Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Implementace algoritmů pro aproximaci potenciálů atomových vazeb pomocí výpočtů kvantové mechaniky

Plzeň 2020 Bc. Vít Mazín

Místo této strany bude zadání práce.

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 20. května 2020

Bc. Vít Mazín

Poděkování

Na tomto místě bych rád poděkoval panu Ing. Janu Očenáškovi Ph.D. za cenné rady, věcné připomínky, ochotu a trpělivost, které mi pomohly tuto práci realizovat.

Abstract

In physics and also in majority fields of science, computational work is a fundamental complement to both experiments and theory, and currently a majority of both experimental and theoretical studies involve some numerical calculations, simulations or computer modeling. Therefore, the use of modern algorithms that allow more efficient calculations is of great benefit. This work is dedicated to the application of optimization algorithms in the field of theoretical physics. The aim of this work is to implement algorithms for optimization of atomic bond potential parameters. These energy potentials are used for computer simulations using molecular dynamic method. The parameters of these potentials will be optimized in respect to the results predicted by quantum mechanics. The work will not focus on the actual calculations using the molecular dynamics and quantum mechanics methods. Program codes for these calculations will be provided.

Abstrakt

Ve fyzice i většině dalších vědních oborů je výpočetní práce důležitým doplňkem k experimentům i teorii a v současnosti převážná většina experimentálních i teoretických prací zahrnuje některé numerické výpočty, simulace nebo počítačové modelování. Proto je použití moderních algoritmů dovolujících efektivnější výpočty velkým přínosem. Tato práce je zaměřena na využití optimalizačních algoritmů v oblasti teoretické fyziky. Cílem práce je implementovat algoritmy pro optimalizaci parametrů potenciálů atomových vazeb. Tyto energetické potenciály jsou využívány pro počítačové simulace metodou molekulární dynamiky a jejich parametry budou optimalizovány vzhledem k výsledkům, které předpovídá kvantová mechanika. Práce nebude zaměřena na samotné výpočty metodou molekulární dynamiky a kvantové mechaniky. Programy pro tyto výpočty budou poskytnuty.

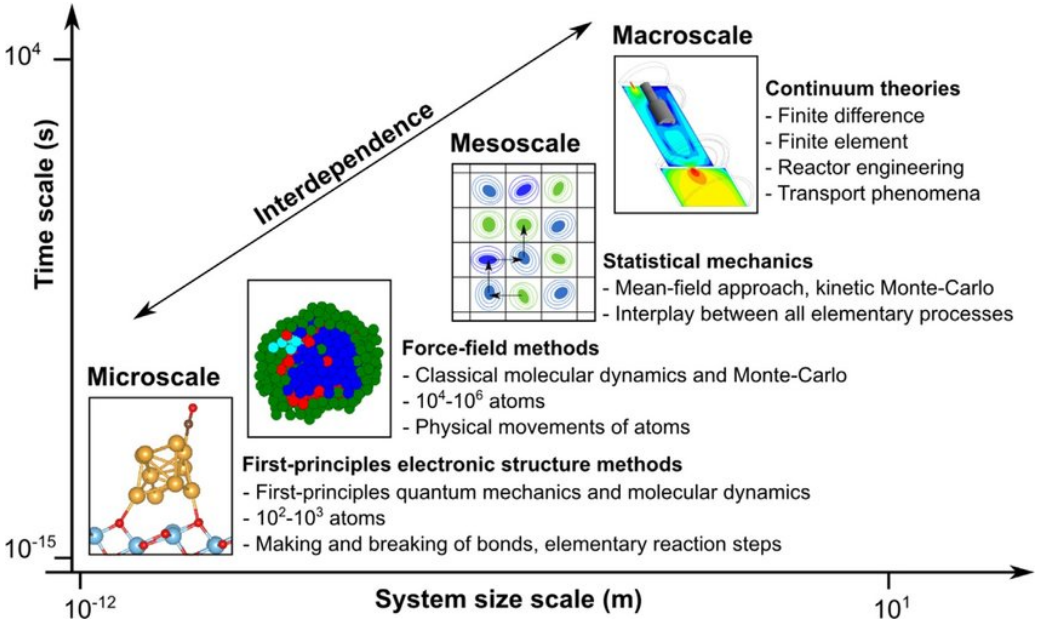
Obsah

1 Úvod

Při hledání nových materiálů s požadovanými vlastnostmi se využívají teoretické výpočty, které dokáží na základě prvkového složení a struktury předpovídat mnoho vlastností materiálu, například elektrické, magnetické, optické nebo mechanické. Pro takovéto výpočty lze použít teorii kvantové mechaniky (více viz kapitola 2.2), ta poskytuje v mnoha ohledech velmi přesné výpočty, ale zároveň je výpočetně velmi náročná. V současné době je limit počtu atomů ve výpočtech řádově 100. Další metodou je metoda molekulární dynamiky (více viz kapitola 2.1), která pomocí jistých zjednodušení dovoluje počítat mnohem větší atomové struktury a studovat jejich vlastnosti, výpočetní limit je díky zjednodušením přibližně 10^9 atomů.

Pro provedení výpočtů metodou molekulární dynamiky je ovšem potřeba znát potenciály atomových vazeb materiálů. Atomový potenciál nám říká, jaké síly působí mezi každou dvojicí atomů v systému. Síla definována, jako derivace potenciálu podle vzdálenosti atomů. Používaných potenciálů je velké množství, ale všechny mají společné to, že je třeba znát jejich parametry. Pouze se správnými parametry lze provést výpočet tak, aby jeho výsledky byly relevantní pro řešení daného problému. Najít tyto parametry analyticky je ovšem většinou příliš náročné, či dokonce nemožné. Jedním ze způsobů, jak nalézt přesnější parametry pro metodu molekulární dynamiky, je využít výsledků z kvantové mechaniky. Tato práce bude se bude tedy pohybovat mezi metodami kvantové mechaniky a molekulární dynamiky (viz 1.1, obrázek převzat z [7]).

Cílem této práce je vytvořit obecnou metodu pro aproximaci parametrů atomových potenciálů. Implementací vytvořené metody vznikne vhodné programové vybavení pro pracovníky výzkumného ústavu NTC Západočeské univerzity, které bude založeno na optimalizaci parametrů potenciálů vzhledem k výsledkům, jež předpovídá kvantová mechanika. Důraz bude kladen na výsledné snadné použití vytvořeného programového vybavení.



Obrázek 1.1: Volba metody v závislosti na měřítku časového úseku (Time scale) a měřítku velikosti systému (System size scale), tato práce se pohybuje mezi metodami kvantové mechaniky a molekulární dynamiky, obrázek převzat z [7]

Proces identifikace parametrů potenciálu meziatomové interakce probíhá následujícím způsobem. Nejprve je provedena sada výpočtů metodou QM, která určuje energii systému v mnoha různých konfiguracích atomů. Dále se zvolí konkrétní typ potenciálu pro MD, čímž jsou dány i neznámé parametry, které je potřeba identifikovat. (Fyzikální pozadí práce, metoda QM a metoda MD a MD potenciály jsou stručně popsána v kapitole 2)

Následuje volba metody v závislosti na měřítku časového úseku (Time scale) a měřítku velikosti systému (System size scale). Tato práce se pohybuje mezi metodami kvantové mechaniky a molekulární dynamiky, obrázek převzat z [7].

Technické požadavky na výpočty MD a podmínky výpočetního prostředí jsou popsány v kapitole 4.

Popis programového řešení je v kapitole 5.

Výsledky, hodnocení výsledků a zhodnocení metody je v kapitole 6 a v závěru práce.

2 Výpočty molekulární dynamiky a kvantové mechaniky

2.1 Molekulární dynamika

V molekulární dynamice (dále MD) je pohyb atomů dán Newtonovými pohybovými zákony. Silové interakce mezi blízkými atomy jsou pak určeny meziatomovými interakčními potenciály. Základem MD je tedy časová integrace Newtonových pohybových rovnic.

Výsledkem jsou potom souřadnice a rychlosti všech simulovaných částic (atomů). Z nich se usuzují další vlastnosti systému. Například teplota, množství defektů v krystalické mřížce a podobně.

Nedílnou součástí MD jsou i další procedury, které mají vztah ke statistické mechanice. Jsou to například numerické termostaty, které ovlivňují statistické rozložení energií atomů tak, aby odpovídalo stavu při teplotní rovnováze.

Pomocí simulací MD lze studovat na atomové úrovni procesy jako jsou krystalizace, tání, laserové odpařování povrchu materiálu, bombardování vysokoenergetickými ionty, iniciace defektů, teplotní vodivost, ve složitějších případech i chemické reakce, nebo relace atomových jader.

Tato metoda má kořeny v teoretické fyzice v padesátých letech minulého století, ale je aplikována dodnes především na poli chemické fyziky, zkoumání materiálů a biomolekulárního modelování.

Protože jsou molekulární systémy obvykle složeny z velkého množství částic, je nemožné určit vlastnosti těchto systémů analyticky. Molekulárně dynamické simulace obcházejí tento problém použitím numerických metod. Nicméně takovéto simulace jsou matematicky špatně podmíněné a obsahují velkou kumulativní chybu v numerické integraci, kterou lze zmenšit výběrem správného algoritmu a parametrů, ale nelze ji zcela odstranit.

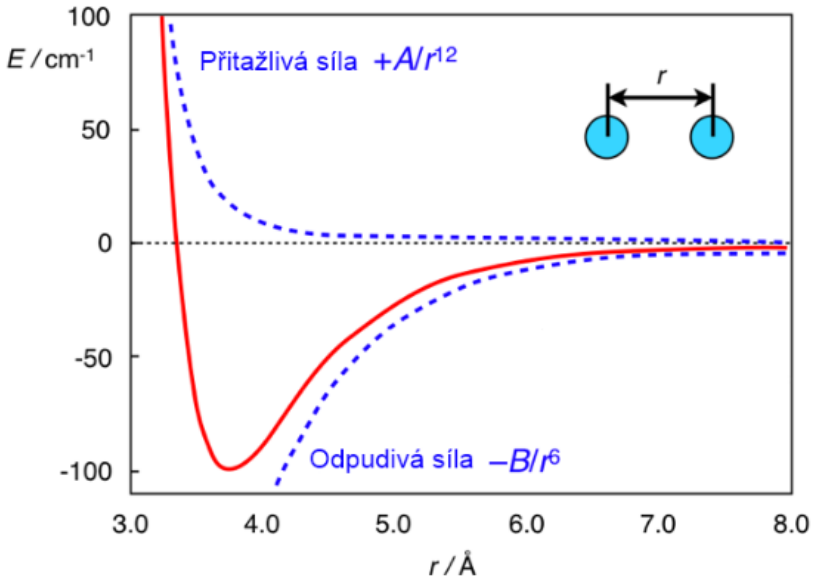
V molekulárně dynamických simulacích je nutné definovat potenciálovou funkci. Touto funkcí je definováno, jak spolu částice v dané simulaci budou interagovat. Potenciály mohou být definované s různými stupni přesnosti.

2.1.1 Lennard-Jonesův potenciál

Lennard-Jonesův potenciál (známý také jako 12-6 potenciál) je matematicky jednoduchý model, který vyjadřuje energii interakce dvou atomů v závislosti na vzdálenosti jejich středů. Potenciál lze popsat následujícím vztahem:

$$\phi_{LJ}(r) = \frac{A}{r^{12}} - \frac{B}{r^{6}} \tag{2.1}$$

Parametry A a B ve vztahu 2.1 jsou závislé na daném materiálu a lze je aproximovat dle výsledků poskytnutých přesnými výpočty např. z kvantové mechaniky. Celková energie je tedy rovna rozdílu překryvové přitažlivosti a odpudivosti. Tento vztah je znázorněn na obrázku 2.1 (obrázek převzat z [3]).



Obrázek 2.1: Hodnota energie v závislosti na vzdálenosti dvou atomů určena Lennard-Jonesovým potenciálem, obrázek převzat z [3]

Poté, co je znám vztah pro výpočet energie mezi dvěma molekulami, lze celkovou energii systému vypočítat jako součet energií mezi všemi atomy:

$$V = \sum_{i} \sum_{j} \phi_{LJ}(r_{ij}) \tag{2.2}$$

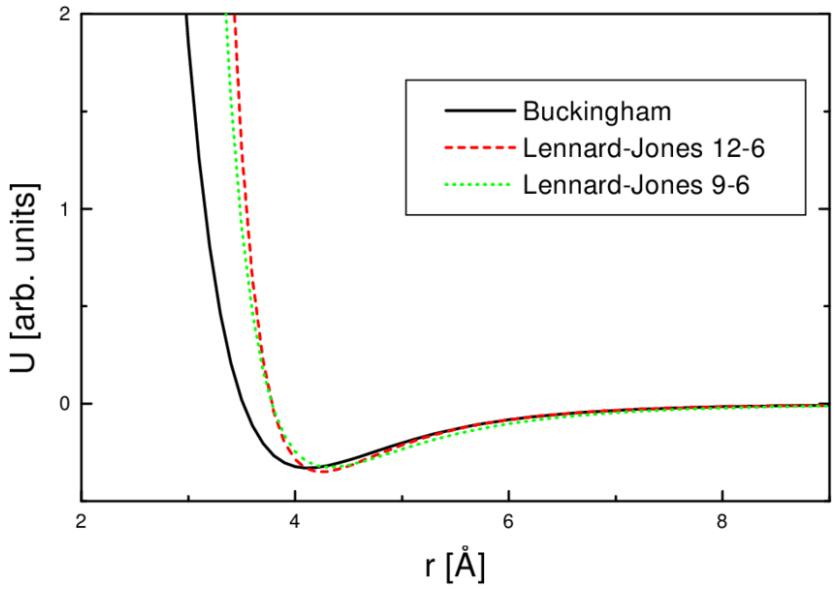
Existuje také tvar 9-6, který se od uvedeného tvaru 12-6 liší pouze mocninami vzdáleností středů atomů. Záleží na jaké vzdálenosti jsou zkoumány vlastnosti atomů v systému. Při výpočtech v této práci bude použit pouze tvar 12-6.

2.1.2 Buckinghamův potenciál

Buckinghamův potenciál je matematicky jednoduchý model, který vyjadřuje energii interakce dvou atomů v závislosti na vzdálenosti jejich středů. Tento potenciál byl navržen Richardem Buckinghamem jako zjednodušení Lennard-Jonesova potenciálu. Potenciál lze popsat následujícím vztahem:

$$\phi_B = A \exp(-Br) - \frac{C}{r^6} \tag{2.3}$$

Parametry A, B a C ve vztahu 2.3 jsou závislé na daném materiálu a lze je taktéž aproximovat dle výsledků z přesných výpočtů např. z kvantové mechaniky. Celková energie je rovna rozdílu přitažlivosti a odpudivosti. Potenciál je znázorněn na obrázku 2.2 (obrázek převzat z [8]) v kontrastu s variacemi Lennard-Jonesova potenciálu.



Obrázek 2.2: Hodnota energie v závislosti na vzdálenosti dvou atomů určena Buckingahmovým potenciálem v kontrastu s variacemi Lennard-Jonesova potenciálu, obrázek převzat z [8]

Poté, co je znám vztah pro výpočet energie mezi dvěma molekulami, lze celkovou energii systému vypočítat jako součet energií mezi všemi atomy:

$$V = \sum_i \sum_j \phi_B(r_{ij}) \tag{2.4}$$

2.1.3 LAMMPS

Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS) je software pro molekulární dynamiku. Tento program je vyvíjen v laboratořích Sandia National Laboratories [1] pod svobodnou licencí GNU GPL [1] a jeho zdrojový kód je umístěn na GitHub [1]. Program je napsán v jazyce C++.

LAMMPS je zaměřen na rychlý výpočet molekulární dynamiky a může provádět výpočty ve dvou módech. V prvním běží veškerý výpočet na jednom procesoru a v tom druhém běží výpočty paralelně na více procesorech. Paralelismu je dosaženo použitím MPI a prostorového rozkladu simulační domény. LAMMPS modeluje např. polymerické, biologické, pevné a další systémy, které jsou složeny z částic v kapalném, pevném nebo plynném skupenství.

Pro řešení Newtonových pohybových rovnic je nutno znát, jaké síly jsou mezi atomy zkoumaného materiálu. V programu LAMMPS je možno využít mnoho předdefinovaných potenciálů. Mezi tyto potenciály patří i Lennard-Jonesův a Buckinghamův potenciál, které budou využity při řešení zadání mé práce. Dále je možno zkoumaný systém zahřívat, stlačovat, roztahovat atd.

2.2 Kvantová mechanika

Pokud je třeba zkoumat materiály např. v excitovaném stavu, při složité chemické reakci nebo pokud je zapotřebí co nejpřesnější výpočet, je třeba použít jiných výpočetních simulačních metod. Chování částic se v takových případech dá získat z výpočtů z tzv. prvního principu za použití metod kvantové mechaniky, jako je například teorie funkcionálu hustoty (DFT). Z této teorie vycházejí ab-initio techniky. Ab-initio techniky vycházejí přímo z úrovně zavedených fyzikálních zákonů a nezakládají se na žádných předpokladech jako jsou empirické modely atd. Například ab-initio výpočet vlastností kapalné vody bude založen na vlastnostech atomů vodíku a kyslíku, zákonu elektrostatiky a kvantové mechanice. Z těchto elementárních informací budou vypočteny vlastnosti izolovaných jednotlivých molekul vody. Dále budou následovat výpočty interakce větších a větších molekul, až bude dosaženo výsledku pro požadované množství tohoto materiálu.

