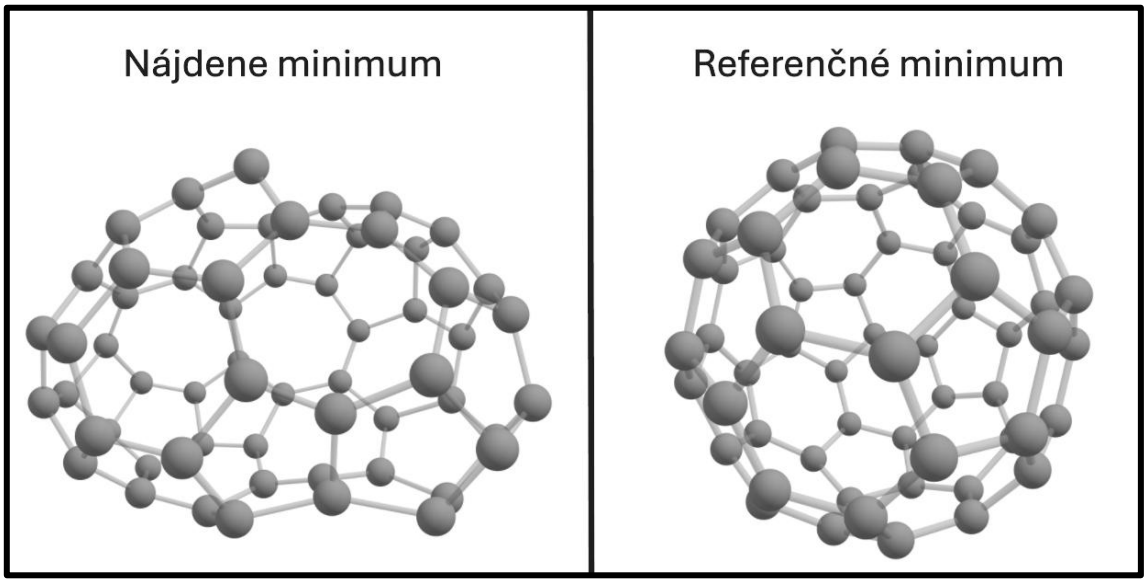


Obrázok 23: Globálna optimalizácia klastrov vody, zdroj: Autor

6.2. Globálna optimalizácia C_{60}

Ako ďalší test bola vybraná globálna optimalizácia uhlíkového klastra C_{60}, ktorý obsahuje veľké množstvo lokálnych miním a predstavuje výrazne náročnejší optimalizačný problém než klastre vody v predchádzajúcom teste. Globálne minimum C_{60} predstavuje štruktúru fulerénu[15], ktorú sa nepodarilo nájsť pri globálnej optimalizácii, lebo algoritmus skonvergoval predčasne. Výpočet programu trval približne 16 hodín pri použití 64 procesorových jadier. Nájdená štruktúra predstavuje štruktúru pripomínajúcu fulerén, ktorý sa ale skladá aj z priamo viazaných päťuholníkov resp. sedemuholníkov. Dosiahnutý výsledok je zobrazený spolu so skutočným globálnym minimom na *Obrázku č.24*. Odchýlka od skutočného globálneho minima je 471.1 $kcal/mol$. Tento veľký rozdiel naznačuje, že algoritmus má problémy pri vyhľadávaní vysoko symetrických štruktúr.

[15] Fulerén predstavuje guľovitú štruktúru, ktorá sa skladá z päť- resp. šesťuholníkov



Obrázok 24: Výsledok globálnej optimalizácie uhlíkového klastra C_{60}, zdroj: Autor

Alternatívne sme sa pokúsili problém obísť a program sme inicializovali tak, aby negeneroval 60 uhlíkatý skelet zo 60 samostatných atómov, ale poskytli sme mu vstupnú geometriu pre *sp2* hybridizovaný systém[16] resp. *sp3*. Program následne generoval 15 molekúl so 4 uhlíkmi (15-krát *sp2* štruktúru) resp. 12 molekúl s 5 uhlíkmi (12-krát *sp3* štruktúra). Predpokladali sme, že vnesením tejto informácie sa konvergencia urýchli.

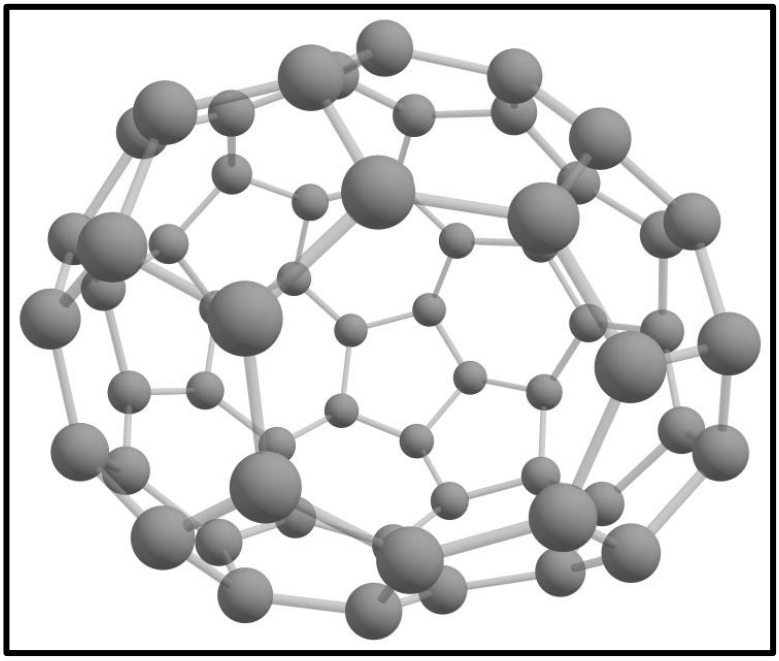
V prípade použitia *sp2* výsekov zo štruktúry grafénu program skonvergoval k podobnej štruktúre, ako je na *Obrázku č.24*, pričom energetický rozdiel voči fulerénu bol ešte výraznejší s odchýlkou 584 $kcal/mol$.

