UNIVERZITA PARDUBICE
Fakulta elektrotechniky a informatiky

Riešenie multikriteriálneho Cutting Stock Problému pomocou metód evolučných algoritmov

Martin Zamba

Diplomová práca
2013

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2012/2013**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Martin Zamba**
Osobní číslo:	**I10428**
Studijní program:	**N2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Řešení multikriteriálního Cutting Stock Problému pomocí metod evolučních algoritmů**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování:

1.Práce řeší problém multikriteriální optimalizace výtěže pořezu dřevní kulatiny pomocí evolučních výpočetních technik.
2.Úvodem práce je třeba provést rešerši odborné literatury týkající se řešení obdobných problémů v technologické praxi se zaměřením na řešení pomocí evolučních algoritmů.
3.Dále je třeba prostudovat modelový problém a navrhnout alternativy řešení problému nejprve jako monokriteriální a následně jako multikriteriální úlohy.
4.Navržené varianty řešení je třeba implementovat v programovacím jazyku JAVA.
5.Použitelnost navržených řešení je třeba ověřit pomocí testovacích scénářů obvyklých při Cutting Stock / Strip Packing problémech. Získaná data je třeba vyhodnotit v přehledné grafické a tabulkové formě.
6.Výstupem práce bude textová dokumentace vypracovaná podle příslušných vyhlášek fakulty s přihlédnutím k normě ČSN ISO 690 a médium se programovým řešením problému.

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1.MACH, M. Evolučné algoritmy. Prvky a princípy.Košice : Elfa, 2009. 250 s. ISBN 978-80-8086-123-0.
2.SHARMA R.; BALACHANDER T.; MCCORD Ch.; ANAND S.; ZHANG Q. Genetic Algorithm for the Non-convex Cutting Stock Problem [online].University of Cincinnati : Computer Aided Manufacturing Laboratory. Dostupné na: http://citeseerx.ist.psu.edu/viewdoc/versions?doi=10.1.1.41.6583.
3.ILLICH S.; WHILE L.; BARONE L. Multi-objective Strip Packing Using an Evolutionary Algorithm [online]. IEEE Congress on Evolutionary Computation, 2007. Dostupné na: http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=4425020
4.COELO C.; LAMONT G. Applications of Multi-Objective Evolutionary Algorithms - Advances In Natural Computation Vol.1, Singapore (Singapúr) : World Scientific Publishing Co. Pte. Ltd., 2004. ISBN 981-256-106-4

Vedoucí diplomové práce: **Ing. Petr Doležel, Ph.D.**
Katedra řízení procesů

Datum zadání diplomové práce: **31. října 2012**
Termín odevzdání diplomové práce: **17. května 2013**

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

prof. Ing. Antonín Kavička, Ph.D.
vedoucí katedry

V Pardubicích dne 15. listopadu 2012

Prehlásenie autora

Prehlasujem, že som túto prácu vypracoval samostatne. Všetky literárne pramene a informácie, ktoré som v práci využil, sú uvedené v zozname použitej literatúry.

Bol som oboznámený s tým, že sa na moju prácu vzťahujú práva a povinnosti vyplývajúce zo zákona č.121/2000 Zb., autorský zákon, hlavne so skutočnosťou, že Univerzita Pardubice má právo na uzavretie licenčnej zmluvy o použití tejto práce ako školného diela podľa § 60 odst. 1 autorského zákona, a s tým, že ak dôjde k použitiu tejto práce mnou, alebo bude poskytnutá licencia o použití inému subjektu, je Univerzita Pardubice oprávnená odo mňa požadovať primeraný príspevok na úhradu nákladov, ktoré na vytvorenie diela vynaložila, a to podľa okolností až do ich skutočnej výšky.

Súhlasím s prezenčným sprístupnením svojej práce v Univerzitnej knižnici.

V Pardubiciach dňa 17. 5. 2013 Martin Zamba

Poďakovanie

Na tomto mieste chcem poďakovať môjmu konzultantovi Petrovi Doleželovi za cenné rady a pripomienky pri vypracovaní tejto práce. Ďalej moje poďakovanie patrí mojej rodine a mojej priateľke za srdečnú podporu počas celého môjho štúdia.

Abstrakt

Táto diplomová práca sa zaoberá použiteľnosťou genetických algoritmov pre dvoj dimenzionálny non-guillotineable Cutting Problem v drevárskom priemysle – 2D non-guillotineable Log Cutting Problem (ngLCP). V práci sú prezentované dva nové prístupy ku konštrukcii riešenia pomocou konštrukčnej heuristiky pre ngLCP a tieto sú následne dôkladne preverené. V práci je ďalej prezentovaná inovatívna fitness funkcia pre ohodnocovanie porezových plánov, ktorá sa zameriava na pomoc evolučnému procesu ku konvergencii k vhodnému riešeniu ngLCP. Kvôli nedostatku literatúry pojednávajúcej o LCP, alebo podobnom probléme a následnej nedostupnosti vhodných testovacích scenárov, boli navrhnuté testovacie scenáre ktoré obsahujú 15 reálnych aj syntetických kontajnerov a 5 sád malých objektov. Tieto scenáre sú následne použité k overeniu a porovnaniu navrhovaných riešení medzi sebou. Práca takisto zvažuje multikriteriálnu optimalizáciu kde je okrem výťaže optimalizovaná aj kvalita reziva

Kľúčové slová

dvoj-rozmerný cutting stock problem, genetický algoritmus, drevná guľatina, rezivo

Title

Solving of Multicriterial Cutting Stock Problem using Evolutionary Computation methods

Abstract

This diploma thesis deals with the usability of genetic algorithms for two dimensional non-guillotine Cutting Problem in wood industry – 2D non-guillotineable Log Cutting Problem (ngLCP). Two novel approaches for construction heuristics of solutions to ngLCP are presented and thoroughly tested. Novel fitness function for evaluation of cutting patterns is presented which aims to help evolution process in finding the right solution for ngLCP. Because of lack of literature about Lumber Cutting Problem or similar problem and resulting lack of testing scenarios, we propose new set of 15 real and synthetic containers and 5 small item sets. These scenarios were later used to compare proposed solution methods between each other. The thesis also consider multicriterial optimization where alongside of utilization the lumber quality is also optimized

Keywords

two-dimensional cutting stock problem, genetic algorithm, lumber, round log

Obsah

Zoznam skratiek

CSP	Cutting Stock Problem
2DCSP	2D Cutting Stock Problem
C&P	Cutting & Packing problems
SPP	Strip Packing Problem
PP	Packing Problem
CP	Cutting Problem
RP	Rectangle Packing
LCP	Log Cutting Problem
ngLCP	non-guillotineable Log Cutting Problem
TS	Tabu Search
GRASP	Greedy Randomized Adaptive Search Procedure
ACO	Ant Colony Optimization
SA	Simulated Annealing
EA	Evolutionary Algorithm
GA	Genetic Algorithm
MOGA	Multi Objective Genetic Algorithm
BCX	Bernoulli Crossover (recombination operator)
BLCX	Bernoulli Local Crossover (recombination operator)
OPSO	One Point Swap Order (mutation operator)
RA	Random Angle (mutation operator)
UA	Uniform Angle (mutation operator)
RH	Random Head (mutation operator)
RIP	Random Insert Point (mutation operator)
RPO	Random Placement Orientation (mutation operator)
RR	Random Rotation (mutation operator)

Zoznam obrázkov

Zoznam tabuliek

Úvod

Aj napriek neustálemu nárastu výpočetnej sily techniky existujú úlohy na ktoré surový výkon nikdy stačiť nebude. Jedná sa o hlavne o rozsiahle kombinatorické a optimalizačné úlohy patriace do skupinu NP-ťažkých problémov. Tieto úlohy nie je možné riešiť žiadnou klasickou výpočetnou metódou v polynomiálnom čase, t.j. v čase uspokojivom. Typickými vlastnosťami takýchto úloh sú:

- stavový priestor je často enormný a vzhľadom na potrebné výpočetné prostriedky ho nie je možné kompletne prehľadať
- neznalosť závislostí jednotlivých premenných neumožňuje redukovať stavový priestor
- zložitosť a viac rozmernosť problému

V priemysle je zastúpenie takýchto úloh obzvlášť bohaté od plánovania úloh (Job Shop Scheduling), cez plánovanie rozloženia VLSI čipov (VLSI Floorplaning) až k obzvlášť bohatej rodine Cutting & Packing problémov (C&P). CP (Cutting Problems) problémy zahŕňajú všetky úlohy u ktorých je potrebné rozdeliť isté množstvo materiálu na menšie kusy pričom je nutné minimalizovať odpad. Pri PP problémoch (Packing Problems) nutné využiť daný priestor čo najlepšie. U obidvoch typov problémov sa však jedná o úlohu šetriť s dostupnými zdrojmi a využiť ich na maximum a tak často tieto dva typy úloh splývajú.

Pre riešenie C&P problémov a iných ťažkých úloh bola vyvinutá celá rada meta-heuristických techník ako napríklad tabu prehľadávanie (Tabu Search), GRASP – Greedy Adaptive Search Procedure, optimaliz8cia pomocou mravenčích kolónií (Ant Colony Optimization), simulované žíhanie (Simulated Annealing) a v neposlednom rade genetické algoritmy (Genetic Algorithms). Rodina C&P problémov je veľmi rozsiahla a každá jej odnož je zvyčajne viazaná na istú aplikáciu v priemysle, pričom požiadavky sa rôznia. Pokusy o návrh univerzálneho prístupu k riešeniu C&P problémov existujú (MANCAPA), avšak takýto univerzálny prístup si prináša nutnosť kompromisov. Keď potom nastane požiadavka nasadiť takýto prístup do reálneho prostredia, zvyčajne sa prijaté kompromisy prejavia negatívne na výkone takéhoto systému, alebo nedostatočnou kvalitou riešení.

Cieľom tejto práce preto nie je navrhnúť, či vybudovať akýsi univerzálny systém na riešenie C&P problémov, ale demonštrovať použiteľnosť genetických algoritmov (GA) na reálnom probléme. Za konkrétnu problematiku z priemyslu bola zvolená problematika optimalizácie porezu guľatiny (Log Cutting Problem - LCP), konkrétne variant ngLCP – non-guillotineable LCP. Jedná sa o ťažký optimalizačný multikriteriálny problém z prostredia drevárskeho priemyslu Dôvodom na výber tejto konkrétnej problematiky je relatívne malá preskúmanosť tejto oblasti, vysoká náročnosť problému a v neposlednom rade autorove osobné zameranie a skúsenosti v danej oblasti.

1 Metódy riešenia C&P problémov

Pre riešenia malých inštancií C&P problémov boli vyvinuté deterministické metódy ako napríklad metódy využívajúce lineárne programovanie (GILMORE, a iní, 1963), (HAESSLER, a iní, 1991), (GILMORE, a iní, 1965). Problém týchto metód je fakt, že sa jedná o deterministický prístup k riešeniu. Na jednej strane síce determinizmu prináša zaručené výsledky, lenže s rastúcou komplexnosťou problému rastie priestor prehľadávania často faktorom $O(n!)$ a tak pri väčších inštanciách tieto metódy zlyhávajú z časového hľadiska.

Pre praktické riešenie C&P problémov začali vznikať rôzne heuristiky neumožňujúce síce zaručiť nájdenie optimálneho riešenia, ale umožnia nájdenie riešenia, ktoré bude takmer optimálne v rozumnom čase. Pomerne dlho existujú heuristiky riešiace C&P problémy ktoré sú založené na lineárnom programovaní, sekvenčné heuristiky a hybridné metódy (KARELAHTI, 2002). Problémom u nich však je, že trpia spoločnou vlastnosťou uviaznuť v lokálnom optime riešenia.

V polovici 70.rokov minulého storočia začali vznikať meta-heuristiky, ktoré používajú rôzne prístupy k tomu, aby vyviedli klasickú heuristiku z lokálneho maxima. Medzi najznámejšie takéto metódy patria:

Tabu Search (TS) – tabu prehľadávanie. Zakladá na generovaní susedných riešení. V každej iterácii sa prejde k najlepšiemu susednému riešeniu aj v prípade, že toto riešenie je horšie ako aktuálne. Toto dovoľuje neuviaznuť v lokálnom optime (KARELAHTI, 2002). Algoritmus si udržiava tzv. Tabu List, v ktorom ukladá už prehľadané riešenia. Je zakázané vrátiť sa k už prehľadanému riešeniu. Algoritmus končí po určitom počte iterácií. Príklad TS algoritmu je možné nájsť v (ALVAREZ-VALDÉZ, et al., 2002).

GRASP (Greedy Randomised Adaptive Search Procedure) je iteratívny algoritmus v ktorom v každej iterácii sú vykonávané konštruktívna a vylepšovacia fáza. V konštrukčnej fáze je riešenie budované krok za krokom pridávaním elementov k čiastočnému riešeniu. Pri vyberaní elementu, ktorý bude pridaný k čiastočnému riešeniu sa používa tzv. greedy (v preklade: hltavý, hrabavý) funkcia. Táto funkcia je vypočítavaná a adaptovaná počas konštrukcie čiastkového riešenia. Greedy heuristika je taká, ktorá za účelom dosiahnutia globálneho optima sa v každom kroku snaží dosiahnuť lokálne optimum. Výberový proces elementu nie je deterministický, ale naopak náhodný a tak pri viacnásobnom spustení konštrukčného kroku dostaneme rôzne riešenia. Zlepšovacia fáza pozostáva z lokálneho prehľadávania, ktoré sa snaží modifikovať vykonštruované náhodné riešenie s cieľom dosiahnuť jeho lepšiu vhodnosť (ALVAREZ-VALDÉZ, et al., 2004). S cieľom uniknúť lokálnemu extrému, algoritmus reštartuje hľadanie z ďalšieho prísľubného regiónu prehľadávacieho priestoru po tom, ako bol nájdený lokálny extrém. GRASP algoritmus bol prvýkrát prezentovaný v neskorých 80.rokoch minulého storočia autormi Feo a Resende (KARELAHTI, 2002) (ALVAREZ-VALDÉZ, et al., 2004).

ACO (Ant Colony Optimization) – optimalizácia použitím mravenčích kolónií. Táto metóda sa začala objavovať začiatkom 90.rokov minulého storočia. Hlavná idea v pozadí tejto metódy spočíva v schopnosti mravcov nájsť najkratšiu cestu medzi zdrojom potravy a hniezdom. Najkratšia cesta je postupne nájdená kvôli schopnosti mravcov zanechávať pri svojom pohybe feromónovú stopu. Mravec má tendenciu vybrať si cestu s veľkou koncentráciou feromónov. Ak existujú dve cesty k potrave s rovnakou koncentráciou feromónov, postupom času sa naakumuluje viac feromónov na kratšej ceste, keďže ju prejde rovnaký počet mravcov viac krát v rovnakom čase. V konečnom dôsledku je mravcami vybraná kratšia cesta. Príklad riešenia C&P problémov pomocou ACO je možné nájsť v (LEVINE, et al., 2003).

Genetické algoritmy (GA) – sú podtypom evolučných algoritmov. Sú to meta-heuristické algoritmy, ktoré v kontraste k prezentovaným heuristikám nespoliehajú na konštrukčné a lokálne prehľadávacie heuristiky. Namiesto toho GA pracujú so súborom *P* náhodne vygenerovaných riešení. Súbor *P* sa v GA nazýva populácia. Všetky potenciálne riešenia problému sú v *P* zastúpené v podobe sekvencií (vektorov) elementov, ktoré napodobňujú genetickú informáciu živých organizmov - chromozómy. Nové riešenia sú kontinuálne konštruované zo stávajúcej populácie *P*. Tieto riešenia sa označujú ako potomkovia a získavajú sa z množiny jedného a viac *rodičovských* riešení prostredníctvom aplikovania genetických operátorov. Takýmito genetickými operátormi sú najmä kríženie a mutácia. Vytváraním selekčného tlaku rodičov, ktorý vytvára simulované podmienky pre súťaž ako vrámci biologického ekvivalentu evolúcie je dosiahnutá postupná konvergencia riešenia smerom ku globálnemu optimu. Problém uviaznutia v lokálnom optime je zmenšený divergenciou riešení obsiahnutých vrámci populácie. Príklad riešenia problémov C&P je možné nájsť v (ANAND, a iní, 1999), (ILLICH, a iní, 2007), (COELO, a iní, 2004).

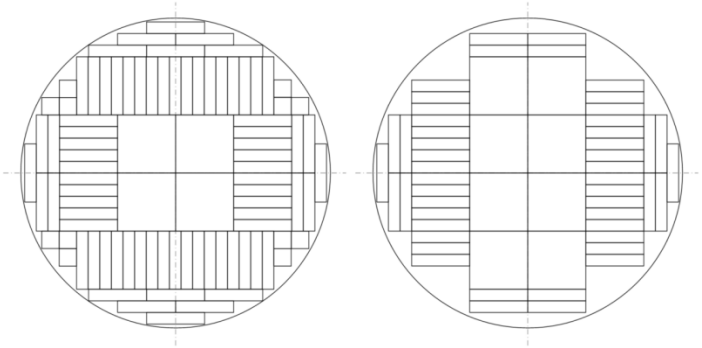
2 Predstavenie problému LCP

Táto práca sa zaoberá aplikáciou genetických algoritmov na konkrétny multikriteriálny problém z rodiny C&P problémov. Za tento problém bol zvolený ngLCP – non-guillotineable Log Cutting Problem ktorý pojednáva o problematike vhodného rozvrhnutia porezu drevnej guľatiny a rovnako ako u ostatných C&P problémov sa u LCP jedná o minimalizáciu odpadu pri spracovávaní zdrojového materiálu.

LCP problém je problém vyrezania istého počtu štvorcových objektov (dosky, hranoly, laty) (WALKER, 2006) z kmeňa ktorý má nedokonalý tvar pripomínajúceho zrezaný kužeľ. Zvyčajná požiadavka je minimalizácia odpadu (odrezkov) v procese rezania ako aj maximálna možná kvalita reziva. Úloha LCP je komplikovaná oproti klasickým C&P problémom faktom, že je nutné uvažovať vnútornú stavbu kmeňa ktorá je nehomogénna. To zapríčiňuje, že niektoré typy reziva ako sú nábytkárske dosky je vhodné vyrezať z oblasti okolo krajov kmeňa tak, aby boli letokruhy v ideálnom prípade kolmo na dlhú stranu dosky, zatiaľ čo stavebné hranoly je vhodné vyrezať zo stredu kmeňa.

V súčasnosti je problém LCP bežne riešený v priemysle proprietárnym softvérom veľkých výrobcov píliarskej technológie ako je napr.Primultini. Avšak musela byť prijatá rada zjednodušení na to aby tento problém bolo bez problémov možné riešiť. Hlavným zjednodušením je, že najbežnejšia technológia používaná na porez guľatiny je pásová píla (WALKER, 2006), ktorá je schopná vykonávať iba gilotínové rezy. Ďalšie zjednodušenie je, že kmeň sa často považuje za dokonale presný zrezaný kužeľ. Takto zjednodušený problém môže byť relatívne jednoducho vyriešený ako viacstupňový 1D CSP (Cutting Stock Problem). Oblasť 1D CSP bola doširoka študovaná v 60 a 70. rokoch minulého storočia a existuje rada deterministických metód na jeho vyriešenie (GILMORE, a iní, 1963) (GILMORE, a iní, 1965)

Táto práca adresuje LCP problém – konkrétne variantu ngLCP pre technológiu uhlového porezu ktorá z princípu vykonáva negilotínové rezy. Zatiaľ čo tento spôsob umožňuje dosiahnuť väčšiu verzatilitu a výťaž, optimalizačný problém návrhu porezového plánu pre túto technológiu je omnoho náročnejší. Vstupom je sada objektov ktoré majú byť vyrezané a nekonvexný polygón reprezentujúci profil kmeňa ktorý je považovaný za nekónický. Tento problém bude adresovaný ako 2D ngLCP.



Obr. 1 – Príklad riešenia LCP ako ngLCP (vľavo) a guillotineable LCP (vpravo)

2.1 Klasifikácia úlohy

U LCP sa jedná sa o špeciálny prípad kombinatorických optimizačných úloh (DYCKHOFF, 1990), ktorý je známy z rôznych odvetví priemyselného inžinierstva. Na problém LCP sa je možné pozerať z dvoch uhlov pohľadu z čoho vyplýva následné zaradenie podľa Dyckhoffovej Typológie (DYCKHOFF, 1990) a Vylepšenej Typológie Cutting a Packing Problémov (C&P) (WÄSCHER, a iní, 2007).

Ak sa na úlohu pozeráme ako na úlohu optimálne s ohľadom na výťažnosť a kvalitu reziva rozložiť množinu výrezov na jednom kmeni, tak sa jedná podľa Dyckhoffovej Typológie o problém 2/B/O/C. Toto označenie znamená, že sa jedná o dvojrozmerný problém umiestnenia malých kongruentných objektov na jeden veľký objekt. Kongruentných z dôvodu, že všetky výrezy sú v priereze obdĺžnikového tvaru. Podľa Vylepšenej Typológie C&P sa jedná o SLOPP problém (Single Large Object Placement Problem) (WÄSCHER, a iní, 2007). Tento typ popisuje skupinu problémov s jedným veľkým objektom, ktorého rozmery sú fixné a úlohou je umiestniť časť, alebo všetky objekty z množiny malých slabo heterogénnych objektov (čiže množiny objektov s malou variáciou).

Z pohľadu druhého je možné sa na úlohu pozerať ako na úlohu v ktorej je potrebné splniť zákazky s čo najmenším porezom na sklad a aby boli všetky zákazky splnené v danom zložení do určitého časového ohraničenia. Zároveň je potrebné dosiahnuť vysokú výťažnosť a kvalitu reziva. V tomto prípade na rozdiel od predchádzajúceho je nutné uvažovať s niekoľkými veľkými heterogénnymi objektmi – kmeňmi a množinou malých objektov – sortimenty na zákazkach. Klasifikácia podľa Dyckhoffovej Typológie je v tomto prípade 2/V/D/C. Čiže dvoj-rozmerná úloha so súborom veľkých objektov rozdielneho tvaru, z ktorých je možné vybrať pre splnenie kritéria najvhodnejšie kusy (čiže je prípustné, že sa nepoužijú všetky veľké objekty), na ktoré je potrebné optimálne umiestniť súbor malých kongruentných objektov. Podľa Vylepšenej Typológie C&P sa jedná o MHLOPP problém (Multiple Heterogeneous Large Object Placement Problem. Tento typ popisuje rovnakú úlohu ako SLOPP, len s tým že je použitý súbor veľkých heterogénnych objektov.

2.2 Rešerš literatúry

Pri rešerši literatúry nebol na prekvapenie autora nájdený žiadny článok, ktorý by sa priamo dotýkal optimalizácie porezu guľatiny (LCP). Vo všeobecnosti bolo problematické nájsť publikáciu riešiacu problém podobný problém ako LCP. Bola nájdená iba jedna práca (ANAND, a iní, 1999) (SHARMA, a iní), ktorá sa zaoberá aplikáciou v kožiarenskom priemysle a ktorá sa vzdialene podobá na problém LCP. Táto práca sa zaoberá optimalizáciou rozloženia malých polygonálnych objektov na jeden veľký polygonálny objekt (kus kože). Práve z tejto práce bola nakoniec čerpaná najväčšia inšpirácia pri návrhu GA pre riešenie LCP.

3 Genetické algoritmy

Genetické algoritmy sú optimizačné a prehľadávacie meta-heuristické algoritmy inšpirované evolučnými procesmi prebiehajúcimi v prírode. Sú založené na Darwinovej teórii evolúcie a simulujú prirodzený výber a genetické procesy. Sú veľmi vhodné na nasadenie u náročných problémov kvôli svojej robustnosti a univerzálnosti. Dokážu bez problémov riešiť aj problémy o ktorých nie sú známe žiadne závislosti ani vnútorné interakcie. Pracujú na princípe prežitia najlepšieho jedinca. Kvalita každého jedinca je hodnotená tzv.fitness funkciou, ktorá by mala byť navrhnutá v zmysle dosahovania lepších hodnôt v prípade lepších riešení.

Základná schéma genetického algoritmu sa dá zhrnúť v krokoch:

1. **Inicializácia** pri ktorej sa vygeneruje prvotná populácia, často náhodným spôsobom, ale existujú aj rôzne sofistikované metodiky pre vytvorenie prvotnej populácie.
2. **Evaluácia** je proces ktorý býva často výpočtovo najnáročnejší z celého GA. Tu totiž dochádza k dekódovaniu riešenia v oblasti reprezentácie GA, tzv.genotypu na reprezentáciu aplikačnú, tzv.fenotyp.
3. **Ohodnotenie** prebieha pomocou hodnoty jednej fitness funkcie u obyčajného algoritmu, avšak u miltikriteriálnych úloh, keď je fitness funkcií väčší počet sa používajú rôzne sofistikované techniky ohodnotenia.
4. **Selekcia** je aplikáciou mechanizmu selekčného tlaku prostredia na populáciu jedincov. Tu existuje niekoľko prístupov ktoré sú uvedené v špeciálnej podkapitole venovanej selekcii
5. **Kríženie** je rekombinácia genetického materiálu rodičov a z neho vytvorenie ich potomkov. Existujú viaceré štandardné operátory kríženia ako sú One Point Crossover, Two Point Crossover, Bernoulli Crossover - BCX, ktoré sú vhodné pre štandardné úlohy, alebo operátory ako Partial Match Crossover – PMX, Order Crossover – OX, ktoré sú vhodné pre permutačné kódovanie jedincov.
6. **Mutácia** je mechanizmom pre vnášanie nového genetického materiálu do populácie. Zvyčajne sú tieto operátory závislé na konkrétnej reprezentácii jedincov
7. **Obnova populácie** sú mechanizmy ako je elitizmus ktorý zabezpečuje, že najlepší jedinec nebude stratený, alebo imigrácia, ktorá prináša nových (náhodných) jedincov do populácie
8. Ak nebola dosiahnutá podmienka ukončenia pokračuj bodom 2

U genetických algoritmov sa štandardne používa binárne zakódovanie jedincov, avšak často sa používa tiež permutačné zakódovanie. U niektorých problémov však použitie binárnej reprezentácie prináša radu komplikácií a tak sa používajú mechanizmy ako Grayovo zakódovanie. Ako však bolo ukázané, tak pre niektoré aplikácie je najvhodnejšie reálne zakódovanie jedincov (MICHALEWICZ, 1996), aj keď neexistujú štandardné operátory kríženia pre toto zakódovanie a preto musia byť často vytvorené podľa konkrétnych špecifík úlohy.

Najväčším problémom GA je často veľmi náročné hľadanie správnej reprezentácie jedinca a implementácia dekódovania genotypu na fenotyp.

3.1 Metodiky selekcie

Existuje niekoľko metodík ktoré pristupujú rôzne k procesu selekcie jedincov pre reprodukciu:

1. **Proporcionálna selekcia** v tomto prípade je pravdepodobnosť výberu jedinca pre účely reprodukcie priamo úmerná jeho ohodnoteniu. Toto je jedna z najzákladnejších techník, ktorá však niekedy trpí problémom, keď sa v populácii nachádzajú jedince s príliš rozdielnym ohodnotením. V bude jedinec s malým ohodnotením úplne potlačený aj keď by inak mohol byť vhodným kandidátom na reprodukciu
2. **Rank-based selekcia** eliminuje problém proporcionálnej selekcie zoradením jedincov podľa ohodnotenia a zostupným pridelením ohodnotenia – ranku úmernému poradiu. Následne sa selekcia vykonáva s pravdepodobnosťou proporcionálnou k tomuto ranku
3. **Turnajová selekcia** nepožaduje úplné zotriedenie populácie apriori. Dokonca ani nepožaduje kvantitatívne ohodnotenie fitness funkciou. Z populácie o N jedincoch sa vyberie náhodne t jedincov medzi ktorými sa vzájomným turnajom určí najlepší ako výsledok selekcie

3.2 Multikriteriálne GA

Klasické genetické algoritmy pracujú s jedným ohodnotením jedinca pomocou fitness funkcie. Ak vyvstane požiadavka na viackriteriálne ohodnotenie jedincov, tak väčšinou je treba použiť iný mechanizmus ako u klasického GA. Najjednoduchšou variantou multikriteriálneho GA je obyčajný GA, ktorý používa váhovanú fitness funkciu:

$$ff = w_1 \times f_1 + w_2 \times f_2 + \cdots + w_n \times f_n \quad (1)$$

Kde ff je výsledná fitness funkcia a w_i sú váhy pri jednotlivých hodnotách čiastkových ohodnotení fitness. Takéto riešenie ale s pravidla neprinesie kompromisné riešenia problému ktorý GA rieši. Často je problematické nastavenie správnych hodnôt váh.