Takovéto výpočty jsou však velmi složité a velmi náročné na výpočetní výkon z mnoha důvodů. Lze je tedy použít na výpočty vlastností menších systémů, než je možné s použitím molekulární dynamiky.

Pro energie stanovené pomocí QM platí ještě jedna zvláštnost. Absolutní hodnota stanovené energie není platná, platné jsou pouze rozdíly energií

mezi jednotlivými konfiguracemi. Důsledkem toho je, že energie stanovené QM mohou mít od námi aproximovaných hodnot libovolný konstantní posun. Znalost absolutní hodnoty energie není pro výpočty MD žádným omezením, protože v matematickém modelu se veličina této energie vyskytuje pouze ve tvaru derivace.

2.2.1 VASP

Program VASP (Vienna Ab initio Simulation Package) je výpočetní program pro modelování materiálů na atomové úrovni, jako jsou např. výpočty elektronové struktury a kvantově mechanická molekulární dynamika z prvních principů. Základní metodologií je teorie funkcionálu hustoty (DFT), ale program umožňuje řešit výpočty založené na dalších teoriích a metodách.

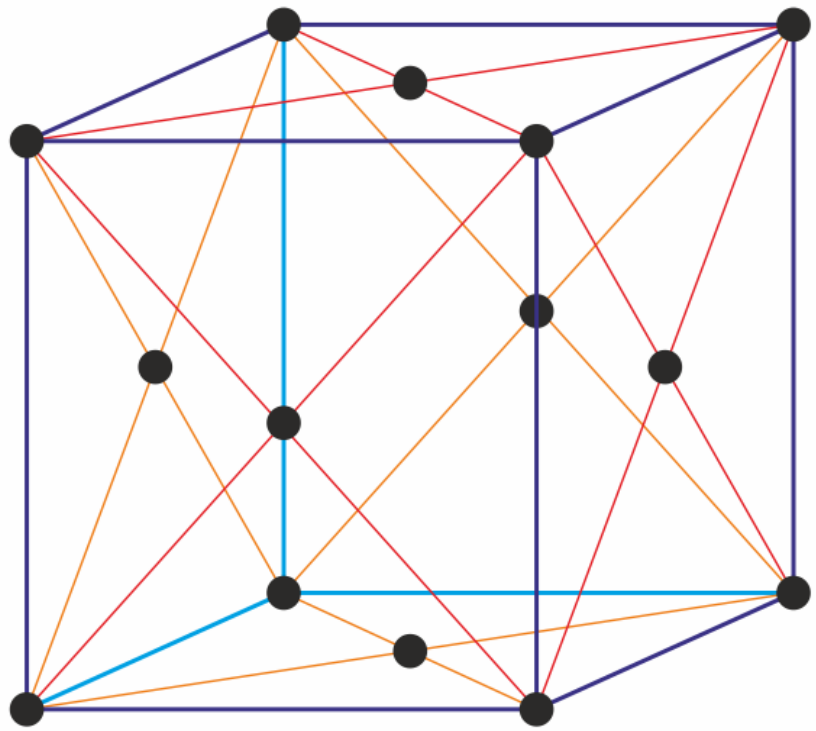
Program je vyvíjen na Vídeňské univerzitě pod proprietární licencí. V této práci budou použity výsledky v něm modelovaných systémů pro aproximaci parametrů pro výpočty molekulární dynamiky.

Program VASP nám poskytne hodnoty energie krystalu v různých konfiguracích (pro různě deformovaný stav krystalu). Naším cílem je (pro stejné konfigurace), co nejvíce se přiblížit těmto hodnotám energií pomocí funkčních potenciálů popsaných v předchozích podsekcích 2.1.1 a 2.1.2.

2.3 Modelový materiál / Krystalická struktura

Jako jeden z hlavních parametrů simulací je struktura materiálu, který bude zkoumaný. Struktura materiálu může dramaticky změnit jeho fyzikální vlastnosti (např. u uhlíku je velký rozdíl mezi grafitem a diamantem, které se liší pouze ve struktuře atomů uhlíku). Proto je třeba předem stanovit jakou strukturu bude mít zkoumaný materiál. Krystalická struktura (polohy atomů) je vstupní parametr pro výpočet molekulární dynamiky v programu LAMMPS a pro výpočet kvantové mechaniky v programu VASP.

Jako modelový materiál bude při tvorbě této práce použit kov stříbra, jehož atomy jsou uspořádané v plošně centrované kubické mřížce (také známá jako *FCC* - face-centered cubic). Jako každá kubická krystalová mřížka, má i tato tzv. „mřížkové body“ ve všech osmi rozích elementární buňky, a navíc má další body ve středech každé stěny elementární buňky [5]. Tato atomová struktura je znázorněna na obrázku 2.3 (obrázek převzat z [5]).



Obrázek 2.3: Plošně centrovaná kubická mřížka, obrázek převzat z [5]

3 Optimalizační algoritmy

V této kapitole budou popsány vybrané optimalizační algoritmy, které budou použity pro tvorbu programového vybavení pro řešení problematiky aproximace parametrů atomových potenciálů. Vybrány byly genetické algoritmy pro jejich schopnost hledat řešení složitých problémů ve velmi velkých stavových prostorech (viz 3.1). Dále byl vybrán algoritmus Spiral Optimization Algorithm pro jeho schopnost hledat lepšího kandidáta (viz 3.2) v postupně se zmenšujícím okolí aktuálního nejlepšího kandidáta, což umožní zpřesnit výsledek genetických algoritmů.

3.1 Genetické algoritmy

Jedná se o heuristický algoritmus, který je založen na myšlenkách evoluční biologie. Genetické algoritmy jsou podmnožinou evolučních algoritmů, které se snaží napodobovat procesy vývoje v přírodě. Základem těchto algoritmů je práce s populacemi řešení, kde využívají mechanismy přirozeného vývoje, jako je např. dědičnost, křížení, mutace, kompetitivní výběr atd. [4]

Genetické algoritmy resp. evoluční algoritmy jsou založené na poznatcích plynoucích z díla *Charles Darwin: O původu druhů*, z něhož plyne, že přirozený výběr je hlavní složkou procesu vývoje organismů. GA jsou vhodné pro řešení problémů, pro které není znám deterministický algoritmus řešení nebo by byl takový algoritmus příliš složitý nebo výpočetně náročný. Zejména jsou využívány pro řešení nelineárních optimalizačních úloh a pro prohledávání rozsáhlých stavových prostorů. GA mají také tu výhodu, že pro řešení cílového problému je potřeba znát, jak ohodnotit jednotlivá řešení, ale není třeba znát podrobně celý řešený problém. Dále je výhodné, že dokážou najít řešení ve velkém stavovém prostoru a jsou odolné proti uváznutí v lokálních optimech. [4]

3.1.1 Průběh genetického algoritmu

Typický průběh genetického algoritmu lze popsat takto:

1. Inicializace prvotní populace
2. Ohodnocení prvotní populace
3. Tvorba nové populace

(a) Výběr kandidátů k vytvoření potomků

(b) Křížení

(c) Mutace

(d) Vyhodnocení nové populace

4. Kontrola zastavovací podmínky; při nesplnění se opakuje bod 3.

5. Vrácení nejlepšího jedince jakožto řešení problému

Nejprve je nutné pro řešený problém vytvořit jedince pro prvotní populaci, kterou bude GA nastartován. Prvotní populaci lze vytvořit náhodně, což k řešení také povede, ale řešení může být hledáno znatelně delší dobu. Pokud lze využít nějaké informace o úloze k vytvoření prvotní populace, je toho vhodné využít, protože jedinci vygenerovaní z nějaké znalosti o úloze mohou být již poměrně blízko k řešení úlohy.

Po vytvoření prvotní populace je třeba ohodnotit veškeré jedince v ní předem definovanou fitness funkcí. Takto ohodnocená populace již může vstoupit do GA a hledání řešení může být zahájeno.

Při tvorbě nové populace jsou vybíráni rodiče, ze kterých jsou tvořeni potomci (prvky nové populace) speciálními metodami pro selekci (viz 3.1.5). Z rodičů jsou poté tvořeni potomci pomocí křížení a následné možné mutace (viz 3.1.4). Když je vytvořena nová populace, je opět nutno ohodnotit její jedince fitness funkcí.

Po vytvoření a ohodnocení nové populace je zkontrolováno, zda bylo dosaženo podmínek pro zastavení algoritmu. Pokud ano, je vrácen nejlépe ohodnocený prvek jako řešení problému. Pokud nebylo dosaženo podmínky pro zastavení, je proces generování nové populace a jejího ohodnocení opakován.

3.1.2 Ohodnocení jedinců

Obecný princip GA je založen na tvorbě populací, které jsou tvořeny jedinci, jimž se říká chromozomy. Každý jedinec v dané populaci představuje jednoho kandidáta řešení problému. Tito jedinci jsou ohodnoceni tak, aby byla jasná jejich životaschopnost. Ohodnocovací funkci se říká fitness funkce a ohodnocení fitness. Ohodnocení si je možno zvolit si dle daného problému, ale musí představovat, jak blízko je kandidát k cíli optimalizace. Ohodnocení je metrika, tj. měla by mít vlastnosti metriky. Obvykle je nutno danou ohodnocovací funkci vymyslet a přizpůsobit danému problému, protože neexistuje žádný přístup, který by byl univerzální pro všechny problémy. Takto ohodnocení jedinci tvoří rodiče pro tvorbu jedinců následující populace. Pro

úspěšné nalezení řešení musí být zajištěn proces, který koresponduje s přirozeným výběrem, tzn. že čím silnější jedinec, tím větší je šance na to, aby stvořil s dalším jedincem nového potomka. Pokud je toto zajištěno, potomci by měli být ještě silnější, což tedy vede k nalezení lepšího řešení problému. [4]

Příklad: Pro řešení problému osmi dam by mohla být fitness funkce rovna počtu dam, které se navzájem ohrožují. Tudíž ohodnocení 0 by představovalo nejlepší řešení.

3.1.3 Kódování chromozomů

Jedinci v populaci musejí být pro potřeby algoritmu nějakým způsobem zakódovaní. Chromozom je vlastně uspořádaná množina hodnot, kterou je třeba zakódovat. Běžně jsou používané tyto způsoby kódování:

- Binární kódování
- Permutační kódování
- Kódování hodnotou
- Kódování stromem

Binární kódování

Jedná se o nejjednodušší způsob zakódování informace. Každý gen chromozomu může nabývat hodnotu nula, nebo jedna. Binární hodnota na daném místě značí, že daná informace v chromozomu (potencionálním řešení problému) je, nebo není přítomná. Takovýto chromozom lze znázornit takto:

$$(0\,0\,1\,1\,0\,1\,0\,0\,1\,0\,1\,1)$$

Tímto způsobem lze například řešit známý *problém batohu*, který je NP-úplný. Každá binární hodnota odpovídá jednomu předmětu, který by mohl být do batohu zabalen, a předměty jsou vybírány tak, aby měli co největší hodnotu a zároveň nebyli těžší, než co batoh unese.

Permutační kódování

V tomto kódování hraje hlavní roli to, na jaké pozici v chromozomu je daná hodnota. Obvykle je tato hodnota nějaké identifikační číslo, které představuje nějaký objekt, a pro řešení je důležité to, v jakém pořadí jsou tyto objekty uspořádány. Chromozom permutačního kódování vypadá takto:

$$(1\,5\,9\,3\,6\,2\,4\,8\,7)$$

Permutačním kódováním lze například řešit další NP-úplný problém známý jako problém obchodního cestujícího. Je nutné obchodnímu cestujícímu naplánovat takovou cestu, která bude nejkratší. K dispozici je tedy seznam měst a vzdálenost mezi nimi. Každá hodnota v chromozomu identifikuje město a pořadí této hodnoty značí, v jakém pořadí budou města navštívena. Při tvorbě potomků je nutno zajistit, aby nevznikaly chromozomy, které kódují nemožné řešení (např. města by spolu nesousedila).

Kódování hodnotou

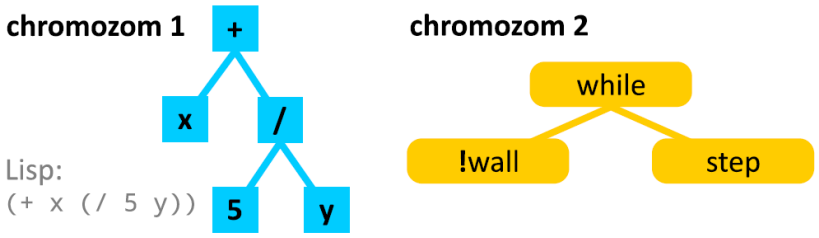
Kódování hodnotou je určeno pro řešení specifických problémů. V chromozomu jsou zakódované parametry řešeného problému. Pro specifické problémy je obvykle nutné definovat speciální metody mutace a křížení (viz 3.1.4). Tímto kódováním lze například řešit problém minimalizace nějaké funkce, pro kterou by bylo analytické řešení příliš složité, nebo dokonce nemožné. Takovýto chromozom by mohl vypadat následovně:

(0.5207 2.2874 3.415 1.9568)

V takovémto kódování je nejdůležitější hodnota, jež představuje obvykle nějaký parametr řešeného problému.

Kódování stromem

Tento druh kódování je speciálním případem kódování, které se využívá např. v genetickém programování ke generování programů atd. Každý chromozom zde představuje strom objektů jako např. na obrázku 3.1 (obrázek převzat z [4]).



Obrázek 3.1: Kódování chromozomu stromem, obrázek převzat z [4]

3.1.4 Křížení a mutace

K tomu, aby se mohli jedinci z populace vyvíjet, a tím mohli vznikat lepší kandidáti k řešení daného problému, je potřeba definovat procesy křížení a

mutace. Tyto procesy velmi napodobují přírodní procesy, díky nimž vznikají noví jedinci. Křížení probíhá s určitou pravděpodobností (že bude potomek vytvořen). Mutace probíhají také, ale pouze s menší pravděpodobností, aby nebyla úloha degradována na pouhé vytváření náhodných řešení.

Křížení

Procesem křížení vznikají nový jedinci tak, že část nesené informace každého rodiče se přenese do jejich potomka. Obvykle je prováděno křížení se dvěma rodiči, jako je to v přírodě, ale není to pravidlem. Pro tvorbu nového potomka lze použít i více než dva jedince. Toto záleží na daném problému, který je řešen. Také to znamená, že křížení bude složitější.

Nejjednodušší případ křížení je ten, že se vybere bod překřížení (může být náhodný i pevně daný - záleží na řešeném problému), a do potomka se zkopíruje první část informace před tímto bodem od prvního rodiče a druhá část za bodem překřížení od druhého rodiče. Tímto křížením nemusí vzniknout jen jeden potomek, ale mohou vzniknout dva. Do druhého se zkopírují opačné části každého rodiče. Toto lze znázornit následovně:

$$(A\ D\ C\ E \mid D\ F)$$

$$(D\ F\ A\ C \mid A\ B)$$

$$\Downarrow$$

$$(A\ D\ C\ E \mid A\ B)$$

$$(D\ F\ A\ C \mid D\ F)$$

Bodů překřížení může být také více než jeden. Specifické úlohy, jež využívají např. kódování hodnotou, nemusí takovéto křížení používat vůbec, ale může být zavedeno speciální, které plně závisí na dané úloze.

Mutace

Stejně tak jako v přírodě s malou pravděpodobností dojde k mutaci, tak k ní dochází také v GA. V GA je mutace zavedena, aby se úloha neustálila v lokálním optimu, ale aby se postupně došlo ke globálnímu optimu. Obvykle je mutována jedna nesená informace, ale v závislosti na úloze jich může být více. V jedincích, kteří jsou kódovaní binárním kódováním, lze jako mutaci provést např. inverzi náhodného bitu. U jedinců kódovaných hodnotou lze např. tuto hodnotu lehce změnit.

$$(0\,1\,0\,0\,1\,1\,1\,0) \Rightarrow (0\,1\,0\,0\,1\,0\,1\,0)$$

$$(0.1\ 2.5\ 1.2\ 0.6\ 3.0) \Rightarrow (0.1\ 2.7\ 1.2\ 0.6\ 3.0)$$

3.1.5 Selekce jedinců

Aby mohla vzniknout nová populace, je třeba vytvořit potomky jedinců ze stávající populace. Aby bylo možné vybrat rodiče, kteří vytvoří nového potomka, musí být nadefinována metoda selekce. Metody mohou být navrženy s ohledem na řešený problém a nějaké mohou pro daný specifický problém fungovat lépe než ostatní. Existuje také několik základních univerzálních metod:

- Náhodný výběr
- Výběr ruletovým kolem
- Turnajový výběr
- Žebříčkový výběr

Náhodný výběr

Tato metoda selekce je velmi jednoduchá na implementaci a vybere různé rodiče pro tvorbu potomků. Její nevýhoda je v tom, že nebere ohled na ohodnocení daných jedinců. Existuje pravděpodobnost, že ze dvou špatných jedinců vznikne dobrý potomek, ale ta je daleko menší, než že vznikne lepší potomek ze dvou dobrých jedinců.