Ani štart z *sp3* štruktúry (základ diamantu) konvergenciu neurýchlil. Výsledné globálne minimum je zobrazené na *Obrázku č.25* a je energeticky blízke minimu na *Obrázku č.24* s odchýlkou od globálneho minima 478.3 $kcal/mol$. Globálna optimalizácia trvala približne 23 hodín a bola ukončená na základe ukončovacej podmienky presiahnutia maximálneho času optimalizácie.

Na tomto systéme sme testovali aj vplyv rôznych užívateľských vstupov, ako je napríklad pridávanie vlastných odhadov do počiatočnej populácie. Ako *initial guess* sme programu poskytli štruktúry výseku z grafénu, diamantu a fulerénu. Fulerén bol identifikovaný ako globálne minimum. S narastajúcim počtom iterácii bolo zaujímavé sledovať, ako *fuleren-like*

[16] Uhlíkové *sp2* štruktúry sú charakterizované tým, že každý uhlík vytvára tri väzby s inými atómami, ako je napríklad možné vidieť v štruktúre fulerénu na *Obrázku č.24*. Uhlíkové *sp3* štruktúry sú charakterizované tým, že každý uhlík vytvára štyri väzby (ako príklad je možné uviesť štruktúru diamantu).

štruktúry vytláčajú málo stabilný výsek z diamantu na nižšie pozície. Z tohto pohľadu môže byť program *GenCluster* užitočný na identifikáciu nízko ležiacich izomérov známej štruktúry.



Obrázok 25: Výsledok globálnej optimalizácie C_{60} z sp3 štruktúry, zdroj: Autor

6.3. Porovnanie s ORCA GOAT-EXPLORE

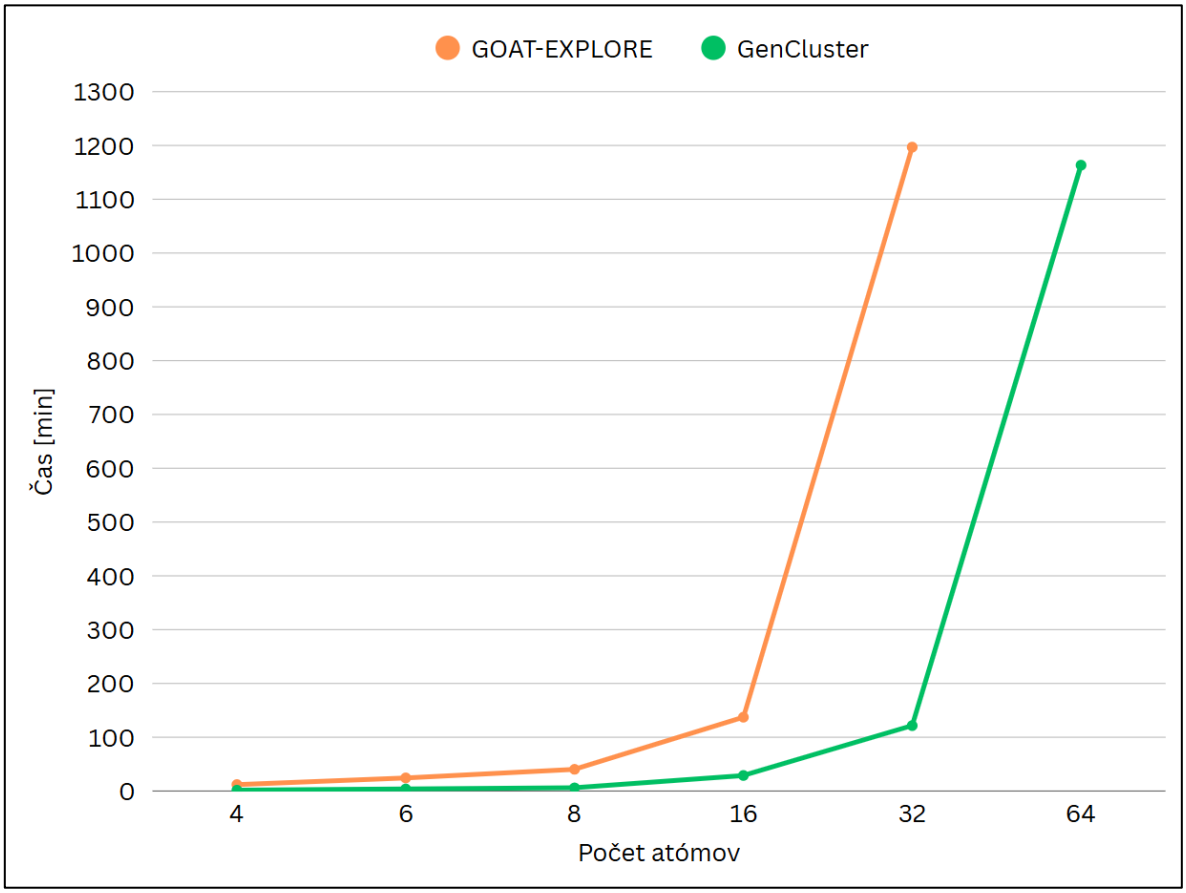
Ako posledný test bolo vykonané porovnanie výsledkov globálnej optimalizácie s programom *ORCA GOAT-EXPLORE*. Na optimalizáciu boli zvolené klastre zlata Au_n, pre $n = 4, 6, 8, 16, 32$ a obidva programy mali dostupných 64 procesorových jadier a používali rovnakú metódu *GFN1-xTB*, pričom *ORCA* nemá vlastnú implementáciu tejto metódy, ale obsahuje v sebe program *XTB*. Výsledky sú zobrazené v *Tabuľke č.2* a časy sú vzájomne porovnané na *Obrázku č.26*. Zrýchlenie *GenCluster* vyjadruje zrýchlenie vzhľadom na čas programu *GOAT-EXPLORE*. Výsledok pre Au_{64} je taktiež zobrazený v tabuľke aj na obrázku pre úplnosť, pričom bol ale inicializovaný vlastnou počiatočnou štruktúrou. Vytvorený