Preto bol zavedený sofistikovanejší prístup k ohodnoteniu jedincov pomocou tzv. Pareto dominanancie.

Riešenie $u = (u_1 \ldots u_n)$ je **dominované** vektorom $v = (v_1 \ldots v_n)$ práve vtedy keď je u čiastocne menšie ako v:

$$\forall i = 1..n, v_i \leq u_i \ \& \ \exists i = 1..n : \ v_i < u_i \quad (2)$$

Riešenie je v je **dominujúce** riešeniu u práve vtedy keď u je dominované riešením v. Riešenia u a v sú si navzájom **nedominujúce**, ak v nie je dominované ani dominujúce u. **Pareto front** je množina navzájom si nedominujúcich riešení.

Pre multikriteriálnu optimalizáciu existuje veľmi veľké množstvo rôznych prístupov (TAN, a iní, 2005). Z nich najzákladnejšie a najstaršie sú algoritmy VEGA a MOGA

3.2.1 Algoritmus VEGA

Tento algoritmus pracuje na princípe klasického genetického algoritmu pričom nepracuje s Pareto dominanciou. Populáciu ohodnotí fitness funkciami osobitne a mechanizmus selekcie sa pre polovicu budúcej populácie vykoná pre ohodnotenie podľa jednej fitness funkcie a pre druhú polovicu podľa druhej fitness funkcie. Tento algoritmus stál na začiatku vývoja miltikriteriálnych GA a inšpiroval k vývoju GA pracujúcich s viacerými nedominujúcimi si riešeniami.

3.2.2 Algoritmus MOGA

Bol prezentovaný v (FONSECA, a iní) v roku 1993. Využíva pareto dominanciu a na základe nej prideľuje jedincom tzv. rank:

$$rank(i) = 1 + \ q_i \tag{3}$$

Kde q_i je počet jedincov ktorí danému jedincovi dominujú. Získaný rank jedinca je prevedený v ďalšom kroku na fitness funkciu inverzným mapovaním funkciou $inver()$, pre ktorú platí:

$$\begin{gathered} fit(i) = inver\big(rank(i)\big) \\ \big(rank(i) < rank(j)\big) \Leftrightarrow \big(rank(i) < rank(j)\big) \end{gathered} \tag{4}$$

Následne sa táto fitness poníži o hodnotu úmernú počtu jedincov ktoré sú dostatočne blízko daného jedinca. Nech je vzdialenosť dvoch ohodnotení X v priestore ohodnotenia daná ako

$$d\big(X_i, X_j\big) = \ ||X_i - X_j||_2 \tag{5}$$

A hodnota sharing funkcie:

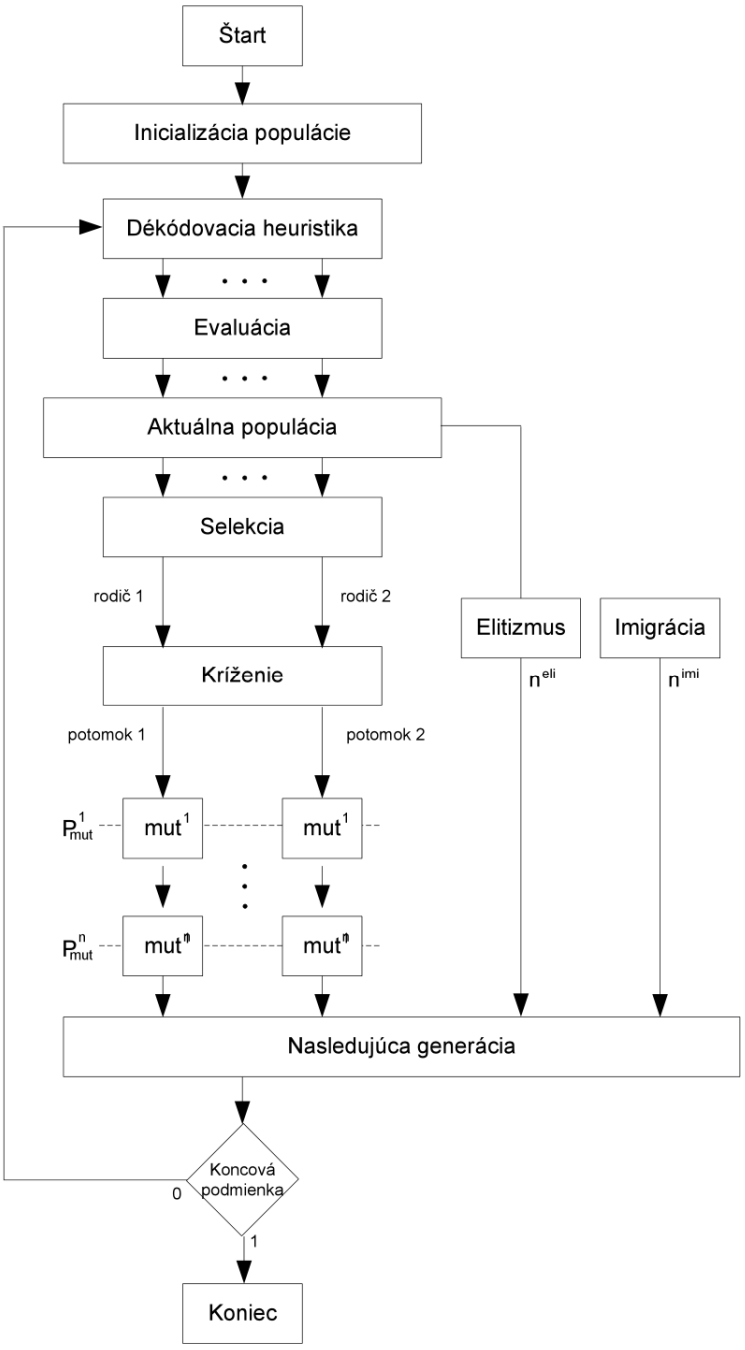
$$SF(i,j) = f(x) = \begin{cases} 1 - \left[\dfrac{d\big(X_i, X_j\big)}{\sigma_{share}}\right]^{\alpha}, & ak \ < d\big(X_i, X_j\big) < \sigma_{share} \\ 0, & inak \end{cases} \tag{6}$$

Potom je výsledná hodnota fitness funkcie:

$$fit'(i) = \frac{fit(i)}{\sum_{j=1}^{n} SF(i,j)} \tag{7}$$

4 Návrh riešenia úlohy

Podľa rešerše literatúry a takisto to je aj autorovo skutočné presvedčenie, neexistuje hotový algoritmus ktorý by dokázal riešiť problém ngLCP (non-guilotinable Log Cuting Problem). Ak takýto algoritmus existuje, tak neboli o ňom nájdené žiadne verejne dostupné články ani iné informácie. V princípe to znamená, že táto práca je v podstate prvým pokusom o riešenie problému LCP. Prístup k riešeniu LCP pomocou genetických algoritmov (GA) bol zvolený kvôli inherentnej univerzálnosti GA a schopnosti riešiť komplexné problémy.



Obr. 2 – Diagram zachycujúci spoločnú štruktúru pre navrhované genetické algoritmy

V rámci tejto práce boli navrhnuté dva genetické algoritmy označené GA1 a GA2 vychádzajúce z rovnakej štruktúry, ale využívajúce rozličné zakódovania jedincov a značne odlišné prístupy k dekódovaniu genotypu na fenotyp. Obidva navrhované genetické algoritmy využívajú pre toto dekódovanie unikátnu konštruktívnu heuristiku. Pre algoritmus GA1 bude kvôli overeniu multikriteriálnej optimalizácie implementovaná varianta MOGA-GA1, ktorá využíva princíp algoritmu MOGA (FONSECA, a iní)

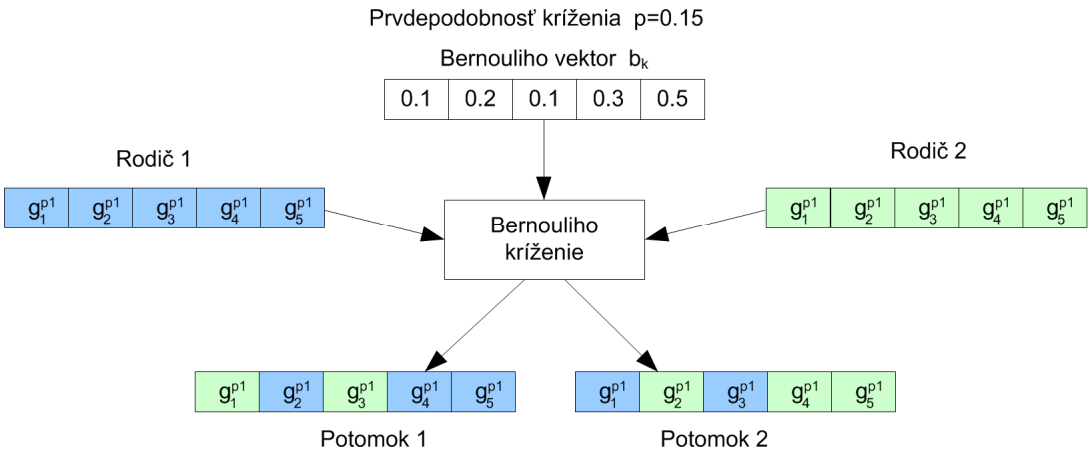
Spoločná štruktúra genetického algoritmu je uvedená na obrázku Obr. 2. V princípe obsahuje všetky elementy normálne sa nachádzajúce u genetických algoritmov. Na začiatku behu sa náhodne vygeneruje prvotná populácia. Následne prebehne dekódovacia heuristika unikátna pre každý algoritmus a ohodnotenie jedincov. V každej iterácii je prenesených n_{eli} najlepších jedincov z aktuálnej generácie do nasledujúcej generácie, čím sa zabezpečí uchovanie najlepšieho riešenia nájdeného do daného bodu. Nastavením $n_{eli} = 1$ sa aktivuje klasický elitizmus ako je u GA bežné. Genetickú diverzitu je možné doplniť pomocou parametra $n_{imig} = m$ ktorý zabezpečí náhodné vygenerovanie m jedincov v každej generácii. Pokiaľ sa nenaplní definovaný počet jedincov v populácii, tak operátor selekcie vyberie dvoch rodičovských jedincov ktorí budú podrobení kríženiu operátorom kríženia BCX (Bernouli Crossover) (Obr. 3). Následne potomkovia prejdú radom mutácií s definovanými pravdepodobnosťami pomocou mutačných operátorov, ktoré sú vo väčšine unikátne pre daný GA.

Pre algoritmy GA1 a GA2 je použitá ruletová selekcia s ohodnotením jedincov pomocou relatívneho poradia v zoradenej postupnosti jedincov podľa hodnoty fitness funkcie – tzv. Rank Based Selection. U MOGA-GA1 je použitá ruletová selekcia s ohodnotením proporcionálnym k ohodnoteniu jedinca ohodnocovacieho pomocou mechanizmu MOGA (3.2.2)

U navrhovaných genetických algoritmov je používané reaálne zakódovanie oproti zvyku u GA používať kódovanie binárne. V práci (MICHALEWICZ, 1996)-str.97 bolo ukázané, že reaálna reprezentácia ak je pre problém prirodzená značne zvyšuje výkon genetického algoritmu. Toto je u navrhovaných algoritmov značne využívané.

4.1.1 Bernoulliho kríženie – operátor BCX

Spoločným prvkom pre všetky navrhované algoritmy je operátor Bernoulliho kríženia – BCX. Použitie tohto operátora bolo inšpirované prácou (SHARMA, a iní) a bol zavedený v (NORMAN, a iní, 1996). Pri tomto krížení sa s normálnou distribúciou náhodne vygeneruje vektor reálnych čísel b_k z intervalu $< 0: 1 >$. Dĺžka tohto vektoru je rovnaká ako počet génov jedincov vstupujúcich do procesu kríženia ako rodičia. Potomkovia vzniknú vzájomnou výmenou génov rodičov u pozícií kde je hodnota odpovedajúcej pozície vektoru b_k väčšia ako pravdepodobnosť kríženia. Tento proces na príklade rodičov s piatimi génmi a pravdepodobnosťou kríženia $p = 0.15$ je znázornený na Obr. 3.



Obr. 3 – Príklad funkcie Bernoulliho kríženia na jedincoch s 5 génmi a pravdepodobnosťou kríženia p=0.15

4.2 Návrh algoritmu GA1

Algoritmus GA1 bol majoritne inšpirovaný publikáciami (SHARMA, a iní) a (ANAND, a iní, 1999). Reprezentácia problému LCP u tohto algoritmu vychádza z reprezentácie problému v uvedených publikáciách. V konečnom dôsledku sa však reprezentácia značne líši.

Každý jedinec populácie je reprezentovaný vektorom n génov rovnakého typu pozostávajúcich z piatich elementov:

$$gene^{GA1}: (id, x, y, rot, \theta) \tag{8}$$

kde:
id – identifikátor malého objektu ktorý môže nadobúdať hodnoty $1..m$, kde m je mohutnosť množiny malých objektov. Toto číslo jednoznačne určuje rozmer melého objektu.
$x, y \in < 0; 1 >$ – súradnica x,y prvotného bodu vloženia malého objektu
$rot \in \{true, false\}$ – určuje či má byť object otočený alebo nie
$\theta \in < 0; 2\pi)$ – uhol určujúci smer korekcie polohy pre daný objekt

Takáto reprezentácia riešenia sa v pôvodných prácach nazýva pseudo-rozloženie. Toto sa pomocou konštruktívnej heuristiky prevedie na reálne rozloženie, ktoré neobsahuje žiadne pretínajúce sa objekty. Ako je uvedené nižšie, veľkou nevýhodou takéhoto spôsobu konštrukcie riešenia je jeho veľká výpočtová náročnosť.

Počas evolúcie je počet chromozómov n u jedincov fixný a značne ovplyvňuje výkonnosť algoritmu. GA1 používa nasledujúci vzorec pre určenie počtu chromozómov inšpirovaný podľa (ANAND, a iní, 1999):

$$n = \frac{A^{container}}{\frac{1}{m}\sum_{i=1}^{m} A_i^{item}} . \frac{chromosome\_multiplier}{100\%} \tag{9}$$

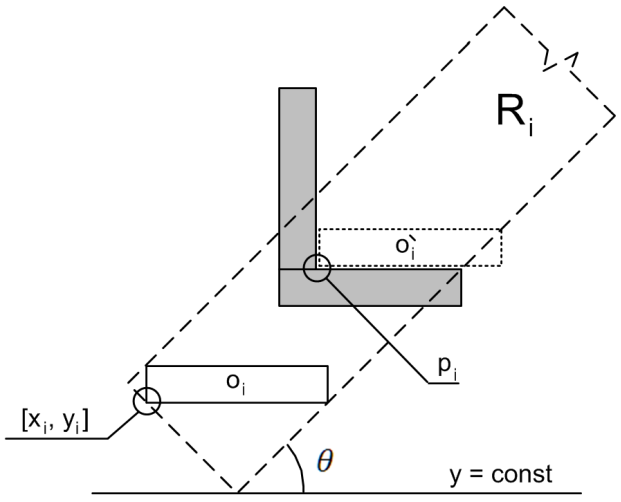
kde:
$A^{container}$ – obsah veľkého objektu (profilu kmeňa)
A_i^{item} – obsah i-teho malého objektu
m – mohutnosť množiny malých objektov
$chromosome\_multiplier$ – parameter algoritmu GA1

Zlomok v rovnici (9) vyjadruje priemerný počet malých objektov, ktoré je možné umiestniť do daného veľké objektu. Experimentálne bolo overené, že počet umiestniteľných objektov dobre korešponduje s touto mierou pre veľké aj malé kmene. Nastavením parametra $chromosome\_multiplier$ je možné ovplyvňovať výkonnosť algoritmu. Pri zvolení malej hodnoty je výkon algoritmu väčší, no klesá optimalita riešenia. Pri nastavení väčších hodnôt dochádza k zvýšeniu optimality, avšak časová náročnosť konštruktívnej heuristiky stúpa.

Každý kandidát riešenia u GA1 predstavuje pseudo-rozloženie v ktorom nie je zabezpečená nutná podmienka LCP a to, že pre všetky malé objekty I_k v rozložení musí platiť:

$$\forall k = 1..n, j = 1..n, k \neq j : B_{ik} \cap B_{ij} = \emptyset, \; B_{ik} = p_{ik} \cap C \tag{10}$$

kde:
i – je index malého objektu v riešení, n je počet malých objektov v rozložení, C je kontajner a B_{ik} je prienik kontajnera s daným objektom p_{ik}. Inými slovami musí byť zabezpečené nepretínanie sa jednotlivých malých objektov vo finálnom rozložení

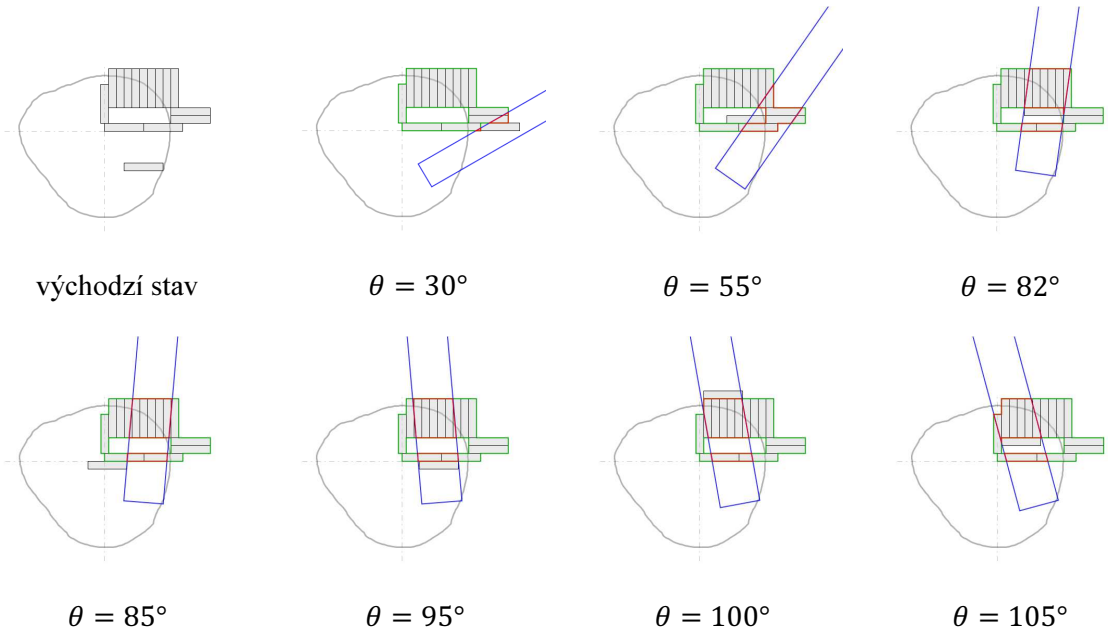


Obr. 4 – Ilustrácia princípu konštruktívnej heuristiky GA1 pre $\theta = \frac{\pi}{8}$

Dekódovanie je pseudo-rozloženia (genotypu) na reálne rozloženie (fenotyp) je prevádzané iteratívne konštruktívnou heuristikou ktorej princíp je ilustrovaný na Obr. 4:

1. placedObjectsPolygon = $\emptyset$
2. for each object $gene_i$:
3. create small object o_i specified by id_i, rot_i
4. place object o_i at $[x_i, y_i]$
5. create such rectangle R_i that its' baseline is tilted against plane y=const by angle θ and it tightly encapsulates placed object o_i from three R_i sides and the last R_i side in direction of angle θ is projected into infinity
6. compute I_i = intersection(R_i, placedObjectsPolygon)
7. find point $p_i \in I_i$ such as distance($[x_i, y_i]$, p_i) = min
8. if such p_i exists then: o_i' = translate(o_i, p_i) else: o_i' = o_i, finish=true
9. compute Io_i' = intersection(intersection(o_i', placedObjectsPolygon), complement(largeObject))
10. if $Io_i' = \emptyset$ then: placedObjectsPolygon = intersection(placedObjectsPolygon, o_i')
11. else: if not(finish) then: remove p_i from I_i, goto (7)
12. end for each
13. *END: placedObjectsPolygon is decoded solution*

Na nasledujú obrázku (Obr. 5) je ukážka uvedenej heuristiky v reálnom algoritme a obrovského vplyvu uhlu vloženia θ na výslednú polohu malého objektu.



Obr. 5 – Ukážka korekcie polohy malého objektu konštruktívnou heuristikou GA1 pre rôzne hodnoty uhlu vloženia θ

Pre algoritmus GA1 boli navrhnuté tri operátory mutácie:

- OPSO – One Point Swap Order Mutation – všeobecne použiteľný operátor pre akékoľvek zakódovanie jedinca. Náhodne sa zvolí index $i \in <1..n>$, kde n je počet génov jedinca: $[g_1, g_2, .., g_i, g_{i+1}, .., g_n]$. Výsledný zmutovaný jedinec má potom tvar: $[g_{i+1}, .., g_n, g_1, g_2, .., g_i]$.
- RA – Random Angle mutation – špeciálne navrhnutý operátor pre GA1. Má za úlohu zmutovať uhol vloženia θ každého génu jedinca s dopredu nastavenou pravdepodobnosťou $p_{mut\theta}$ na náhodnú hodnotu. Pre GA1 bola táto pravdepodobnosť fixne nastavená na $p_{mut\theta} = 0.2$
- UA – Uniform Angle mutation – je varianta RA, ktorý takisto mutuje uhol vloženia θ, s rozdielom že tento uhol sa dopredu zvolí a každý gén jedinca ktorý je zmutovaný s pravdepodobnosťou $p_{mut\theta}$ bude nastavený na túto hodnotu.

V časti experimentov bude potrebné preveriť vhodnosť použitia buď RA, alebo UA operátoru.

4.3 Návrh algoritmu GA2

Návrh algoritmu GA2 vychádza z návrhu GA1, avšak používa iné zakódovanie jedinca a inú konštrukčnú heuristiku. Jeho návrh bol čiastočne inšpirovaný článkom (ILLICH, a iní, 2007), v ktorom bolo využívané zakódovanie použitia rôznych konštrukčných heuristík do jedinca populácie a taktiež článkom (CONCALVES, 2007).

Každý jedinec je rozdelený na dve časti:

- hlavu, ktorá určuje objekt ktorý bude uložený ako prvý:

$$head\_gene^{GA2}: (id, offset_x, offset_y, rot) \tag{11}$$

kde:
$id \in <1; n^{small\_objects}>$ – je identifikátor malého objektu ktorý môže nadobúdať hodnoty $1..m$, kde m je mohutnosť množiny malých objektov. Toto číslo jednoznačne určuje rozmer melého objektu.
$offset_x, offset_y \in <0; 1>$ určuje posunutie stredu objektu voči stredu kmeňa
$rot \in \{true, false\}$ – určuje, či bude object otočený o 90°, alebo nie

- a telo, ktoré pozostáva z n-1 génov rovnakého typu:

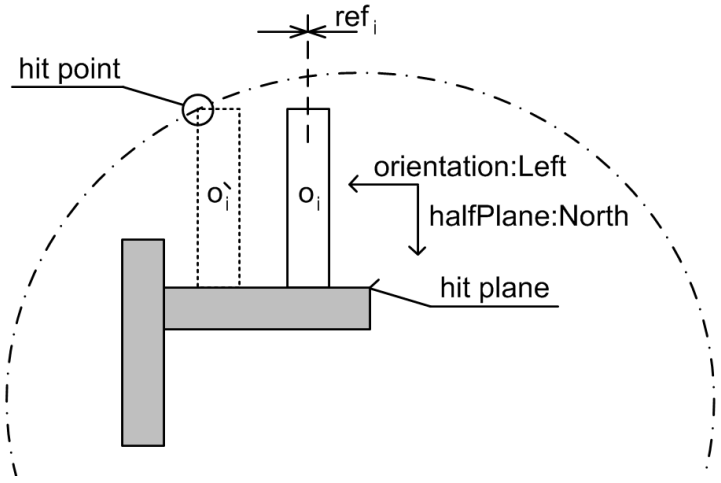
$$body\_gene^{GA2}: (id, ref, rot, method) \tag{12}$$

kde:
$id \in <1; n^{small\_objects}>$ – je identifikátor malého objektu
$rot \in \{true, false\}$ – určuje, či bude object otočený o 90°, alebo nie
$ref \in <0; 1>$ - referenčný bod pre vloženie objektu používaný heuristikou
$method$ – indikuje heuristickú metódu ktorá bude použitá pre vloženie objektu do riešenia.

U tohto algoritmu je použitých celkovo 8 kombinácii konštrukčnej heuristiky. Konštrukčná heuristika môže objekt vložiť zo 4 smerov: $\{north, east, south, west\}$ a v dvoch orientáciách $\{right, left\}$.

Dekódovanie z pseudo-rozloženia (genotypu) na reálne rozloženie (fenotyp) je rovnako ako u GA1 prevádzané iteratívne konštruktívnou heuristikou ktorej princíp je ilustrovaný na Obr. 6:

1. place object o_h specified by head_gene
2. placedObjectsPolygon = $intersection(\emptyset,\ o_h)$
3. for each $body\_gene_i$:
4. create small object o_i specified by id_i, rot_i
5. slide in object o_i from direction determined $method_i.halfPlane$ until it hits placedObjectsPolygon
6. if no surface is hit then: goto (3)
7. else slide object o_i along previously hit surface in direction specified by $method_i.orientation$ until it hits placedObjectsPolygon or complement(largeObject) and place it there as o_i'
8. if o_i does not hit anything then place it at the end of surface below it in specified direction as o_i'
9. placedObjectsPolygon = intersection(placedObjectsPolygon, o_i')
10. end for each
11. END: placedObjectsPolygon is decoded solution



Obr. 6 – Ilustrácia princípu konštruktívnej heuristiky GA2 pre metódu vloženia $north/left$

Konštruktívna heuristika algoritmu GA2 bola navrhnutá so zreteľom na vlastnosti problem LCP a jeho riešenia. V princípe by táto heuristika mala zabezpečiť lepšie vlastnosti GA2 oproti GA1, keďže GA1 používa heuristiku pôvodne navrhnutú pre umiestnenie nekonvexných malých polygonálnych objektov.

Algoritmus GA2 využíva rovnako ako GA1 operátor mutácie OPSO (One Point Swap Order mutation) a hneď niekoľko typov mutačných operátorov špeciálne navrhnutých pre reprezentáciu GA2:

- RH – Random Head mutation – špeciálne navrhnutý operátor pre GA2. Má za úlohu zmutovať náhodne vybranú časť hlavy jedinca GA2
- RIP – Random Insert Point mutation – s danou pravdepodobnosťou p_{mutRIP} zmutuje všetky gény jedinca určujúce bod vloženia objektu
- RPO – Random Placement Orientation mutation - s danou pravdepodobnosťou p_{mutRPO} zmutuje všetky gény jedinca určujúce typ vkladacej heuristiky
- RR – Random Rotation mutation - s danou pravdepodobnosťou p_{mutRR} zmutuje všetky gény jedinca určujúce natočenie malého objektu

Pravdepodobnosti p_{mutRR}, p_{mutRIP} a p_{mutRPO} boli u GA2 fixne nastavené na hodnotu 0.2.