Výběr ruletovým kolem

Tento výběr silně zvýhodňuje dobře ohodnocené jedince. Součet fitness hodnot všech jedinců si lze představit jako ruletové kolo. Každý jedinec má na tomto kole výseč, která je proporčně rovna velikosti jeho ohodnocení. Po zjištění součtu všech hodnot je vygenerováno náhodné číslo od nuly do hodnoty celkového součtu. Toto číslo si lze představit jako číslo, na kterém se zastaví kulička na ruletovém kole. Po vytvoření této zastavovací podmínky jsou jedinci procházeni a je sčítáno jejich ohodnocení. Jedinec, u kterého je tento součet roven nebo vyšší tomu, kde se kulička zastavila, je vybraný jedinec. Pokud má několik jedinců daleko vyšší ohodnocení, jsou ty s menším znevýhodněni. Tato metoda se dá dále upravovat a vylepšovat.

Turnajový výběr

Turnajový výběr funguje tak, že je náhodně vybráno n jedinců z populace. Mezi těmito jedinci je „uspořádán turnaj“ a ten nejlepší vyhraje. Nejlepší jedinec je ten, který má nejlepší ohodnocení fitness funkcí. Tato metoda selekce také zvýhodňuje lépe ohodnocené jedince.

Žebříčkový výběr

Tento výběr je vlastně vylepšení výběru ruletového kola. Chromozomy jsou seřazené podle své životaschopnosti od nejhoršího po nejlepší. Nejhoršímu se dává váha 1, druhému 2, atd. Nejlepší bude mít váhu N. Součet vah pak představuje 100% (celé ruletové kolo) a každý chromozom pak zabírá plochu odpovídající poměru své váhy k součtu všech vah [4]. Kdyby nebyly takto přidělené váhy jedincům a byla by použita čistě hodnota ohodnocení, mohlo by se stát, že by dva jedinci, kteří mají oproti ostatním skvělé ohodnocení, byli vybíráni stále znovu, protože by zabírali většinu ruletového kola, a byla by jen malá šance vybrat někoho dalšího. Tento výběr tedy dává šanci k vybrání i horším jedincům, kteří ale mohou vytvořit skvělého nového jedince.

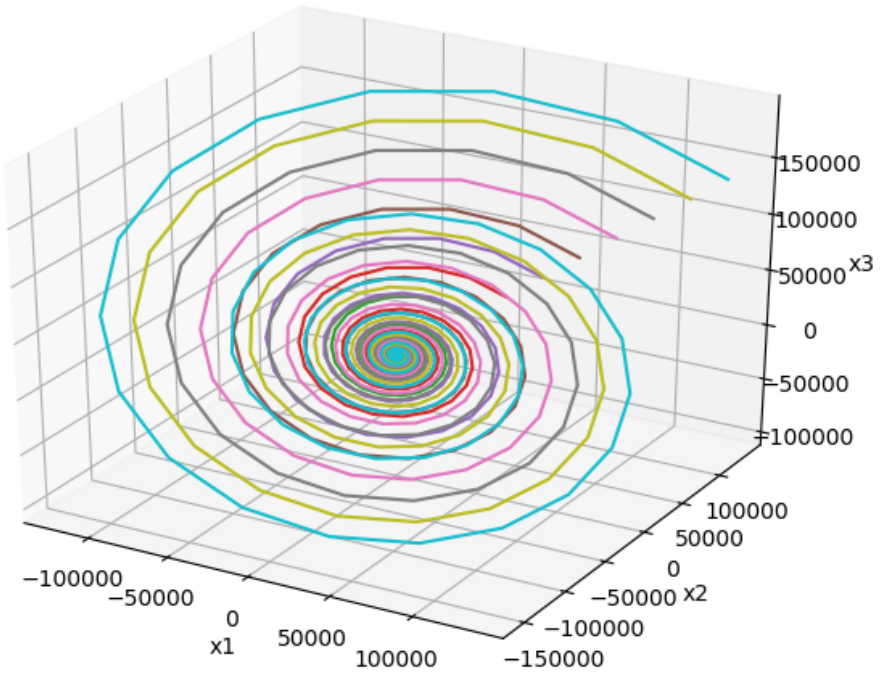
3.1.6 Elitářství

Myšlenka elitářství vychází z toho, že silnější jedinec má větší možnost k přežití než slabší jedinec. V GA to znamená, že do další populace je nejdříve zkopírováno n nejlepších jedinců a až poté jsou tvořeni další jedinci z vybraných rodičů křížením a případnou mutací. Elitářství může poměrně výrazně urychlit nalezení řešení, protože při tvorbě nové populace nejsou ztracena dosud nalezená nejlepší řešení.

3.2 Spiral Optimization Algorithm

3.2.1 Obecný popis

Algoritmus SPO je založen na konceptu, který je inspirován fenoménem spirál v přírodě. Tento v přírodě poměrně běžný jev (schránky měkkýšů, tropické cyklóny, ramena spirálních galaxií atd. [2]) je algoritmem SPO aproximován logaritmickými spirálami. První SPO algoritmus byl navržen pro prohledávání ve 2D v neohraničeném prostoru [9], ale později byl upraven tak, aby jej bylo možné použít pro n-dimenzionální problémy [10].



Obrázek 3.2: Ilustrace algoritmu SPO

Algoritmus funguje tak, že je nalezen nejlepší bod, který je prohlášen za střed spirály. K tomuto bodu jsou poté rotovány všechny ostatní body. Při této rotaci se ke středu body přibližují a pokud je nalezen v průběhu bod, který je lepší kandidát na řešení, je tento bod prohlášen za střed spirály a všechny další body jsou rotovány směrem k novému středu. Tímto způsobem by mělo být nalezené řešení daného problému. [10]

3.2.2 Použití SPO

Tím, že algoritmu SPO hledá vhodnější kandidáty na řešení okolo nejlepšího aktuálního kandidáta, jeví se jako vhodná volba pro upřesnění výsledku. Genetické algoritmy jsou schopné hledat řešení ve velkém stavovém prostoru a mají velmi dobré řešení možnosti uvíznutí v lokálním optimu v podobě mutací. GA se tedy mohou velmi dobře přiblížit přesnému řešení, ale nemusí se vždy dostat do přesného řešení. Z tohoto důvodu se jeví použití algoritmu SPO jako dobrá volba pro upřesnění výsledku nalezeného genetickým algoritmem.

3.2.3 Matematický popis

Konvergenční SPO algoritmus je popsán následujícími body [11]:

1. Nastavte počet bodů vyhledávání $m \geq 2$ a zastavovací kritérium. Vytvořte rotační matici $R(\theta)$ (viz „Tvorba rotační matice“).

2. Specifikujte prvotní body vyhledávání $x_i(0)$ $(i = 1, ..., m)$, ohodnocovací funkci $f(x)$ a určete aktuální střed spirály $x^\star(0) = x_{i_b}(0)$, $i_b = \min\{\arg\min_{i=1,...m}\{f(x_i(0))\}\}$

3. Nastavte parametr $r(k)$ dle následující podmínky (h značí míru konvergence ke středu, viz „Míra konvergence ke středu“):

$$r(k) = \begin{cases} 1 & (k^\star \leq k \leq k^\star + 2n - 1) \\ h & (k \geq k^\star + 2n). \end{cases} \tag{3.1}$$

4. Aktualizujte body vyhledávání dle následujícího vztahu:

$$x_i(k+1) = x^\star(k) + r(k)R(\theta)(x_i(k) - x^\star(k))$$

$$(i = 1, ..., m). \tag{3.2}$$

5. Aktualizujte střed dle následujícího vztahu:

$$x^\star(k+1) = \begin{cases} x_{i_b}(k+1) & (\text{pokud } f(x_{i_b}(k+1)) < f(x^\star(k))) \\ x^\star(k) & (\text{jinak}). \end{cases} \tag{3.3}$$

   kde $i_b = \min\{\arg\min_{i=1,...m}\{f(x_i(0))\}\}$. Dále, jestliže $x^\star(k+1) \neq x^\star(k)$, pak $k^\star = k + 1$.

6. Nastavte $k = k+1$. Pokud bylo splněno zastavovací kritérium, ukončete algoritmus a jeho výstup určete jako $x^\star(k)$. V opačném případě se vraťte na krok č. 3.

Tvorba rotační matice

Aby bylo možné rotovat body, je třeba vytvořit matici rotace tak, jak je popsáno v [10]. Rotace bodu x ve 2-dimenzionálním ortogonálním systému souřadnic doleva okolo počátku o úhel θ lze zapsat takto:

$$x' = R^{(2)}_{1,2}(\theta)x \tag{3.4}$$

Rotační matice pro tuto rotaci vypadá takto:

$$R_{1,2}^{(2)}(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \tag{3.5}$$

Každá 2-dimenzionální matice v n-dimenzionálním prostoru bude tedy vypadat následovně:

$$R_{i,j}^{(n)}(\theta_{i,j}) = \begin{array}{c} \\ \\ i \\ \\ \\ \\ \\ j \\ \\ \\ \end{array} \begin{array}{c} \begin{array}{ccccccccccc} & & & & i & & & & j & & \end{array} \\ \begin{pmatrix} 1 & & & & & & & & & & \\ & \ddots & & & & & & & & & \\ & & 1 & & & & & & & & \\ & & & \cos\theta_{i,j} & & & & & -\sin\theta_{i,j} & & \\ & & & & 1 & & & & & & \\ & & & & & \ddots & & & & & \\ & & & & & & 1 & & & & \\ & & & \sin\theta_{i,j} & & & & & \cos\theta_{i,j} & & \\ & & & & & & & & & 1 & \\ & & & & & & & & & & \ddots \\ & & & & & & & & & & & 1 \end{pmatrix} \end{array}$$

V případě, že $n = 3$, je třeba vytvořit matici rotace pro všechny možné kombinace.

$$R_{1,2}^{(3)}(\theta_{1,2}) = \begin{pmatrix} \cos\theta_{1,2} & -\sin\theta_{1,2} & 0 \\ \sin\theta_{1,2} & \cos\theta_{1,2} & 0 \\ 0 & 0 & 1 \end{pmatrix} \text{(Rotace v rovině } x_1\text{-}x_2\text{)}$$

$$R_{1,3}^{(3)}(\theta_{1,3}) = \begin{pmatrix} \cos\theta_{1,3} & 0 & -\sin\theta_{1,3} \\ 0 & 1 & 0 \\ \sin\theta_{1,3} & 0 & \cos\theta_{1,3} \end{pmatrix} \text{(Rotace v rovině } x_1\text{-}x_3\text{)}$$

$$R_{2,3}^{(3)}(\theta_{2,3}) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_{2,3} & -\sin\theta_{2,3} \\ 0 & \sin\theta_{2,3} & \cos\theta_{2,3} \end{pmatrix} \text{(Rotace v rovině } x_2\text{-}x_3\text{)}$$

Pokud máme například bod $x \in \mathbb{R}^3$, můžeme vznik bodu $x' \in \mathbb{R}^3$, který vznikl rotací bodu x v rovině x_1-x_3 o úhel $\theta_{1,3}$ a poté rotací v rovině x_2-x_3 o úhel $\theta_{2,3}$, zapsat jako $x' = R_{2,3}^{(3)}(\theta_{2,3})R_{1,3}^{(3)}(\theta_{1,3})x$.

Pro rotaci v n-dimenzionálním prostoru je nutné vytvořit kompoziční matici rotace $R^{(n)}$, která vznikne složením rotačních matic pro všechny kombinace dvou os. Rotace bodu má poté tvar:

$$x(k+1) = rR^{(n)}(\theta_{1,2}, \theta_{1,3}, ..., \theta_{n-1,n})x(k) \tag{3.6}$$

kde

$$R^{(n)}(\theta_{1,2},\theta_{1,3},...,\theta_{n-1,n}) = R^{(n)}_{n-1,n}(\theta_{n-1,n}) \times R^{(n)}_{n-2,n}(\theta_{n-2,n})R^{(n)}_{n-2,n-1}(\theta_{n-2,n-1})$$
$$\times ... \times R^{(n)}_{2,n}(\theta_{2,n}) \times ... \times R^{(n)}_{2,3}(\theta_{2,3})R^{(n)}_{1,n}(\theta_{1,n}) \times ... \times R^{(n)}_{1,3}(\theta_{1,3})R^{(n)}_{n-1,n}(\theta_{1,2})$$
$$= \prod_{i=1}^{n-1}(\prod_{j=1}^{i} R^{(n)}_{n-i,n+1-j}(\theta_{n-i,n+1-j})) \tag{3.7}$$

$0 < \theta_{i,j} < 2\pi$ jsou úhly rotace okolo středu v každé rovině a $0 < r < 1$ je míra konvergence ke středu.

Míra konvergence ke středu

Aby byly body přitahovány ke středu (a ne se jen okolo něj otáčely), je třeba určit míru konvergence ke středu. Ve výše popsaném algoritmu je toto číslo označeno jako r (pokud je r nastavováno dle podmínky, je toto číslo označováno jako h). Pro míru konvergence platí, že $0 < h < 1$. Tento parametr je dobré nastavit v závislosti na dimenzi problému dle vztahu [11]:

$$h = \sqrt[2n]{\omega} \tag{3.8}$$
$$\omega \in (0,1)$$

4 Požadavky zadavatele na vytvořené programové vybavení

Zadavatel požadoval, aby byl program zhotoven v programovacím jazyce Python 3 (konkrétně verze 3.6) a aby bylo možné jej co nejjednodušeji použít pro různé druhy materiálů při různém zacházení. Z tohoto důvodu bude program realizován jako jednotlivé skripty v jazyce Python, které bude uživatel používat.

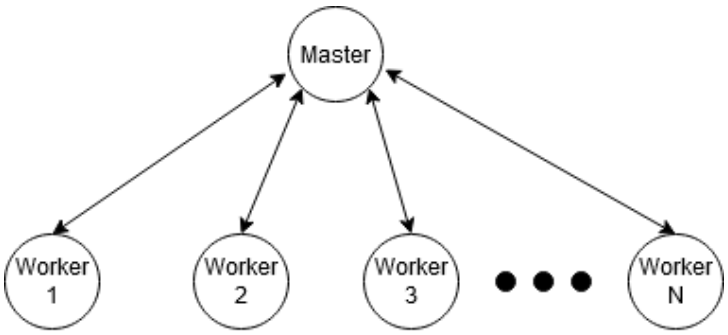
Další požadavek byl na samotný výpočet ohodnocovací funkce v použitých optimalizačních algoritmech (viz kapitola 5), jejíž hlavní část je třeba vypočítat v programu LAMMPS. Z důvodu zadavatelova intenzivního používání výpočetního clusteru a rozhraní MPI musí být pro masivní paralelizaci výpočtu použito právě rozhraní MPI spolu s programem LAMMPS pro složité výpočty.

5 Realizace

V této kapitole je popsáno, jak bylo realizované programové vybavení vytvořené pro potřebu zadavatele.

5.1 Architektura

V programu je použita Master-Worker architektura (která je znázorněna na obrázku 5.1). Tato architektura pracuje na principu, že jeden hlavní proces (dále master proces) rozděluje práci pracovníkům, kteří práci vykonávají (master proces může také vykonávat dále stejnou práci jako ostatní). Poté, co je práce vykonána, je toto sděleno master procesu, který se postará o vyhodnocení a rozdělení další práce.



Obrázek 5.1: Master-Worker architektura, master přiděluje práci pracovníkům

5.2 Struktura programu

Program je strukturován do několika balíčků, které mohou být vloženy k uživatelské implementaci hledaného problému, nebo může být nový výpočet přidán jako další balíček. Základní balíčky programu jsou tyto:

- `core` - Tento balíček obsahuje základní implementace algoritmů (viz sekce 5.3 a 5.4), třídu `Solver` (viz 5.5) a přidružené pomocné funkce.
- `runners` - Balíček obsahuje skripty pro spuštění programu LAMMPS a čtení základních výsledků (viz sekce 5.5).

- `ga_implementations` - Zde jsou dvě třídy GA vycházející ze základní abstraktní třídy z balíku `core`. Jedna třída obsahuje pouze spuštění GA pod rozhraním MPI a je abstraktní a druhá třída dědí od této abstraktní třídy, kde je pouze spuštění pod MPI, a navíc implementuje křížení a mutaci, které jsou vhodné pro úlohy uživatele. Třídu s křížením a mutací již použije zadavatel/uživatel beze změny.
- `spo_implementations` - V tomto balíku je třída SPO vycházející ze základní abstraktní třídy z balíku `core`, která implementuje běh pod MPI, a zadavatel/uživatel ji již použije beze změny.