Tabuľka 2: Porovnanie výsledkov s GOAT-EXPLORE, zdroj: Autor

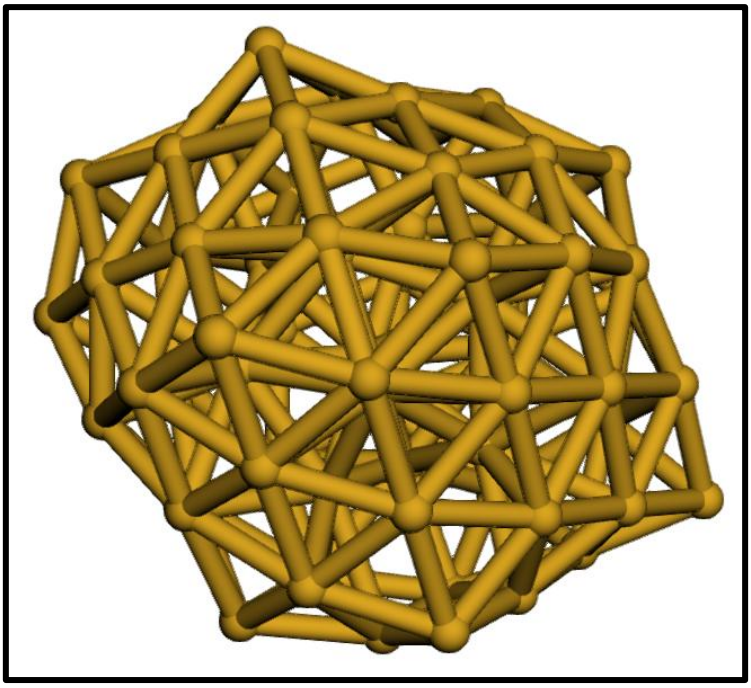
Počet atómov	Čas GOAT-EXPLORE [min]	Čas GenCluster [min]	Energia GOAT-EXPLORE	Enegia GenCluster	Rozdiel energií [kcal/mol]	Zrýchlenie GenCluster
4	12	1.56	-15.8574	-15.8574	0	7.681
6	24.38	3.8	-23.8512	-23.8512	0	6.412
8	40.37	6	-31.8288	-31.8288	0	6.719
16	137.23	28.91	-63.7939	-63.7939	0	4.746
32	1196.85	121.55	-127.8489	-127.8489	0	9.846
64	—	1163.32	—	-256.0793	—	—

program dokázal v každom testovacom prípade nájsť energeticky ekvivalentné globálne minimum rýchlejšie než *GOAT-EXPLORE*.

Vzhľadom na systematické zväčšovanie klastrov sme skúmali aj možnosti, ako využiť predchádzajúci výsledok. Klaster Au_{64}, ktorý bol už mimo možností *GOAT-EXPLORE*, sme pre náš program vytvorili zopakovaním štruktúr Au_{32} prepojených viacerými väzbami. Výsledný najstabilnejší klaster je vizualizovaný na *Obrázku č. 27*. Aj keď samotný štruktúrny motív sa tým objavil už v počiatočných iteráciách, prehľadávanie ostatných možností bolo podobne časovo náročné. Pokiaľ je cieľom užívateľa mať iba jedno z mnohých lokálnych miním, ktoré obsahuje podobné charakteristiky ako plne optimalizované nižšie klastre, môže byť tento prístup užitočný spolu s nižšími konvergenčnými kritériami.