4.4 MOGA varianta algoritmu GA1

Tento algoritmus je navrhnutý ako modifikácia algoritmu GA1. MOGA algoritmus podľa článku (FONSECA, a iní) je jedným z najzákladnejších multikriteriálnych algoritmov. Integrácia princípov MOGA s algoritmom GA1 spočíva v zmene mechanizmu ohodnotenia jedincov podľa (3.2.2) a ich selekcie. Algoritmus GA1 využíva ruletovú selekciu s pravdepodobnosťou úmernou poradiu jedinca v zoradenej postupnosti jedincov podľa ich ohodnotenia. Takýto mechanizmus je robustný dostatočne na to, aby neznevýhodňoval slabých jedincov príliš, avšak pre algoritmus MOGA je nevhodný. Preto je potrebné zmeniť tento mechanizmus na ruletovú selekciu s pravdepodobnosťou úmernou ohodnoteniu jedinca.

4.5 Fitness funkcia

Fitness funkcia u genetického algoritmu hraje jednu z najdôležitejších rolí. V princípe genetické algoritmy pracujú bez akejkoľvek znalosti riešeného problému a práve fitness funkcia im sprostredkováva potrebnú spätnú väzbu. U nesprávne navrhnutej fitness funkcie GA buď nebude pracovať správne, alebo nebude podávať požadované výsledky. U niektorých problémov postačuje celkom triviálna fitness funkcia na to aby pracovali správne. Napríklad vo väčšine C&P (Cutting & Packing) problémov postačuje triviálna fitness funkcia - pomer medzi využitou plochou kontajneru a jeho celkovou plochou (ALVAREZ-VALDÉZ, a iní, 2001) (BURKE, a iní, 2004):

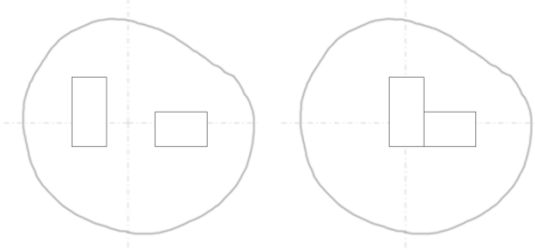
$$fitness = \frac{A^{used}}{A^{container}} \tag{13}$$

kde
A^{used} – je využitá plocha kontajnera
$A^{container}$ – celková plocha kontajnera

Obdobne u SPP (Strip Packing Problem) problémov je fitness funkcia priamo úmerná výške uloženia na pás (ILLICH, a iní, 2007), (KARELAHTI, 2002).

Zvoliť fitness funkciu vhodnú pre problém optimalizácie porezu guľatiny (LCP) nie je vôbec triviálna záležitosť. Naopak zdá sa, že fitness funkcia je jedným z kľúčových prvkov pri riešení LCP. Od tvaru fitness funkcie závisí rýchlosť konvergencie a kvalita výstupu genetického algoritmu a vôbec spoľahlivosť konvergencie u heuristických algoritmov.

Pri experimentoch s GA1 a jednoducho definovanou fitness funkciou podľa rovnice (13), ktorá u LCP reprezentuje výťaž z kmeňa nedochádzalo ku spoľahlivej konvergencii algoritmu. Často dochádzalo k vzniku oddelených ‚ostrovov' malých objektov, alebo vzniku 'dutých' miest v riešeniach. Tento problem ilustruje nasledujúci obrázok (Obr. 7). Ako vidno v obidvoch prípadoch je využitie plochy kontajnera rovnaké, ale kompaktnosť riešenia vpravo je veľmi malá. Prirodzene najlepšie riešenie CLP musí byť kompaktné čo najviac, čo však takto definovaná fitness funkcia nezabezpečí.



Obr. 7 – Ilustrácia problému s rovnakým ohodnotením menej vhodného riešenia (vľavo) a vhodnejšieho (vpravo) jednoduchou fitness funkciou v podobe výťaže

Namiesto jednoduchej výťaže bola teda navrhnutá nasledujúca fitness funkcia:

$$ff = U \times \frac{e}{cr}$$

$$U = \frac{A^{used}}{A^{container}}$$

$$cr = \frac{\text{circumference}(placedObjectsPolygon)}{\sum_{i=1}^{n_{phenotype}} circumference(o_i')} \tag{14}$$

$$e = \sqrt[3]{\frac{n_{phenotype}}{n}}$$

kde
ff – je fitness funkcia
U – je výťaž s rovnakým významom ako v rovnici (13)
e – je efektivita uloženia množiny malých objektov
$n_{phenotype}$ – je počet objektov ktoré sa podarilo uložiť pri konštruktívnej heuristike dekódujúcej genotyp na fenotyp
n – je celkový počet génov jedinca, ktorý reprezentuje maximálnu počet malých objektov ktoré môžu byť uložené
cr – je pomer obvodu polygónu, ktorý je zjednotením všetkých položených malých objektov k sume ich jednotlivých obvodov

Prvok cr bol navrhnutý s cieľom tlačiť evolúciu k riešeniam bez prázdnych miest v strede zoskupení malých objektov. Tento prvok zabezpečí, že aj keď za vytvorí minimálna medzera medzi dvoma malými objektmi, tak celková fitness sa okamžite zhorší úmerne k dvojnásobku dĺžky hrán medzi ktorými bola vytvorená medzera. Ak sú v rozložení ojedinelé zhluky malých objektov, tak evolučný tlak vyvinutý týmto prvkom zabezpečí, že tieto zhluky sa spoja a tým sa vytvorí voľný priestor pre ďalšie objekty, čím sa sekundárne zvýši výťažnosť. Pri použití čistej výťažnosti ako fitness funkcie by algoritmus na čas uviazol na lokálnom optime.

Efektivita uloženia e vytvára evolučný tlak s cieľom dostať do finálneho rozloženia čím väčší počet malých objektov. Toto je vhodné z dôvodu, že takýmto spôsobom sa sekundárne ovplyvňuje výťažnosť. Veľký počet objektov znamená použiť čo najmenšie objekty k dispozícii. To na druhej strane zabezpečí lepšiu výťažnosť, pretože je jednoduchšie zaplniť veľký objekt objektmi s menšou granularitou. Na druhej strane však nie je vhodné aby mal tento člen príliš veľkú váhu, lebo by to malo negatívny vplyv na celkovú evolúciu. Experimentálne bola zvolená a overená tretia odmocnina z pomeru $\frac{n_{phenotype}}{n}$ ako vhodný kompromis.

4.5.1 Kritérium kvality vo fitness funkcii

U praktického problému LCP je okrem kritéria výťažnosti nutné zvažovať sekundárne kritériu kvality výsledného reziva. Z praxe vyplýva, že masívnejšie typy stavebného reziva sú vhodnejšie zo stredu kmeňa, zatiaľ čo dosky sú vhodnejšie z okrajových častí. Problematika kvality reziva je značne komplikovaná a zahŕňa orientáciu dosky a jej presné umiestnenie vzhľadom na letokruhy, avšak pre účely tejto práce postačí rozdelenie na stavebné rezivo zo stredu kmeňa a dosky z okrajových častí.

Kvalita reziva je separátnym kritériom u LCP a tým pádom vzniká potreba optimalizovať dve sledované veličiny – vhodnosť umiestnenia a výťažnosť. Z problému LCP s takto definovanými požiadavkami sa tým pádom stáva multikriteriálny problém. Pre algoritmy GA1 a GA2 bola preto navrhnutá nasledujúca upravená fitness funkcia ktorá dáva možnosť pre váhovanie daných čiastkových fitness funkcií (15). Algoritmus MOGA umožňuje optimalizáciu viacerých kritérií priamo a tak je použitá forma podľa rovníc (16):

$$ff = \left(w_u \times U + w_{ps} \times PS\right) \times \frac{e}{cr} \tag{15}$$

$$ff_U = U \times \frac{e}{cr}, \qquad ff_{PS} = PS \times \frac{e}{cr} \tag{16}$$

U týchto fitness funkcií pribudol nový člen PS ktorý vyjadruje vhodnosť umiestnenia malých objektov. Matematicky vyjadriť vhodnosť umiestnenia nie je veľmi jednoduché, ale napriek tomu bola navrhnutá nasledujúca funkcia ktorá sa o toto pokúša:

$$PS = \sum_{i=i}^{n_{phenotype}} PSF(r_i, d_i) \frac{A_{o_i'}}{A^{container}} \tag{17}$$

kde:
$n^{phenotype}$ – počet uložených malých objektov v rozložení
$A^{container}$– plocha profilu kmeňa
$A_{o_i'}$ – obsah daného umiestneného malého objektu
d_i – je vzdialenosť stredu výrezu od stredu kmeňa a je daná ako:

$$d_i = \sqrt{(xc_i - xc_C)^2 + (yc_i - yc_C)^2}$$

r_i – je pomer medzi dlhšou a kratšou stranov malého objektu o_i'

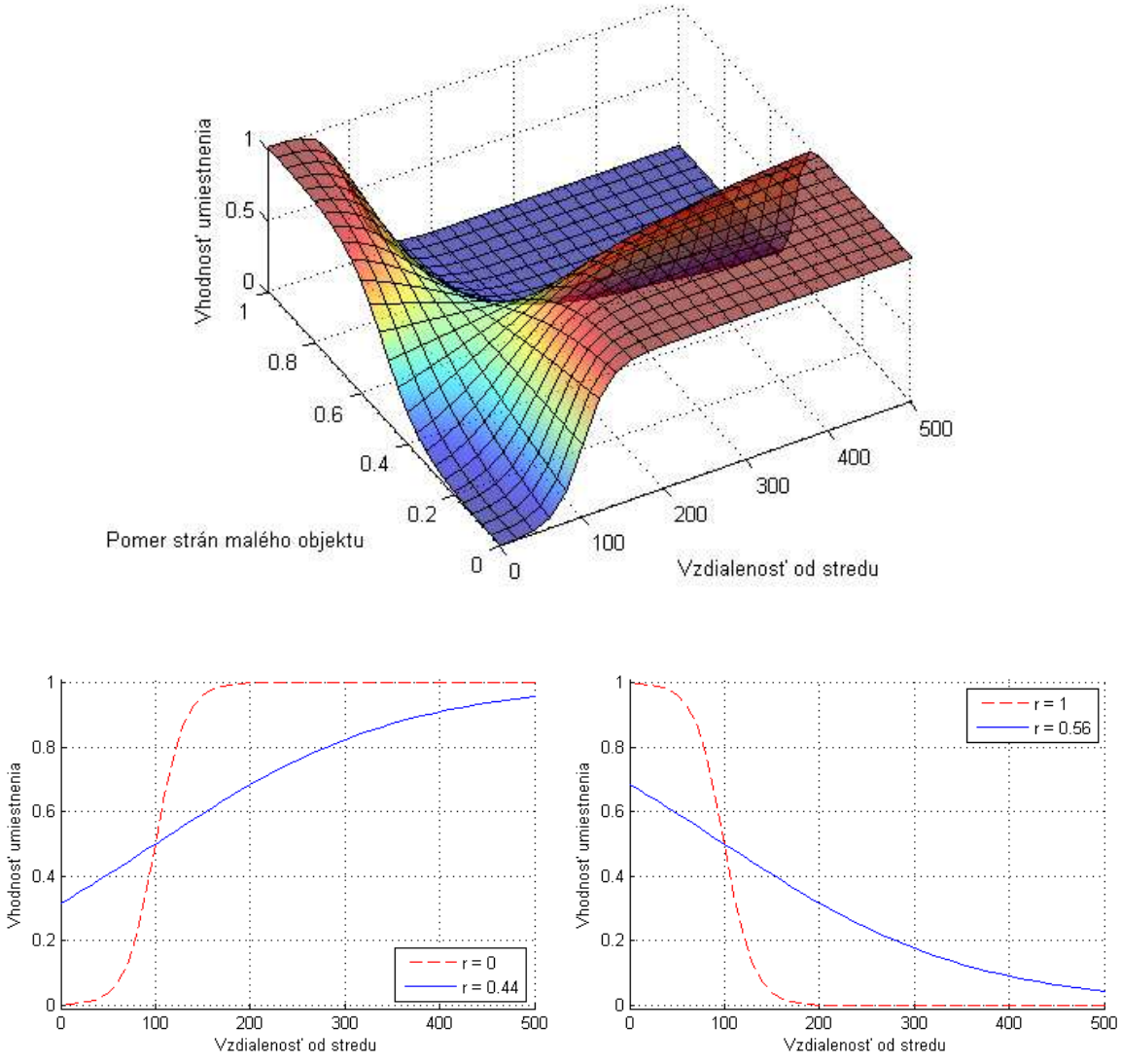
PSF je funkcia vyjadrujúca vhodnosť umiestnenia konkrétneho malého objektu. Bola navrhnutá vďaka inšpirácii zo sigmoidálnej funkcie a jej zápis je v nasledujúcej rovnici (18). Dôvodom na takto ‘zložito’ definovanú funkciu je potreba hladkej funkcie ktorá zabezpečí menej roviniek vo výslednej fitness funkcii a tým pomôže k lepšej konvergencii GA. Táto funkcia rozhodne nie je najvhodnejšia, keďže rovnako ohodnotí väčšie štvorcové objekty (stavebné hranoly) ktoré naozaj patria do stredu kmeňa a malé štvorcové objekty (stavebné laty). Pre účely tejto práce a pre overenie multikriteriálnej optimalizácie však uvedená funkcia postačuje.

$$PSF(r,d) = \frac{1}{1 + e^{\left(\frac{100\ .steepness^2}{extent}\right)\left(((a-b)*r+b)-d\right)(inertR-r)}}$$

$$a = \frac{A}{100} \times extent \quad b = \frac{B}{100} \times extent \tag{18}$$

kde:
$extent$ – je miera záberu funkcie a je volená tak, aby bola rovná maximálnemu polomeru kmeňa
$steepness, A, B, inertR$ – sú statické parametre funkcie a pre účely tejto práce boli fixne zvolené nasledujúce konštanty: $steepness = 0.8,\ A = B = 20\%,\ inertR = 0.5$

Na nasledujúcom obrázku (Obr. 8) je zobrazený priebeh navrhnutej funkcie pri daných parametroch pre $extent = 500$.



Obr. 8 – Funkcia určujúca vhodnosť umiestnenia malého objektu vrámci kontajneru. 3D plot (hore), priebeh funkcie pre rôzne pomery strán malého objektu (dole).

5 Experimentálne overenie

5.1 Dátové sady

U C&P problémov (Cutting & Packing problems) je bežnou praxou porovnávanie prístupov k ich riešeniu pomocou testovacích scenárov. Takéto scenáre majú presne definované obmedzenia, pričom každý zo scenárov sa snaží reprezentovať typické 'chytáky' (pitfalls) pre daný C&P problém. Takéto testovacie sety vznikajú na základe potreby autorov článku a postupne sa stávajú štandardizovanými setmi pre overenie a porovnanie funkcie algoritmu na riešenie daného konkrétneho problému aj bez dostupnosti zdrojových kódov konkurenčného riešenia. Pravdepodobnosť existencie takýchto štandardizovaných scenárov je zdá sa byť priamo úmerná počtu publikácií, ktoré sa danej odnože C&P týkajú. U problémov, ktoré sú dobre popísané ako napríklad SPP v oceliarskom priemysle, kde je tento problém prirodzený z dôvodu princípu produkcie oceľových plechov ako ‚nekonečných' pásov existuje hneď viacero štandardizovaných testovacích scenárov. U týchto problémov je potrebné umiestniť konečný počet malých pravouhlých objektov na čo najkratší pás materiálu. Príkladmi z literatúry je testovací set *C* (HOPPER, a iní, 2000), ktorý obsahuje 21 inštancií o veľkosti množiny malých objektov 16-197 pre SPP (Strip Packing Problem) alebo testovací set *Burke* (BURKE, a iní, 2004), ktorý obsahuje 13 inštancií o veľkosti množiny malých objektov 10-3153 rovnako pre SPP problém.

Výsledkom z rešerše literatúry však je, že nie je k dispozícii len veľmi malé množstvo publikácií, ktoré riešia problém podobný LCP. Z tohto dôvodu nie je k dispozícii ani žiadny štandardizovaný súbor testovacích scenárov vhodný pre overenie riešenia problému LCP. Z tohto dôvodu bolo nutné najprv pripraviť tieto scenáre a následne pomocou nich navrhované riešenie otestovať.

5.1.1 Vytvorenie testovacích scenárov

Pre potreby overenia navrhovaných algoritmov je potrebné pripraviť:

- profily kmeňov – kontajnery (v terminológii C&P)
- druhy potrebného reziva - sady malých objektov (v terminológii C&P)

Pri LCP probléme je v praxi štandardom, že vstupom do procesu optimalizácie sú dáta o fyzických rozmeroch kmeňa. V princípe sa používajú dve metódy:

1. **2D model** – táto reprezentácia má dva podtypy:
    - v zjednodušenej variante sa zmeria priemer kmeňa a ďalej sa uvažuje kmeň ako perfektný kruh s presným stredom kmeňa v strede kruhu. Toto má výhodu v tom, že nie je potrebné žiadne zložité technické vybavenie na zmeranie profilu kmeňa, avšak má za následok menšiu efektivitu vnesenú často veľkou diferenciou modelu od reality. Takisto dochádza k menšej kvalite reziva kvôli nepresným odkrojom. Preto sa táto naivná metóda prakticky nepoužíva

- v zložitejšej variante sa zmeria profil prierezu kmeňa (napr. vision systémom) a zistí sa jeho stred a vyextrahuje sa v princípe konvexný polygón reprezentujúci profil prierezu. Ďalej sa uvažuje o kmeni ako o valci s podstavou v tvare tohto profilu. Toto riešenie je akýmsi kompromisom medzi komplexnou 3D a jednoduchou 2D reprezentáciou. Z toho vyplývajú benefity vo forme vyššej kvality reziva a nižšieho podielu odpadu ako u jednoduchej 2D reprezentácie a nižšie cenové nároky technického vybavenia vzhľadom na plnú 3D reprezentáciu.

2. **3D model** – pri tejto reprezentácii je profil kmeňa oskenovaný 3D skenerom. Ide o takmer najdokonalejšiu reprezentáciu aká sa v súčasnosti používa. Výhodou je možná veľmi presná optimalizácia, kde je možné zabezpečiť vyhnutie sa oblastiam poškodenia kmeňa, či rešpektovať dĺžkové prehnutie kmeňa. Nevýhodou je pridanie jednej dimenzie do prehľadavacieho priestoru, keďže malé objekty sa ukladajú ako kvádre v 3D, nie ako obdĺžniky v 2D.

Táto práca sa zaoberá problémom LCP s 2D modelom kmeňa. Práca s 3D modelom je len generalizáciou LCP s 2D modelom a prináša so sebou značne vyššie výpočtové nároky. Prípadné rozšírenie na 3D LCP by malo byť možné aplikovaním poznatkov získaných pri riešení 2D LCP.

5.1.2 Získanie reálneho 2D profilu kmeňa

Aby sa mohlo overenie reálnej funkčnosti navrhovaných algoritmov previesť čo najvernejšie, bolo potrebné získať reálne profily kmeňov. Pre tento účel bol navrhnutý a implementovaný algoritmus extrakcie profilu z dodanej snímky kmeňa. V nasledujúcom texte bude tento algoritmus v krátkosti opísaný. Ako modelový príklad poslúži extrakcia profilu norm_medium_1.

Algoritmus extrakcie profilu kmeňa pracuje v niekoľkých na seba nadväzujúcich krokoch:

1. **Preprocesing**

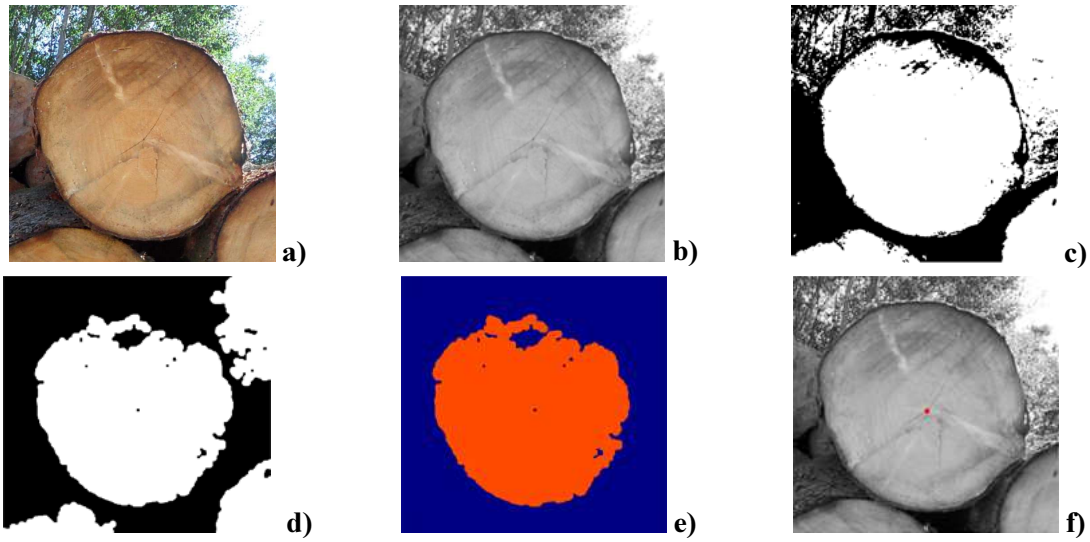
V tejto fáze je obraz kmeňa (Obr. 9-a) filtrovaný gausovským filtrom za účelom odstránenia šumu a následne prevedený do šedotónového obrazu (Obr. 9-b) podľa nasledujúcej rovnice (19). U takto vyjadrenej intenzity obrazu $I_{i,j}$ bolo empiricky preverené, že veľmi dobre vystihuje farbu kmeňa a umožňuje lepšiu detekciu hrán

$$I_{i,j} = \frac{R_{i,j} + G_{i,j}}{\max_{i,j}\left(R_{i,j} + G_{i,j}\right)} \quad i = 1..w, j = 1..h \tag{19}$$

kde
I – je šedotónová intenzita
R, G – sú hodnoty červeného a zeleného kanálu
w, h – šírka a výška obrázku

2. Estimácia stredu kmeňa

Obrázok sa prevedie na binárny pomocou metódy Otsu (Obr. 9-c). Následne sú za použitia morfologických operátorov ako sú dilatácia a erózia odstránené artefakty (Obr. 9-d) a pomocou analýzy spojených komponentov (Connected Components Analysis) sú odstránené objekty na hrane snímku (Obr. 9-e). V poslednom kroku je určený centroid objektov na scéne a tento je prehlásený za stred kmeňa $[x_c, y_c]$ (Obr. 9-f).



Obr. 9 – Proces estimácie stredu kmeňa. Pôvodná snímka (a), prevod do šedotónovej oblasti (b), prevod na binárny obraz (c), odstránenie artefaktov (d), odstránenie objektov na hranách snímku (e), výpočet centroidu a určenie stredu kmeňa (f)

Uvedená metóda sa však na príliš nehodí na robustné určenie stredu kmeňa preto museli byť v niektorých prípadoch zistené stredy korigované. Neskôr bola experimentálne implementovaná iná metóda, ktorá používa Houghovu transformáciu pre kružnice a pomocou štatistického vyhodnotenia dosahuje lepších výsledkov.

3. Transformácia obrazu kmeňa

V tomto kroku sa prevedie transformácia obrazu kmeňa z polárnych do karteziánskych súradníc pričom sa využíva stredu $[x_c, y_c]$ odhadnutého v predchádzajúcom kroku. Je použitá transformácia podľa rovnice (20) . Výsledok je na obrázku Obr. 10-a.

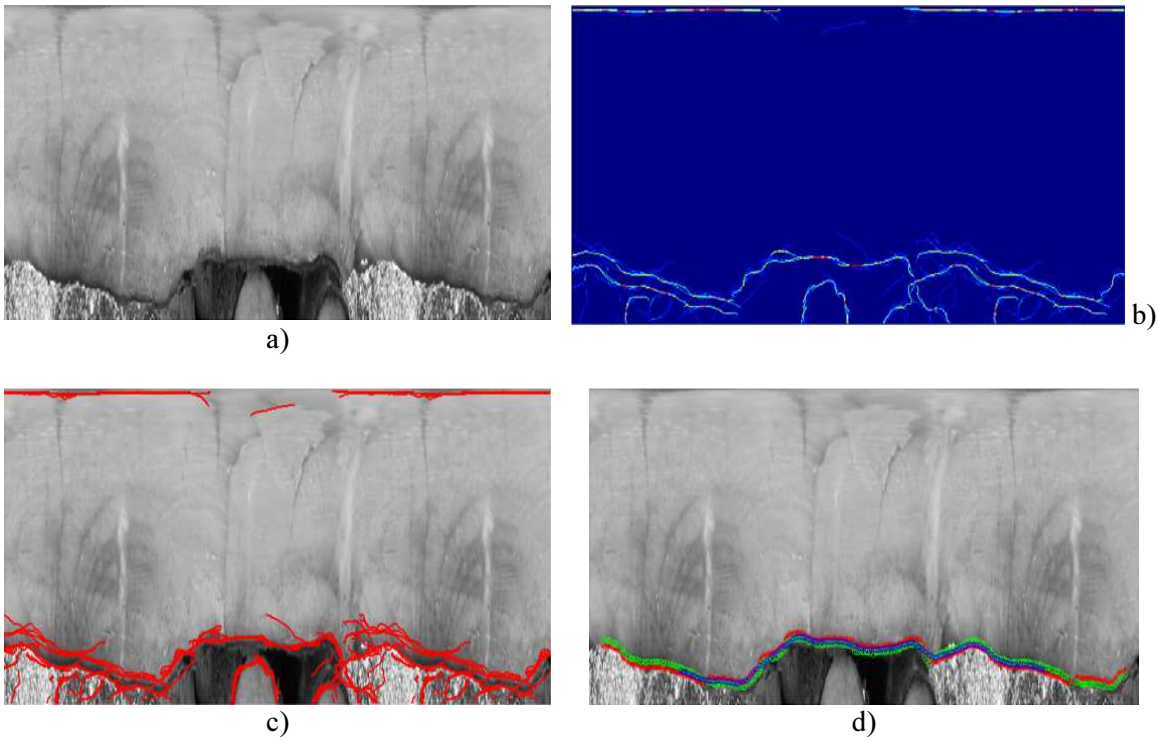
$$S'(\varphi, r) = S(x_c + cos(\varphi).r,\ \ y_c + sin(\varphi).r) \qquad \varphi, r \in R \tag{20}$$

4. Detekcia hrán a nájdenie obrysu kmeňa

Detekcia hrán na rozvinutom kmeni (Obr. 10-a) vychádza z myšlienky, že na rozvinutom kmeni by mal okraj kmeňa smerovať horizontálne, prípadne s malou uhlovou diferenciou od horizontálneho smeru. Pre niekoľko málo uhlov θ napr. v rozpätí +/-10° od horizontálneho smeru sa pomocou operácie motion blur zvýraznia hrany a prevedie sa detekcia hrán pre príslušný uhol detektorom *canny*. Následne sa pixely, kde bola hrana

nájdená zapíšu svojím príspevkom do hlasovacieho priestoru podľa rovnice (21). Hlasovací priestor je zobrazený na Obr. 10-b a všetky nájdené hrany na Obr. 10-c.

$$S^{vote} = S^{vote} + canny(motionBlur(S', \theta)) \tag{21}$$



a) b) c) d)

Obr. 10 – Detekcia obrysu kmeňa. Transformácia z polárnych do karteziánskych súradníc (a), hlasovací priestor detekcie hrán (b), všetky zdetekované hrany (c), postprocessing nájdeného obrysu (d)

Úloha nájsť obrys kmeňa spočíva v nájdení správnej hrany v hlasovacom priestore S^{vote} ktorá ho bude reprezentovať najlepšie. Najprv sa odstránia všetky hrany v S^{vote} , pre ktoré platí $r < r_{max}.0.15$. Empiricky bolo zistené, že v tejto oblasti, ktorá reprezentuje oblasť okolo stredu kmeňa existuje tendencia vytvárať veľké množstvo hrán pri opisovanom postupe. To môže viesť až k nesprávnemu výsledku.