Další balíky jsou již hotové ukázkové příklady, jež budou dále použity pro demonstraci hledání parametrů pro dané potenciály a příklady. Jedná se o tyto balíky:

- `buck_1d`
- `buck_3d`
- `lj_1d`
- `lj_3d`

V těchto balících jsou naprogramovány příklady, ve nichž jsou hledány parametry pro Lennard-Jonesův potenciál a Buckinghamův potenciál.

5.3 Genetický algoritmus

V této sekci je popsáno, jak byl realizován GA podle potřeby zadavatele.

5.3.1 Výpočet fitness

Program byl vytvořen tak, aby uživatel implementoval pouze svoji specifickou fitness funkci a mohl použít GA bez dalšího programování. Toto byl jeden z požadavků zadavatele (viz kapitola 4), protože bylo třeba splnit co nejjednodušší použití (samozřejmě lze implementovat i vlastní křížení, mutaci a hlavní řídící smyčku GA, ale tyto funkce byly navrženy dle požadavků zadavatele a konkrétního druhu řešeného problému, proto výsledný uživatel bude potřebovat dodat pouze fitness funkci).

Podoba výpočtu se může lišit dle konkrétního úkolu (fitness bude odlišná pokud je materiál deformován ve všech osách stejně, nebo v každé jinak), ale ze zadání plyne, že se vždy bude porovnávat podobnost vypočtených

funkčních hodnot z kvantové mechaniky vůči novým hodnotám z molekulární dynamiky. Proto bylo implementováno čtení vstupních souborů s hodnotami funkce z kvantové mechaniky, které mají předem dohodnutý formát, a jako jeden z parametrů fitness funkce (a samotného GA) je vkládáno pole polí s hodnotami této funkce. Jednotlivé chromozomy představují parametry pro výpočet molekulární dynamiky, proto jsou ohodnocovány tak, že jsou parametry dosazeny do konkrétního výpočtu, jenž je proveden v programu LAMMPS, a jeho výsledek se porovnává s vstupními daty z kvantové mechaniky. Čím je menší odlišnost, tím je chromozom lepší.

Například daný materiál bude stlačován, tudíž se bude měnit celková energie systému v závislosti na velikosti mřížkové konstanty mřížkové struktury (viz sekce 2.3), protože celková energie je závislá na vzdálenostech atomů v systému. Toto lze pro malý systém vypočítat kvantovou mechanikou, jejíž výstup tedy bude závislost energie na mřížkové konstantě. V molekulární dynamice bude provedena také simulace stlačování, ale pro její průběh je třeba najít parametry pro příslušný atomový potenciál (viz sekce 2.1). Ohodnocení tedy bude vypočítáno tak, že jsou za parametry potenciálu ve vstupu programu LAMMPS dosazeny hodnoty, které obsahuje ohodnocovaný chromozom. Program LAMMPS vypočítá energii stlačovaného systému a jeho výstup bude tedy také závislost energie na mřížkové konstantě. Poté je k dispozici pro každou hodnotu mřížkové konstanty hodnota energie v systému, což je to samé jako výstup kvantové mechaniky. Když jsou vypočítány hodnoty jak z kvantové mechaniky (které jsou dané již dopředu jako jeden z parametrů programu), tak z molekulární dynamiky (vypočítáno programem LAMMPS pro daný chromozom, který obsahuje parametry potenciálu), lze ohodnotit chromozom dle rozdílu těchto dvou výstupních sad závislostí energií na mřížkových konstantách.

5.3.2 Generování počáteční populace

Při inicializaci GA je jako jeden z parametrů konstruktoru pole, ve kterém je předána základní hodnota pro každou dimenzi problému. Hodnota pro každou dimenzi je předávána proto, že v mnoha případech existuje odhad podle různých materiálů, v jakých hodnotách by se mohly parametry s největší pravděpodobností pohybovat, a bylo by tak kontraproduktivní prohledávat příliš velký stavový prostor.

Dále je v parametrech GA předán koeficient, který určuje, o kolik procent se může ± hodnota pro každou dimenzi maximálně lišit oproti předané základní. Rozdíl oproti základní hodnotě se určí náhodně z tohoto koeficientu. Do počáteční generace je vždy přidáno potenciální řešení problému,

jenž je vytvořené beze změny z prvotního odhadu, který byl předán do GA v konstruktoru.

Aby byly potenciální řešení rozprostřeny rovnoměrně, je při jejich generování použit generátor náhodných čísel s rovnoměrným rozdělením.

5.3.3 Selekce a elitářství

Výběr jedinců je jednou z esenciálních částí GA. Aby mohla vzniknout nová populace, je třeba vybrat z populace jedince, kteří spolu budou kříženi. Kvůli specifickému křížení, které bylo implementováno po konzultaci se zadavatelem (viz 5.3.4), jsou pro křížení vždy vybíráni pouze dva jedinci. V rámci této práce byly naprogramovány tři metody selekce (Přesný popis metod viz 3.1.5):

- Turnajový výběr
- Žebříčkový výběr
- Výběr ruletovým kolem

Jakou metodu má GA použít, lze specifikovat v konstruktoru GA. Parametr s metodou selekce je nepovinný. Pokud není specifikován, je pro selekci chromozomů, které budou spolu kříženy, použita metoda žebříčkového výběru. Tuto metodu jsem zvolil jako základní, protože je schopna dát lepší šanci na vybrání i slabším jedincům v populaci. Tím, že jsou vybíráni i slabší jedinci, vzniká větší variabilita nových jedinců.

Aby nezanikli nejlepší dosavadní jedinci, je v GA implementováno elitářství. Elitářství je nepovinný parametr v konstruktoru GA. Pokud není specifikován, je automaticky elitářství zapnuto. Pokud je elitářství zapnuto, je požadováno, aby velikost populace byla minimálně 20 jedinců, jinak by hrozilo, že bude kopírována příliš velká část populace, což by degradovalo GA.

5.3.4 Křížení

Křížením chromozomů stávající populace vzniká nová populace s novými potenciálními řešeními problému. Tento GA bude používán k hledání parametrů pro výpočet meziatomových potenciálů. Jednotlivé parametry spolu souvisejí a není tak možné, aby křížením vznikaly naprosto odlišné hodnoty jednotlivých parametrů. Takovéto parametry by s největší pravděpodobností nevedly k žádnému použitelnému řešení a výpočet ohodnocení takovéhoto řešení by ani reálně nemohl proběhnout korektně. Z tohoto důvodu je křížení

řešeno následujícím algoritmem (každý jeden parametr představuje jednu dimenzi problému, počet dimenzí se tedy liší dle počtu hledaných parametrů) 1:

```
Data: chromozom1 a chromozom2 - kopie vybraných předků
Result: chromozom1 a chromozom2 s novými hodnotami pro každou
        dimenzi
for každou dimenzi problému - dim do
    if chromozom1[dim] == chromozom2[dim] then
        continue;
    end
    diff = |chromozom1[dim] - chromozom2[dim]|;
    for oba chromozomy - c do
        rnd = random.uniform(0.1, 1.0);
        new_diff = diff * rnd;
        rnd = random.uniform(0.0, 1.0);
        if rnd < 0.5 then
            c[dim] += new_diff;
        else
            c[dim] -= new_diff;
        end
    end
end
```

Algoritmus 1: Křížení dvou chromozomů

Tímto způsobem vznikne z každého předka nový potomek, který je zařazen do populace.

5.3.5 Mutace

Další ze základních operátorů GA je mutace. Mutace slouží k zabránění uvíznutí v lokálním optimu a k lepšímu prozkoumání stavového prostoru problému.

Jako dva povinné parametry GA jsou pravděpodobnost mutace a koeficient, určující, o kolik procent se může mutovaná hodnota lišit od té původní (obvykle se jedná o malé procento). Pro každého z potomků vzniklých křížením je určováno zda budou mutováni zvlášť. Mutace je řešena následujícím algoritmem 2:

```
Data: chromozom, variance (aktuální koeficient určující, o kolik
      procent se může mutovaná hodnota lišit od té původní)
Result: zmutovaný chromozom
var_half = variance * 0.5;
for každou dimenzi problému - dim do
    rnd = random.uniform_exclude_zero(0.0, variance);
    diff = chromozom[dim] * rnd;
    if rnd < var_half then
        diff = -diff;
    end
    chromozom[dim] += diff;
end
```

Algoritmus 2: Mutace chromozomu

Mutace je obvykle prováděna v průběhu tvorby nové populace dle pravděpodobnosti mutace jednotlivce. Z důvodu specifického křížení, při kterém mohou vznikat stejní jedinci, bylo třeba mutací tomuto jevu co nejvíce zabránit. Pro jeho zabránění je populace kromě nejlepších jedinců mutována, pokud je v populaci více než třetina stejných chromozomů. Tento přístup se ukázal jako velmi efektivní pro zabránění degradace GA.

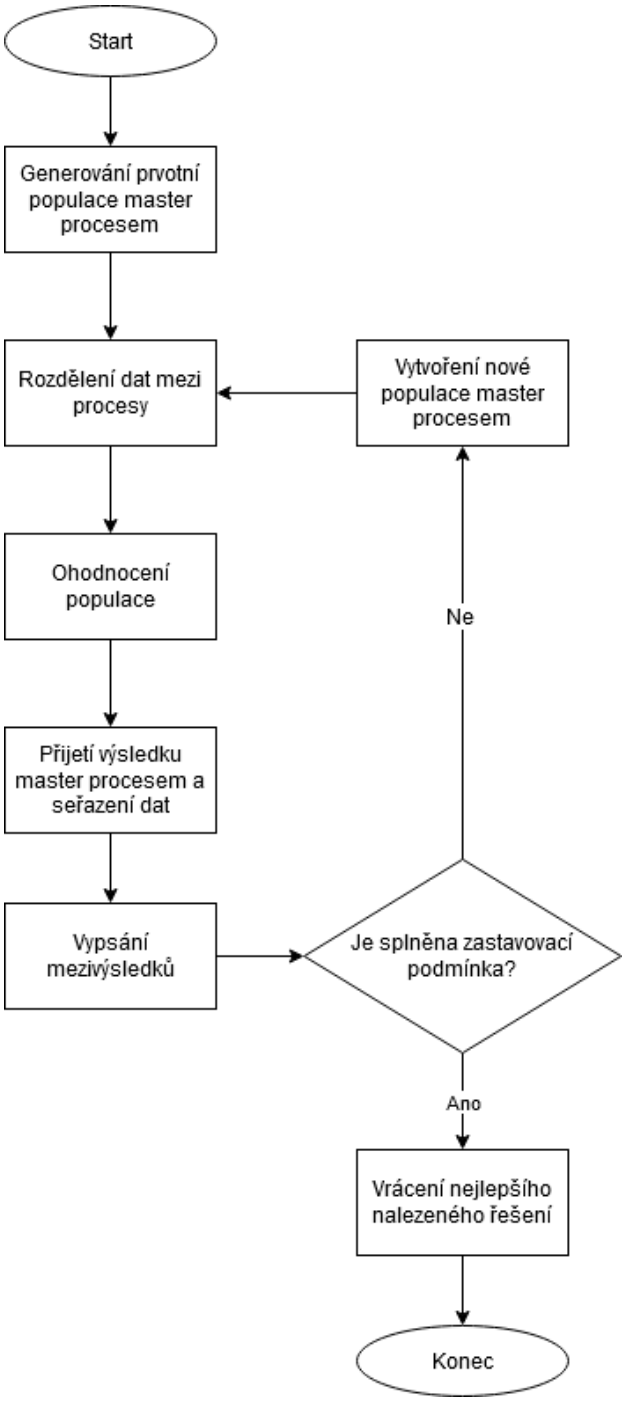
Další z možných nežádoucích stavů je, pokud se dlouho nemění nejlepší jedinec. To sice může znamenat, že GA nalezl nejlepší řešení, ale také že uvázl v lokálním optimu. Tomuto stavu má mutace předcházet a předchází, ale může se stát, že mutované hodnoty jsou ve výsledku odlišné od původních hodnot moc a globální optimum je „přestřelováno“. Z tohoto důvodu je koeficient, o kolik se mutovaná hodnota smí lišit od té původní, postupně snižován a pravděpodobnost mutace je zvýšena na vyšší hodnotu. Po nalezení nového optima jsou tyto hodnoty vráceny na původní.

5.3.6 Paralelizace a průběh algoritmu

Jeden z hlavních požadavků zadavatele byl, aby program běžel na výpočetním clusteru s rozhraním MPI. Toto rozhraní umožňuje masivní paralelizaci s jednoduchou komunikací mezi jednotlivými procesy. V genetickém algoritmu bylo třeba rozložit mezi více výpočetních jednotek výpočet fitness funkce. Její výpočet je časově náročný zejména kvůli složitosti výpočtů molekulární dynamiky a množství chromozomů. Jejich množství se může lišit, ale běžně je počítáno v testovacích výpočtech (viz kapitola 6) v řádu stovek. Pro ohodnocení každého chromozomu je třeba provést příslušný výpočet v

programu LAMMPS, díky jehož výsledku může být chromozom ohodnocen. Program LAMMPS běží v řádech sekund až desítek sekund, proto je třeba ohodnocení chromozomů co nejvíce rozložit. Pro práci s MPI rozhraním byla použita knihovna `mpi4py`.

GA implementuje architekturu Master-Worker (viz sekce 5.1) tak, že je vždy vyhrazen jeden proces (dále jako master proces), který se stará o přidělování chromozomů ostatním procesům, které vypočítají jejich ohodnocení. Samotný master proces také provádí výpočet svých chromozomů. Po dokončení výpočtu zašlou ostatní procesy masterovi své výsledky. Ten se postará o zařazení chromozomů do populace, zkontroluje, zda nebyly splněny zastavovací podmínky GA, a případně pokračuje ve výpočtu. O vytvoření nové populace křížením a mutacemi se také stará tento master proces. Pokud se vyskytla chyba nebo bylo dosaženo zastavovací podmínky, informuje master proces ostatní, aby se korektně ukončily. Celý tento proces je znázorněn na obrázku 5.2. Zastavovací podmínka je tvořena dalšími dvěma podmínkami. První je ta, zda je fitness hodnota nejlepšího jedince menší, než je stanovena uživatelská hodnota (která je předána do GA v konstruktoru). Druhou podmínkou je to, že pokud se nejlepší jedinec nezmění po určitý počet iterací, GA skončí. Tato hodnota je opět parametrem algoritmu (jedná se o nepovinný parametr, pokud není nastaven, jeho hodnota je automaticky 200).



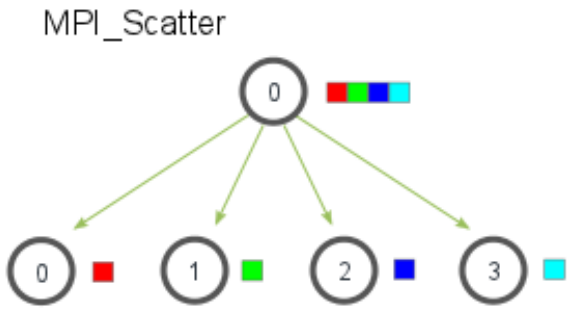
Obrázek 5.2: Vývojový diagram implementovaného genetického algoritmu

Pro komunikaci v rámci MPI byly využity hlavně tyto funkce:

- `MPI_Scatter`
- `MPI_Gahter`

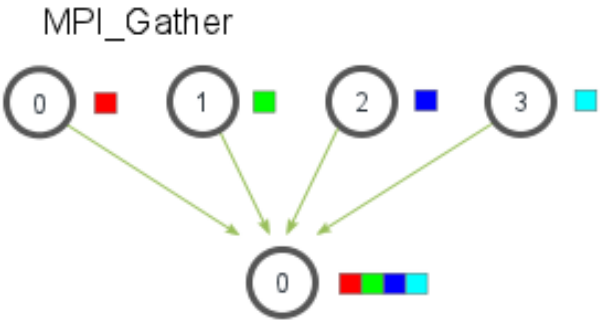
- `MPI_Bcast`

Pro rozeslání chromozomů pro výpočet ohodnocovací funkce z master procesu do ostatních byla použita funkce `MPI_Scatter`. Jako vstup pro tuto funkci je pole o stejné velikosti jako je počet procesů, které master proces touto funkcí rozdělí a rozešle všem, takže má každý proces přidělenou svoji část výpočtu. Funkce je znázorněna na obrázku 5.3 (obrázek převzat z [6]):



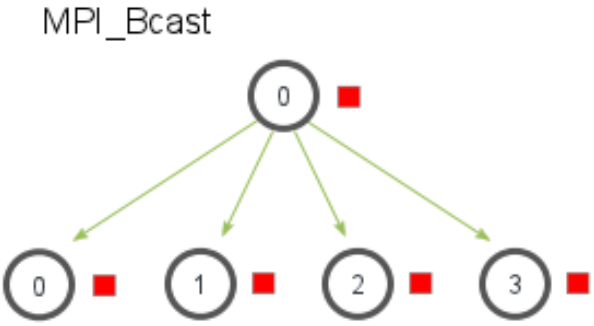
Obrázek 5.3: Funkce `MPI_Scatter`, která rozdělí data rovnoměrně mezi všechny procesy, obrázek převzat z [6]

Poté, co všechny procesy provedou svoji část výpočtu a ohodnotí chromozomy, zašlou zpět data do master procesu. Tento proces má vždy `rank` (identifikační číslo v rámci MPI) 0, proto lze předem specifikovat, do jakého procesu budou data odesílána. Odesílána jsou funkcí `MPI_Gather`, což je de facto inverzní funkce k funkci `MPI_Scatter`. Data jsou přijata do pole o stejné velikosti jako je počet procesů a master proces si jej transformuje zpět na populaci chromozomů. Funkce je znázorněna na obrázku 5.4 (obrázek převzat z [6]):



Obrázek 5.4: Funkce `MPI_Gather`, která ze všech procesů odešle jejich data do jednoho zvoleného procesu, obrázek převzat z [6]

Pro komunikaci master procesu s ostatními je použita funkce `MPI_Bcast`. Funkce je využíváno, aby bylo zajištěno korektní ukončení všech procesů, pokud GA skončí, nebo pokud se vyskytne chyba. `MPI_Bcast` rozešle jednu zprávu všesměrově do všech procesů, takže mají všechny procesy jednu a tu samou informaci. Funkce je znázorněna na obrázku 5.5 (obrázek převzat z [6]):



Obrázek 5.5: Funkce `MPI_Bcast`, která zajišťuje zaslání identických dat do všech procesů, obrázek převzat z [6]

5.4 Spiral Optimization Algorithm

V této sekci jsou popsány implementační záležitosti algoritmu Spiral Optimization Algorithm (dále jen SPO). Algoritmus SPO byl implementován kvůli zpřesnění výsledku nalezeného GA.