Obrázok 26: Porovnanie časov GOAT-EXPLORE a GenCluster pre globálnu optimalizáciu klastrov zlata, zdroj: Autor



Obrázok 27: Výsledok globálnej optimalizácie klastra Au_{64}, zdroj: Autor

ZÁVER

V práci bola spracovaná problematika globálnej optimalizácie štruktúr chemických klastrov. Táto téma bola predstavená od základov všeobecnej problematiky optimalizácie až po analógiu so Schrödingerovou rovnicou, ktorá vyjadruje účelovú funkciu tohto problému globálnej optimalizácie. Z uvedených optimalizačných algoritmov boli vybrané genetické algoritmy. Hlavným cieľom tejto časti bolo priblížiť stratégie skúmania priestoru riešení a štruktúru týchto algoritmov. V tretej, poslednej teoretickej, kapitole boli uvedené základné definície štruktúr chemických klastrov a princípy dôležité pre efektívne vykonávanie globálneho prehľadávania. Jedným z najdôležitejších poznatkov tejto kapitoly je transformácia súčasne prakticky neriešiteľnej Schrödingerovej rovnice pomocou rôznych aproximácií, ktoré umožňujú získavať aspoň približné hodnoty potenciálnej energie klastrov a vykonávať globálnu optimalizáciu pomocou algoritmov.

V druhej časti práce bol opísaný proces návrhu a tvorby programu na globálnu optimalizáciu štruktúr chemických klastrov. Tento proces začínal od návrhu riešenia, ktorý zahrňoval definovanie štruktúry vytváraného programu spolu s požadovanými funkcionálnymi a nefunkcionálnymi požiadavkami. Následne bola v ďalšej kapitole opísaná výsledná implementácia tohto programu a uvedené príklady fungovania jeho jednotlivých častí. V poslednej kapitole boli vykonané testy na overenie kvality vytvoreného programu. Program dokázal úspešne vyriešiť globálnu optimalizáciu klastrov zlata a vody. V prípade náročnejšieho problému optimalizácie uhlíkového klastra C_{60} program dosiahol len suboptimálne riešenie, ktoré pripomínalo hľadanú štruktúru globálneho minima. Je možné, že v prípade prísnejších ukončovacích podmienok a úpravou niektorých konfiguračných nastavení algoritmu by bolo možné nájsť lepšie riešenie. Hlavným problémom môže byť vysoká symetria optimálneho riešenia, ktorá je ťažko dosiahnuteľná bežnými genetickými algoritmami [37], [38]. Pri porovnaní s výsledkami globálnej optimalizácie iného programu GOAT-EXPLORE dosiahol vytvorený program výsledky rovnakej kvality v kratšom čase.

Hlavným výstupom tejto práce je genetický algoritmus, ktorý dokáže hľadať globálne minimum štruktúr chemických klastrov. Medzi hlavné výzvy pri používaní tohto algoritmu patrí správne nastavenie konfiguračných parametrov, ako sú napríklad veľkosť populácie alebo ukončovacie podmienky. Riešenie tohto problému bolo opísané v druhej kapitole, kde boli uvedené rôzne stratégie a postupy na získanie optimálnych parametrov.

BIBLIOGRAFIA

[1] M. Kitti, “History of Optimization,” 13 Apríl 2012. [Online]. Available: http://www.mitrikitti.fi/opthist.html. [Accessed 4 March 2025].

[2] M. G. Sahab, V. V. Toropov and A. H. Gandomi, “2 - A Review on Traditional and Modern Structural Optimization: Problems and Techniques,” in *Metaheuristic Applications in Structures and Infrastructures*, Oxford, Elsevier, 2013, pp. 25-47.

[3] T. Bartz-Beielstein, J. Branke, J. Mehnen and O. Mersmann, “Evolutionary Algorithms,” *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery,* vol. 4, 2014.

[4] A. Deshpande and M. Kumar, Artificial Intelligence for Big Data, Packt Publishing, 2018.

[5] S. Ronald, “Robust encodings in genetic algorithms: a survey of encoding issues,” in *Proceedings of 1997 IEEE International Conference on Evolutionary Computation (ICEC '97)*, 1997.

[6] S. Mirjalili, “Genetic Algorithm,” in *Evolutionary Algorithms and Neural Networks*, vol. 780, Cham, Springer International Publishing, 2019, pp. 43-55.

[7] S. Katoch, S. S. Chauhan and V. Kumar, “A review on genetic algorithm: past, present, and future,” *Multimedia Tools and Applications,* vol. 80, no. 5, pp. 8091-8126, 2021.

[8] H. Kour, P. Sharma and P. Abrol, “Analysis of Fitness Function in Genetic Algorithms,” *Journal of Scientific and Technical,* vol. 1, no. 3, pp. 87-90, 2015.

[9] K. Sastry, D. E. Goldberg and G. Kendall, “Genetic Algorithms,” in *Search Methodologies*, Boston, Springer, 2013.

[10] M. Burkhardt and R. V. Yampolskiy, *Death in Genetic Algorithms,* 2021.

[11] C. Li and J. Wu, *Subpopulation Diversity Based Selecting Migration Moment in Distributed Evolutionary Algorithms,* 2017.

[12] O. Kramer, "Genetic Algorithm Essentials," in *Genetic Algorithms*, Cham, Springer, 2017.