Pre každý bod v hlasovacom priestore S^{vote}, pre ktorý platí, že jeho hodnota je väčšia ako 0 sa v niekoľko málo uhloch φ napr. v rozpätí +/-10° od horizontálneho smeru sa nájde najbližší sused s najvyššou hodnotou v S^{vote}. Následne sa do neorientovaného grafu G vloží hrana ohodnotená vzdialenosťou k susedovi a jeho hodnotou v S^{vote}, spájajúca uzol reprezentujúci aktuálny bod s uzlom reprezentujúcim suseda. Djikstrovým algoritmom sa následne nájde najkratšia cesta, ktorá reprezentuje hranu kmeňa (Obr. 10-d-červená).

5. Postprocessing

V tomto kroku prebieha nájdenie maximálnej zmeny (hodnoty derivácie) v gradiente kolmom na nájdenú hranu pre každý jej bod v obmedzenom okolí tohto bodu

(Obr. 10-d-zelená). Následne sa vyhodnotí výsledná hrana ako priemer nájdenej hrany a hrany získanej lokálnym prehľadávaním (Obr. 10-d-modrá).

5.1.3 Súbor testovacích profilov kontajnerov

Pre overenie reálnej použiteľnosti algoritmov bolo pripravených 11 reálnych profilov kmeňov pomocou algoritmu opísaného v predchádzajúcej kapitole (5.1.2). Rozdelené sú podľa scenára, ktorý opisujú a podľa veľkosti. K ním bolo vygenerované 4 ďalšie profily modelujúce ideálne okrúhle kmene. Tieto majú za úlohu overiť, či je algoritmus schopný využiť symetriu pre dosiahnutie lepšieho riešenia. V nasledujúcej tabuľke (Tab. 1) sú uvedené názvy testovacích profilov ich parametre a popis.

Tab. 1 – Kompletná tabuľka testovacích sád profilov kontajnerov

Označenie profilu	Plocha [mm^2]	Priemer [mm]		Popis
		min	max	
250mm	49 077	250	250	Umelo vygenerované, idealistické presne kruhové kmene. Sú navrhnuté s cieľom odhaliť, či je algoritmus schopný nájsť symetrické riešenia ktoré sú často najlepšou voľbou u dokonalo kruhových kmeňov.
375mm	110 424	375	375	
500mm	196 310	500	500	
625mm	306 734	625	625	
asymmetric	104 929	345	385	Najnepravidelnejší profil v celom súbore. Mal by slúžiť na overenie robustnosti algoritmu, ktorý by mal dokázať pracovať s kýmkoľvek konvexným polygónom.
egg_large	306 747	576	661	Profily sploštených kmeňov ktoré sa často vyskytujú vo veterných oblastiach. Je nutné robustne navrhovať porezový plán aj pre tento typ kmeňov.
egg_small	77 779	295	334	
excenter_medium	174 014	443	483	Profily s excentricky umiestneným stredom. Modelujú situáciu kde je nutné umiestňovať rezivo silne asymetricky pre dosiahnutie potrebnej kvality.
excenter_small	53 626	246	275	
norm_large_1	226 618	517	555	Prrofily normálnych kmeňov ktoré nie sú príliš deformované a majú stred umiestnený zhruba kocentricky
norm_large_2	335 284	635	683	
norm_medium_1	168 770	437	481	
norm_medium_2	102 831	346	376	
norm_small_1	41 154	224	236	
norm_small_2	68 407	291	300	

Pre vizuálne porovnanie sú uvedené obrázky jednotlivých kontajnerov v prílohe A. Príklady detekcie obrysu u niektorých z nich sú na Obr. 11.


a)

b)

c)

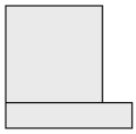
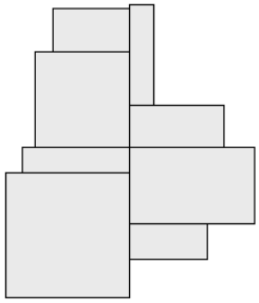
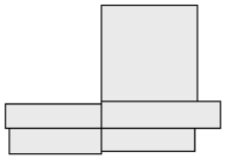
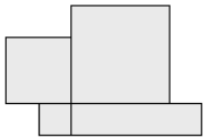
Obr. 11 – Príklad extrakcie profilov pre norm_medium_1 (a), excenter_medium (b), norm_small_2 (c)

5.1.4 Súbor testovacích sád malých objektov

Pre účely testovania algoritmov bolo navrhnutých 5 sád malých objektov, každá s iným zámerom:

- *simple* má za úlohu preveriť kombinačné schopnosti algoritmov kvôli vlastnosti, že niektoré jej objekty je možné skonštruovať kombináciou iných, menších.
- *boards_only* bola vytvorená ako syntetický záťažový test. Keďže ich priemerná plocha je najmenšia spomedzi všetkých sád, tak je ich možné umiestniť do daného kontajnera veľké množstvo.
- *beam_and_board* je sada určená hlavne na preverenie schopnosti algoritmu umiestniť štvorcové hranoly do centra kmeňa a obdĺžnikové dosky po stranách
- *complex* je sada určená na preverenie komplexných rekombinačných schopností algoritmu pri netriviálnom počte typov malých objektov
- *real* je sada objektov, ktorá je prebraná z reálnej praxe

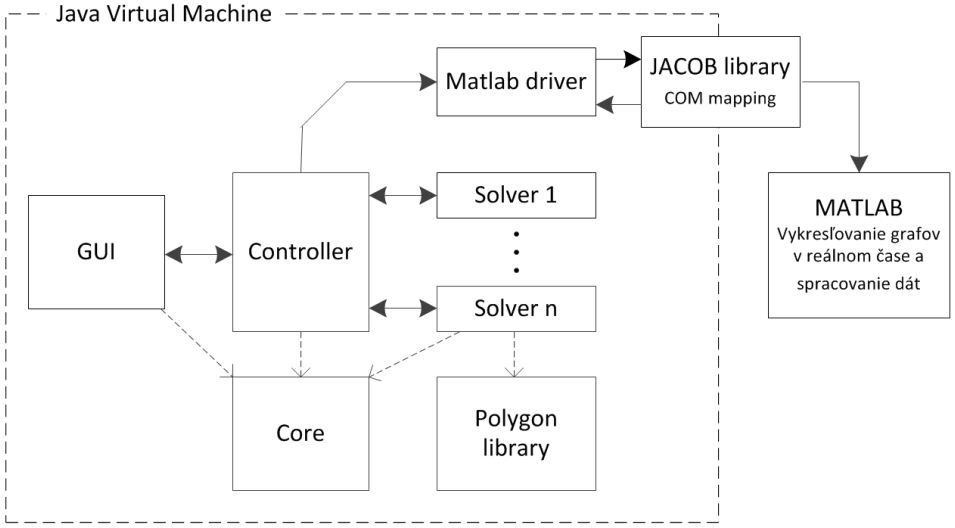
Tab. 2 – Kompletná tabuľka testovacích sád malých objektov

Názov sady	Rozmery objektov [mm]	Priemerná plocha objektu [mm^2]	Sada objektov
beam_and_board	76 x 76 100 x 17	4436	
boards_only	95 x 15 50 x 20	1688	
complex	75 x 30 100 x 60 75 x 75 86 x 17 15 x 80 60 x 25 60 x 32 100 x 100	4388	
real	95 x 17 73 x 15 76 x 15 73 x 17 76 x 76	2731	
simple	77,5 x 77,5 105 x 22,5 50 x 50 22,5 x 22,5	3403	

5.2 Implementácia

Pre implementáciu bol použitý jazyk JAVA SE verzie 7 s frameworkom Netbeans Platform od spoločnosti Oracle. Na vyhodnotenie experimentov slúžilo prostredie Matlab R2013a. Dôvodom na implementáciu vo frameworku Netbeans Platform je veľmi dobrá podpora modularizácie väčšieho softvéru. Veľmi dôležitou vlastnosťou tohto frameworku je veľmi dobrá pripravenosť na komplexné GUI aplikácie. To umožnilo relatívne jednoducho implementovať rôzne vizualizačné pomôcky potrebné na odladenie algoritmov. Obyčajné ladenie jednoducho zďaleka nepostačovalo. Navrhované algoritmy využívajú komplexné konštrukčné heuristiky a pomoc GUI na vizualizáciu procesu konštrukcie bola v konečnom dôsledku nutnosťou. Vizualizácia procesu evolúcie a spätná väzba o skutočnom tvare riešenia veľmi pomohla, hlavne v procese návrhu fitness funkcie.

Softvér je rozdelený do niekoľkých viac-menej nezávislých modulov (Obr. 12). Modulárna štruktúra bola zvolená kvôli jednoduchej výmene prípadne dopĺňaniu modulov v budúcnosti. Základná kostra softvéru pritom ostáva nedotknutá. Táto modularizácia má niekoľko nenahraditeľných vlastností. V prvom rade sa odseparujú prípadné chyby v moduloch, ktoré sa môžu ladiť v podstate osobitne. V rade druhom je možné napríklad modul *Polygon library* vymeniť za lepšiu implementáciu, ktorá ani nemusí bežať vo virtuálnom stroji Javy a na zvyšok aplikácie to nebude mať žiadny vplyv. Podobne môže byť modul *Matlab driver* nahradený iným modulom, ktorý bude napríklad vykresľovať grafy pomocou natívnych funkcií Javy.

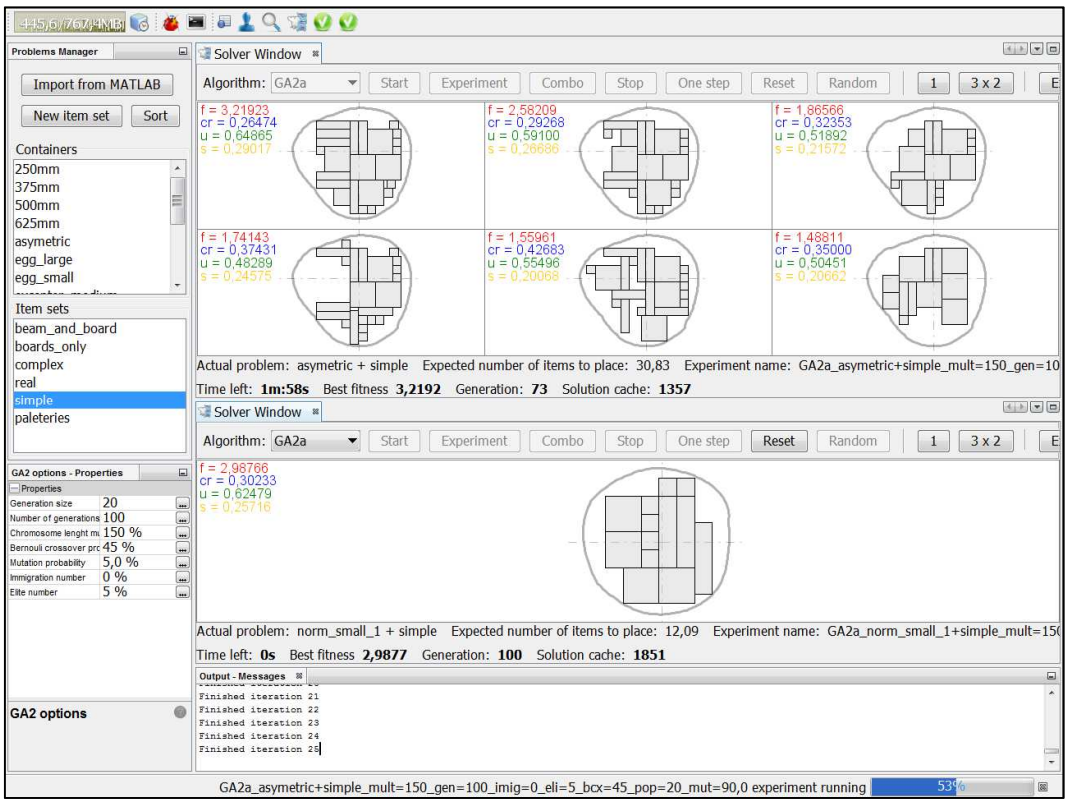


Obr. 12 – Základná modulárna štruktúra implementovaného riešenia

Spoločným modulom, na ktorom sú ostatné závislé je modul *Core,* ten obsahuje základné spoločné triedy a definície. Tento modul takisto obsahuje definície rozhraní a abstraktných tried. Niektoré rozhrania umožňujú transparentnú komunikáciu medzi modulmi bez toho, aby na sebe boli závislé. Moduly GUI a Controller sú v podstate integrované do jedného modulu. V prípade potreby by ale nemal byť problém vytvoriť modul headlessController, ktorý by bol úplne nezávislý na GUI.

Moduly Solver$_i$ reprezentujú jednotlivé algoritmy. U tejto konkrétnej implementácie sú použité dva moduly pre GA1 a GA2. Tieto moduly sú okrem modulu *Core* hlavne závislé na knižnici *Polygon library* , ktorá im poskytuje podporu pre ich konštruktívne heuristiky. Táto knižnica je využívaná veľmi intenzívne – konštruktívna heuristika je v prípade navrhnutých GA najnáročnejším krokom. Modul *Core* poskytuje modulom Solver$_i$ podporu pre paralelné výpočty a konštruktívna heuristika tak môže byť na mnohojadrovom procesore značne urýchlená. Tu prichádza k slovu Java a Netbeans Platform, ktoré majú excelentnú podporu pre bezproblémovú integráciu paralelných výpočtov.

Prostredie Matlab je využívané na ukladanie a analýzu dát a na vykresľovanie grafov v reálnom čase. Aplikácia s Matlabom komunikuje pomocou mechanizmu OLE Automation. Na strane aplikácie zabezpečuje obluhu volaní pre posielanie dát, vykresľovanie grafov atď. modul *Matlab Driver.* Tento modul predstavuje abstraktné API pre aplikáciu, takže aplikácia vôbec netuší, že jej dáta spracováva v konečnom dôsledku Matlab. V prípade potreby by mohol byť tento modul nahradený iným, ktorý by zabezpečoval spracovanie dát iným spôsobom.



Obr. 13 – Snímka pracovného okna aplikácie za behu experimentov

Medzi modulom *Matlab Driver* a Matlabom figuruje na rozhraní virtuálneho stroja Javy (JVM) knižnica JACOB ktorá vytvára JAVA-COM most pomocou mechanizmu JNI (Java Native Interface). Matlab pracuje v režime headless – bez hlavného okna a volá sa na princípe on-demand, takže iba v prípade, že je ho potreba.

Na Obr. 13 je uvedená snímka hlavného okna aplikácie. Ako vidno, tak aplikácia vďaka prepracovanému užívateľskému rozhraniu frameworku Netbeans Platform a podpore paralelného behu umožňuje prevádzať viacero experimentov naraz s grafickou spätnou väzbou.

Celkovú náročnosť implementácie je možné odhadnúť pomocou metodiky SLOC (Source Lines Of Code). Celkovo JAVA implementácia aplikácie bez knižníc je realizovaná pomocou približne 8000 riadkov kôdu a MATLAB skripty pre spracovanie dát a vykresľovanie grafov obsahujú približne 6000 riadkov kódu.

5.3 Experimenty

Cieľom experimentov je vzájomné porovnanie navrhovaných genetických algoritmov. Keďže neexistuje žiadna iná publikácia riešiaca podobný problém, tak nie je možné porovnať výsledky s výsledkami iných autorov ako je to zvykom u štandardných problémov C&P. Pre objektívne porovnanie algoritmov je najprv vhodné zabezpečiť optimálne naladenie ich parametrov. Ako je u GA známe, tak priestor parametrov algoritmu je značne netriviálny a úloha nájsť správnu kombináciu parametrov je samostatnou optimalizačnou úlohou, na ktorú sú často nasadzované nadradené GA (MACH, 2009). Avšak u navrhovaných algoritmov nie je možné z praktického hľadiska preskúmať veľké množstvo kombinácii parametrov kvôli ich časovej náročnosti. Preto nasadenie nadradeného GA na naladenie parametrov navrhovaných GA nepripadá ako vhodný variant. Vhodnejšie sa javí zvoliť primeranú metodiku pre ich naladenie. Presný návod ako zostaviť takúto metodiku neexistuje, a tak zvolená metodika vychádzala zo skúsenosti autora práce a z empirického pozorovania správania algoritmov.

5.3.1 Metodika experimentov

Metodika hľadania vhodných parametrov algoritmov bola závislá na štruktúre daného GA a často museli experimenty odhaliť nielen správne nastavenie daného parametra, ale aj to, ktorý operátor (kríženia, mutácie) je vhodný na nasadenie.

Parametre boli ladené postupne (vodopádovým modelom): Najprv boli vybrané vhodní operátory kríženia a mutácie za použitia empiricky zvolených parametrov. Následne bola overená vhodná hodnota pravdepodobnosti kríženia a s touto aktuálne optimálnou hodnotou sa ďalej prevádzali experimenty na určenie vhodnej hodnoty pravdepodobnosti mutácie. Získané hodnoty sa používali v ďalších prípadných krokoch.

Pri hľadaní parametrov GA boli často využívané intervaly spoľahlivosti, ktoré slúžili na vytvorenie lepšej predstavy o skutočnej polohe strednej hodnoty pri opakovanom experimente. Zvyčajne sa jednalo o dosiahnutú hodnotu fitness funkcie po istom počte generácií pri istom nastavení algoritmu. Predpokladalo sa pritom, že táto hodnota sa správa ako náhodná premenná s normálnym rozložením pravdepodobnosti a pre jej strednú hodnotu μ je možné zostrojiť intervalový odhad (ŠEDIVÁ, 2007):

$$\left(\bar{x} - u_{1-\frac{\alpha}{2}}\frac{\sigma}{\sqrt{n}}\right) < \mu < \left(\bar{x} + u_{1-\frac{\alpha}{2}}\frac{\sigma}{\sqrt{n}}\right) \tag{22}$$

kde
$\bar{x}$ – je priemerná zmeraná hodnota
σ – je smerodajná odchýlka zmeraných hodnôt
n – je počet meraní
$u_{1-\frac{\alpha}{2}}$ – je hodnota kvantilu normálneho normovaného rozdelenia N(0,1) odpovedajúca hodnote α, ktorá reprezentuje hladinu významnosti. Pre všetky experimenty bola zvolená hladina významnosti $\alpha = 0.99$ (99%), čomu odpovedá hodnota $u_{1-\frac{\alpha}{2}} = 2.58$.

Často boli výsledné priebehy závislostí pri jednotlivých experimentoch veľmi nespoľahlivé a pre praktické určenie pravdepodobných priebehov musela byť použitá aproximácia priebehu. Avšak aproximácia polynómom alebo inou analytickou funkciou nepripadala do úvahy, keďže priebehy závislostí majú často veľmi nelineárny charakter. Preto bola na aproximáciu priebehov použitá metóda *Smoothing Spline* v podstate u všetkých experimentov s vhodne nastaveným parametrom vyhladzovania.

Parametre uvedené v nasledujúcej tabuľke (Tab. 3) boli zvolené štandardne na základe skúsenosti autora a na základe správania sa navrhovaných GA. S týmito parametrami sa počíta pri porovnaní algoritmov navzájom.

Tab. 3 – Štandardne zvolené parametre genetických algoritmov

Počet generácii	300
Veľkosť populácie	50
Násobiteľ dĺžky chromozómu	2

U počtu generácií bolo odsledované, že pri správnom nastavení parametrov navrhovaných GA, algoritmus konverguje k finálnemu riešeniu približne už po 120 generáciách. 300 generácií bolo zvolených pre umožneniu algoritmu dopracovať sa k lepšiemu riešeniu. U bežných GA nie je veľmi bežné, že sa algoritmus dopracuje k riešeniu po takom malom počte generácií. Navrhovaným GA však k rýchlej konvergencii zrejme veľmi napomáha robustná konštruktívna heuristika.

Kvôli veľkej výpočtovej náročnosti experimentov, u ktorých je potrebné jeden beh opakovať často aj 100x pre rôzne nastavenia nezávislého parametra nie sú parametre v Tab. 3 veľmi vhodné. Preto je navrhovaný postup pri experimentovaní s nastaveniami algoritmov podľa odľahčených parametrov (Tab. 4), ktoré umožnia väčšiu voľnosť. Overenie sa však prevedie pri plnohodnotnom nastavení GA podľa Tab. 3.

Tab. 4 – Odľahčené parametre genetických algoritmov pre experimenty s nastavením parametrov

Počet generácii	100
Veľkosť populácie	20
Násobiteľ dĺžky chromozómu	1.5x

V kapitole 5.1 popisujúcej dátové sady je uvedených 15 kontajnerov a 5 sád malých objektov. Spolu teda existuje 75 kombinácií na ktorých je možné algoritmy preveriť. Samozrejme zďaleka nie je možné prevádzať experimenty pre všetky dátové sady. Preto boli vybrané nasledujúce kombinácie dátových sád ako testovacie sady pre experimenty s parametrami GA:

1. *asymmetric / simple*
2. *norm_small_1 / simple*
3. *norm_small_2 / complex*
4. *excenter_small / complex*

Počas experimentov sa ukázalo, že niektoré závislosti v parametroch GA je relatívne ľahké odhaliť. Preto sa pre prvotné preskúmanie parametrického priestoru zvyčajne volili prvé dve sady a počet opakovaní experimentu 20x. Ak bolo potrebné spresniť výsledky, tak bolo prevádzaných až 100 opakovaní. Ak získané hodnoty neboli postačujúce v prípade náročnejších experimentov, tak bolo volené použitie všetkých štyroch testovacích sád.

5.4 Experimenty s GA1

5.4.1 Voľba operátoru kríženia

V časti 4.2 boli navrhnuté dve varianty rekombinačného operátora kríženia:

5. BCX – Bernouli Crossover
6. BLCX – Bernouli Local Crossover

Experimentálne bola overená vhodnosť uvedených operátorov u algoritmu GA1. Keďže sa jednalo o jeden z prvých experimentov, nebol a priori známy takmer žiadny vhodný parameter daného GA. V nasledujúcej tabuľke (Tab. 5) sú uvedené empiricky zvolené parameter použité v experimente. Parametre n_{imig} / n_{eli} boli zvolené na neelitistickú variantu bez imigrácie 0 / 0. V experimente nebol použitý žiadny operátor kríženia, preto $p_{MX} = 0$.

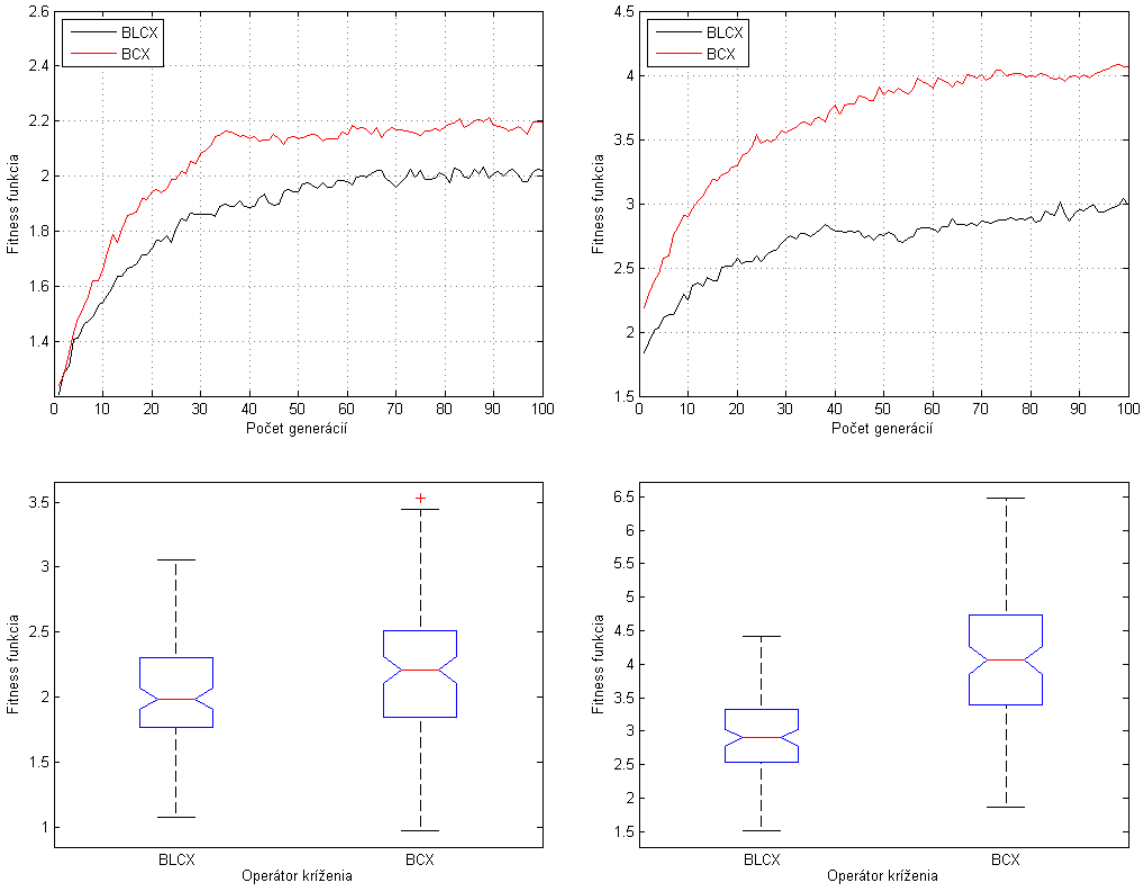
Tab. 5 – Nastavenie experimentov s GA1 pre výber operátoru kríženia

Testovacie sady	asymmetric / simple norm_small_1 / simple
p_{CX}	0.2
p_{MX}	0
n_{imig} / n_{eli}	0 / 0
Počet opakovaní	100

Na Obr. 14 sú vyobrazené výsledky experimentu. GA bol spustený 100x a priemerná fitenss počas behu algoritmu je vyobrazená na Obr. 14-hore. Ako vidno, tak pri použití operátora BCX dochádza konzistentne k získaniu lepších výsledkov. Potvrdzujú to aj krabicové grafy (Obr. 14-dole), kde hlavne u kontajneru *asymmetric* (Obr. 14-vpravo) jasne vidno lepšie výsledky operátoru BCX.

BCX tieto gény ako celky prehadzuje medzi jedincami, pričom zoskupenia týchto informácií uchováva. BLCX naproti tomu náhodne vyberá, ktoré z týchto informácií prehodí medzi jednotlivými génmi jedincov a tak zrejme dochádza k rozbíjaniu stavebných blokov riešenia.

Pre ďalšie experimenty bol teda zvolený vhodnejší operátor kríženia BCX, a to ako pre GA1, tak aj pre GA2.



Obr. 14 – Výber operátoru kríženia – porovnanie medzi BLCX a BCX: priebeh fitness funkcie počas evolúcie (hore) a krabicové grafy hodnoty fitness funkcie na konci evolúcie (dole) u kontajnera *norm_small_1* (vľavo) a *asymmetric* (vpravo) pre sadu objektov *simple*

5.4.2 Pravdepodobnosť kríženia

Pri experimentoch s cieľom nájsť vhodnú pravdepodobnosť kríženia p_{CX} bolo použité nastavenie podľa Tab. 6. Sledovaná bola hodnota fitness funkcie na konci evolúcie. Úloha nájsť optimálnu hodnotu p_{CX} sa ukázala ako problematická, keďže experimenty na rôznych testovacích úlohách dávajú nekonzistentné výsledky. Preto bolo nutné pristúpiť k experimentom s viacerými kombináciami malých objektov a kontajnerov. Konkrétne boli vykonané experimenty s kombináciami *asymmetric/simple, norm_small_1/simple, norm_small_2/complex, excenter_small/complex*.

