Univerzita Komenského v Bratislave

Fakulta matematiky, fyziky a informatiky

Riešenie problému RSA – teória čísel vs. globálna optimalizácia

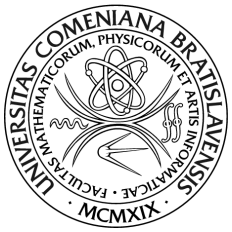
Diplomová práca

Bc. Kristína Ciesarová

25. apríla 2014

Univerzita Komenského v Bratislave

Fakulta matematiky, fyziky a informatiky



Riešenie problému RSA – teória čísel vs. globálna optimalizácia

Diplomová práca

Študijný program:	Ekonomická a finančná matematika
Študijný odbor:	1114 Aplikovaná matematika
Školiace pracovisko:	Katedra aplikovanej matematiky a štatistiky
Vedúci práce:	RNDr. Mária Trnovská, PhD.

Bc. Kristína Ciesarová

25. apríla 2014

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Kristína Ciesarová
Študijný program: ekonomická a finančná matematika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: 9.1.9. aplikovaná matematika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: slovenský

Názov: Riešenie problému RSA - teória čísel vs. globálna optimalizácia / *Solution to RSA problem - Number theory versus Global Optimization*

Cieľ: S kryptografiou súvisí niekoľko problémov diskrétnej optimalizácie, ako napríklad problém celočíselnej faktorizácie súvisiaci s RSA kryptosystémami. Algoritmy, ktoré dokážu riešiť tieto problémy v 'rozumnom' čase môžu byť užitočným nástrojom kryptoanalýzy. Cieľom práce je naštudovať známe prístupy a pokúsiť sa navrhnúť alternatívnu metódu na riešenie takýchto problémov.

Vedúci: RNDr. Mária Trnovská, PhD.
Katedra: FMFI.KAMŠ - Katedra aplikovanej matematiky a štatistiky
Vedúci katedry: prof. RNDr. Daniel Ševčovič, CSc.

Dátum zadania: 25.01.2013

Dátum schválenia: 04.02.2013 prof. RNDr. Daniel Ševčovič, CSc.
garant študijného programu

.....
študent

.....
vedúci práce

Poďakovanie

Na tomto mieste by som sa chcela poďakovať mojej školiteľke, RNDr. Márii Trnovskej, PhD. za všetky jej cenné rady a pripomienky, ochotu a ústretovosť počas konzultácií aj mimo nich. Tiež ďakujem mojej rodine a budúcemu manželovi za to, že mali so mnou trpezlivosť počas písania tejto práce a za ich neustálu podporu.

Abstrakt

CIESAROVÁ, Kristína: Riešenie problému RSA – teória čísel vs. globálna optimalizácia [Diplomová práca], Univerzita Komenského v Bratislave, Fakulta matematiky, fyziky a informatiky, Katedra aplikovanej matematiky a štatistiky; školiteľ: RNDr. Mária Trnovská, PhD., Bratislava, 2014, 56s.

V práci sa venujeme problému RSA kryptosystému – faktorizácii čísel na prvočísla. Na tento problém sa pozeráme z perspektívy teórie čísel, ktorá ako riešenie ponúka známy algoritmus kvadratického sita. Ďalej úlohu modifikujeme na úlohu diskrétnej globálnej optimalizácie a riešime ju pomocou evolučných algoritmov. Konkrétne sme použili algoritmy Particle Swarm Optimization (PSO) a Self-Organizing Migrating Algorithm (SOMA), ktoré sme porovnávali s náhodným prehľadávaním. Nakoniec úlohu pomocou poznatkov z teórie čísel transformujeme do jedného rozmeru a optimalizujeme pomocou upraveného evolučného algoritmu SOMA. Tento prístup taktiež porovnávame s náhodným prehľadávaním. Výsledkom našej práce je zhrnutie použitých metód a ich porovnanie. V nami navrhnutej jednorozmernej modifikácii úlohy sa nám podarilo úspešnosťou algoritmu SOMA poraziť náhodné prehľadávanie.

Kľúčové slová: problém RSA, faktorizácia, kvadratické sito, diskrétna optimalizácia, PSO, SOMA

Abstract

CIESAROVÁ, Kristína: Solution to RSA problem – Number theory versus Global Optimization [Master Thesis], Comenius University in Bratislava, Faculty of Mathematics, Physics and Informatics, Department of Applied Mathematics and Statistics; Supervisor: RNDr. Mária Trnovská, PhD., Bratislava, 2014, 56p.