5.4.1 Výpočet fitness

Ohodnocení jedince je počítáno stejně jako v případě GA (viz 5.3.1). SPO je zamýšleno používat pro zpřesnění výsledku z GA, proto obě funkce používají stejnou fitness funkci, kterou uživatel definuje (teoreticky může používat i jinou, ale to ve většině případů použití nebude dávat smysl). Funkce je předána do SPO skrze parametr (stejně jako v GA), aby uživatel mohl použít připravené třídy, které jsou vytvořené pro potřeby zadavatele, a nemusel programovat více než hlavní skript a ohodnocující funkci.

5.4.2 Generování počáteční populace

Jako jeden z parametrů algoritmu SPO je pole se základními hodnotami. Pokud je algoritmus požíván pro zpřesnění výsledku, který je získán z GA, může být toto pole při inicializaci vynecháno, protože program si jej později dosadí sám (předpokládá se použití připraveného `solver` skriptu, který od této práce odstiňuje uživatele, viz 5.5). Pokud by byl algoritmus SPO použit sám, je třeba pole se základními hodnotami při inicializaci nevynechat. Důvod předávání počáteční hodnoty pro každou dimenzi je ten, že obvykle existuje počáteční odhad nebo protože se jedná o výsledek z GA.

Dále je jeden z parametrů opět koeficient, jenž určuje, o kolik procent se může $\pm$ hodnota pro každou dimenzi maximálně lišit oproti základní. Do počáteční generace je vždy přidán jedinec, který je vytvořen z počátečních hodnot beze změn. Tímto je zajištěno, že se například výsledek předaný z GA neztratí při generování počáteční populace.

5.4.3 Tvorba rotační matice

Rotační matice je tvořena při inicializaci algoritmu, protože se již dále nebude měnit a její tvorba při rotaci by zpomalovala algoritmus. Algoritmus tvorby matice byl naprogramován podle matematického popisu z podsekce

3.2.3. Algoritmus lze popsat následujícím pseudokódem 3:

```
Data: dim_size - počet dimenzí problému;
theta - úhel rotace;
matrix - prázdná matice velikosti dim_size × dim_size
Result: naplněná matice rotace - matrix
n = dim_size - 1;
i = 1;
while i ≤ n do
    j = 1;
    while j ≤ i do
        matrix[dim_size - i -1][dim_size - i - 1] *= cos(theta);
        matrix[dim_size - j][dim_size - j] *= cos(theta);
        if matrix[dim_size - i - 1][dim_size - j] != 0 then
            matrix[dim_size - i - 1][dim_size - j] *= -sin(theta);
        else
            matrix[dim_size - i - 1][dim_size - j] = -sin(theta);
        end
        if matrix[dim_size - j][dim_size - i - 1] != 0 then
            matrix[dim_size - j][dim_size - i - 1] *= sin(theta);
        else
            matrix[dim_size - j][dim_size - i - 1] = sin(theta);
        end
        j += 1;
    end
    i += 1;
end
```

Algoritmus 3: Tvorba rotační matice

Rotační matice je před rotací vynásobena mírou konvergence ke středu podle toho, kdy se naposledy měnil střed způsobem, který je popsán v rovnici 3.1 v podsekci 3.2.3. Pokud se tedy střed nemění, je po několik iterací rotační matice násobena číslem 1. To znamená, že body okolo středu rotují. Pokud je podmínka (rovnice 3.1 v podsekci 3.2.3) vyhodnocena tak, že body již déle nemají rotovat, je rotační matice vynásobena mírou konvergence ke středu a tím se při rotaci body přibližují k danému středu. Aby nebylo před každou rotací nutně násobit matici, je v algoritmu uložena jak originální matice rotace, která pouze body rotuje, tak i matice, která body rotuje a zároveň přitahuje ke středu.

5.4.4 Rotace

Rotace je vždy uskutečněna pomocí nejlepšího jedince, který je brán jako střed rotace. Tento nejlepší jedinec je hledán vždy po každém ohodnocení populace před následnou rotací. Rotace je popsána následujícím algoritmem 4:

Data: chromosomes - pole chromozomů;
best_c - nejlepší chromozom;
r - míra konvergence ke středu;
rot_mat - rotační matice;
I - matice identity
Result: aktualizované pole chromozomů
rotacni_matice = r * rot_mat;
rotacni_matice_bez_I = rotacni_matice - I;
for *i = 0; i < chromosomes.length; i++* **do**
    chromosomes[i] = dot_multiplication(rotacni_matice,
     chromosomes[i]) - dot_multiplication(rotacni_matice_bez_I,
     best_c)
end

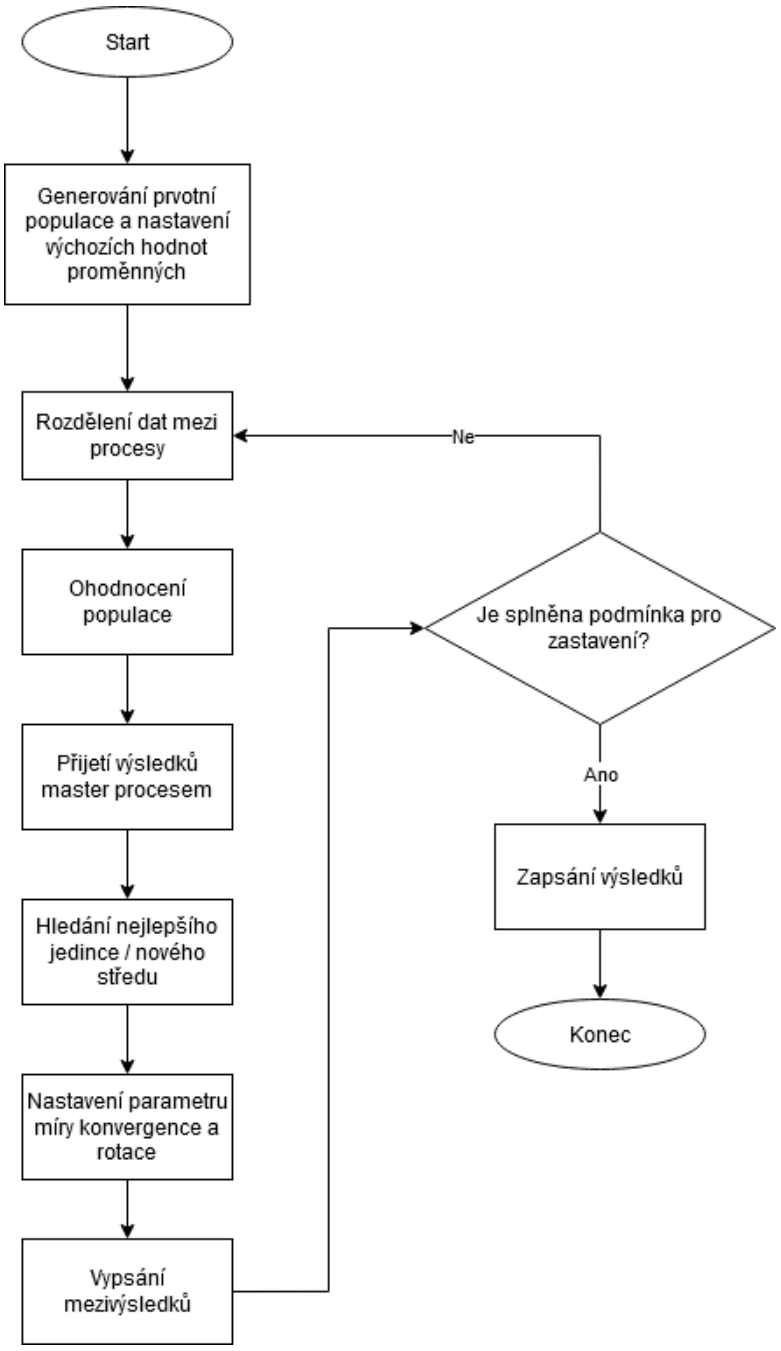
Algoritmus 4: Rotace

5.4.5 Paralelizace a průběh algoritmu

Paralelizace je provedena stejným způsobem jako v případě GA (viz podsekce 5.3.6). Opět bylo nutné použít rozhraní MPI a Master-Worker architektury (viz sekce 5.1). Pro rozesílání práce z master procesu do ostatních procesů a přijímání dat od ostatních je opět zařízeno funkcemi `MPI_Scatter` a `MPI_Gather`. K realizaci zasílání řídících informací z master procesu do ostatních byla opět použita funkce `MPI_Bcast`.

Master proces vygeneruje prvotní populaci, inicializuje potřebné proměnné a rozešle data k ohodnocení ostatním procesům. Ty je po ohodnocení zašlou zpět do master procesu. Ten se postará o nalezení nejlepšího jedince a rotaci okolo tohoto jedince. Před rotací ovšem ještě musí určit míru konvergence ke středu dle podmínky v rovnici 3.1 z podsekce 3.2.3. Dále musí master proces zkontrolovat, zda nebyla splněna podmínka pro zastavení algoritmu. Zastavovací podmínka je opět tvořena dalšími dvěma podmínkami. První podmínkou je, zda je fitness hodnota nejlepšího jedince menší, než je stanovena uživatelská hodnota (která je předána do SPO v konstruktoru). Druhou podmínkou je, že pokud se nejlepší jedinec nezmění po určitý počet iterací, SPO skončí. Tato hodnota je opět parametrem algoritmu (je nepovinným parametrem, pokud není nastaven, jeho hodnota je automaticky 200). Celý

tento proces je znázorněn na 5.6.



Obrázek 5.6: Vývojový diagram implementovaného algoritmu SPO

5.5 Třída `Solver`, hlavní skript a třída `LammpsRunnerBase`

Třída `Solver` byla vytvořena z důvodu, aby byl uživatel co nejvíce odstíněn od nadbytečného programování. Poté, co implementuje fitness funkci, která je parametrem GA i SPO, stačí načíst data funkce z kvantové mechaniky (na což jsou také připraveny funkce) a inicializovat objekty GA a SPO. Třída `Solver` slouží k tomu, aby spustila řešení problému, předala výsledek z GA do SPO a na konci vrátila finální výsledek uživateli. Uživatel je tak odstíněn od případného řešení problému, když například nedoběhne GA korektně, nebo od předávání výsledku z GA do SPO. Zjednodušeně může vypadat hlavní skript takto:

```
import...  # import all necessary packages

def main():
    qm_func = read_qm_func_by_columns("qm.dat", 2)

    ga = GeneticAlgorithm(fitness_function=compute_fitness,
                          qm_func=qm_func,
                          ...)

    spo = SPO(fitness_function=compute_fitness,
              qm_func=qm_func,
              ...)

    solver = Solver(ga, spo)
    result = solver.solve()
    print(result)


def compute_fitness(chromosomes: List[Chromosome],
                    runner_id: int, input_file: str,
                    output_path: str,
                    qm_func: List[List[float]]) -> None:
    if qm_func is None:
        raise QM2MDException("QM function is not defined")

    runner = LammpsRunner(runner_id)

    for c in chromosomes:
        ...
        # filing variables for LAMMPS input file
        variables = [["x", c.genes[0]], ["y", c.genes[1]]]
        if runner.run(input_file, output_path,
                      variables) == 0:
            ...
    ...
if __name__ == '__main__':
    main()
```

Kód 5.1: Ukázka hlavního skriptu výpočtu

V hlavním skriptu je též vidět použití třídy `LammpsRunner`. Tato třída je potomkem třídy `LammpsRunnerBase`, která zajišťuje spouštění výpočtů v programu LAMMPS. Potomek pouze rozšiřuje funkčnost předka o vrácení formátovaných příslušných hodnot z výstupního souboru, který vytvoří jako výsledek výpočtu program LAMMPS. V předkovi jsou naprogramovány funkce pro spuštění výpočtu, úklid po výpočtu a čtení nějakého sloupce

z výstupního souboru. Pokud by uživatel tedy chtěl mít rovnou pohodlně funkce, jež vracejí nějakým způsobem upravená výstupní data, musí si takovéto funkce vytvořit. V ukázkových výpočtech je toto provedeno právě děděním třídy `LammpsRunnerBase`, kde jsou v potomkovi pouze implementované metody pro úpravu výstupních dat z výpočtu. Ovšem pro spuštění výpočtu, úklid a čtení nějakého sloupce z výstupního souboru stačí pouze použít třídu `LammpsRunnerBase`.

Dále je instanci třídy `LammpsRunner` při spouštění výpočtu předáno pole s proměnnými. Tyto proměnné jsou definovány ve vstupním souboru pro LAMMPS a jedná se o parametry příslušného potenciálu. Vyplněním názvu proměnné a její hodnoty v poli proměnných je tak definováno, za jakou proměnnou se má ve vstupním LAMMPS souboru dosadit jaká hodnota. V ukázce hlavního skriptu jsou tyto proměnné pouze parametry potenciálu, které jsou hledány. Co pro daný výpočet uživatel potřebuje a použije, je ovšem čistě na něm.

5.6 Spuštění pod MPI

Pro spuštění pod rozhraním MPI nestačí pouze hlavní skript, ale je třeba vytvořit tzv. „job file“. V tomto souboru je definováno, kolik procesorů je pro spouštěný program požadováno (podle této hodnoty musí uživatel nastavit velikosti populací, aby byly všechny procesy rovnoměrně zatíženy při výpočtu ohodnocení). Dále je definováno, v jaké složce výpočet začíná, přesměrování `stdout` a `stderr` do souborů a příslušná fronta (záleží na výpočetním clusteru). V tomto souboru jsou definovány tyto věci (tento soubor byl použit při vývoji):

1. Shebang - řádek na začátku skriptu začínající znaky křížek a vykřičník, za nimiž (po případné mezeře) je jméno programu, kterým má být skript interpretován
2. Definice pracovní složky - aktuální složka je pracovní
3. Přesměrování `stdin` do souboru
4. Přesměrování `stderr` do souboru
5. Fronta ve výpočetním clusteru
6. Počet požadovaných procesorů
7. Spuštění Python interpretu s hlavním skriptem pod MPI

A takovýto soubor vypadá následovně:

```
1 #! /usr/bin/bash
2 #$ -cwd
3 #$ -o out
4 #$ -e err
5 #$ -q sprkkr.q
6 #$ -pe mpi 100
7 mpirun -np 100 python3 main.py
```

Kód 5.2: Ukázka skriptu pro spuštění programu pod rozhraním MPI

6 Výsledky

V této kapitole jsou uvedeny výsledky vytvořeného programu, tj. výsledky optimalizačního procesu, který aproximuje parametry zvolené funkce popisující energetický potenciál.

Základem pro optimalizační proces jsou výpočty energie krystalové mřížky v mnoha různých konfiguracích metodou kvantové mechaniky. Tyto konfigurace se liší malou změnou parametrů mřížky (vzdáleností atomů v mřížce). Takové změny v mřížce krystalu lze reálně vyvolat působením vnějších sil. V této kapitole jsou kombinovány dvě různé sady dat a dva různé potenciály. Rozdělení do podkapitol je následující.

V podkapitole 6.2 jsou použity výpočty QM, kde elementární buňka krystalu měnila svůj objem, přičemž tvar buňky zůstával krychlový. Tato sada dat měla 42 konfigurací. Hledány byly parametry pro Lennard-Jonesův a Buckinghamův potenciál.