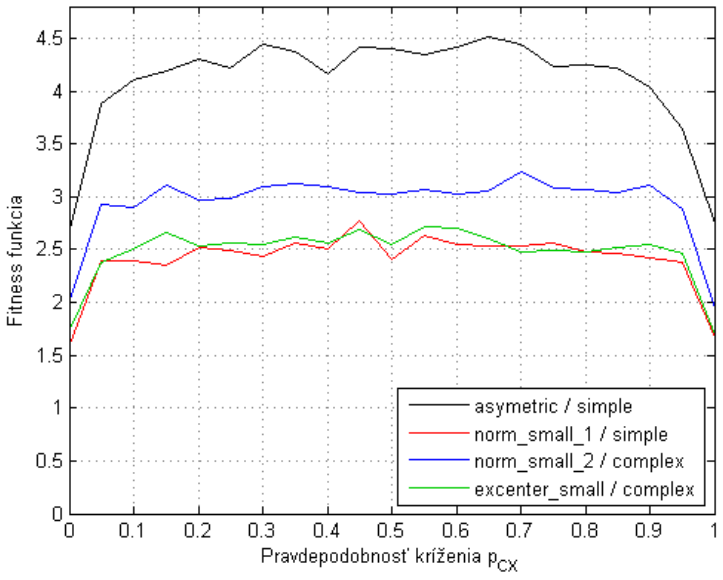
Tab. 6 – Nastavenie experimentov s GA1 pre určenie vhodnej pravdepodobnosti kríženia

Testovacie sady	asymmetric / simple norm_small_1 / simple norm_small_2 / complex excenter_small / complex
p_{CX}	0.0-1.0
p_{MX}	0.1
n_{imig} / n_{eli}	0 / 0
Počet opakovaní	100

Takisto bolo empiricky zistené, že pri zakázaní mutácie nastavením $p_{MX} = 0$ dochádza k zániku vhodných rekombinačných vlastností GA (kapitola 5.4.3), preto boli experimenty vykonávané s nastavením $p_{MX} = 0.1$. Malá pravdepodobnosť mutácie nerozruší markantne evolučný proces a zabezpečí dostatočnú diverzitu pri evolúcii.

V prvom kroku boli vykonané experimenty s hodnotou p_{CX} v plnom rozmedzí 0.0-1.0 s odstupňovaním 0.1. Ako vidno na Obr. 15, ukázalo sa, že závislosť dosiahnutej fitness funkcie je symetrická vzhľadom na jej stred. Po bližšom preskúmaní funkcie operátora BCX je toto správanie pochopiteľné. Pri $p_{CX} = 0.10$ bude 10% génov vymenených, pričom pri $p_{CX} = 0.90$ bude vymenených 90% génov, výslední potomkovia sú rovnakí. Vďaka tejto symetrii je možné zredukovať hľadanie vhodnej hodnoty p_{CX} na interval $< 0.0; 0.5 >$.

Ako vidno z Obr. 15, tak hodnoty fitness funkcii pre jednotlivé experimenty sú rôzne. To komplikuje určenie vhodnej hodnotu p_{CX}, ktorá by bola vo všeobecnosti čo najvhodnejšia. Nie je možné jednoducho uvažovať o hodnotách fitness funkcie pri rôznych experimentoch pri danom parametri p_{CX} ako o hodnotách z jedného náhodného výberu.



Obr. 15 – Závislosť hodnoty fitness funkcie u GA1 na pravdepodobnosti kríženia p_{CX}

Za účelom vyriešenia tohto problému bola navrhnutá nasledujúca normalizácia hodnôt fitness funkcií:

$$ff'_{i,j,k} = \frac{ff_{i,j,k}}{\max_j(\overline{ff}_{j,k})}$$

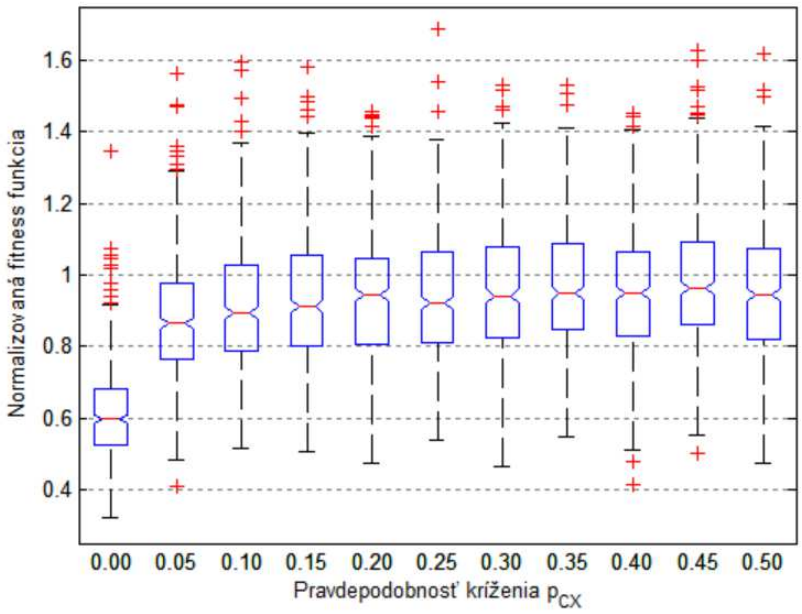
$$\overline{ff}_{j,k} = \frac{1}{n}\sum_{i=1}^{n} ff_{i,j,k} \qquad (23)$$

kde:
$ff'_{i,j,k}$ – je normalizovaná hodnota fitness funkcie
k – je index daného experimentu (*asymmetric/simple, excenter_small/complex* ..)
j – je index experimentu pre istú hodnotu p_{CX}
i – je index jedného konkrétneho behu GA v rámci experimentu
$ff_{i,j,k}$ – je hodnota fitness funkcie
$\overline{ff}_{j,k}$ – je hodnota fitness funkcie vypočítaná ako priemer zo všetkých behov v rámci jedného experimentu

Normalizovaná hodnota fitness funkcie sa určí ako pomer medzi hodnotou fitness funkcie a maximálnou dosiahnutou priemernou hodnotou fitness funkcie v rámci jedného experimentu. Tým by sa mala dosiahnuť normalizácia hodnôt do intervalu $< 0; 1 >$, avšak vzhľadom na fakt, že sa normalizuje maximom z priemernej hodnoty, tak v skutočnosti niektoré hodnoty normalizovanej fitness funkcie môžu byť u odľahlých hodnôt ďaleko väčšie ako 1 (Obr. 16).

Všetky normalizované hodnoty fitness funkcii boli následne zoskupené do množín podľa parametra p_{CX}. Navyše vďaka symetrii závislosti boli zoskupené aj merania pre $p_{CX} = 0.0$ spolu s $p_{CX} = 1.0$, $p_{CX} = 0.1$ spolu s $p_{CX} = 0.9$ atď. Z takto spracovaných dát boli vykreslené krabicové grafy zobrazené na nasledujúcom obrázku (Obr. 16).

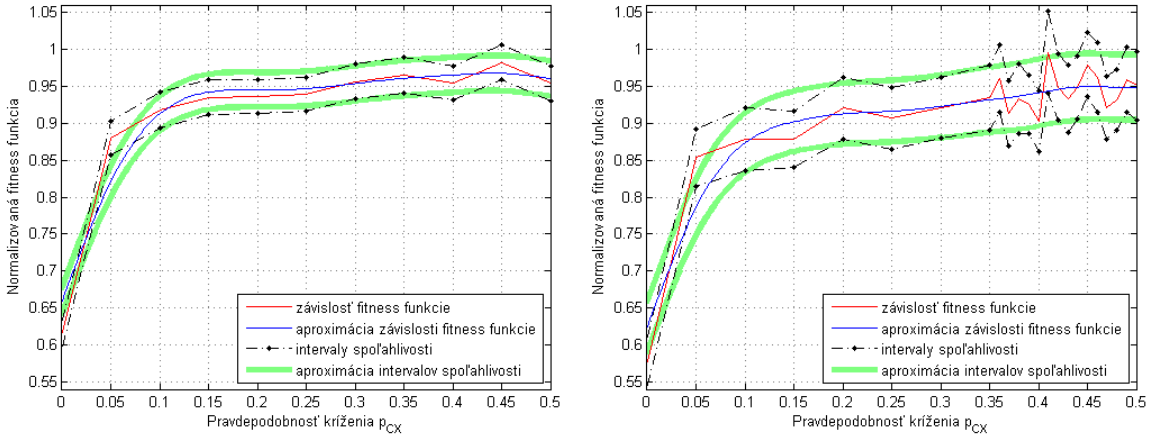
Z grafov je zrejmé, že rozptyl získaných hodnôt je značný a takisto, že hodnota fitness funkcie na konci behu zrejme nenasleduje normálne rozdelenie. Napriek tomu je z grafov viac-menej vidno, že GA pracuje lepšie s rastúcou hodnotou p_{CX} a najlepšie z experimentov dopadol experiment s $p_{CX} = 0.45$. Vzhľadom na rozptyl, ktorý krabicové grafy vykazujú sa však nedá s istotou tvrdiť že toto je optimálna hodnota.



Obr. 16 – Krabicové grafy normalizovaných hodnôt fitness funkcie pre pravdepodobnosti kríženia p_{CX} z intervalu <0.0;0.5>.

Pre presnejší odhad vhodnej pravdepodobnosti kríženia p_{CX} boli dáta ďalej analyzované s cieľom získať aproximáciu závislosti fitness funkcie pomocou krivky. Napriek indíciám krabicových grafov, že u fitness funkcie na konci behu GA sa nejedná o normálne rozloženie, bolo považované toto rozdelenie v ďalšej analýze za normálne. Keďže je potrebné porovnať výkonnosť algoritmu vzhľadom na hodnotu p_{CX}, a nie je potrebné presne určiť stredné hodnotu, tak takéto zjednodušenie by nemalo vniesť signifikantnú chybu.

Na Obr. 17-vľavo je vyobrazený priebeh normalizovanej fitness funkcie (červená), ktorý vznikol jednoduchým spriemerovaním nameraných hodnôt a jeho intervalový odhad (čierna), ktorý bol počítaný podľa rovnice (22) pre normálne rozdelenie. Modrou a zelenou farbou sú vykreslené aproximácie priebehu a intervalových odhadov, ktoré vznikli aproximáciou pomocou smoothing-spline metódy. Pomocou tohto prístupu bola získaná optimálna hodnota $p_{CX} = 0{,}443$.



Obr. 17 – Aproximácia závislosti normalizovanej fitness funkcie u pôvodného experimentu (vľavo) a u spresňujúceho experimentu (vpravo).

Pre presnejšie určenie vhodnej pravdepodobnosti p_{CX} boli vykonané dodatočné experimenty s kontajnermi a malými objektami: *asymmetric/simple, norm_small_1/simple* pre $p_{CX} \in < 0.35; 0.5 >$ s rozlíšením 0.01. Výsledok je na Obr. 17-vpravo. Ako vidno, tak intervaly spoľahlivosti sú širšie ako u pôvodného experimentu čo je spôsobené menším počtom meraní na jeden bod. U tohto experimentu bola získaná optimálna hodnota $p_{CX} = 0{,}453$.

Výsledkom zo všetkých uvedených experimentov je fakt, že optimálna hodnota p_{CX} leží s najväčšou pravdepodobnosťou niekde v intervale $< 0.3; 0.5 >$ a s veľkou pravdepodobnosťou nebude rovná 0.5 (Obr. 17-vľavo). Pre ďalšie experimenty s GA1 bola ako optimálna hodnota zvolená $p_{CX} = 0{,}45$. Rovnaká pravdepodobnosť kríženia bola následne použitá aj u GA2.

5.4.3 Pravdepodobnosť mutácie u GA1

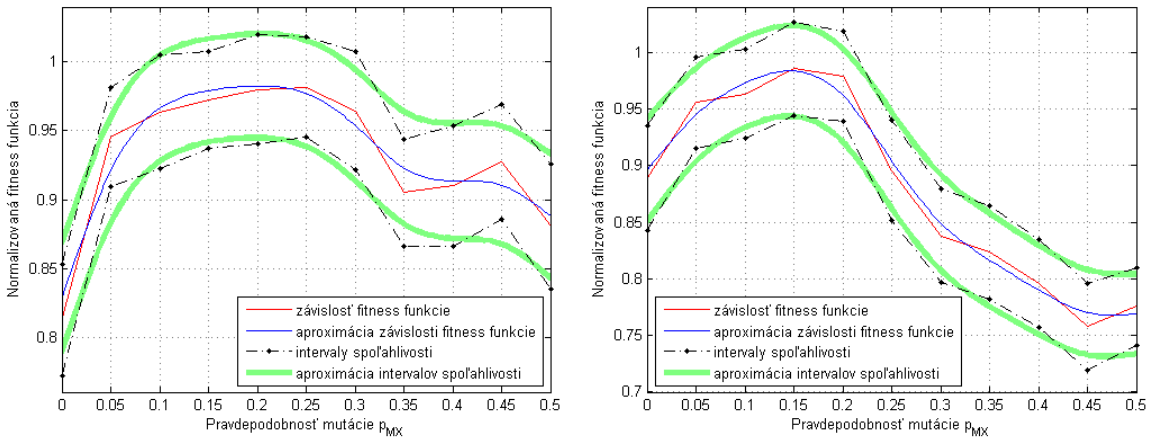
U algoritmu GA1 boli navrhnuté dva mutačné operátory:

7. OPSO – One Point Swap Order mutation – jednobodový mutačný operátor meniaci poradie génov jedinca. Jedná sa o mutačný operátor relatívne štandardný u GA.
8. RA/UA – Random Angle / Uniform Angle mutation – jedná sa o špecificky navrhnuté oprátory pre GA1, ktoré majú za úlohu meniť uhol vloženia pri konštruktívnej heuristike. RA náhodne prechádza všetky gény jedinca a s pravdepodobnosťou 0.2 zmení daný gén o náhodne vygenerovaný uhol. UA mení všetky gény jedinca o náhodne vygenerovaný uhol.

Úlohou v tejto časti experimentov je zistiť ktorý z operátorov RA/UA je vhodnejší a zistiť optimálne nastavenie pravdepodobností mutačných operátorov pre GA1. Pri týchto experimentoch sa ukázalo, že závislosti fitness funkcií relatívne dobre reagujú na zmenu parametrov a fitnes funkcie vykazovali malý rozptyl na konci behu GA a tak postačovalo vykonať 10 opakovaní pri jednom nastavení. Efektivita mutačných operátorov bola preverená pre každý osobitne – t.j. v jednom experimente bol povolený iba daný operátor podrobovaný testu. Nastavenie experimentov je zhrnuté v nasledujúcej tabuľke (Tab. 7). Pri experimentoch boli použitý obdobný postup ako v prípade kapitoly 5.4.2. T.j. bola použitá normalizácia fitness funkcie podľa rovnice (23) a výpočet intervalov spoľahlivosti podľa rovnice (22).

Tab. 7 – Nastavenie experimentov s GA1 pre určenie vhodných operátorov mutácie a ich pravdepodobností

Testovacie sady	asymmetric / simple, norm_small_1 / simple
p_{CX}	0.45
p_{MX}	0-0.5
n_{imig} / n_{eli}	0 / 0
Počet opakovaní	20

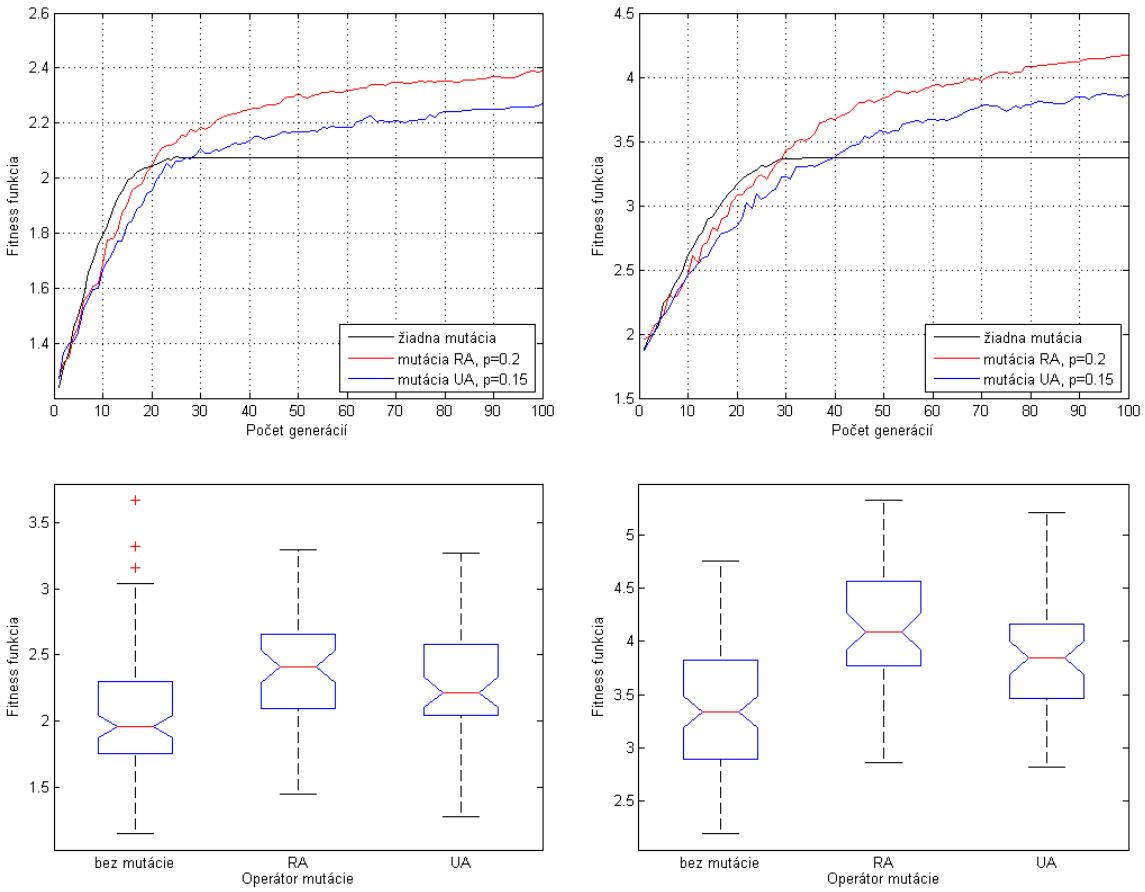


Obr. 18 – Aproximácia závislosti normalizovanej fitness funkcie u GA1 od pravdepodobnosti mutácie pre mutačný operátor RA (vľavo) a UA (vpravo)

Na Obr. 18 sú uvedené závislosti normalizovanej fitness funkcie od pravdepodobnosti mutácie pre operátory RA a UA. Z týchto grafov boli určené optimálne pravdepodobnosti pre RA: $p_{MX}^{RA} = 0.201$ a pre UA: $p_{MX}^{UA} = 0.147$. Pre ďalšie experimenty boli tieto pravdepodobnosti zaokrúhlené a ďalej sú použité $p_{MX}^{RA} = 0.2$ a $p_{MX}^{UA} = 0.15$.

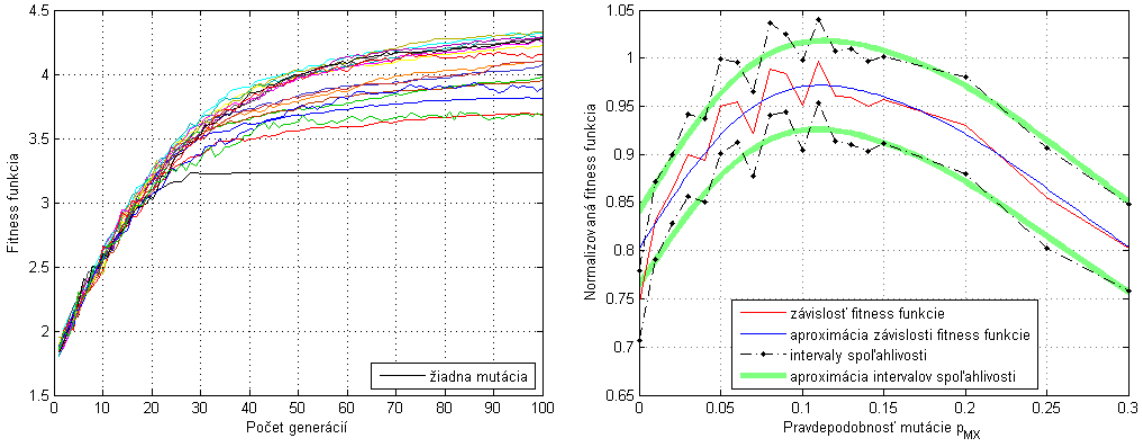
Operátory RA a UA boli navzájom porovnané pri zistených optimálnych nastaveniach a výsledok experimentov je zobrazený v nasledujúcich grafoch (Obr. 19). Podľa priebehu fitness funkcie počas evolúcie GA, sa operátor RA javí byť jasne lepší. Štatisticky signifikantné však toto tvrdenie veľmi nie je, pretože ako vidno tak zárezy na krabicových grafoch , ktoré zobrazujú intervaly spoľahlivosti sa čiastočne prekrývajú u obidvoch testov. Ako jasne vidno, tak obidva operátory prinášajú pre GA1 pozitívne výsledky. Podľa priebehu fitness funkcie sa dá povedať, že evolúcia sa po 30 generáciách zastaví a ďalej sa už nevyvíja. Je to celkom prirodzené, keďže v algoritme pracuje iba operátor kríženia BLX a počet imigrantov $n_{imig} = 0$, takže do populácie nepribúda žiadny nový genetický materiál.

Pre ďalšie použitie v algoritme GA1 bol podľa týchto výsledkov zvolený operátor RA s pravdepodobnosťou $p_{MX}^{RA} = 0.2$.



Obr. 19 – Výsledky porovnania mutačných operátorov RA a UA pre algoritmus GA1. Priebeh fitness funkcie počas evolúcie (hore) a krabicové grafy hodnoty fitness funkcie na konci evolúcie (dole) u kontajnera *norm_small_1* (vľavo) a *asymmetric* (vpravo) pre sadu objektov *simple*

Ďalším operátorom mutácie pre GA1 je operátor OPSO. Pre tento operátor boli prevedené rovnaké experimenty ako v prípade operátorov RA a UA. Výsledky sú zobrazené na Obr. 20. V ľavej časti je ilustrovaný priebeh hodnoty fitness funkcie počas evolúcie pre testovací súbor *asymmetric/simple.* Účel tohto grafu je hlavne zachytiť fakt, že akákoľvek hodnota pravdepodobnosti mutácie je lepšia ako žiadna mutácia. V pravej časti je zahytená závislosť fitness funkcie od p_{MX}^{OPSO} a z aproximácie tejto závislosti bola určená optimálna hodnota $p_{MX}^{OPSO} = 0.11$.



Obr. 20 – Priebeh fitness funkcie počas evolúcie GA1 pre testovací súbor *asymmetric/small* pre rôzne pravdepodobnosti mutácie operátoru OSPO (vľavo) a závislosť normalizovanej fitness funkcie na tejto pravdepodobnosti (vpravo)

Pre praktické nastavenie GA1 museli byť uvedení operátory mutácie ešte navzájom porovnané pri vzájomnej interakcii. Výsledky týchto experimentov sú na Obr. 21. Kvôli lepšej presnosti boli dáta vyhodnotené zo 100 behov algoritmu pre každú kombináciu. Okrem behov algoritmu s jednotlivými operátormi boli prevedené tri experimenty C1-C3, pri ktorých boli tieto operátory kombinované:

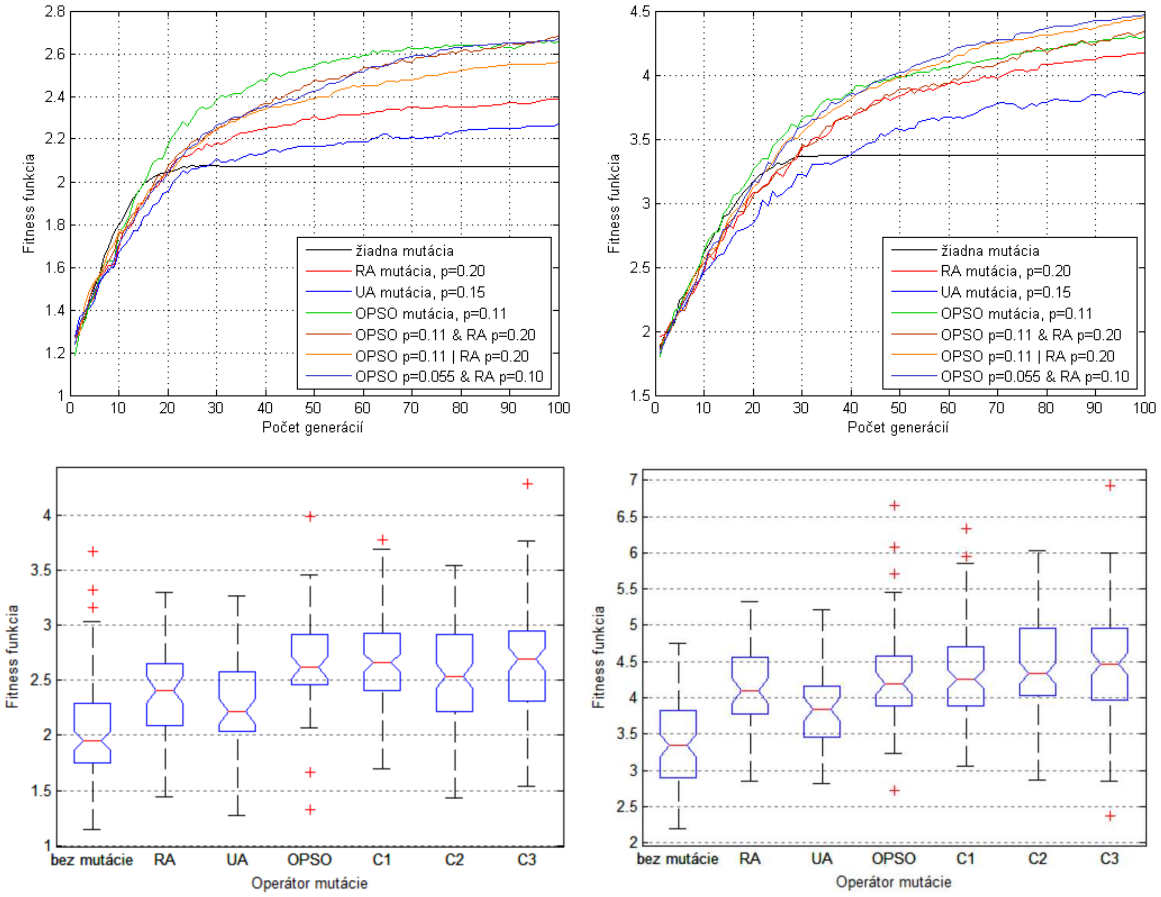
9. C1 – nasadené boli obidva operátory OPSO aj RA so zistenými najlepšími pravdepodobnosťami $p_{MX}^{OPSO'} = 0.11$ a $p_{MX}^{RA'} = 0.2$. Mutácia bola prevedená najprv operátorom OSPO , a potom operátorom RA
10. C2 – podobne ako u C1 s rozdielom, že mutácia sa vykonala exkluzívne buď s jedným, alebo druhým operátorom
11. C3 – rovnaká funkčnosť ako u C1, ale pravdepodobnosti boli ponížené na polovicu, čiže na $p_{MX}^{OPSO'} = 0.055$ a $p_{MX}^{RA'} = 0.1$

Ako vidno, tak C3 dopadol najlepšie. Samozrejme štatisticky je tento výsledok nevýznamný, ale zrejme táto kombinácia pracuje pre GA1 najlepšie na daných testovacích sadách. Vysvetlenie, prečo kombinácia s polovičnými pravdepodobnosťami pracuje najlepšie sa opiera o fakt je , že GA vo všeobecnosti potrebuje istú hodnotu diverzity pre správnu funkciu. Ak jeden operátor pri istej nastavenej pravdepodobnosti pracuje optimálne, znamená to, že pre GA prináša akurát vhodnú úroveň diverzity. Ak sa takéto dva optimálne naladene operátory skombinujú, tak v podstate prinesú pre GA viac

diverzity ako je potrebné. Pri znížení pravdepodobností u obidvoch operátorov na polovicu došlo k zlepšeniu vlastností GA, avšak nie je možné tvrdiť, že výsledok je štatisticky významný.