[13] M. Sipper, W. Fu, K. Ahuja and J. H. Moore, "Investigating the parameter space of evolutionary algorithms," *BioData Mining*, vol. 11, no. 2, 2018.

[14] B. Doerr and C. Doerr, "Optimal Parameter Choices Through Self-Adjustment: Applying the 1/5-th Rule in Discrete Settings," in *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, New York, 2015.

[15] J. Zhang and V. Glezakou, "Global optimization of chemical cluster structures: Methods, applications, and challenges," *International Journal of Quantum Chemistry*, vol. 121, no. 7, 2021.

[16] W. J. Hehre, A Guide to Molecular Mechanics and Quantum Chemical Calculations, Irvine: Wavefunction, Inc., 2003.

[17] S. Heiles and R. L. Johnston, "Global optimization of clusters using electronic structure methods," *International Journal of Quantum Chemistry*, vol. 113, no. 18, pp. 2091-2109, 2013.

[18] D. Larsen, Physical Chemistry, LibreTexts, 2014.

[19] M. Remko, Molekulové modelovanie, Bratislava: Slovak Academic Press, s.r.o., 2000.

[20] S. Ehlert, M. Stahn, S. Spicher and S. Grimme, "Robust and Efficient Implicit Solvation Model for Fast Semiempirical Methods," *Journal of Chemical Theory and Computation*, vol. 17, no. 7, pp. 4250-4261, 2021.

[21] M. Najafabadi, T. Khoshgoftaar, F. Villanustre and J. Holt, "Large-scale distributed L-BFGS," *Journal of Big Data*, vol. 4, no. 22, 2017.

[22] D. C. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization," *Mathematical Programming*, vol. 45, no. 1, pp. 503-528, 1989.

[23] E. Bitzek, P. Koskinen, F. Gähler, M. Moseler and P. Gumbsch, “Structural Relaxation Made Simple,” *Physical Review Letters,* vol. 97, no. 17, 2006.

[24] J. Guénolé, W. G. Nöhring, A. Vaid, F. Houllé, Z. Xie, A. Prakash and E. Bitzek, “Assessment and optimization of the fast inertial relaxation engine (fire) for energy minimization in atomistic simulations and its implementation in lammps,” *Computational Materials Science,* vol. 175, 2020.

[25] Q. Yang, G. Jiang and S. He, “Enhancing the Performance of Global Optimization of Platinum Cluster Structures by Transfer Learning in a Deep Neural Network,” *Journal of Chemical Theory and Computation,* vol. 19, no. 6, pp. 1922-1930, 2023.

[26] H. Kim, S. Ryu, N. Jung, J. Yang and C. Seok, “CSearch: chemical space search via virtual synthesis and global optimization,” *Journal of Cheminformatics,* vol. 16, no. 1, 2024.

[27] C. John and R. S. Swathi, “Global Optimization of Dinitrogen Clusters Bound to Monolayer and Bilayer Graphene: A Swarm Intelligence Approach,” *The Journal of Physical Chemistry A,* vol. 127, no. 21, pp. 4632-4642, 2023.

[28] Z. Zhang, W. Gee, H. R. Lavroff and A. N. Alexandrova, “GOCIA: a grand canonical global optimizer for clusters, interfaces, and adsorbates,” *Physical Chemistry Chemical Physics,* vol. 27, no. 2, pp. 696-706, 2025.

[29] FACCTs, “Orca - FACCTs,” 2025. [Online]. Available: https://www.faccts.de/orca/. [Accessed 6 April 2025].

[30] FACCTs, “6.4. GOAT: global geometry optimization and ensemble generator,” 2025. [Online]. Available: https://www.faccts.de/docs/orca/6.0/manual/contents/typical/GOAT.html. [Accessed 6 April 2025].

[31] Grimme group, “GFN-Force-Field (GFN-FF),” 2023. [Online]. Available: https://xtb-docs.readthedocs.io/en/latest/gfnff.html. [Accessed 7 April 2025].

[32] P. O. Dral, “MLatom AI-enhanced computational chemistry,” 2025. [Online]. Available: http://mlatom.com. [Accessed 10 April 2025].

[33] A. H. Larsen et al., “The Atomic Simulation Environment—A Python library for working with atoms,” *Journal of Physics: Condensed Matter,* vol. 29, no. 27, 2017.

[34] Q. Sun et al., “PySCF: the Python-based simulations of chemistry framework,” *WIREs Computational Molecular Science,* vol. 8, no. 1, 2018.

[35] K. V. Shopov and D. V. Markova, “Application of Hungarian Algorithm for Assignment Problem,” in *2021 International Conference on Information Technologies (InfoTech),* Varna, 2021.

[36] R. M. Shields et al., “Accurate Predictions of Water Cluster Formation, (H2O)n=2−10,” *The Journal of Physical Chemistry A,* vol. 114, no. 43, pp. 11725-11737, 2010.

[37] S. Han et al., “Unfolding the structural stability of nanoalloys via symmetry-constrained genetic algorithm and neural network potential,” *npj Computational Materials,* vol. 8, no. 1, 2022.

[38] A. Marino, “Genetic Search on Highly Symmetric Solution Spaces: Preliminary Results,” in *Theoretical Aspects of Evolutionary Computing*, Berlin, Heidelberg, Springer Berlin Heidelberg, 2001, pp. 387-407.