V podkapitole 6.3 jsou použity výpočty QM, kde elementární buňka krystalu měnila svůj tvar neizotropně, kontrakce a dilatace mřížky byla různá v základních směrech. Tato sada dat měla 150 konfigurací. Hledány byly opět parametry pro Lennard-Jonesův a Buckinghamův potenciál.

Pro aproximaci potenciálu je potřeba celá sada konfigurací, které poskytují informaci o energii systému (krystalu) při různých vzdálenostech atomů. Jen tak bude potenciál přesně popisovat situace při deformaci mřížky, při poruchách mřížky apod.

6.1 Metoda hledání parametrů

Pro potenciály byly hledány parametry tak, že pro dané rozměry krystalické mřížky byla vypočítána v MD celková energie v systému (bylo počítáno s použitím daného potenciálu a parametrů potenciálního řešení). Energie byla vypočítána pro všechny rozměry, pro které byla data z QM. Celkové energie systému pro dané rozměry krystalické mřížky byly od sebe odečteny tak, aby vznikla co nejmenší akumulovaná chyba. Chyba byla akumulována tím způsobem, že byly sčítány čtverce rozdílů energie v daných bodech s optimálním posunem dat vůči sobě. Data z QM mohou mít výslednou energii posunutou o konstantní hodnotu, ale tvar musí mít výsledná funkce stejný, viz 2.2. Tato akumulovaná chyba byla použita jako fitness hodnota pro dané potenciální řešení problému.

6.2 Izotropní konfigurace krystalické mřížky

Izotropní konfigurace krystalické mřížky znamená, že se například při stlačování systému mění krystalická mřížka ve všech směrech jednotlivých os stejně.

6.2.1 Lennard-Jonesův potenciál

V této podkapitole jsou uvedeny výsledky aproximace parametrů Lennard-Jonesova potenciálu pro materiál s izotropní konfigurací krystalické mřížky.

Lennard-Jonesův potenciál má tvar:

$$\phi_{LJ}(r) = \frac{A}{r^{12}} - \frac{B}{r^6} \tag{6.1}$$

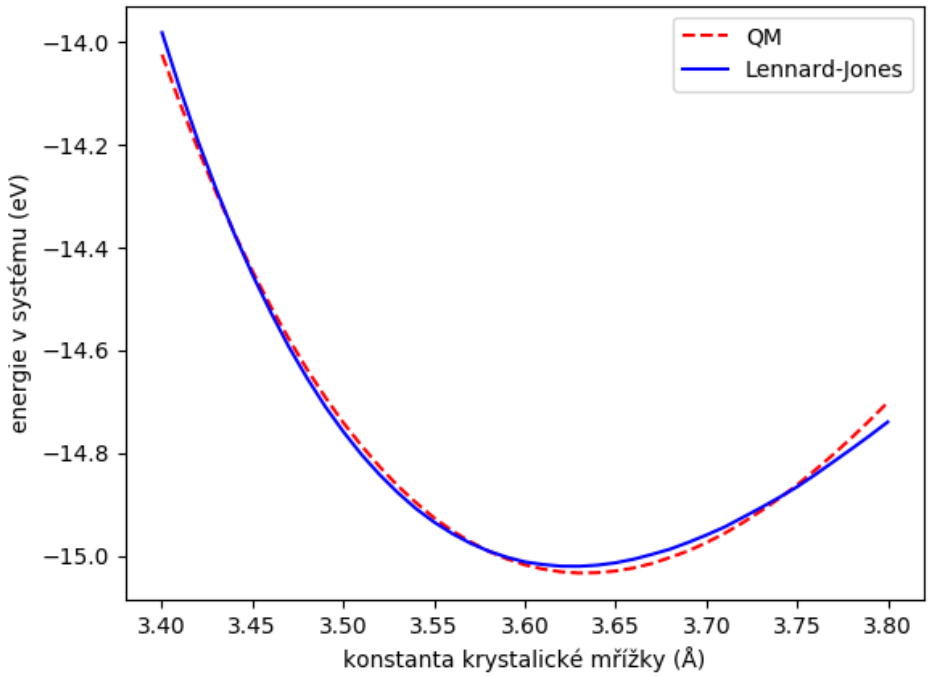
V tomto vzorci jsou A a B identifikované parametry. Výsledky optimalizace jsou uvedeny v tabulce 6.1.

parametr	A	B
hodnota	0.7810362591914752	2.2842911903136005

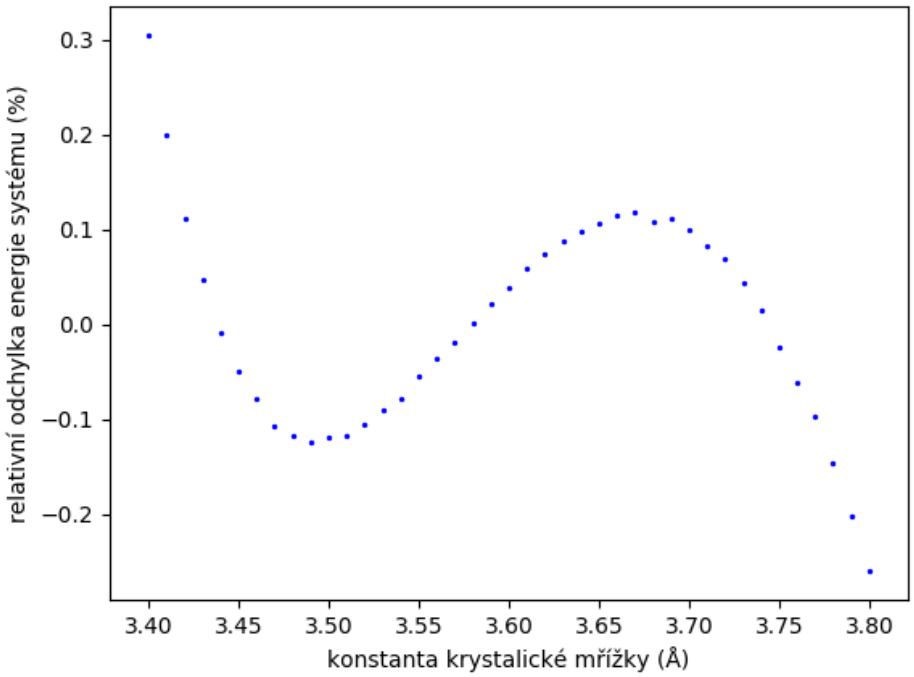
Tabulka 6.1: Aproximované parametry Lennard-Jonesova potenciálu pro izotropní konfiguraci krystalické mřížky

Na obrázku 6.1 je srovnání hodnot energií stanovených výpočtem QM a pomocí Lennard-Jonesova potenciálu s aproximovanými parametry.

Na dalším obrázku 6.2 je zobrazena relativní odchylka energie v systému v závislosti na velikosti mřížkové konstanty od hodnot z QM.



Obrázek 6.1: Výsledek aproximace parametrů Lennard-Jonesova potenciálu pro izotropní konfiguraci krystalické mřížky, plná čára označuje energie aproximované Lennard-Jonesovým potenciálem, čárkovaně jsou hodnoty určené metodou QM



Obrázek 6.2: Znázornění relativní odchylky energie v systému v závislosti na velikosti mřížkové konstanty, odchylka je určena z dat vypočítaných pomocí Lennard-Jonesova potenciálu a aproximovaných parametrů oproti datům z QM

6.2.2 Buckinghamův potenciál

V této podkapitole jsou uvedeny výsledky aproximace parametrů Buckinghamova potenciálu pro materiál s izotropní konfigurací krystalické mřížky.

Buckinghamův potenciál má tvar:

$$\phi_B = A \exp(-Br) - \frac{C}{r^6} \tag{6.2}$$

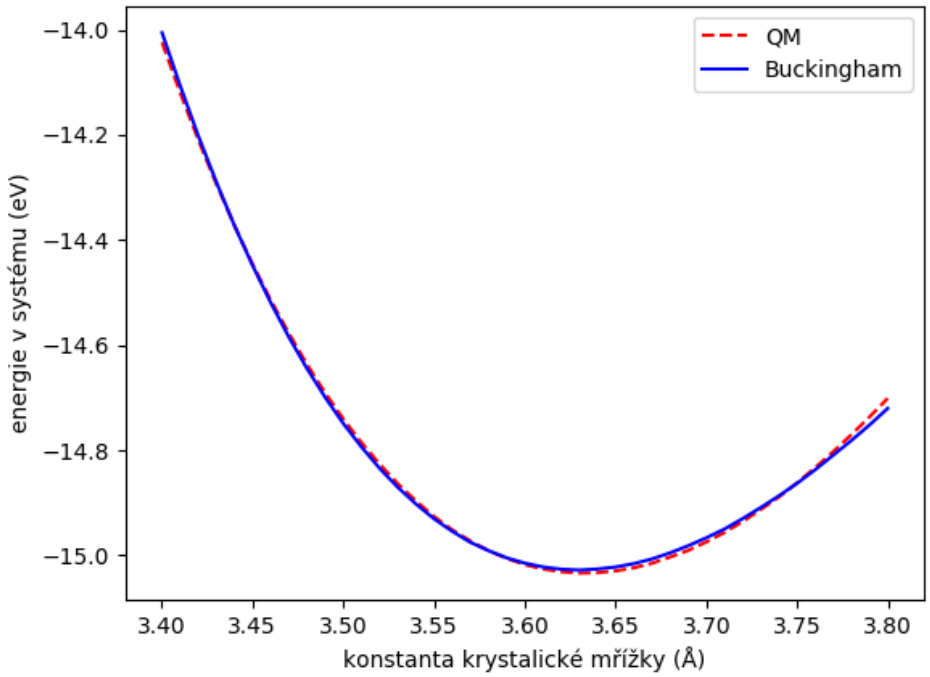
V tomto vzorci jsou A, B a C identifikované parametry. Výsledky optimalizace jsou uvedeny v tabulce 6.2.

parametr	A	B	C
hodnota	87093.46644358922	0.23117640833879705	694.6944500699174

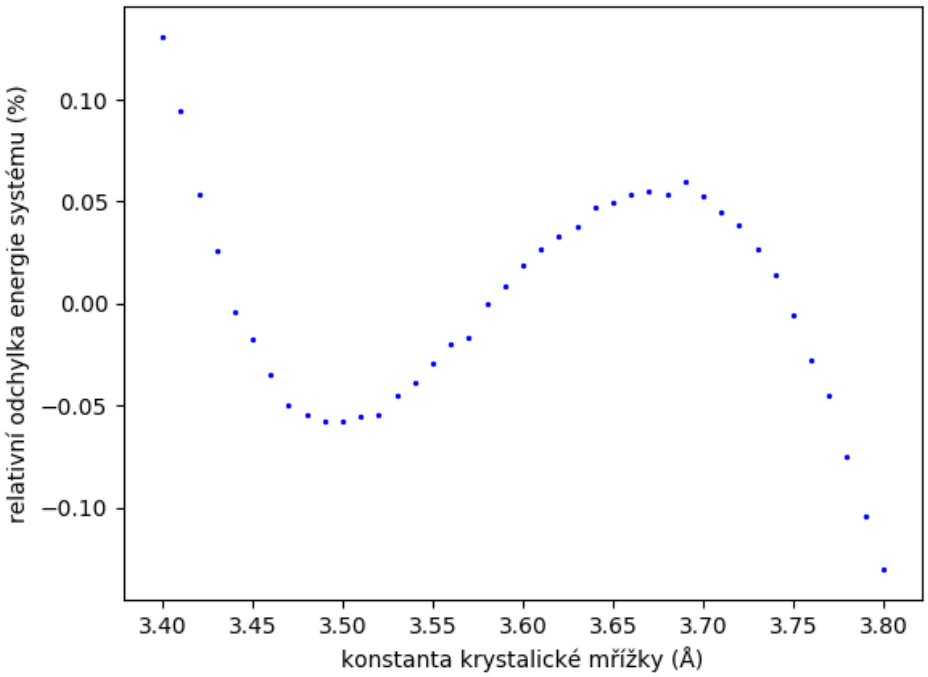
Tabulka 6.2: Aproximované parametry Buckinghamova potenciálu pro izotropní konfiguraci krystalické mřížky

Na obrázku 6.3 je srovnání hodnot energií stanovených výpočtem QM a pomocí Buckinghamova potenciálu s aproximovanými parametry.

Na dalším obrázku 6.4 je zobrazena relativní odchylka energie v systému v závislosti na velikosti mřížkové konstanty od hodnot z QM.



Obrázek 6.3: Výsledek aproximace parametrů Buckinghamova potenciálu pro izotropní konfiguraci krystalické mřížky, plná čára označuje energie aproximované Buckinghamovým potenciálem, čárkovaně jsou hodnoty určené metodou QM



Obrázek 6.4: Znázornění relativní odchylky energie v systému v závislosti na velikosti mřížkové konstanty, odchylka je určena z dat vypočítaných pomocí Buckinghamova potenciálu a aproximovaných parametrů oproti datům z QM

Podle obrázků 6.2 a 6.4 je chyba o něco větší pro extrémní (maximální i minimální) hodnoty konstanty krystalické mřížky. Relativní chyba je v intervalu od -0.25% do 0.3% pro Lennard-Jonesův potenciál (viz obrázek 6.2) a od -0.15% do 0.15% pro Buckinghamův potenciál (viz obrázek 6.4).

6.3 Neizotropní konfigurace krystalické mřížky

Neizotropní konfigurace krystalické mřížky znamená, že se například při stlačování systému mění krystalická mřížka ve všech směrech jednotlivých os jinak.

Konfigurace mřížky je charakterizována třemi parametry (a_x, a_y, a_z) a znesnadňuje tak jednoduché grafické zobrazení. Zjednodušeně je proto zobrazena závislost energie na objemu mřížky ($a_x \times a_y \times a_z$), viz obrázky 6.5 a 6.8. Dále na obrázcích 6.6 a 6.9 je znázorněno porovnání velikosti odchylky energie v jednotlivých konfigurujících.

6.3.1 Lennard-Jonesův potenciál

V této podkapitole jsou uvedeny výsledky aproximace Lennard-Jonesova potenciálu pro materiál s neizotropní konfigurací krystalické mřížky.

Hodnoty aproximovaných parametrů A a B (viz vzorec 6.1) jsou uvedeny v tabulce 6.3.

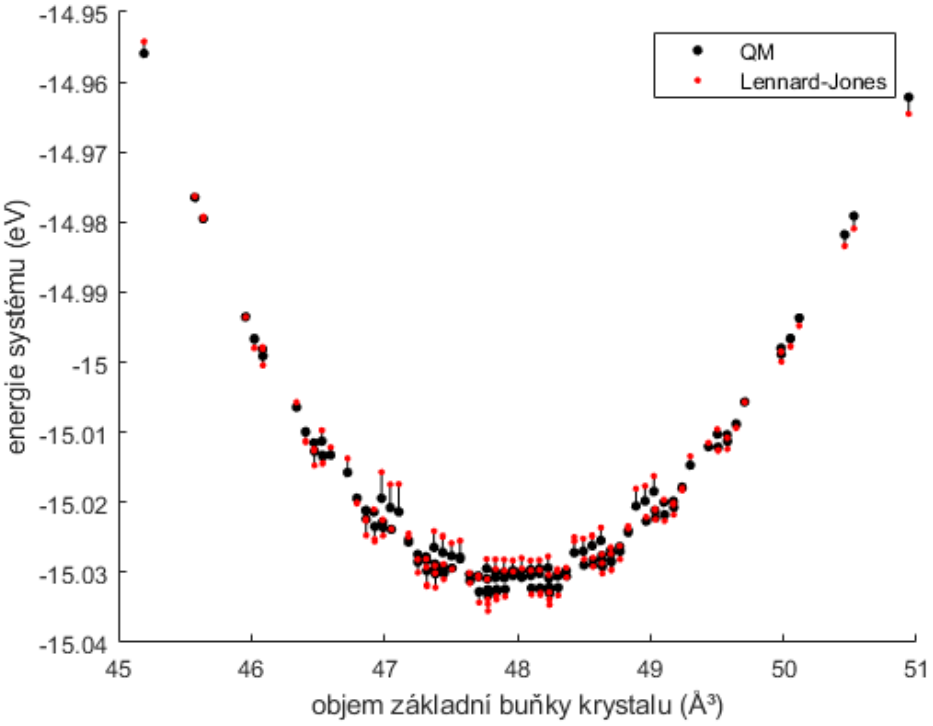
parametr	A	B
hodnota	0.8887180038045444	2.2879780313731852

Tabulka 6.3: Aproximované parametry Lennard-Jonesova potenciálu pro neizotropní konfiguraci krystalické mřížky

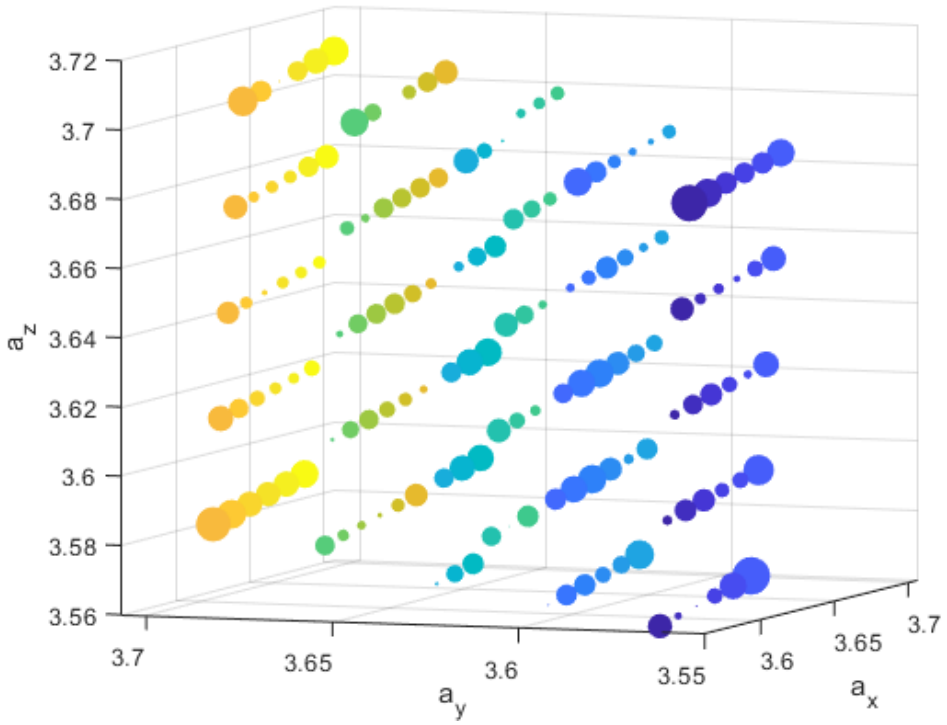
Na obrázku 6.5 je ukázáno, jak se liší energie systému (v závislosti na objemu buňky krystalu) vypočítaná z dat z MD oproti datům z QM.