Tab. 8 – Najlepšia zistená konfigurácia mutačných a rekombinačných operátorov pre GA1

BCX – Bernoulli Crossover	$p_{CX} = 0.45$
OPSO – One Point Swap Order mutation	$p_{MX}^{OPSO} = 0.055$
RA – Random Angle mutation	$p_{MX}^{RA} = 0.1$



Obr. 21 – Výsledky porovnania mutačných operátorov pre GA1 u kontajnera *norm_small_1* (vľavo) a *asymmetric* (vpravo) pre sadu objektov *simple*. Priebeh fitness funkcie počas evolúcie (hore) a krabicové grafy hodnoty fitness funkcie na konci evolúcie (dole). Nastavenie pravdepodobnosti: RA:0.2, UA:0.15, OPSO:0.11, experimenty C1-C3 sú pre kombinované použitie operátorov (popis viď v texte)

5.4.4 Elitizmus a imigrácia u GA1

Elitizmus u GA znamená, že algoritmus zachováva najlepšieho jedinca z aktuálnej generácie a prenáša ho automaticky bez prechodu procesom selekcie. U GA zvyčajne dôjde k zlepšeniu konvergencie, keďže neskartujeme potenciálne najlepšieho jedinca, avšak algoritmus takisto stráca schopnosť uniknúť z lokálneho maxima. Preto bol v štruktúre GA (Obr. 2) navrhnutý proces imigrácie, ktorý vnesie do populácie nastavený počet náhodne vygenerovaných jedincov a tým by mal pomôcť zachovať diverzitu populácie potrebnú pre únik algoritmu z lokálneho optima. Pre overenie týchto tvrdení boli vykonané experimenty s rôznym nastaveniami parametrov n_{imig} a n_{eli}. Nastavenie experimentov je zhrnuté v nasledujúcej tabuľke (Tab. 9).

Tab. 9 – Nastavenie experimentov s GA1 pre overenie elitizmu a mechanizmu imigrácie

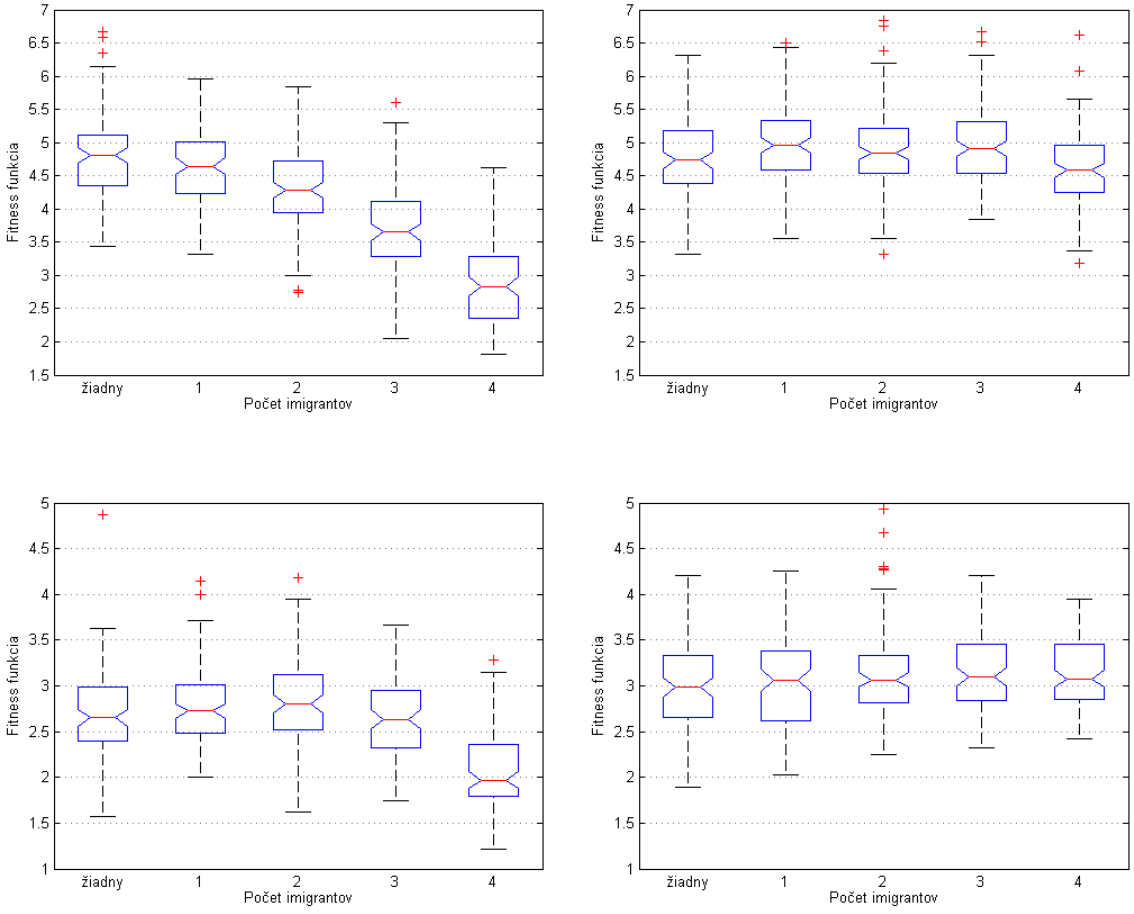
Testovacie sady	asymmetric / simple, norm_small_1 / simple
BCX – Bernoulli Crossover	$p_{CX} = 0.45$
OPSO – One Point Swap Order mutation	$p_{MX}^{OPSO} = 0.055$
RA – Random Angle mutation	$p_{MX}^{RA} = 0.1$
$\boldsymbol{n_{imig}}$	0 - 4
$\boldsymbol{n_{eli}}$	0, alebo 1
Počet opakovaní	100

Výsledky experimentov sú zhrnuté vo forme krabicových grafov na Obr. 22. Ako vidno, tak zavedním imigrácie v neelitistickej variante došlo u testovacieho súboru *asymmetric / small* k degradácii evolučného procesu. Dôvodom je zrejme fakt, že parametre operátorov mutácie boli nastavené na optimálne hodnoty bez použitia imigrácie a akákoľvek väčšia diverzita vnesená imigráciou má nepriaznivý dopad. U testovacieho súboru *norm_small_1* zjavne GA1 potrebuje väčšiu diverzitu a tak najlepšie výsledky dosahuje pri dvoch imigrantoch. Pri elitistických variantoch experimentov však vidno, že nielen že elitizmus zabezpečí dosiahnutie lepších výsledkov, ale zavedenie imigrácie ešte ďalej napomôže procesu evolúcie. Zatiaľ čo u testovacieho súboru *norm_small_1 / simple* pracuje pozitívne akýkoľvek počet imigrantov, tak u zložitejšieho problému *asymmetric / simple* znova nastáva degradácia procesu evolúcie pri väčších počtoch imigrantov. Pre ďalšie použitie algoritmu GA1 bola teda zvolená elitistická varianta z parametrom $n_{eli} = 1$ a použitie jedného imigranta, $n_{imig} = 1$. Množstvo nového genetického materiálu zabezpečovaného genetickými operátormi v GA je úmerné veľkosti populácie. Preto by aj počet imigrantov mal byť závislý na jej veľkosti. V Tab. 10, kde je uvedené najlepšie zistené nastavenie GA1, je uvedená percentuálna hodnota 5% imigrantov, keďže experimenty prebiehali s populáciou 20 jedincov.

Tab. 10 – Najlepšia zistená konfigurácia algoritmu GA1

BCX – Bernoulli Crossover	$p_{CX} = 0.45$
OPSO – One Point Swap Order mutation	$p_{MX}^{OPSO} = 0.055$
RA – Random Angle mutation	$p_{MX}^{RA} = 0.1$
n_{eli}	1
n_{imig}	1 / (5%)

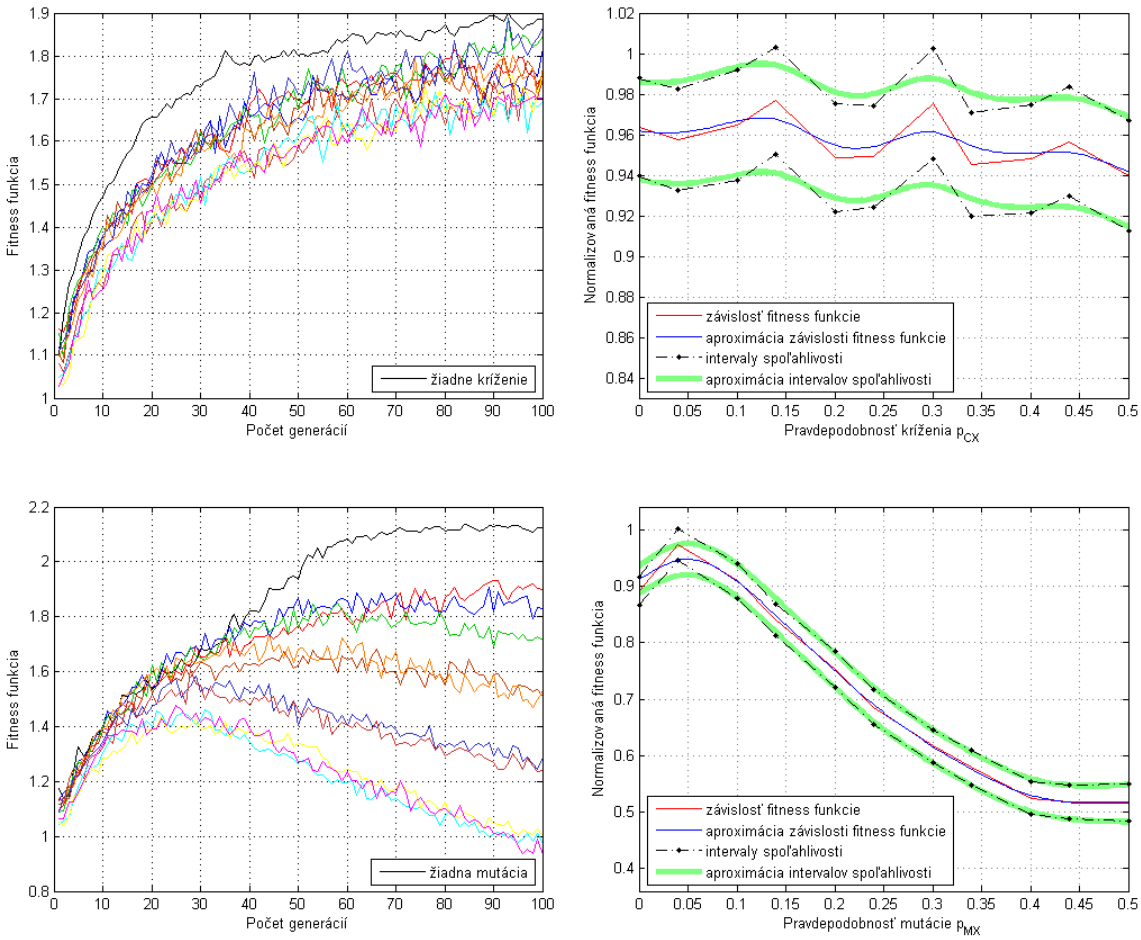
Ako bolo podotknuté na začiatku kapitoly 0, tak nastavenie parametrov GA je vo všeobecnosti veľmi zložitá úloha, keďže parametre ako sú pravdepodobnosti jednotlivých operátorov veľkosť populácie, počet generácií, či je algoritmus elitistický atď. sa navzájom ovplyvňujú. Rozsiahlejšie experimenty a sofistikovanejšie techniky sú však mimo rozsah tejto práce a preto sa budú zistené parametre algoritmu GA1 považovať za optimálne.



Obr. 22 – Výsledky experimentov pre overenie elitizmu a machanizmu imigrácie u GA1 pre testovaciu sadu *asymmetric / simple* (hore) a *norm_small_1* (dole). Neelitistická varianta (vľavo), elitistická varianta (vpravo)

5.5 Experimenty s GA2

Ako úplne prvé, čo bolo zistené pri experimentoch s GA2 je fakt, že naladenie parametrov tohto algoritmu je omnoho problematickejšie ako u GA1. Zo začiatku experimentov často dochádzalo k tomu, že získané merania boli úplne nesmerodajné a pripomínali skôr náhodné dáta a to aj pri viacerých testovacích sadách, či veľkom množstve opakovaní. Príklad takýchto získaných dát je na Obr. 23, kde boli experimenty prevedené na testovacách sadách *asymmetric / simple, norm_small_1 / simple, norm_small_2 / complex* a *excenter_small / complex* s počtom opakovaní 100x.



Obr. 23 – Výsledky neúspešných experimentov o nájdenie optimálnej pravdepodobnosti kríženia (hore) a mutácie pre operátor OPSO (dole) u GA2. Priebeh fitness funkcie u testovecieho súboru *asymmetric / simple* počas evolúcie GA2 (vľavo) a závislosť normalizovanej fitness funkcie na pravdepodobnosti kríženia/mutácie (vpravo)

Jedným z prvých experimentov ktoré boli s GA2 vykonané bolo hľadanie vhodnej pravdepodobnosti pre operátor Bernoulliho kríženia BCX. Podobne ako u algoritmu GA1 bola zvolená neelitistická varianta s nastavením $n_{eli} = 0$ a s natavenými malými pravdepodobnosťami mutácie u všetkých operátorov na $p_{MX} = 0{,}05$ aby sa zabezpečil nový dostupný genetický materiál v GA. Ako vidno z výsledných grafov (Obr. 23-hore), tak nie je z nich možné určiť rozumnú závislosť. Dokonca na priebehu fitness funkcie

u testovacieho súboru *asymmetric / simple* je vidno, že bez kríženia bol dosahovaný najlepší výsledok.

Ďalej bol prevedený pokus o nájdenie optimálnej pravdepodobnosti mutácie pre operátor OSPO. Výsledky tohto experiemntu sú na Obr. 23-dole. Graf závislosti fitness funkcie na p_{MX}^{OPSO} zachycuje, že optimálna pravdepodobnosť by mala byť takmer nulová. Na Obr. 23-vľavo-dole je zachytená priemerná hodnota fitness funkcie pre testovací súbor *asymmetric / simple*, na ktorom je vidno ako radikálne zaostáva hodnota fitness funkcie pri akomkoľvek nastavení p_{MX}^{OPSO} a dokonca pri niektorých hodnotách dochádza k deštrukcii riešenia.

Dôvod tohto správania bol hľadaný v prílišnej diverzite populácie ktorá zapríčiní až skĺznutie GA k náhodnému prehľadávaniu stavového priestoru. Experimenty s nižšími hodnotami $p_{MX} < 0.005$ však toto nepotvrdili. Zaujímavý je fakt, že pri vhodnej pravdepodobnosti mutácie OPSO ($p_{MX}^{OPSO} = 0.12$) sa začne GA2 správať ‘relatívne rozumne‘. Avšak konzistentné správanie algoritmu bolo docielené až po rozsiahlom experimentovaní zavedením elitizmu nastavením parametru $n_{eli} = 1$ a za použitia pravdepodobnosti kríženia $p_{CX} = 0.45$ pre operátor BCX, ktorá bola overená u algoritmu GA1 (kap.5.4.2). Výsledky algoritmu boli konzistentné natoľko, že postačovalo použitie dvoch testovacích sád a počet opakovaní experimentu bol nastavený na 20. Nastavenia algoritmu GA2 pre účely experimentov s jeho nastavením sú zhrnuté v nasledujúcej tabuľke (Tab. 11). Ostatné parametre použité štandardne pri experimentoch s parametrami GA sú uvedené v Tab. 4 v kap. 5.3.1.

Tab. 11 – Štandardné nastavenie experimentov s GA2

Testovacie sady	asymmetric / simple norm_small_1 / simple
p_{CX}	0.45
n_{imig} / n_{eli}	0 / 1
Počet opakovaní	20

5.5.1 Pravdepodobnosť mutácie u GA2

U algoritmu GA2 bol použitý rovnako ako u algoritmu GA1 operátor mutácie OPSO – One Point Swap Order mutation. Navyše boli pre tento algoritmus vytvorené špecificky stavané operátory mutácie, ktoré súvisia s konštruktívnou heuristikou GA2 a štruktúrou jeho jedincov:

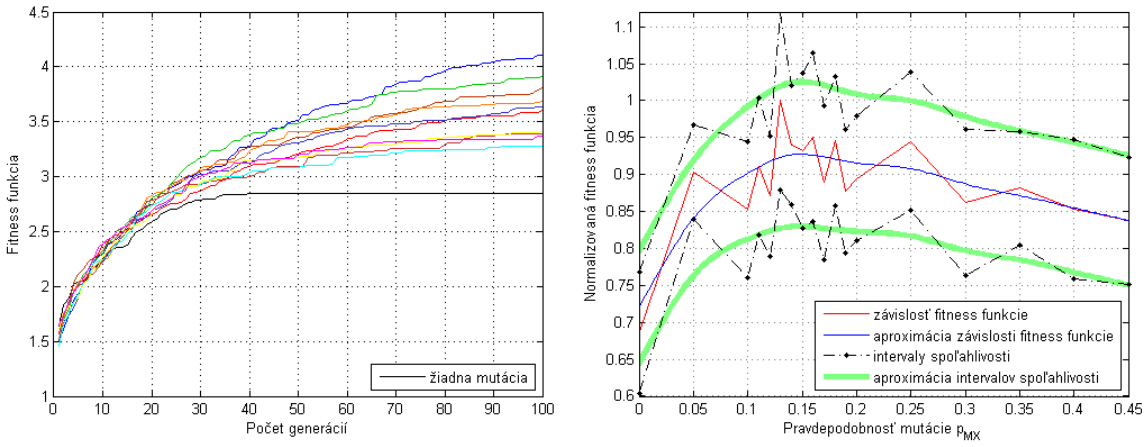
12. RH – Random Head mutation – je operátor ktorý špecificky mutuje prvý gén jedinca, ktorý ma v sebe zakódovanú pozíciu vloženia prvého malého objektu. Od tohto operátora je očakávané doladenie riešenia na najvhodnejšiu pozíciu
13. RR – Random Rotation mutation ovplyvňuje smer odkiaľ bude bude malý objekt vložený pri konštruktívnej heuristike
14. RPO – Random Placement Orientation mutation ovplyvňuje orientáciu malého objektu

15. RIP – Random Insert Point mutation ovplyvňuje polohu bodu v ktorom bude malý objekt vkladaný

Všetky tieto operátory boli vytvorené na zaistenie možnosti konvergencie GA ku globálnemu optimu. Každý z nich umožňuje zmenu iného typu génu jedinca. Preto nie je možné žiaden z nich vynechať podobne ako to bolo u operátorov RA vs. UA pre algoritmus GA1. Operátory RR, RPO a RIP boli navrhnuté na základe úspechu operátora RA u GA1 (viď Obr. 19 v kapitole 5.4.3). Podobne ako operátor RA, aj tieto operátory mutujú všetky gény daného jedinca s pravdepodobnosťou 0.2.

Úspešnosť každého operátora bola preverená nezávisle na ostatných – t.j., ak bol jeden operátor aktívny, tak všetky ostatné boli deaktivované. Pri experimentoch bol použitý obdobný postup ako v prípade kapitoly 5.4.2. t.j. bola použitá normalizácia fitness funkcie podľa rovnice (23) a výpočet intervalov spoľahlivosti podľa rovnice (22). Ako prvý bol preverený operátor OPSO ktorý pracuje trochu inak ako ostatné operátoru u GA2. OPSO má za úlohu kompletne prehadzovať poradie génov jedinca a tým potenciálne vytvárať lepšie riešenia z už nájdených stavebných blokov, zatiaľ čo ostatné operátory sa zameriavajú na mutáciu jednotlivých typov génov a tým pádom vnášajú nový genetický materiál do GA. Preto bol operátor OPSO vyhodnotený s trochu väčším zreteľom.

Na Obr. 24 sú vyobrazené výsledky experimentu pre nájdenie vhodnej p_{MX}^{OPSO} . V ľavej časti je zobrazný priebeh fitness funkcie počas evolúcie. Špeciálne treba poukázať na rozdiel medzi týmito priebehmi a priebehmi fitness funkcie na Obr. 23-vľavo-dole. U tohto experimentu je zreteľný prínos operátora, zatiaľ čo na Obr. 23 pôsobí operátor skôr deštruktívne. Toto správanie bolo zabezpečené primárne zavedením spomínaného elitizmu a sekundárne zakázaním ostatných mutačných operátorov, čo zabezpečí lepšiu konvergenciu fitness funkcie. S cieľom spresniť odhad bolo prevedených viacero meraní v oblasti, kde sa nádejné maximum nachádzalo – ako vidno grafu. Výsledná najvhodnejšia pravdepodobnosť mutácie odhadnutá aproximačným modelom je $p_{MX}^{OPSO} = 0.149$.

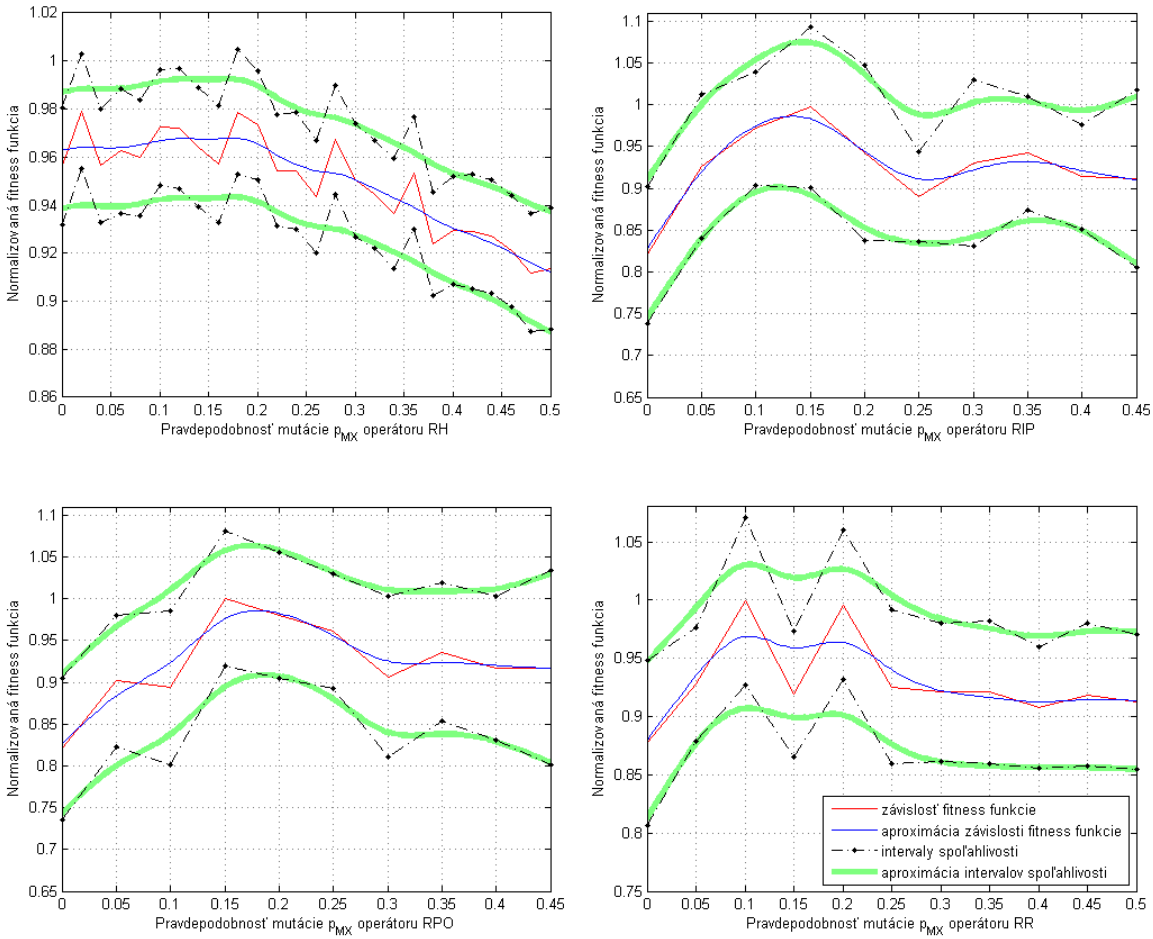


Obr. 24 – Priebeh fitness funkcie počas evolúcie GA2 pre testovací súbor *asymmetric/small* pre rôzne pravdepodobnosti mutácie operátoru OSPO (vľavo) a závislosť normalizovanej fitness funkcie na tejto pravdepodobnosti (vpravo)

Experimenty pre ostatné mutačné operátory GA2: RH, RIP, RPO a RR boli prevedené rovnakou metodikou a výsledné priebehy fitness funkcií sú zhrnuté na Obr. 25. V Tab. 12 sú zhrnuté najlepšie zistené hodnoty pravdepodobnosti mutácie pre jednotlivé operátory. Jedná sa však o pravdepodobnosti zistené pri sólo nasadení daného operátora.

Tab. 12 – Najlepšie hodnoty pravdepodobností pre jednotlivé mutačné operátory GA2

OPSO – One Point Swap Order mutation	$p_{MX}^{OPSO} = 0.15$
RH – Random Head mutation	$p_{MX}^{RH} = 0.17$
RIP – Random Insert Point mutation	$p_{MX}^{RIP} = 0.13$
RPO – Random Placement Orientation mutation	$p_{MX}^{RPO} = 0.17$
RR – Random Rotation mutation	$p_{MX}^{RR} = 0.15$

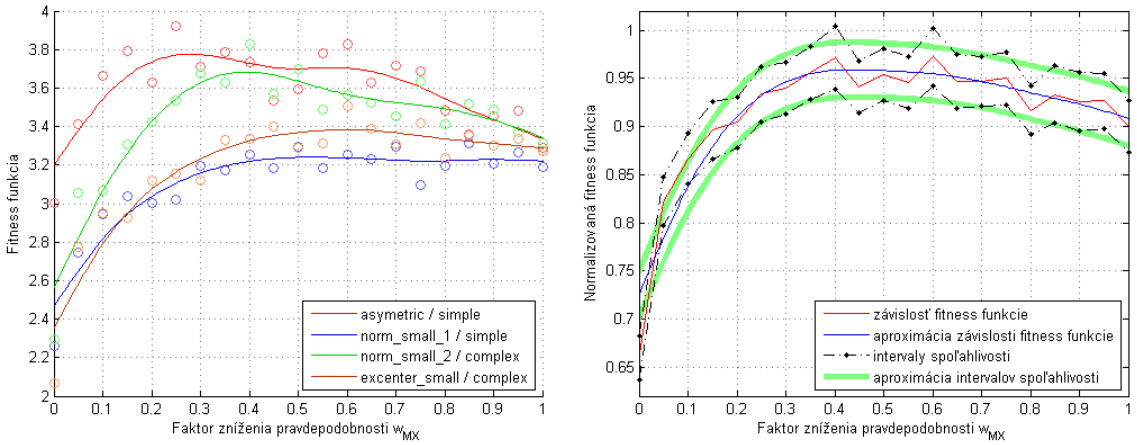


Obr. 25 – Výsledky experimentov na určenie optimálnych pravdepodobností mutácie operátorov GA2. Operátory: RH (hore-vľavo), RIP (hore-vpravo),RPO (dole-vľavo),RR (dole-vpravo)

Ak by boli tieto operátory nasadené s pravdepodobnosťami podľa Tab. 12 naraz do GA2, tak by s najväčšou pravdepodobnosťou spôsobili príliš veľkú diverzitu populácie a následnú degradáciu procesu evolúcie. Kvôli tomu bol prevedený ďalší experiment, v ktorom bol preskúmaný faktor w_{MX} zníženia jednotlivých pravdepodobnosti podľa nasledujúcej rovnice (24).

$$p'^{op}_{MX} = p^{op}_{MX}.w_{MX} \qquad op \in \{OPSO, RH, RIP, RPO, RR\} \tag{24}$$

Predpokladá sa teda, že výsledné optimálne pravdepodobnosti mutácie budú lineárnou kombináciou najvhodnejších sólo pravdepodobností z Tab. 12. Na nasledujúcich grafoch (Obr. 26) sú zachytené výsledky experimentu, ktorý musel byť spustený pre všetky 4 štandardne používané testovacie sady: *asymmetric/simple, norm_small_1/simple, norm_small_2/complex, excenter_small/complex.* Podobnou metodikou ako u predchádzajúcich experimentov bol vykreslený graf závislosti normovanej fitness funkcie od faktoru w_{MX} (Obr. 26-vpravo). Predpoklad o prílišnej diverzite vnášanej do GA2 pri spoločnom použití všetkých mutačných operátorov sa potvrdil. Ako optimálna bola z tohto priebehu určená hodnota faktoru $w_{MX} = 0{,}429$. V pravej časti Obr. 26 sú uvedené aproximácie závislosti fitness funkcie od faktoru w_{MX} pre jednotlivé testovacie sady. Z týchto priebehov vidno, že pre dve datové sady: *asymmetric / small* a *norm_small_2 / complex* je vysoká hodnota mutácie škodlivá, zatiaľ čo u dvoch zvyšných dátových sád sa toto nepotvrdilo a viac im škodí malá hodnota mutácie. Priebeh v pravej časti obrázku teda zachycuje vhodný kompromis.