In the thesis we analyse the problem of RSA cryptosystem – the factorization of numbers. We inspect this problem through the number theory perspective, which provide us with a well-known quadratic sieve algorithm. We further modify the problem to meet the standards of discrete global optimization and then we solve the problem using evolutionary algorithms. We used namely the Particle Swarm Optimization (PSO) and the Self-Organizing Migrating Algorithm (SOMA) and compared them to the random search. Finally, using findings of number theory, we transform the problem to only one dimension and optimize it by modified SOMA algorithm. Likewise, we compare this approach to the random search. The result of our thesis is a summarization of methods used and their comparison. The success rate of SOMA algorithm we suggested on one-dimensional modification of the problem managed to outperform random search.

Keywords: RSA problem, factorization, quadratic sieve, discrete optimization, PSO, SOMA

Obsah

Úvod **7**

Tabuľka pojmov a označení **9**

1 Kryptografia **10**

1.1 Historický náhľad 10

1.2 Základné pojmy kryptografie 10

2 RSA **13**

3 Motivácia **17**

3.1 Prístup teórie čísel 18

3.2 Faktorizácia ako úloha diskrétnej optimalizácie 19

4 Kvadratické sito **21**

5 Globálna optimalizácia **25**

5.1 Formulácia úlohy 25

5.2 Evolučné algoritmy 27

5.2.1 Particle swarm optimization – PSO 28

5.2.2 Self-Organizing Migrating Algorithm – SOMA 30

6 1D prístup **34**

7 Praktická časť **37**

7.1 Kvadratické sito 37

7.2 Náhodné prehľadávanie (Random search) 38

7.3 PSO 39

7.4 SOMA 42

7.5 1D prístup 45
7.6 Zhrnutie 48

Záver **51**

Literatúra **53**

Dodatok **55**

Programy v prostredí Python 55
Tabuľky s výsledkami 56

Úvod

Najrozšírenejšie a najúčinnejšie metódy moderného šifrovania sú systémy s tzv. verejným kľúčom. Ich bezpečnosť sa zakladá na tom, že nie je možné správu dešifrovať, ak človek pozná iba šifrovaciu transformáciu, ale nepozná istú informáciu navyše. Jedným z takýchto systémov je aj kryptosystém RSA, ktorý je momentálne považovaný za veľmi ťažko prelomiteľný. V RSA problém nájdenia dešifrovacej funkcie nazývame RSA problém. Túto úlohu však vieme abstrahovať na matematický problém rozkladania čísel na prvočísla. Násobenie čísel je totižto z hľadiska výpočtovej náročnosti veľmi jednoduchá operácia, avšak opačný proces – faktorizácia čísel – je úloha pre veľké čísla mimoriadne výpočtovo náročná. Preto ak by sa našiel spôsob efektívneho rozkladania čísel, tak by RSA prestalo byť bezpečnou šifrou.

V tejto práci sa budeme venovať rôznym prístupom k riešeniu RSA problému. Predstavíme prístup teórie čísel, ktorý je známy už od roku 1981 (pozri [11]). Na tento problém sa ale dá pozerať aj z inej perspektívy – ako na úlohu globálnej diskrétnej optimalizácie. Táto oblasť je zatiaľ pomerne málo preskúmaná, my sme sa s týmto prístupom oboznámili v článku [2] z roku 2005. V tomto článku autori na problém RSA aplikujú evolučné algoritmy, ktorých popularita narastá od začiatku 21. storočia. Keďže táto oblasť je ešte pomerne mladá a predstavované sú stále nové evolučné algoritmy, ponúka široké možnosti na skúmanie v tomto smere. Preto sme sa aj rozhodli túto tému študovať a pokúsiť sa ju nejakým spôsobom vylepšiť. Z množstva rôznych algoritmov a prístupov sme si vybrali niekoľko, ktoré sme aplikovali na problém RSA a výsledky sme porovnali. Cieľom našej práce je pokúsiť sa navrhnúť alternatívny spôsob riešenia problému RSA, ktorý by potenciálne mohol byť užitočným nástrojom kryptoanalýzy.