[39] Chegg Inc., “https://www.chegg.com/,” [Online]. Available: https://www.chegg.com. [Accessed 28 March 2025].

[40] N. C. Albanese, “Implementing the Steepest Descent Algorithm in Python from Scratch,” Towards Data Science, 20 Február 2023. [Online]. Available: https://towardsdatascience.com/implementing-the-steepest-descent-algorithm-in-python-from-scratch-d32da2906fe2/. [Accessed 1 April 2025].

[41] S. Banihashemian and F. A. Adibnia, “A Novel Robust Soft-Computed Range-Free Localization Algorithm Against Malicious Anchor Nodes,” *Cognitive Computation*, vol. 13, pp. 992–1007, 2021.

[42] A. Honkela, T. Raiko, M. Kuusela, M. Tornio and J. Karhunen, “Approximate Riemannian Conjugate Gradient Learning for Fixed-Form Variational Bayes.,” *Journal of Machine Learning Research,* vol. 11, pp. 3235-3268, 2010.

[43] A. Umam, “Newton’s Method Optimization: Derivation and How It Works,” 27 September 2017. [Online]. Available: https://ardianumam.wordpress.com/2017/09/27/newtons-method-optimization-derivation-and-how-it-works/. [Accessed 4 April 2025].

UNIVERZITA MATEJA BELA V BANSKEJ BYSTRICI
FAKULTA PRÍRODNÝCH VIED

OPTIMALIZÁCIA ŠTRUKTÚR CHEMICKÝCH KLASTROV POMOCOU SOFT COMPUTING METÓD

Používateľská príručka

Banská Bystrica, 2025 **Bc. Michal Mojžiš**

Inštalácia a spustenie aplikácie

Vytvorenú aplikáciu je možné stiahnuť na nasledovnej stránke: GenCluster. Na nainštalovanie knižníc, ktoré sú využívané v aplikácii, je potrebné mať nainštalovaný Python 3.11. Odporúčaný spôsob inštalácie je vytvorenie virtuálneho prostredia *venv*. Návod je spísaný v nasledovných bodoch:

1. Stiahnuť a rozbaliť projekt z archívu
2. Príprava virtuálneho prostredia *venv*
    a. Prejsť do priečinku, kde sa nachádza projekt (súbor main.py)
    b. Vytvoriť prostredie = "python3 -m venv .venv"
    c. Aktivovať prostredie = "source .venv/bin/activate"
    d. Nainštalovanie knižníc = "pip install -r requirements.txt"
3. Spustenie programu = "python3 main.py genetic -config <cesta>"

Následne pri budúcich spusteniach stačí aktivovať vytvorené virtuálne prostredie pomocou rovnakého príkazu a spustiť *main.py*. Aplikácia obsahuje príklady, ktoré sa nachádzajú v priečinku *data/examples*. Spustenie jedného z príkladov je možné vykonať nasledovným príkazom:

```
python3 main.py genetic -config data/examples/Au_6/au_6.yaml
```

Aplikácia taktiež podporuje spúšťanie v prostredí *Jupyter Notebook*, kde je možné vstupné a výstupné klastre vizualizovať. V prípade spustenia v tomto prostredí nie je potrebné vykonávať žiadne dodatočné kroky a aplikácia sama deteguje dostupnosť *Ipykernel*.

V prípade spúšťania aplikácie z externého kontextu je potrebné nastaviť buď absolútne cesty k súborom alebo použiť prepínač *-project_dir*, ktorý špecifikuje cestu k projektu a upraví relatívne cesty na absolútne. Ďalší podporovaný prepínač je *-work_dir*, ktorý umožňuje dynamicky nastaviť adresár, v ktorom sa bude počítať a do ktorého budú uložené výsledné súbory. Tento prepínač má prednosť pred definovaným pracovným adresárom vo vstupnom konfiguračnom súbore.

Konfiguračné súbory

Aplikácia využíva vstupné *yaml* konfiguračné súbory, ako jediný povinný vstup algoritmu. Pri spustení sa zadáva pomocou prepínača *-config* spolu s cestou k súboru. Tieto súbory obsahujú všetky potrebné konfigurácie vstupného problému, paralelizácie, výberu modelu a genetických operátorov. Vytvorený konfiguračný súbor je možné jednoducho uložiť na neskoršie použitie alebo zdieľanie. Dokumentácia ku každému nastaveniu sa nachádza v súbore *data/examples/example_config.yaml*.