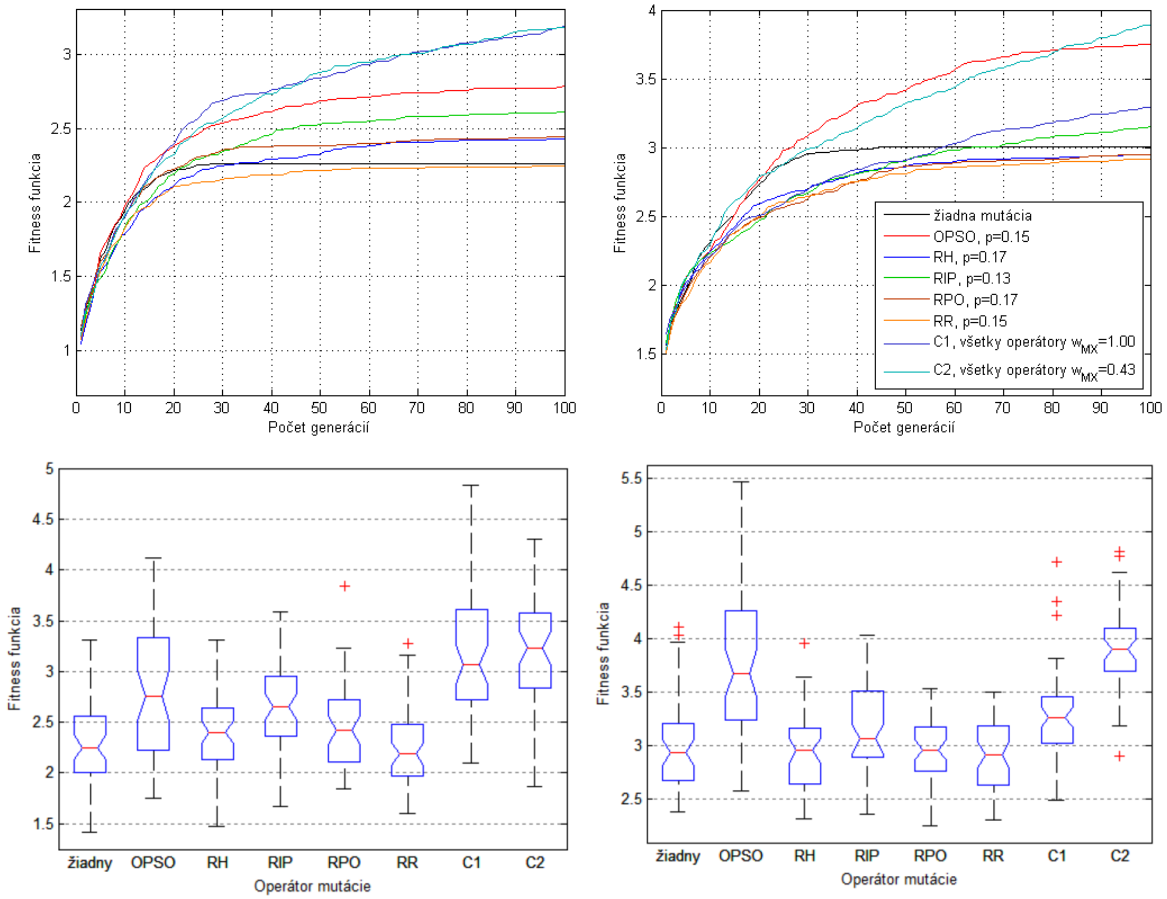
Obr. 26 – Závislosť fitness funkcií od faktoru zníženia pravdepodobnosti w_{MX} u GA2 (vľavo) a výsledná aproximácia závislosti fitness funkcie od w_{MX}. Najlepšia hodnota $w_{MX} = 0,429$.

Na záver experimentov s mutáciou u GA2 boli prevedené experimenty , ktoré majú celkovo zhodnotiť vhodnosť zvolených pravdepodobností mutácie. Výsledky týchto experimentov sú na Obr. 27. Kvôli lepšej presnosti boli dáta vyhodnotené zo 100 behov algoritmu pre každú kombináciu. U tohto experimentu bola preverená efektivita mutačných operátorov nasadených sólovo, beh GA2 bez mutačného operátora a dva experimenty s nasadením všetkých operátorov mutácie:

16. C1 – pravdepodobnosti mutácie boli nastavené podľa Tab. 12
17. C2 – podobne ako u C1, ale pravdepodobnosti boli ponížené podľa rovnice (24) faktorom $w_{MX} = 0{,}43$ zisteným v predchádzajúcom experimente

Výsledky týchto experimentov sú veľmi zaujímavé, pretože potvrdzujú hneď niekoľko tvrdení o funkcii mutačných operátorov vo všeobecnom GA ako aj tvrdenia o konkrétnych operátoroch u GA2.

Na prvý pohľad zaujme efektivita samostatne nasadeného operátoru OPSO, ktorú na konci behu algoritmu u testovacej sady *asymmetric / simple* (Obr. 27-vpravo) prekonáva všetky ostatné experimenty okrem C2. Ako však aj u testovacej sady *norm_small_1 / simple* vidno, tak algoritmus ku konci behu stráca prehľadávaciu schopnosť a hodnota fitness funkcie sa začína stabilizovať okolo jednej úrovne. Je tomu tak preto, že operátor OSPO nevnáša do GA2 žiadny nový genetický materiál iba pracuje v súčinnosti s operátorom kríženia BCX na rekombinácii genetického materiálu, ktorý sa nachádza v populácii. Prirodzene takýto algoritmus uviazne v lokálnom optime. Na to aby sa GA mohol z lokálneho optima dostať potrebuje prísun nového genetického materiálu na čo u GA slúžia operátory RH, RIP, RPO a RR. Pri experimentoch s týmito operátormi boli nasadzované takisto sólovo. Ako však vidno, bez operátoru OPSO ktorý podľa všetkého veľmi efektívne zabezpečuje nachádzanie nových stavebných blokov riešenia, je ich efektivita veľmi slabá. Dokonca horšia ako u experimentu bez akejkoľvek mutácie.



Obr. 27 – Výsledky porovnania mutačných operátorov pre GA2 u kontajnera *norm_small_1* (vľavo) a *asymmetric* (vpravo) pre sadu objektov *simple*. Priebeh fitness funkcie počas evolúcie (hore) a krabicové grafy hodnoty fitness funkcie na konci evolúcie (dole). Nastavenie pravdepodobnosti: OPSO:0.15, RH:0.17, RIP:0.13, RPO:0.17, RR:0.15, experimenty C1 a C2 sú pre kombinované použitie operátorov – viď text.

Ako sa však ukazuje, pri vhodnom nastavení parametrov nastane veľmi pozitívny zvrat a ako vidno na priebehoch fitness funkcie, tak pri ukončení GA má evolučný proces ešte stále veľkú silu. Pri testovacej sade *asymmetric / simple* si je však u krabicového grafu možné všimnúť obrovské rozpätie hodnôt fitness funkcie na konci behu algoritmu u operátoru OPSO. To sa môže zdať ako plus pre tento operátor, avšak znamená to len to, že operátor OPSO bez intervencie v zmysle prílevu novej genetickej informácie dokáže veľmi dobre lokálne rekombinovať stavebné bloky riešenia v prípade, ak má šťastie na vhodnú populáciu po inicializácii algoritmu. Naopak u experimentu C2 a testovacej sady *asymmetric / simple* sa podľa krabicového grafu ukazuje, že evolúcia pri 100. generácii prebieha veľmi konzistentne a podľa priebehu fitness funkcie je možné predpokladať, že evolúcia sa ešte zďaleka nezastavila a zrejme bude konzistentne pokračovať k lepším hodnotám fitness funkcie.

V nasledujúcej tabuľke (Tab. 13) sú zhrnuté najlepšie zistené nastavenia pre algoritmus GA2.

Tab. 13 – Najlepšia zistená konfigurácia pre GA2

n_{imig} / n_{eli}	0 / 1
BCX – Bernoulli Crossover	$p_{CX} = 0.45$
OPSO – One Point Swap Order mutation	$p_{MX}^{OPSO} = 0.0645$
RH – Random Head mutation	$p_{MX}^{RH} = 0.0731$
RIP – Random Insert Point mutation	$p_{MX}^{RIP} = 0.0559$
RPO – Random Placement Orientation mutation	$p_{MX}^{RPO} = 0.0731$
RR – Random Rotation mutation	$p_{MX}^{RR} = 0.0645$

5.5.2 Elitizmus a imigrácia u GA2

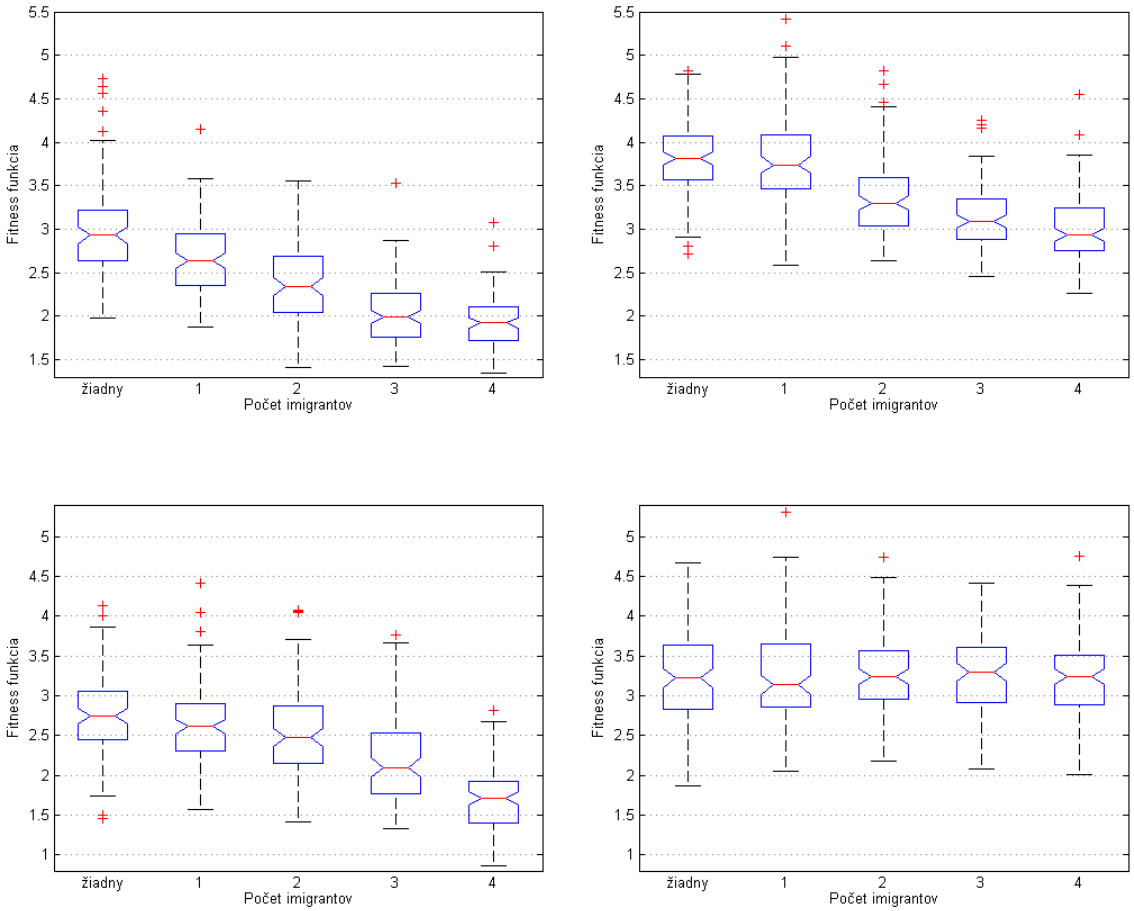
Podobne ako pre GA1 v kapitole 5.4.4 bolo potrebné u GA2 previesť experimenty pre overenie prípadnej vhodnosti zavedenia elitizmu imigrácie. U GA1 sa potvrdil pozitívny dopad zavedenia elitizmu. U GA2 však kvôli problémom s jeho nastavením bol elitizmus zavedený už na začiatku kapitoly 5.5. Nastavenie experimentov je zhrnuté v nasledujúcej tabuľke (Tab. 14), ostatné nastavenia sú podľa tabuľky Tab. 13.

Tab. 14 – Nastavenie experimentov s GA2 pre overenie elitizmu a mechanizmu imigrácie

Testovacie sady	asymmetric / simple, norm_small_1 / simple
n_{imig}	0 - 4
n_{eli}	0, alebo 1
Počet opakovaní	100

Výsledky experimentov sú zhrnuté vo forme krabicových grafov na Obr. 28. Zaujímavý je drastický rozdiel medzi medzi experimentmi bez použitia elitizmu (Obr. 28-vľavo) a s použitím elitizmu (Obr. 28-vpravo). Tento rozdiel tým pádom len potvrdzuje nutnosť a vhodnosť zavedenia elitizmu po prvotných neúspechoch s nastavením GA2. U elitistickej

varianty experimentov sa ukazuje, že zavedenie imigrácie nastavením parametru $n_{imig} = 1$ vedie k širšiemu prehľadávaniu stavového priestoru algoritmom GA2, ale zároveň zvyšuje rozptyl ohodnotenia riešení. To znamená, že algoritmus sa dostane k dobrému riešeniu menej často, no niekedy sa mu podarí veľmi dobré riešenie ako ukazuje horná vonkajšia hradba daných krabicových grafov a ich odľahlé hodnoty (Obr. 28-vpravo, 2.zľava). Avšak kvôli menšiemu výslednému mediánu bolo za najvhodnejšie nastavenie GA2 zvolené nastavenie $n_{eli} = 1, n_{imig} = 0$. Najlepšia zistená konfigurácia GA2 podľa tabuľky Tab. 13 teda ostáva nedotknutá a zistené parametre GA2 sa budú od tohto bodu považovať za optimálne. Samozrejme optimálne nastavenie GA v pravom slova zmysle nie je možné nájsť.



Obr. 28 – Výsledky experimentov pre overenie elitizmu a machanizmu imigrácie u GA2 pre testovaciu sadu *asymmetric / simple* (hore) a *norm_small_1* (dole). Neelitistická varianta (vľavo), elitistická varianta (vpravo)

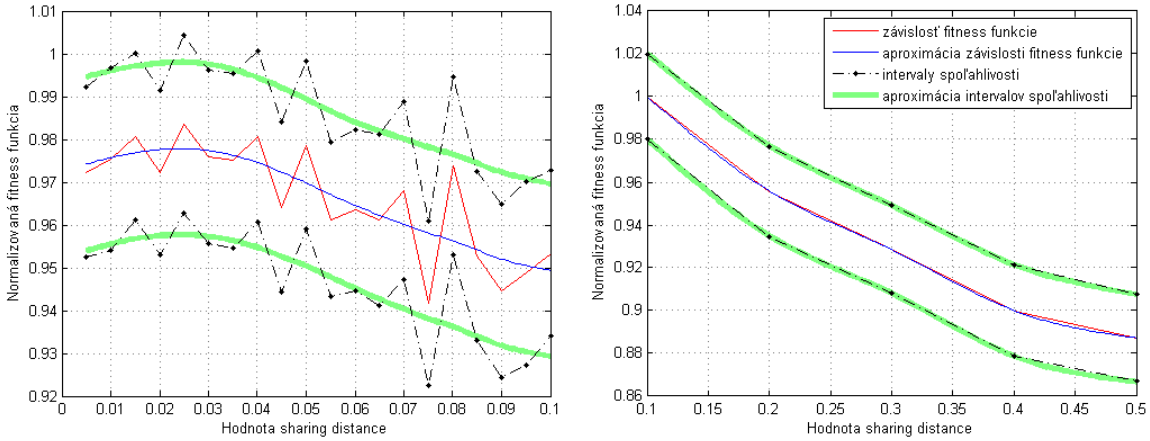
5.6 Experimenty s MOGA variantou GA1

MOGA variant GA1 bol implementovaný rovnako ako pôvodný GA1. Procesy ohodnotenia jedincov a selekcie sú však nahradené multikriteriálnym variantom algoritmu MOGA (FONSECA, a iní). Tieto prvky algoritmu MOGA sú popísané v kap.3.2.2. Z pohľadu praktického použitia je podstatné nastavenie parametra σ_{share} – sharing distance, ktorý ovplyvňuje hodnotu fitness funkcie v prípade jedincov, ktorí sú v multidimenzionálnom priestore ohodnotenia blízko pri sebe. Ak sú ohodnotenia jedincov bližšie ako σ_{share}, tak hodnota ich fitness funkcie bude proporcionálne znížená. Preto bolo potrebné vykonať experimenty s jeho nastavením. Algoritmus MOGA používa ešte jeden parameter – α, ktorý má ovplyvňovať tvar penalizačnej funkcie. Z praktických dôvodov však bol tento parameter ponechaný na hodnote $\alpha = 1$. V princípe mohlo dôjsť zavedením ďalšieho parametra do GA1 a zmenou selekčného mechanizmu a mechanizmu ohodnotenia k zmene vlastností pôvodného algoritmu GA1, tým pádom k potrebe prehodnotiť zistené najlepšie nastavenia parametrov GA1. Z praktických dôvodov však tieto parametre neboli menené a pre experimenty s algoritmom MOGA-GA1 bolo použité nastavenie podľa nasledujúcej tabuľky (Tab. 15). Bolo použité nastavenie neelitistického variantu GA1, keďže algoritmus MOGA je prirodzene neelitistický.

Tab. 15 – Konfigurácia MOGA-GA1 pre experimenty s nastavením parametru σ_{share}

Testovacie sady	asymmetric / simple, norm_small_1 / simple norm_small_2 / complex excenter_small / complex
BCX – Bernoulli Crossover	$p_{CX} = 0.45$
OPSO – One Point Swap Order mutation	$p_{MX}^{OPSO} = 0.055$
RA – Random Angle mutation	$p_{MX}^{RA} = 0.1$
n_{imig}	0
n_{eli}	0
σ_{share}	0.005-0.5
Počet opakovaní	100

Výsledky experimentov sú zobrazené na Obr. 29. Určenie priebehu tejto závislosti bolo problematické a tak museli byť experimenty vykonávané na všetkých štyroch testovacích sadách: *asymmetric / simple, norm_small_1 / simple, norm_small_2 / complex* a *excenter_small / complex* s počtom opakovaní 100x. Prvý pokus o nájdenie závislosti indikoval optimálnu hodnotu $\sigma_{share} = 0$ Obr. 29-vpravo. Táto hodnota je však pre algoritmus MOGA neplatná, keďže vyústi v delenie nulou. Následne boli prevedené spresňujúce experimenty bližšie k oblasti nuly (Obr. 29-vľavo). Zdá sa, že závislosť bola predsa len objavená a že parameter $\sigma_{share} < 0.01$ už prináša zhoršenie v dosahovenej hodnote fitness funkcie. Ako najlepšia hodnota predikovaná modelom je $\sigma_{share} = 0{,}02476$. Ako najlepšie nastavenie bude použitá hodnota $\sigma_{share} = 0{,}025$. V tabuľke Tab. 16 je uvedená finálna najlepšia zistená konfigurácia algoritmu MOGA-GA1.



Obr. 29 – Závislosť normalizovanej fitness funkcie od parametru σ_{share} – sharing distance algoritmu MOGA-GA1. Výsledky prvotných experimentov(vpravo) a spresnenie výsledkov (vľavo). Najlepšia hodnota určená aproximačným modelom: $\sigma_{share} = 0.02476$

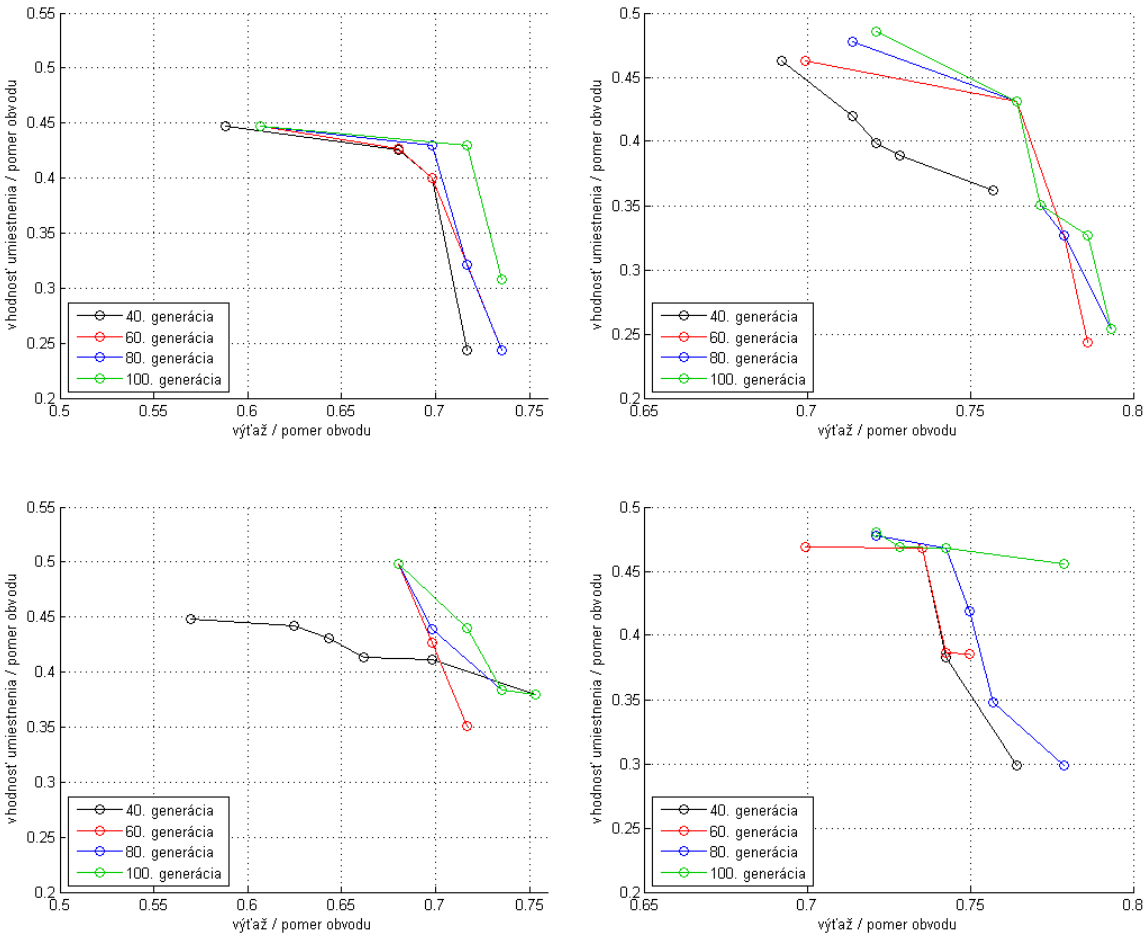
Tab. 16 – Najlepšia zistená konfigurácia algoritmu MOGA-GA1

BCX – Bernoulli Crossover	$p_{CX} = 0.45$
OPSO – One Point Swap Order mutation	$p_{MX}^{OPSO} = 0.055$
RA – Random Angle mutation	$p_{MX}^{RA} = 0.1$
n_{eli} / n_{imig}	0 / 0
σ_{share}	0.025

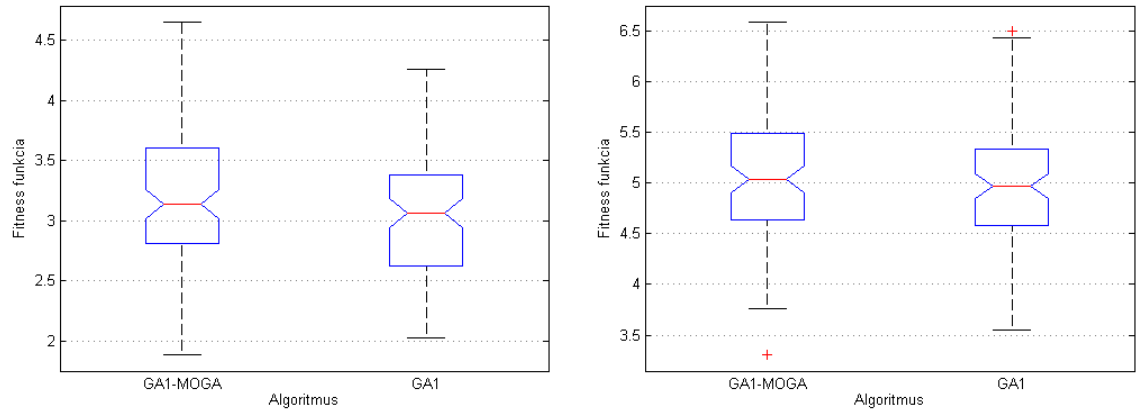
MOGA algoritmus je založený na hodnotení jedincov podľa Pareto dominancie, rovnako ako veľké množstvo iných multikriteriálnych algoritmov. Preto sa často sleduje rovnomerný vývoj pareto frontu, ktorý by mal pokrývať riešenia odpovedajúce rôznym kompromisom. Na obrázku Obr. 30 je vyobrazený vývoj pareto frontu u algoritmov GA1 (hore) a MOGA-GA1 (dole). Je nutné podotknúť, že tieto grafy boli vykreslené zo súboru 100x opakovaných experimentov. Pre každú vykreslenú generáciu (40,60,80,100) boli vyhodnotené všetky jedince zo všetkých behov a z nich bol určený Pareto-front. Problémom je totiž vykresliť Pareto-front u algoritmu GA1, ktorý vďaka nastavenému elitizmu udržuje jedného najdominantnejšieho jedinca.

Ako vidno, tak vývoj pareto frontu u MOGA-GA1 sa zdá byť horší oproti GA1. Výsledok je prekvapivý a je v rozpore s očakávaním, že algoritmus MOGA by mal udržiavať lepší Pareto-front vzhľadom na mechanizmy do neho zabudované ako obyčajný GA. U GA1 je použité váhovanie zložiek fitness funkcie, ktoré sa považuje tiež za vhodný mechanizmus multikriteriálnej optimalizácie a zrejme má schopnosť predurčiť MOGA-GA1 vďaka dobrému prehľadávaniu stavového priestoru – aj keď vo viacerých separátnych behoch.