Dále obrázek 6.6 znázorňuje chybu velikosti energie v systému v závislosti na velikosti buňky krystalu vypočtené z dat z MD oproti datům z QM.

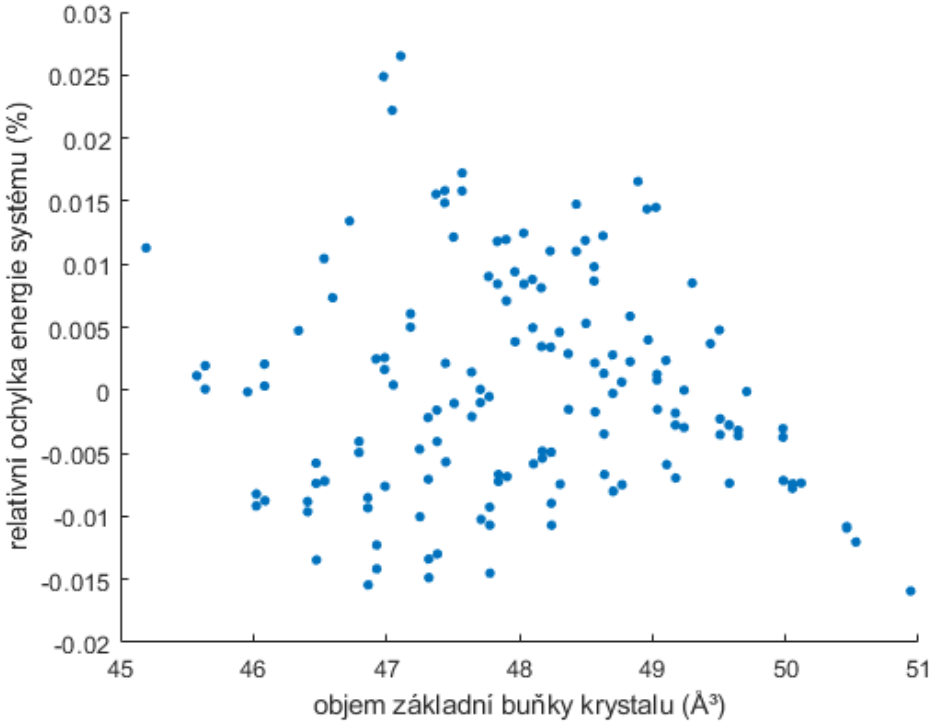
Obrázek 6.7 zobrazuje relativní odchylky energie v systému v závislosti na objemu základní buňky krystalu. Odchylka je určena z dat vypočítaných pomocí Lennard-Jonesova potenciálu a aproximovaných parametrů oproti datům z QM.



Obrázek 6.5: Graf velikosti energie systému v závislosti na objemu základní buňky krystalu, zobrazena jsou data z QM a k nim znázorněn rozdíl vypočtených dat z MD za použití aproximovaných parametrů pro Lennard-Jonesův potenciál



Obrázek 6.6: Znázornění chyby velikosti energie v systému v závislosti na velikosti buňky krystalu vypočtené z dat z MD oproti datům z QM, data z MD byla vypočtena pomocí Lennard-Jonesova potenciálu s použitím aproximovaných parametrů, velikost symbolu je úměrná velikosti odchylky, barva symbolu odpovídá hodnotě a_y (pro přehlednější zobrazení)



Obrázek 6.7: Znázornění relativní odchylky energie v systému v závislosti na objemu základní buňky krystalu, odchylka je určena z dat vypočítaných pomocí Lennard-Jonesova potenciálu a aproximovaných parametrů oproti datům z QM

6.3.2 Buckinghamův potenciál

V této podkapitole jsou uvedeny výsledky aproximace Buckinghamova potenciálu pro materiál s neizotropní konfigurací krystalické mřížky.

Hodnoty aproximovaných parametrů A, B a C (viz vzorec 6.2) jsou uvedeny v tabulce 6.4.

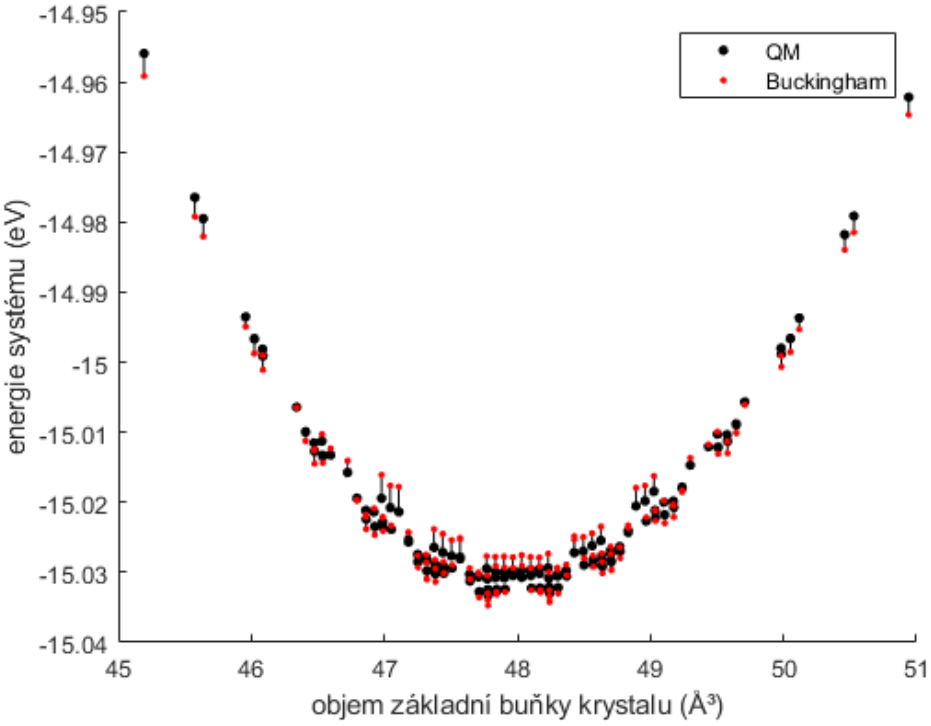
parametr	A	B	C
hodnota	26378.102523759742	0.28926542719909776	1560.1442729703322

Tabulka 6.4: Aproximované parametry Buckinghamova potenciálu pro neizotropní konfiguraci krystalické mřížky

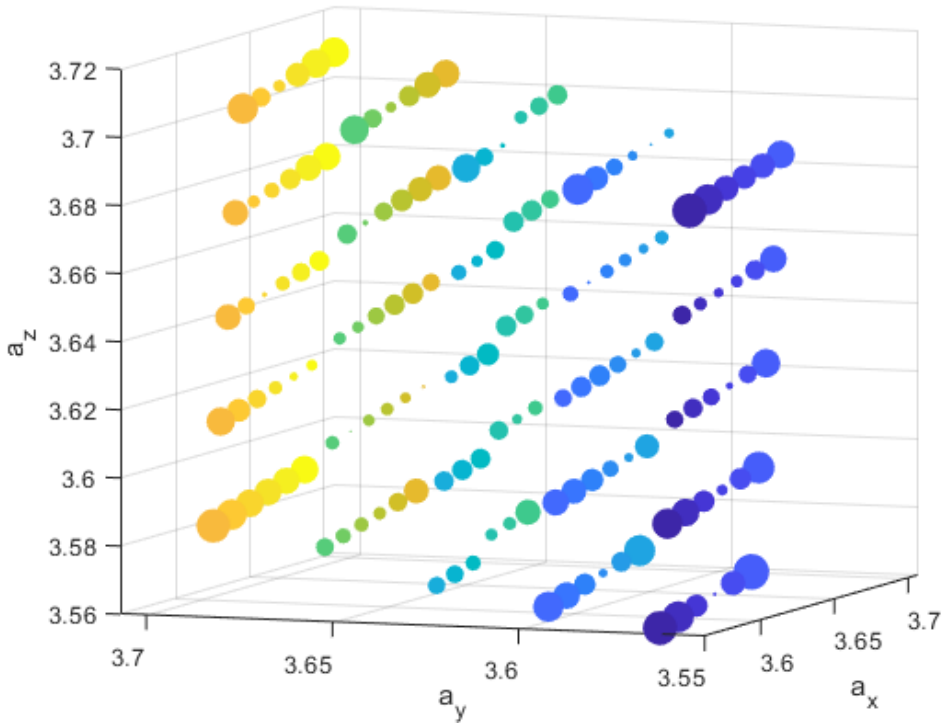
Na obrázku 6.8 je ukázáno, jak se liší energie systému (v závislosti na objemu buňky krystalu) vypočítaná z dat z MD oproti datům z QM.

Dále obrázek 6.9 znázorňuje chybu velikosti energie v systému v závislosti na velikosti buňky krystalu vypočtené z dat z MD oproti datům z QM.

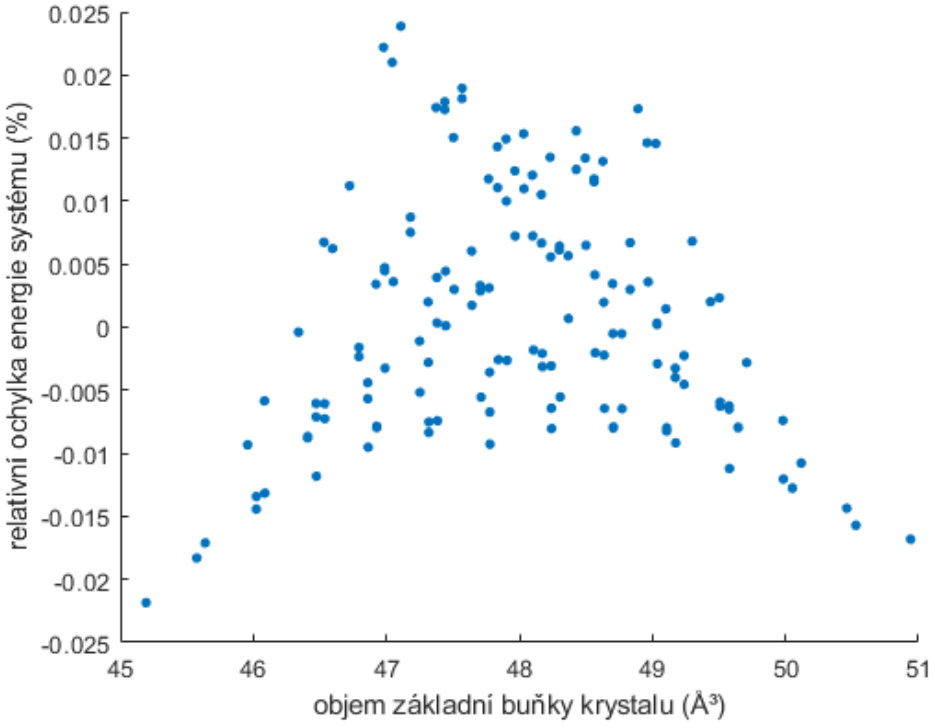
Obrázek 6.7 zobrazuje relativní odchylky energie v systému v závislosti na objemu základní buňky krystalu. Odchylka je určena z dat vypočítaných pomocí Buckinghamova potenciálu a aproximovaných parametrů oproti datům z QM.



Obrázek 6.8: Graf velikosti energie systému v závislosti na objemu základní buňky krystalu, zobrazena jsou data z QM a k nim znázorněn rozdíl vypočtených dat z MD za použití aproximovaných parametrů pro Buckinghamův potenciál



Obrázek 6.9: Znázornění chyby velikosti energie v systému v závislosti na velikosti buňky krystalu vypočtené z dat z MD oproti datům z QM, data z MD byla vypočtena pomocí Buckinghamova potenciálu s použitím aproximovaných parametrů, velikost symbolu je úměrná velikosti odchylky, barva symbolu odpovídá hodnotě a_y (pro přehlednější zobrazení)



Obrázek 6.10: Znázornění relativní odchylky energie v systému v závislosti na objemu základní buňky krystalu, odchylka je určena z dat vypočítaných pomocí Buckinghamova potenciálu a aproximovaných parametrů oproti datům z QM

Podle obrázků 6.7 a 6.10 je chyba o něco větší pro extrémní (maximální i minimální hodnoty) a_x, a_y a a_z. Relativní chyba je v intervalu -0.02% do 0.03% pro Lennard-Jonesův potenciál (viz obrázek 6.7) a -0.025% do 0.025% pro Buckinghamův potenciál (viz obrázek 6.10).

6.4 Diskuze

Lze konstatovat, že pro všechny čtyři uvedené úlohy algoritmus úspěšně aproximoval parametry zvolených potenciálů. Maximální relativní odchylka energií je od -0.25% do -0.3% pro izotropní konfigurace krystalické mřížky a od -0.025% do 0.03% pro neizotropní konfigurace krystalické mřížky. Relativní odchylka je větší v případech s izotropní konfigurací krystalické mřížky, protože v sadě dat z QM jsou konfigurace, kde jsou atomy mnohem blíže a mnohem dále od sebe.

Výpočetní nároky na jednu optimalizaci jsou poměrně značné, průměrně 6 hodin pro izotropní konfigurace krystalické mřížky a 20 hodin pro neizotropní konfigurace krystalické mřížky při použití 100 CPU. Ovšem dobu výpočtu zásadně ovlivňuje počet konfigurací a také, jak moc dobrý je prvotní odhad parametrů. Dále dobu ovlivňuje, kolik jedinců bude pro ohodnocování připadat na jeden CPU. Z těchto důvodů nelze předem určit, jak dlouho bude výpočet trvat.

Přesnost parametrů jednotlivých potenciálů lze dále hodnotit. Například podle toho, jak přesně předpovídají konkrétní fyzikální vlastnosti materiálu (tepelná roztažnost, modul pružnosti apod.). Tyto výpočty ovšem nejsou zcela triviální, a tak nebyly cílem této práce.

7 Závěr

Cílem této práce bylo vytvořit metodu pro aproximaci parametrů atomových potenciálů. Tato vytvořená metoda měla pomoci vědcům z výzkumného centra NTC ZČU při zkoumání nových materiálů.

Na počátku práce jsem se seznámil s fyzikálním pozadím problému. Byl jsem seznámen s technikami molekulární dynamiky a kvantové mechaniky. Pro obě metody jsem byl dále seznámen s výpočetními programy, které se používají při jejich výpočtech. Konkrétně jsem byl seznámen s programy LAMMPS a VASP.

Po seznámení se s fyzikálním pozadím problému byly popsány optimalizační algoritmy, které jsem vybral pro řešení problematiky hledání parametrů atomových potenciálů. Vybral jsem genetické algoritmy pro jejich schopnost hledat řešení složitých problémů ve velmi velkých stavových prostorech a algoritmus Spiral Optimization Algorithm díky jeho schopnosti hledat lepšího kandidáta v postupně se zmenšujícím okolí aktuálního nejlepšího kandidáta, což umožnilo zpřesnit výsledek genetických algoritmů.

Dále práce obsahuje popis požadavků zadavatele, způsob implementace, popis programu a propojení optimalizačních algoritmů.

Na konci práce jsou uvedené výsledky vytvořených ukázkových výpočtů, které byly řešené vyvinutým programovým vybavením. Výsledky jsou zde popsány a zhodnoceny.