Konfiguračný súbor je rozdelený do 9 hlavných skupín. Každá skupina konfiguruje inú časť algoritmu a spolu s ich popisom sú uvedené v nasledovnom zozname:

- *problem* – nastavuje všeobecné parametre algoritmu a vstupu
- *population* – konfiguruje populáciu genetického algoritmu
- *optimizer* – konfiguruje model a parametre optimalizácie
- *evaluator* – konfiguruje model pre výpočet potenciálnej energie
- *generator* – definuje spôsob generovania klastrov
- *selector* – slúži na výber operátora selekcie a jeho parametrov
- *crossover* – slúži na výber operátora kríženia a jeho parametrov
- *mutators* – slúži na výber skupiny operátorov mutácie a ich parametrov
- *terminators* – definuje ukončovacie podmienky algoritmu

Dokumentácia skupiny *problem* je zobrazená na nasledovnej strane. Každé nastavenie (vyznačené oranžovou farbou) má pri sebe opis jeho funkcie (komentár vyznačený zelenou farbou) a značku (vyznačenú červenou farbou), ktorá vyjadruje, či je dané nastavenia povinné nastaviť. V prípade nenastavenia nepovinných nastavení sa buď táto hodnota inicializuje predvolenou hodnotou alebo sa daný proces neuplatní počas behu algoritmu. V súbore s dokumentáciou je správanie každého nastavenia opísané podrobnejšie. Príklady reálnych konfiguračných súborov je možné nájsť v *data/examples*.

Aplikácia obsahuje validáciu konfiguračných súborov, čím sa predchádza problémom pri ich tvorbe. V prípade, že súbor nie je validný, je na začiatku algoritmu program ukončený a používateľ dostáva hlášku, kde nastala chyba.

```yaml
# Specifies general problem parameters
problem:
  # Path to the input molecule (.xyz) [Required]
  input_molecule_path: data/examples/Au_8/molecule.xyz
  # Specifies the cluster's charge [Required]
  molecule_charge: 0
  # Specifies the cluster's multiplicity [Required]
  molecule_multiplicity: 1
  # The number of input molecules inside a single cluster [Required]
  cluster_size: 8
  # Directory which is used to store results and temp files [Optional]
  working_directory: data/out
  # Creates a checkpoint after every n-th generation [Optional]
  checkpoint_every_nth_generation: 2
  # Defines the maximum number of output clusters [Required]
  leaderboard_size: 10
  # Defines the minimum RMSD threshold for saving clusters in the leaderboard [Optional]
  leaderboard_rmsd_threshold: 1
  # Defines the internuclear bond distances in the cluster [Required]
  internuclear_angstrom_distances:
    'Au-Au': 2.5, 3  # Defines the minimum and maximum Angstrom distances between two Au atoms
    'Au-*':   2.4, 3  # Wildcards with * are supported
    '*-*':      2.3, 3  # The distance is chosen in order from the most specific to the most general one
  # Specifies a list of RMSD methods to be used for calculations [Optional]
  rmsd_methods:
    - eigen
  # Specifies a list of alignment methods to be used for calculations [Optional]
  alignment_methods:
    - rdkit
    - hungarian_rdkit
    - inertia_hungarian_rdkit
```

UNIVERZITA MATEJA BELA V BANSKEJ BYSTRICI
FAKULTA PRÍRODNÝCH VIED

OPTIMALIZÁCIA ŠTRUKTÚR CHEMICKÝCH KLASTROV POMOCOU SOFT COMPUTING METÓD

Systémová dokumentácia

Banská Bystrica, 2025 **Bc. Michal Mojžiš**

Zdrojový kód

Zdrojový kód aplikácie je voľne dostupný na adrese:

https://github.com/misko3001/GenCluster

Knižnice aplikácie

V aplikácii sa využíva viacero knižníc, ktoré je potrebné nainštalovať pre spustenie programu. V hlavnom adresári projektu sa nachádza súbor *requirements.txt*, ktorý obsahuje všetky hlavné knižnice. Pomocou príkazu *pip install -r requirements.txt* je možné nainštalovať všetky potrebné knižnice a ich závislosti. Zoznam použitých knižníc:

```
ase==3.24.0
fortranformat==2.0.0
geometric==1.0.2
h5py==3.13.0
ipykernel==6.29.5
joblib==1.4.2
matplotlib==3.10.1
MDAnalysis==2.8.0
mdapackmol==0.1.0
mlatom==3.16.2
networkx==3.4.2
numpy==1.26.4
pandas==2.2.3
prettytable==3.14.0
py3Dmol==2.4.2
pydantic==2.10.6
pyscf==2.7.0
PyYAML==6.0.2
rdkit==2024.3.2
rmsd==1.5.1
scikit-learn==1.5.2
scipy==1.15.2
sgdml==1.0.2
tensorflow==2.18.0
watchdog==6.0.0
xgboost==2.1.2
```

Integrácia s MLatom

Na výpočet energie a vykonávanie lokálnych optimalizácií sa primárne využíva knižnica *MLatom*. Jednou z funkcionalít tejto knižnice sú rozhrania pre chemické výpočtové softvéry, ktoré sa využívajú v aplikácii. Podporované externé programy sú *XTB, ORCA* a knižnica *PySCF* napísaná priamo v Pythone. Volania týchto programov sú implementované v triede *GeometryOptimizer*.