Na Obr. 31 sú porovnané distribúcie hodnôt fitness funkcie na konci behu GA1 a MOGA-GA1. Z týchto grafov nie je možné vyčítať žiadnu signifikantnú informáciu, avšak vidno, že variant MOGA-GA1 napriek zjavne menšej schopnosti preskúmať Pareto-front dosahuje ešte o trochu lepšie výsledky ako samotný GA1.



Obr. 30 – Vývoj Pareto-frontu u algoritmu GA1 (hore) a MOGA-GA1 (dole) pre testovacie sady *asymmetric / simple* (vpravo) a *norm_small_1 (vľavo)*



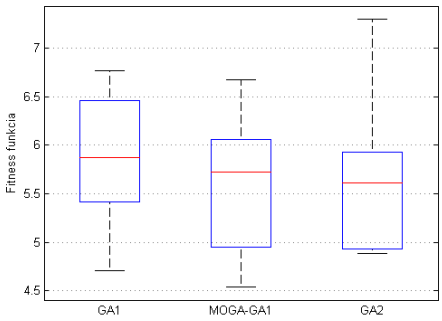
Obr. 31 – Krabicové grafy porovnávajúce hodnotu fitness funkcie na konci behu algoritmov GA1 a GA1-MOGA pre testovacie sady *asymmetric / simple* (vpravo) a *norm_small_1 (vľavo)*

6 Výsledky a zhodnotenie

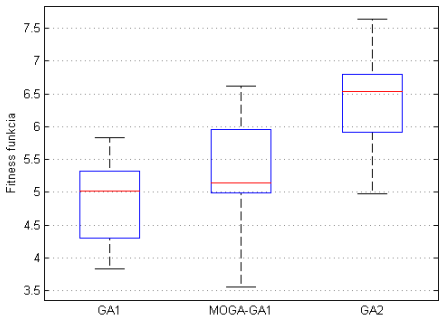
Pre účely výsledného porovnania boli všetky algoritmy spustené s plnohodnotnými parametrami podľa Tab. 3. Celkovo bolo vykonaných 20 experimentov pre všetky sady malých objektov v kombinácii s kontajnermi:

- asymetric
- excenter_medium
- norm_small_2
- egg_small

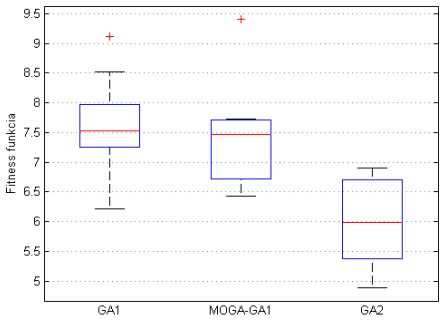
Výsledky sú zachytené na nasledujúcich obrázkoch (Obr. 32-Obr. 36). Ako vidno, tak žiaľ algoritmus MOGA_GA1 dopadol veľmi zle. Skoro pri každom teste sa ukázalo, že buď algoritmus GA1,alebo GA2 dosahuje lepšie výsledky. MOGA_GA1 dopadol najlepšie len v dvoch prípadoch u testovací sád norm_small_2 / simple a egg_small / simple. Zrejme je tak preto že polovica experimentov na určenie parametru ρ_{share} bola prevedená práve s testovacou sadou malých objektov *simple* a na relatívne malých kontajneroch. Jednoznačne sa dá však povedať že rozšírenie algoritmu GA1 o MOGA selekciu a ohodnotenie neprinieslo zlepšenie dosahovanej fitness funkcie riešení. A ako vidno na obrázku Obr. 30 v predchádzajúcej kapitole, tak nedá sa žiaľ ani tvrdiť, že by MOGA priniesol lepšie prehľadanie Pareto frontu.



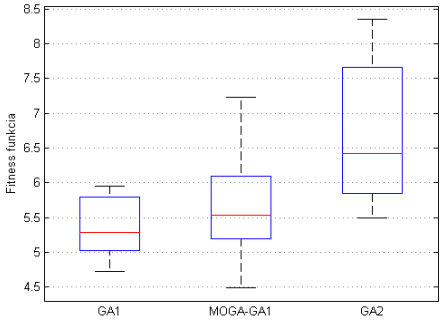
asymetric



egg_small

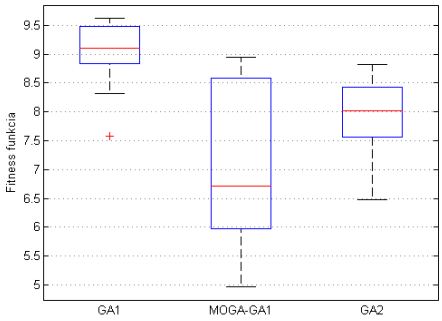


excenter_medium

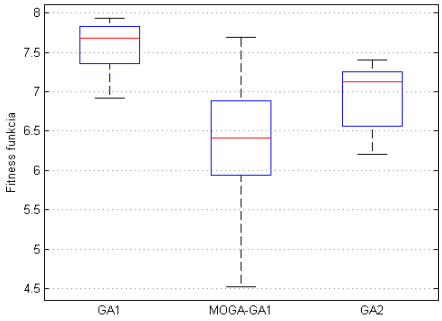


norm_small_2

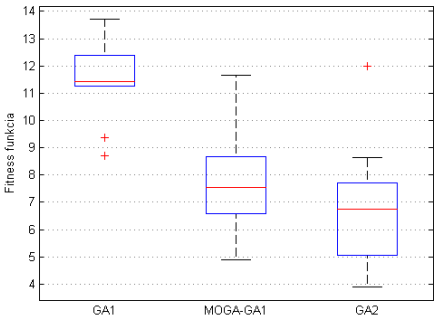
Obr. 32 – Krabicové grafy pre záverečné porovnanie GA pre sadu malých objektov *beam_and_board*



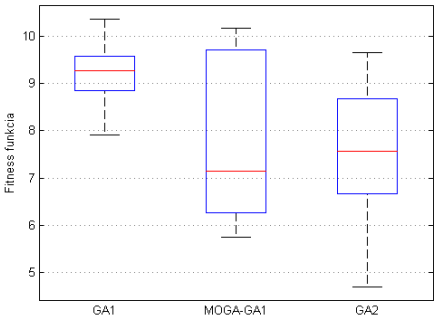
asymetric



egg_small

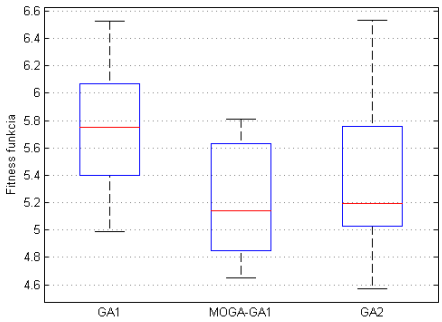


excenter_medium

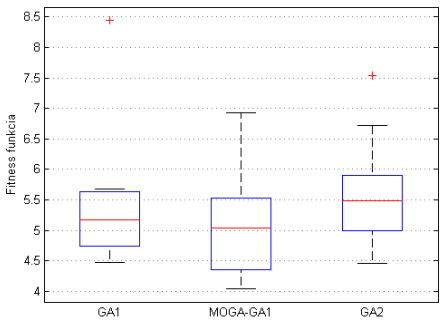


norm_small_2

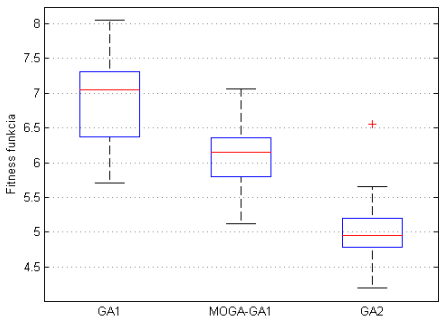
Obr. 33 – Krabicové grafy pre záverečné porovnanie GA pre sadu malých objektov *boards_only*



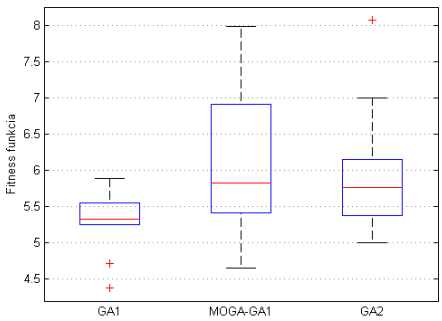
asymetric



egg_small

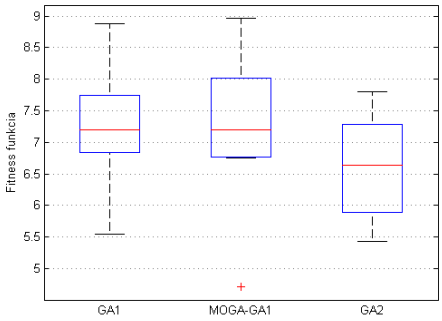


excenter_medium

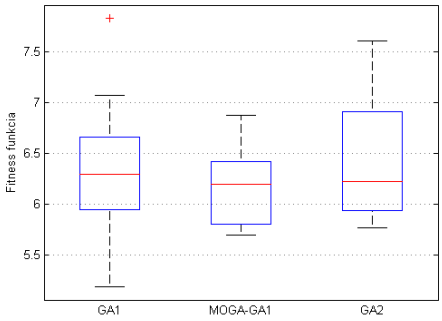


norm_small_2

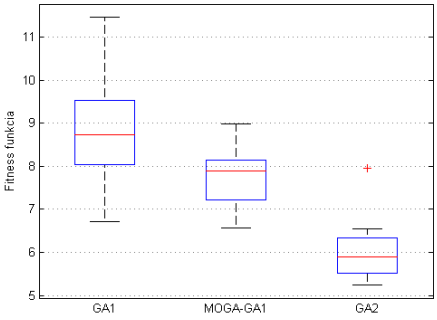
Obr. 34 – Krabicové grafy pre záverečné porovnanie GA pre sadu malých objektov *complex*



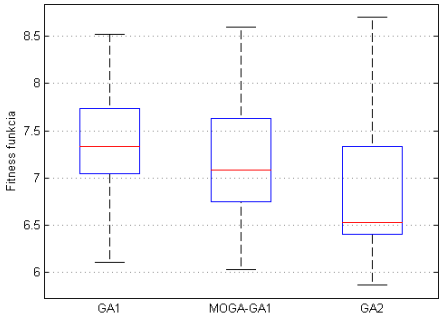
asymetric



egg_small

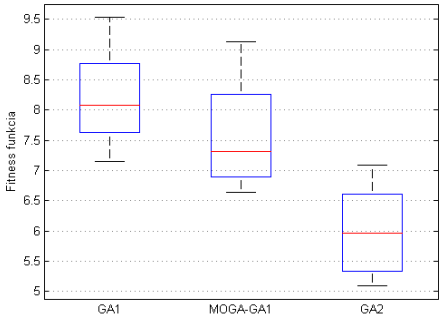


excenter_medium

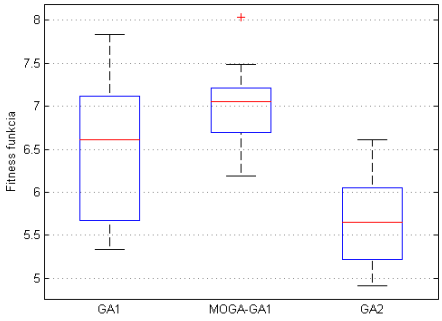


norm_small_2

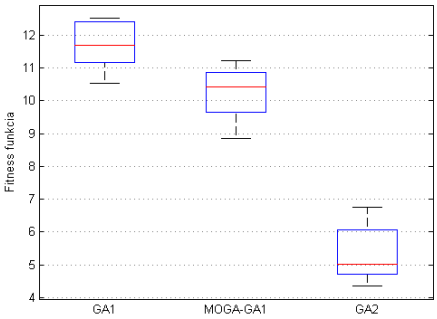
Obr. 35 – Krabicové grafy pre záverečné porovnanie GA pre sadu malých objektov *real*



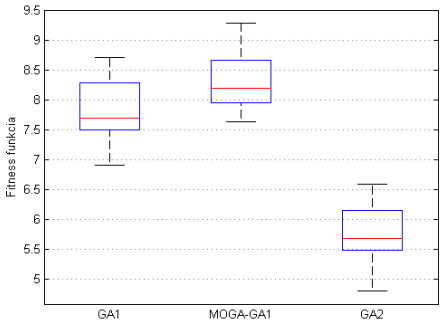
asymetric



egg_small



excenter_medium

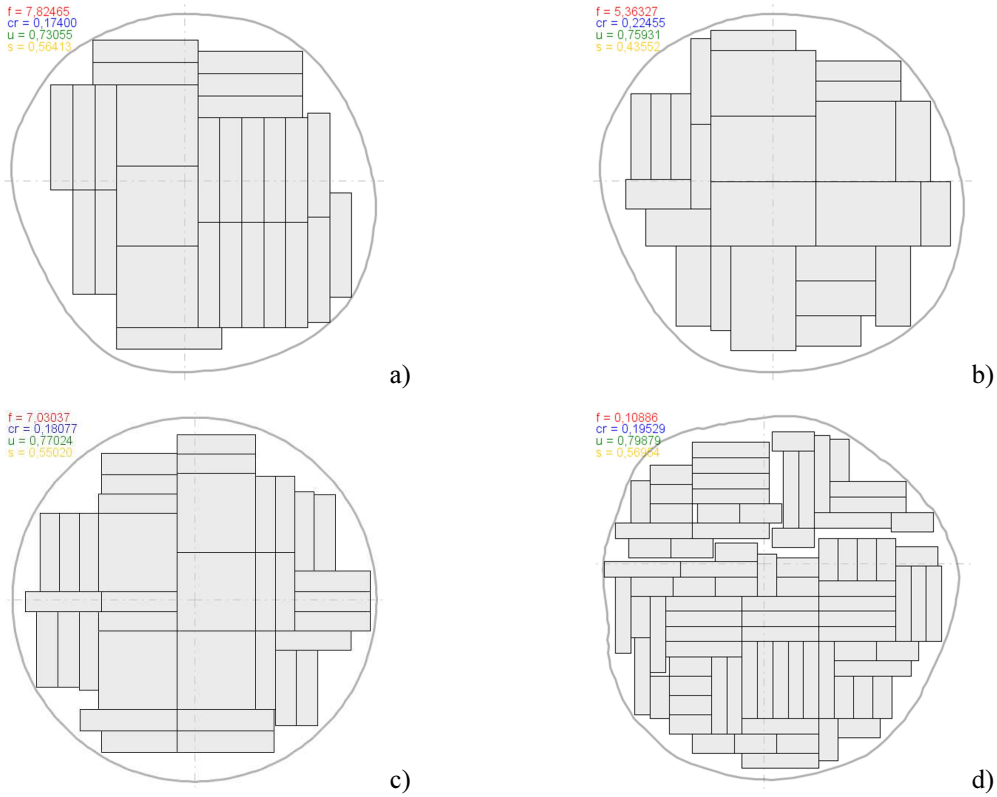


norm_small_2

Obr. 36 – Krabicové grafy pre záverečné porovnanie GA pre sadu malých objektov *simple*

Pri porovnaní algoritmov GA1 a GA2 je vidno, že niekedy algoritmu GA2 dosahuje lepšie výsledky ako GA1 a to hlavne u malých testovacích sád. Z uvedených grafov je možné vysledovať, že pri väčšom počte umiestňovaných objektov dominuje jednoznačne GA1.Vidno to hlavne na Obr. 33 vľavo, kde je počet ukladaných objektov najväčší. Pri znižovaní tohto počtu ako napr. na Obr. 34-vpravo a Obr. 35-vpravo dosahuje GA2 lepšie výsledky. Celkovo bol však GA1 úspešnejší v 16 prípadoch z 20 ako GA2. Preto je možné tvrdiť, že algoritmus GA1 sa ukazuje byť najefektívnejší.

Na záver zhodnotenia sú na Obr. 37 uvedené niektoré porezové plány vybrané počas evolúcie. Na všetkých uvedených plánoch je dosiahnutá naozaj dobré uloženie objektov, aj keď v niektorých prípadoch by sa zrejme dali vylepšiť – napr. vyplnením prázdneho miesta u plánu c) jednou doskou. Zároveň je možné si všimnúť veľmi sofistikovane využitú šírku a výšku kontajnera – hlavne v jeho pravej časti. U porezových plánov a), b) a c) si je možné všimnúť uloženie hranolov do stredu kmeňa tak ako je požadované. Porezový plán d) zobrazuje pretrvávajúci problém s nezaplnením medzery po čiastočnej konvergencii algoritmu. Pritom bol tento snímok vyhotovený približne v 250.generácii.



Obr. 37 – Ukážka niektorých vygenerovaných porezových plánov.

Záver

V tejto práci boli preskúmané možnosti riešenia problému porezu drevnej guľatiny (Log Cuting Problem) vo variante ngLCP – non-guilotinable Log Cutting Problem. Ako sa ukázalo pri rešerši literatúry, tak žiadna práca riešiaca tento problem nebola nájdená. Môže to byť prekvapujúce keďže problematika Cuting & Packing (C&P) problémov je v literatúre veľmi dobre preskúmaná. Jedná sa však väčšinou o problémy kde sú používané pravouhlé kontajnery, prípadne nekonečný pás materiálu. Prezentovaný problém ale rieši umiestnenie malých pravouhlých kongruentných objektov do kontajnera reprezentovaného konvexným polygónom. Použité metódy ani zakódovania problémov u klasických C&P probl0mov tak nesú vhodné. Zlomovým bodom bolo objavenie publikácie o riešení CSP problému (Cutting Stock Problem) v kožiarenskom priemysle (ANAND, a iní, 1999) (SHARMA, a iní). Na základe získanej inšpirácie boli navrhnuté dva genetické algoritmy využívajúce úplne rozdielne konštrukčné heuristiky – GA1 a GA2. Postupne bola vyvinutá fitness funkcia ktorá dokázala doviesť algoritmy k veľmi dobrým výsledkom. Následne bola navrhnutá fitness funkcia ktorá má za úlohu ohodnotiť vhodnosť umiestnenia malých objektov. Použitím tejto fitness funkcie bolo dosiahnuté vhodnejšie ukladanie malých objektov s cieľom zabezpečiť lepšiu kvalitu reziva. Táto dodatočná fitness funkcia vnáša ďalšie hodnotiace kritérium a tak bola do fitness funkcie GA1 a GA2 zakomponovaná váhovaním. Ďalej bola implementovaná varianta algoritmu GA1 - MOGA-GA1 (FONSECA, a iní) využívajúca princíp pareto dominancie namiesto váhovania za účelom multikriteriálnej optimalizácie.

V práci boli vytvorené syntetické testovacie sady, ako aj testovacie sady založené na reálnych profiloch kontajnerov (kmeňov). Pomocou týchto testovacích sád bolo prevedené množstvo experimentov s cieľom zistiť optimálne nastavenie navrhnutých GA. Na konci práce boli navrhnuté GA navzájom porovnané. Z tohto porovnania vyplýva, že najlepšie dopadol GA1 (u 16 z 20 experimentov) a MOGA-GA1 naopak priniesol veľké sklamanie. V budúcnosti by možno bolo dobré pristúpiť sofistikovanejších a efektívnejších metód k multikriteriálnej optimalizácii - napr. algoritmu NSGAII, alebo PAES (TAN, a iní, 2005).

Pre reálne nasadenie systému optimalizácie ngLCP nie je však dané riešenie vhodné. Táto práca síce dokázala, že pomocou GA1 by bolo možné optimalizovať porez guľatiny, avšak prakticky iba u kmeňov z priemerom $\emptyset < 400mm$, čo nemusí postačovať. Už aj pri tomto priemere trvá optimalizácia približne 3-4 minúty. Pre reálne nasadenie systému by bolo nutné algoritmus GA1 implementovať efektívnejšie a na vhodnejšom hardvéri – napr. GPU.

Najväčším prínosom práce však je vytvorenie funkčných algoritmov schopných reálne riešiť problém porezu guľatiny.

Literatúra

ALVAREZ-VALDÉZ, Ramón, PARAJÓN, Antonio a TAMARIT, Jose'e Manuel. 2001. A Computational Study of Heuristic Algorithms for Two-Dimensional Cutting Stock Problems. *4th Metaheuristics International Conference.* 2001.

—. 2002. A tabu search algorithm for large-scale guillotine (un)constrained two-dimensional cutting problems. *Computers & Operations Research 29.* 2002, s. 925-947.

ALVAREZ-VALDÉZ, Ramón, PAREŇO, F. a TAMARIT, Jose'e Manuel. 2004. A GRASP algoritmh for constrained two-dimensional non-guillotine cutting problems. *Journal of Operational Research Society.* 2004.

ANAND, Sam, McCORD, Christopher a SHARMA, Rohit. 1999. An Integrated Machine Vision Based System for Solving the Nonconvex Cutting Stock Problem Using Genetic Algorithms. *Journal of Manufacturing System Vol. 18/No. 6.* 1999, s. 396-415.

BURKE, E. K., KENDALL, G. a WHITWELL, G. 2004. A new placement heuristic for the orthogonal stock-cutting problem. *Operations Research, 52.* 2004, s. 655-671.

COELO, Carlos A. a LAMONT, Gary. 2004. *Applications Of Multi-Objective Evolutionary Algorithms (Advances in Natural Computation).* Singapore : World Scientific Publishing Co. Pte. Ltd., 2004. ISBN 981-256-106-4.

CONCALVES, Jose Fernando. 2007. A hybrid genetic algorithm-heuristic for a two-dimensional orthogonal packing problem. *European Journal of Operational Research 183.* Porto : s.n., 2007, s. 1212-1229.

DYCKHOFF, H. 1990. A typology of cutting and packing problems. *European Journal of Operational Research 44.* 1990, s. 144-159.

FONSECA, Carlos M. a FLEMING, Peter J. Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization. Sheffield : University of Sheffield.

GILMORE, P.C. a GOMORY, R.E. 1963. A linear programming approach to the cutting stock problem. *Operations Research Vol. 11, No. 6.* 1963, s. 863-888.

—. 1965. Multi-stage cutting stock problems of two or more dimensions. *Operations Research 13.* 1965, s. 94-120.

HAESSLER, R.W. a SWEENEY, P.E. 1991. Cutting stock problems and solution procedures. *European Journal of Operational Research 54.* 1991, s. 141-150.

HOPPER, E. a TURTON, B. C. H. 2000. An empirical investigation of meta-heuristic and heuristic algorithms for a 2D packing problem. *European Journal of Operational Research 128.* Cardiff : s.n., 2000, s. 34-57.

ILLICH, Simon, WHILE, Lyndon a BARONE, Luigi. 2007. Multi-objective Strip Packing Using an Evolutionary Algorithm. *IEEE Congress on Evolutionary Computation (CEC 2007).* 2007, s. 4201-4214.

KARELAHTI, Janne. 2002. *Solving the cutting stock problem in the steel industry.* Espoo : s.n., 2002. Diploma Thesis.

LEVINE, J. a DUCATELLE, F. 2003. Ant colony optimisation and local search for bin packing and cutting stock problems. *Journal of the Operational Research Society.* 2003.

MACH, Marián. 2009. *Evolučné algoritmy - prvky a princípy.* Košice : Elfa, 2009. 978-80-8086-123-0.

MANCAPA, V. *General Genetic Algorithm for One and Two Dimensional Cutting and Packing Problems.* s.l. : Nelson Mandela Metropolitan University.

MICHALEWICZ, Zbigniew. 1996. *Genetic Algorithms + Data Structures = Evolution Programs.* Charlotte : Springer-Verlag, 1996. ISBN 3-540-60676-9.

NORMAN, B. A. a BEAN, J. C. 1996. *Random Keys Genetic Algorithm for Job Shop Scheduling.* s.l. : University of Michigan, 1996.

PARMEE, Ian C. 2001. *Evolutionary and Adaptive Computing in Engineering Design.* Exeter : Springer-Verlag, 2001. ISBN 1-85233-029-5.

SHARMA, Rohit, a iní. *Genetic Algorithm for the Non-convex Cutting Stock Problem.* Cincinnati : University of Cincinnati.

ŠEDIVÁ, Blanka. 2007. Intervalové odhady. [Online] 2007. [Dátum: 5. 5 2013.] http://home.zcu.cz/~sediva/stav/kap06.pdf.

TAN, K.C., KHOR, E.F. a LEE, T.H. 2005. *Multiobjective Evolutionary Algorithms and Applications.* Singapore : Springer-Verlag, 2005. ISBN 1-85233-836-9.

TAUFER, Ivan, KOTYK, Josef a JAVŮREK, Milan. 2009. *Jak psát a obhajovat závěrečnou práci bakalířskou, diplomovou, rigorózní, disertační, habilitační.* Pardubice : Univerzita Pardubice, 2009. ISBN 978-80-7395-157-3.

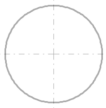
WALKER, J.C.F. 2006. *Primary Wood Processing – Principles and Practice 2nd edition.* s.l. : Springer, 2006. ISBN-10 1-4020-4393-7.

WÄSCHER, Gerhard, HAUßNER, Heike a SCHUMANN, Holger. 2007. An improved typology of cutting and packing problems. *European Journal of Operational Research 183.* Magdeburg : Elsevier, 2007, s. 1109-1130.

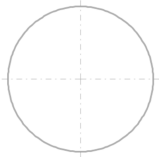
ZELINKA, Ivan, a iní. 2009. *Evoluční výpočetní techniky - princípy a aplikace.* Praha : BEN, 2009. ISBN 978-80-7300-218-3.

ZITZLER, Eckart, THIELE, Lothar a DEB, Kalyanmoy. 2000. Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. *Evolutionary Computation 8(2).* s.l. : Massachusetts Institute of Technology, 2000, s. 173-195.

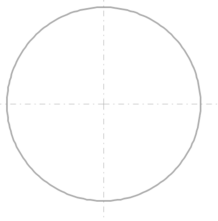
Príloha A – kompletný testovací súbor profilov kontajnerov



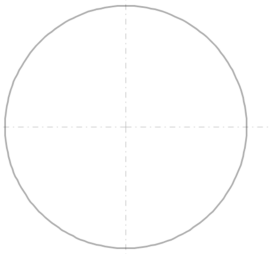
250mm



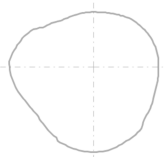
375mm



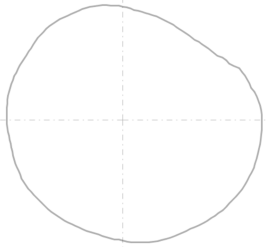
500mm



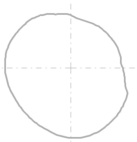
625mm



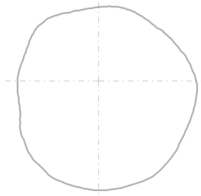
asymetric



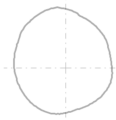
egg_large



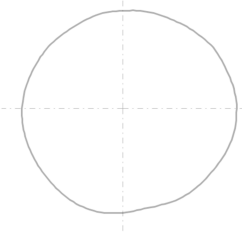
egg_small



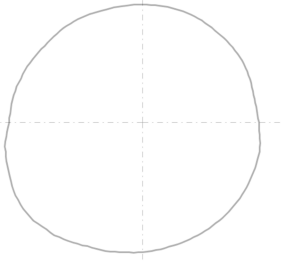
excenter_medium



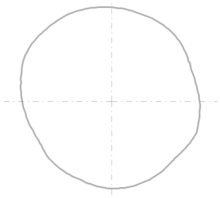
excenter_small



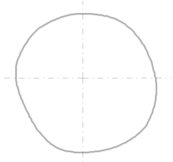
norm_large_1



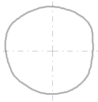
norm_large_2



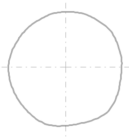
norm_medium_1



norm_medium_2



norm_small_1



norm_small_2