Výsledkem práce je vytvořená metoda pro aproximaci parametrů atomových potenciálů a implementace této metody a s tím spojených optimalizačních algoritmů. Tímto práce splnila zadání v jeho plném rozsahu.

Přehled zkratek

- DFT - Density functional theory (teorie funkcionálu hustoty)
- FCC - Face-centered cubic - Plošně centrovaná kubická mřížka
- GA - Genetické algoritmy
- LAMMPS - Large-scale Atomic/Molecular Massively Parallel Simulator - program
- MD - Molekulární dynamika
- MPI - Message Passing Interface
- SPO - Spiral Optimization Algorithm
- QM - Kvantová mechanika

Literatura

[1] LAMMPS Molecular Dynamics Simulator. https://lammps.sandia.gov/. [cit. 2019/08/20].

[2] Where Equiangular Spirals Are. http://jwilson.coe.uga.edu/EMT668/EMAT6680.F99/Erbas/KURSATgeometrypro/nature%26logspiral/nature%26logspi.html. [cit. 2020/04/07].

[3] Dvořák, M. *Depozice Ga a GaN ultratenkých vrstec na grafenový substrát* [online]. Vysoké učení technické v Brně, 2013. [cit. 2019/08/25]. Dostupné z: https://dspace.vutbr.cz/xmlui/handle/11012/27952.

[4] Ekštein, K. *Přednáška: Genetické algoritmy* [online]. Západočeská univerzita v Plzni, 2018. [cit. 2019/08/25]. Dostupné z: https://courseware.zcu.cz/portal/studium/courseware/kiv/su.

[5] Generalic, E. *Face-centered cubic lattice* [online]. Croatian-English Chemistry Dictionary & Glossary, 2018. [cit. 2019/09/02]. Dostupné z: https://glossary.periodni.com/glossary.php?en=face-centered+cubic+lattice.

[6] Kendall, W. *MPI Scatter, Gather, and Allgather* [online]. 2019. [cit. 2020/03/30]. Dostupné z: https://mpitutorial.com/tutorials/mpi-scatter-gather-and-allgather/.

[7] Lamoureux, P. *Computational Modeling in Heterogeneous Catalysis*. 01 2017. doi: 10.1016/B978-0-12-409547-2.14273-8. ISBN 9780124095472.

[8] Sutmann, G. Classical Molecular Dynamics. *Quantum Simul Complex Many-body Syst: Theory Algorithms*. 08 2009, 10.

[9] Tamura, K. – Yasuda, K. Spiral Dynamics Inspired Optimization. *Journal of Advanced Computational Intelligence and Intelligent Informatics*. 10 2011, 15, s. 1116–1122.

[10] Tamura, K. – Yasuda, K. Spiral optimization -A new multipoint search method. s. 1759–1764, 10 2011. doi: 10.1109/ICSMC.2011.6083926.

[11] Tamura, K. – Yasuda, K. The Spiral Optimization Algorithm: Convergence Conditions and Settings. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*. 05 2017, PP, s. 1–16. doi: 10.1109/TSMC.2017.2695577.

Seznam obrázků

Seznam tabulek

Seznam algoritmů

Seznam ukázek kódu

A Návod k použití programu

V návodu jsou označovány vytvořené příklady s izotropní konfigurací krychlové mřížky jako „1D“ a příklady s neizotropní konfigurací krychlové mřížky jako „3D“.

A.1 Balíky programu

Následovat bude popis jednotlivých balíků a skriptů v nich nacházejících se. Pokud nebudou uvedeny bližší informace o některé třídě nebo funkci v daném skriptu, jsou v programovém kódu vytvořeny dokumentační komentáře, které popisují funkčnost jednotlivých tříd, funkcí a metod a také obsahují popis proměnných vstupujících do daných funkcí nebo metod. Pro bližší informace je tedy vhodné těchto dokumentačních komentářů využít.

A.1.1 `core`

V tomto balíku jsou naprogramovány základní třídy a funkce. Jednotlivé skripty jsou tyto:

- `chromosome` - Obsahuje třídu `Chromosome`, jejíž instance představují jednotlivé potenciální řešení daného problému.
- `ga` - Zde je naprogramována základní abstraktní třída GA `GABaseClass`.
- `qm2mdexception` - Tento skript obsahuje implementaci výjimky používané v tomto programu.
- `qm_func_reader` - Obsahuje funkce pro čtení vstupního souboru s daty z QM. Jsou zde dvě funkce. Jedna čte vstupní soubor po jednotlivých řádcích a druhá po sloupcích.
- `selections` - Zde jsou naprogramovány různé metody selekce pro GA. GA je předána metoda selekce v konstruktoru třídy (implicitně je použit žebříčkový výběr).
- `solver` - V tomto skriptu je třída `Solver`, která se stará o spuštění GA a SPO a o korektní ukončení programu po výpočtech.
- `spo` - Tento skript obsahuje základní abstraktní třídu `SPOBaseClass` SPO algoritmu.

- `util` - Zde jsou naprogramovány různé pomocné funkce.

A.1.2 `ga_implementations`

Tento balík obsahuje implementace abstraktní třídy `SPOBaseClass` ze skriptu `core.spo`. Pokud by uživatel potřeboval vytvořit novou specifickou implementaci GA, lze v tomto balíku vytvořit nový skript a použít třídy vytvořené v tomto balíku, nebo předka z balíku `core`. Balík obsahuje následující skripty:

- `ga_basic_mpi` - Zde je implementován GA tak, jak je popsán v kapitole 5 - Realizace diplomové práce.
- `ga_mpi_run_only` - Tento skript obsahuje implementaci pouze spuštěné pod MPI. Další abstraktní metody předka `GABaseClass` ze skriptu `core.ga` nejsou zde implementovány.

A.1.3 `runners`

Balík obsahuje implementace třídy `LammpsRunnerBase` a další třídy, které ji dědí. Tato třída zajišťuje spuštění výpočtu v programu LAMMPS a umožňuje číst výsledky ze souboru, ve kterém jsou uloženy. Obecně může mít soubor s výsledky jakýkoliv formát, ale platí, že soubor **musí být strukturován do sloupců**. Záleží na tom, jak si jej uživatel definuje ve vstupním souboru pro LAMMPS.

Specializované třídy jsou zde hlavně proto, aby četly specifický formát výsledků a byly vytvořeny pro ukázkové příklady.

Uživatel si zde může vytvořit případně další specializované třídy pro pohodlnější čtení výsledků. Balík obsahuje následující skripty:

- `lammps_runner` - Skript obsahuje implementaci základní třídy pro spuštění výpočtu v programu LAMMPS a čtení jednotlivých sloupců ze souboru s výsledky.
- `lammps_runner_1d` - Zde je potomek třídy `LammpsRunnerBase`, který byl vytvořen pro „1D“ příklady. To jsou příklady, v nichž se mřížková konstanta mění ve všech směrech stejně.
- `lammps_runner_3d` - Zde je potomek třídy `LammpsRunnerBase`, který byl vytvořen pro „3D“ příklady. To jsou příklady, v nichž se mřížková konstanta mění v každém směru jinak.

A.1.4 `spo_implementations`

Tento balík obsahuje implementace abstraktní třídy `GABaseClass` ze skriptu `core.spo`. Pokud by uživatel potřeboval vytvořit novou specifickou implementaci SPO, lze v tomto balíku vytvořit nový skript a použít třídy vytvořené v tomto balíku nebo předka z balíku `core`. Balík obsahuje následující skripty:

- `spo_mpi_run` - Skript obsahuje implementaci základní třídy SPO algoritmu, která je přizpůsobena pro běh pod MPI.

A.2 Požadavky na soubor s výsledky z QM

Soubor s daty z QM **musí být strukturován do sloupců**. Například pro vytvořené příklady „1D“ potenciálů jsou v tomto souboru v prvním sloupci hodnoty mřížkové konstanty a ve druhém celková energie v systému dle dané mřížkové konstanty.

A.3 Požadavky na vstupní soubor pro program LAMMPS

Vstupní soubor programu LAMMPS muže být libovolný. Uživatel musí ovšem za parametry potenciálů dosadit názvy proměnných, které budou do výpočtu předávány z fitness funkce. Pro název a cestu ukládaného souboru s výsledky je vždy **vyhrazena proměnná `$1`**.

A.4 Ukázka vytvoření výpočtu

Tato ukázka je založena na zdrojových souborech naprogramovaného hotového výpočtu „lj_1d“. Pro vytvoření nového výpočtu je nejprve potřeba učinit následující kroky:

1. Na úrovni balíků programu vytvořit nový balík pro vytvářený výpočet. V tomto případě balík `lj_1d`.
2. Je třeba vytvořit hlavní skript výpočtu, který lze pojmenovat libovolně a v naprogramovaných ukázkách se jmenuje `main.py`.
3. Vytvořit je také třeba tzv. „job file“, jenž zajišťuje spuštění programu pod MPI, který se taktéž může jmenovat libovolně a v naprogramovaných ukázkách se jmenuje `job.sh`.

4. Vytvořit vstupní soubor pro program LAMMPS a v něm místo konkrétních parametrů potenciálů definovat proměnné.

5. Mít soubor se vstupními daty z QM. Pro ukázku „1D potenciálů“ tento soubor obsahuje mřížkovou konstantu v jednom sloupci a v druhém odpovídající energii systému. Pro „3D potenciály“ obsahuje soubor v prvních třech sloupcích mřížkovou konstantu pro každý rozměr a ve čtvrtém energii systému.

A.5 Zbylé balíky

Zbylé balíky `lj_1d`, `lj_3d`, `buck_1d` a `buck_3d` obsahují hotové naprogramované příklady.

A.5.1 Hlavní skript výpočtu

Na začátek skriptu je třeba přidat základní importy a příkazy, které zajistí, aby byly nastaveny správně cesty pod MPI.

```
import sys
import os

path = os.path.join(os.path.dirname(__file__), os.pardir)
sys.path.append(path)
```

Dále je třeba připojit importy na implementace GA, SPO, `solver` skriptu pro čtení dat z QM a `LammpsRunner`.

```
from ga_implementations.ga_basic_mpi import *
from spo_implementations.spo_mpi_run import *
from core.solver import *
from core.qm_func_reader import *
from scipy import interpolate
from runners.lammps_runner_1d import LammpsRunner
from scipy.optimize import fmin
```

Po importu všech výše uvedených potřebných skriptů je třeba vytvořit fitness funkci, která bude ohodnocovat chromozomy v předané kolekci. Fitness funkce má vždy stejný kontrakt, protože je vkládána jako parametr pro GA a SPO. Ve funkci je třeba si vytvořit instanci použité třídy pro spuštění výpočtu v programu LAMMPS a projít všechny chromozomy v předané kolekci. Třídě pro spuštění programu LAMMPS je třeba předat pole s proměnnými, které budou dosazeny do výpočtu v programu LAMMPS (viz A.3). Z výsledků výpočtů programu LAMMPS je třeba vypočítat fitness daného chromozomu. Čím je menší hodnota fitness, tím je chromozom lepší. V tomto

příkladu je fitness vypočítána jako čtverec rozdílů funkčních hodnot (energie v systému) ve stejných bodech na ose x (mřížková konstanta) funkcí z QM a funkcí, která vznikla jako výsledek z MD (LAMMPS). Protože můžou být data z MD posunuta o konstantu oproti datům z QM (viz 2.2), je třeba najít toto posunutí. K tomu slouží použitá funkce `fmin` z balíku `scipy` a pro její potřebu vytvořená chybová funkce `error_func`. Tato metoda může také vyhodit výjimku `GAException` v případě nějaké chyby, která způsobí korektní ukončení výpočtu. Její konkrétní podoba v příkladu `lj_1d` je takováto:

```
def compute_fitness(chromosomes: List[Chromosome], runner_id:
    int, input_file: str, output_path: str,
                    qm_func: List[List[float]]) -> None:
    """
    Computes and updates fitness value of each chromosome in
   given list of chromosomes
    :param chromosomes: list of chromosomes
    :param runner_id: ID for LAMMPS runner
    :param input_file: full path to input LAMMPS file
    :param output_path: output path for various logs
    :param qm_func: function from quantum mechanics which is
   used for computation of fitness values
    :raises QM2MDException: if fitness computation goes to
   error state QM2MDException is thrown, if exception is \
    thrown whole program stops
    """

    if qm_func is None:
        raise QM2MDException("No QM function")  # There is no
    valid QM function
    else: # save values from QM for better readability
        x1 = qm_func[0]
        y1 = qm_func[1]

    y1 = util.sub_mean(y1, np.mean(y1))
    f1 = interpolate.interp1d(x1, y1)

    # create runner object
    runner = LammpsRunner(runner_id)

    for c in chromosomes:
        fitness = 0.0

        # filing variables for LAMMPS input file
        variables = [["x", c.genes[0]], ["y", c.genes[1]]]
        # compute fitness if LAMMPS ended with 0
        if runner.run(input_file, output_path, variables) ==
   0:
```

```
            # fitness computation
            x2 = runner.get_v_d0()
            y2 = runner.get_v_E()

            y2 = util.sub_mean(y2, np.mean(y2))
            f2 = interpolate.interp1d(x2, y2)

            x_both_new = np.linspace(max(min(x1), min(x2)),
   min(max(x1), max(x2)), len(x2))
            try:
                y1_new = f1(x_both_new)
                y2_new = f2(x_both_new)
                params = (y1_new, y2_new)
                xopt, fopt, iter, funcalls, warnflag = fmin(
   func=error_func, x0=0, args=params, disp=False,

   full_output=True)
                fitness = fopt
            except ValueError:
                fitness = float('inf')

            c.fitness = fitness
            runner.cleanup()
        else:
            # if LAMMPS computation did not ended correctly
            c.fitness = float('inf')

def error_func(x, p1, p2):
    accumulator = 0
    for i in range(len(p1)):
        accumulator += (p1[i] - p2[i] + x) ** 2

    return accumulator
```

S hotovou fitness funkcí již chybí pouze vytvořit instance algoritmů a spustit výpočet. Tuto úlohu má hlavní metoda ve skriptu. Nejprve jsou načtena data z QM a poté jsou vytvořeny instance GA a SPO. Pro bližší informace o tom, jak jsou vytvořeny instance GA a SPO, si lze přečíst programovou dokumentaci konstruktoru tříd. Tyto instance jsou předány instanci třídy `Solver`, který se postará o korektní průběh a ukončení. Nakonec je výsledek vypsán. Hlavní metoda vypadá takto:

```
def main():
    qm_func = read_qm_func_by_columns("qm.dat", 2)

    ga = GeneticAlgorithm(...)

    spo = SPO(...)
```

```
    solver = Solver(ga, spo)
    solver.solve()
    solver.print_result()

if __name__ == '__main__':
    main()
```

Pro představu, jak vypadá celý funkční skript, se prosím podívejte na přiložené vytvořené příklady v balících `lj_1d`, `lj_3d`, `buck_1d` a `buck_3d`.

A.5.2 Job file

Pro spuštění pod rozhraním MPI nestačí pouze hlavní skript, ale je třeba vytvořit tzv. „job file“. V tomto souboru je definováno, kolik procesorů je pro spouštěný program požadováno (podle této hodnoty musí uživatel nastavit velikosti populací, aby byly všechny procesy rovnoměrně zatíženy při výpočtu ohodnocení). Dále je definováno v jaké složce výpočet začíná, přesměrování `stdout` a `stderr` do souborů a příslušná fronta (záleží na výpočetním clusteru). V tomto souboru jsou definovány tyto věci (tento soubor byl použit při vývoji):

1. Shebang - řádek na začátku skriptu začínající znaky křížek a vykřičník, za nimiž (po případné mezeře) je jméno programu, kterým má být skript interpretován
2. Definice pracovní složky - aktuální složka je pracovní
3. Přesměrování `stdin` do souboru
4. Přesměrování `stderr` do souboru
5. Fronta ve výpočetním clusteru
6. Počet požadovaných procesorů
7. Spuštění Python interpretu s hlavním skriptem pod MPI

A takovýto soubor vypadá takto:

```
1 #! /usr/bin/bash
2 #$ -cwd
3 #$ -o out
4 #$ -e err
5 #$ -q sprkkr.q
```

```
6 #$ -pe mpi 100
7 mpirun -np 100 python3 main.py
```

A.5.3 Spuštění pod MPI v clusteru

Výpočet je v clusteru spuštěn příkazem `qsub job_file`, pro zjištění běžících výpočtů uživatele je použit příkaz `qstat` a pro vymazání je použit příkaz `qdel job_id`.

B Obsah přiloženého CD

- DP_Mazin_A18N0096P.pdf - PDF diplomové práce
- zdrojove_kody/qm2md/ - zdrojové kódy vytvořeného programového vybavení
- zdrojove_kody/latex/ - zdrojové kódy textu DP
- funkcni_ukazka/pokyny.txt - pokyny pro možnost předvedení funkčního programu
- Poster/Mazin_Vit_2020.pdf - PDF soubor s posterem
- Poster/Mazin_Vit_2020.pub - Microsoft Publisher soubor s posterem