Pre využívanie týchto dvoch externých programov je potrebné ich nainštalovať na zariadenie, kde beží program *GenCluster*. Po inštalácií týchto programov je následne potrebné definovať systémovú premennú, ktorá obsahuje cestu k adresáru daného programu. Tieto premenné je možné definovať príkazmi:

- ORCA: export orcabin="/<cesta-k-ORCA>/orca"
- XTB: export xtb="/<cesta-k-XTB>/bin/xtb"

Pridávanie nových konfigurácií

Do konfiguračného súboru je možné pridávať nové konfigurácie pomocou nasledovného spôsobu (príklad pre *population*):

1. V súbore *core/ConfigParser.py* sa nachádzajú triedy, ktoré zodpovedajú názvom nastavení v konfiguračnom *yaml* súbore.
2. Pre pridanie novej konfigurácie pre *population* je potrebné nájsť triedu *PopulationConfig* a pridať do nej novú premennú, ktorá bude zodpovedať názvu novej konfigurácie.
3. Na komplexnú kontrolu vstupných hodnôt je možné vytvoriť funkciu s anotáciou *@field_validator('<názov-premennej>', mode='before/after')*.
4. Rozšírenie existujúcej dokumentácie o pridanú premennú.
5. Nová premenná je následne dostupná v súbore (a triede) *core/EvolutionContext.py* a je možné k nej pristupovať ako *config.population.<názov-premennej>*.

Trieda *EvolutionContext* inicializuje základnú triedu *EvolutionEngine*, ktorá slúži na riadenie genetického algoritmu. V tejto triede je možné nastavovať všetky časti genetického algoritmu, ako napríklad použité genetické operátory. Vzor triedy *PopulationConfig* je zobrazený v nasledovnom bloku:

```
class PopulationConfig(BaseModel):
    """
    Represents a configuration for managing the population, including size and optional
    attributes such as maximum age.

    :ivar size: The size of the population. Must be greater than or equal to 2. [Required]
    :type size: int
    :ivar max_age: The maximum age of invididuals in the population.
                   If defined, it must be greater than or equal to 1. [Optional]
    :type max_age: Optional[int]
    :ivar initial_guess_path: Path to the initial guesses file for initial population [Optional]
    :type initial_guess_path: Optional[str]
    """
    size: int = Field(ge=2)
    max_age: Optional[int] = None
    initial_guess_path: Optional[str] = None

    @field_validator('max_age', mode='after')
    def check_optional_max_age(cls, value: Optional[int]) -> Optional[int]:
        if value is not None and value < 1:
            raise ValueError('max_age must be greater than 0')
        return value

    @field_validator('initial_guess_path', mode='after')
    def check_optional_initial_guess_path(cls, value: Optional[str]) -> Optional[str]:
        if value is not None and is_absolute_path(value) and not os.path.isfile(value):
            raise ValueError('initial_guess_path does not point to a valid file')
        return value
```

Pridávanie nových operátorov, generátorov a ukončovacích podmienok

Vytváranie nových genetických operátorov, generátorov a ukončovacích podmienok predstavuje celkom jednoduchý proces. Na začiatok je napríklad pre pridanie nového operátora selekcie potrebné vytvoriť novú triedu pomocou rozšírenia základnej triedy *Selector* v súbore *optimization/genetic/operations/Selector.py*. Táto základná trieda obsahuje jednu funkciu *select*, ktorú implementuje používateľ. Aplikácia automaticky deteguje novú triedu v súbore a je možné ju podľa názvu triedy definovať vo vstupnom konfiguračnom súbore. Vzor triedy pre operátor selekcie je:

```
class Selector:
    rmsd_select_limit: int = 2

    def select(self, population: List[molecule], n_ind: int) -> List[molecule]:
        raise NotImplementedError

    @staticmethod
    def check_unique_constraint(population: List[molecule], requested: int) -> None:
        if requested > len(population):
            raise ValueError(f'Cant select {requested} unique molecules from population of
                             {len(population)} molecules')

    def check_rmsd_constraint(self, requested: int) -> None:
        if requested != self.rmsd_select_limit:
            raise ValueError(f'Rmsd selection can only be performed for
                             {self.rmsd_select_limit} molecules')
```

V prípade potreby dodatočných premenných pre vytvárané rozšírenie je potrebné vytvoriť nové premenné v konfiguračnom súbore podľa rovnakého postupu, ktorý je uvedený v predošlej kapitole. Ako predlohu implementácie je možné využiť existujúce operátory. Vytvorenú triedu je následne potrebné inicializovať s požadovanými premennými v triede *EvolutionContext*.

Ostatné základné triedy (*Crossover*, *Mutator* a *Terminator*) sa nachádzajú v priečinku *optimization/genetic/operations*. Základná trieda *MoleculeGenerator* sa nachádza v *optimization/genetic/molecule/MoleculeGenerator.py*. Postupy na vytvorenie nových implementácií týchto tried sú obdobné s postupom na vytvorenie nového operátora selekcie.

Pridávanie nových RMSD metód

RMSD a *alignment* metódy sa štandardne nachádzajú v súbore *optimization/genetic/molecule/MoleculeUtils.py*. Nové metódy je možné pridávať pomocou nasledovného postupu:

1. Vytvorenie a implementovanie novej funkcie v *MoleculeUtils.py*.
2. Pridanie nového identifikátora vytvorenej funkcie do premennej *AlignmentMethods* resp. *RMSDMethods* v rovnakom súbore.
3. Pridanie prípadu pre nový identifikátor v triede *MoleculeIndividual* vo funkcii *__align* resp. *__calculate_rmsd*.
4. Vytvorený identifikátor je následne možné použiť vo vstupnom konfiguračnom súbore v *problem.alignment_methods* resp. *problem.rmsd_methods*.