Naša práca je rozdelená na dve časti. V prvej časti, ktorá je viac teoretická, na začiatok predstavujeme základné pojmy v kryptografii a stručný pohľad do jej histórie. Ďalej sa venujeme RSA kryptosystému a definícii RSA problému aj jeho ekvivalentu v globálnej

optimalizácii. Predstavujeme jednotlivé známe prístupy na riešenie tohto problému a tiež pridávame vlastný, ktorý podrobne odvádzame.

V druhej časti práce sme sa venovali praktickému aplikovaniu predstavených metód na daný problém. Jednotlivé algoritmy sme programovali v prostredí Python a aplikovali sme ich na náš problém. Výsledky sme potom porovnávali a skúmali.

Práca je určená nielen pre tých, ktorí sa zaujímajú o aktuálne problémy kryptografie, konkrétne kryptosystému RSA. Oblasť, ktorej sa v diplomovej práci venujeme je zatiaľ málo preskúmaná, preto veríme, že zaujímavými výsledkami sa nám podarí zvýšiť záujem vedeckej komunity o danú tému.

Tabuľka pojmov a označení

$\varphi(n)$	Eulerova funkcia
$N = pq$	prirodzené číslo, ktoré je súčinom dvoch rôznych nepárnych prvočísel
$\phi = \varphi(N) = (p-1)(q-1)$	hodnota Eulerovej funkcie v bode N
$f(P) = P^e \pmod N$	šifrovacia transformácia v RSA kryptosystéme
$f^{-1}(C) = C^d \pmod N$	dešifrovacia transformácia v RSA kryptosystéme
$NSD(a, b)$	najväčší spoločný deliteľ čísel a a b
$a \equiv b \pmod n$	a je kongruentné s b modulo n, teda $n\|(a-b)$
$\mathbb{Z}_n$	zvyšková trieda $\mathbb{Z} \pmod n$
$\mathbb{Z}_n^* = \{a \in \mathbb{Z}_n;\ NSD(a, n) = 1\}$	multiplikatívna grupa zvyškovej triedy $\mathbb{Z}_n$
Q_n	množina kvadratických zvyškov modulo n
$\overline{Q}_n$	množina kvadratických nezvyškov modulo n

Tabuľka 1: Tabuľka pojmov a označení

1 Kryptografia

1.1 Historický náhľad

História kryptografie má svoje začiatky už takmer pred 4000 rokmi v starovekom Egypte. Najstaršie známe šifrované správy pochádzajú z roku okolo 1900 p.n.l. Od tohto času sa kryptografia výrazne a prudko rozvíjala. Veľkým krokom vpred pre oblasť kryptografie bolo obdobie svetových vojen, kedy dôležitosť účinného šifrovania narastala. Avšak najväčšiu zmenu v kryptografii priniesol práve rozmach počítačov v 60-tych rokoch 20. storočia [6]. Rozširovaním komerčného využitia počítačov sa definitívne zmenil princíp fungovania kryptografie. Jedným z problémov vtedajších kryptografov bola otázka štandardizácie - potreby jednotného šifrovacieho systému pre utajenú komunikáciu aj medzi rôznymi inštitúciami. Moderné šifrovanie sa preto zakladá na používaní verejne známych algoritmov s utajenými kľúčmi, ktoré nám umožňujú správy šifrovať a dešifrovať.

1.2 Základné pojmy kryptografie

Existuje mnoho rôznych definícií samotného pojmu kryptografia. Jednoducho by sme mohli povedať, že kryptografia je šifrovanie. V skutočnosti je ale kryptografia širší pojem. Definícia z knihy [1] znie:

> *Kryptografia je štúdium matematických techník súvisiacich s rôznymi aspektami informačnej bezpečnosti, ako je dôvernosť informácií, dátová integrita, autentifikácia totožnosti a autentifikácia pôvodu dát.*

Znamená to teda, že kryptografia slúži nielen na šifrovanie a dešifrovanie správ, ale aj na overovanie totožnosti odosielateľa, prípadne autentifikáciu toho, či s dátami počas ich prenosu nemanipulovala tretia strana. V našej práci sa budeme sústrediť ale iba na konkrétnu metódu posielania správ v šifrovanej podobe, ktorú môže odtajniť a prečítať len určený príjemca. Na začiatok je však potrebné objasniť niektoré základné pojmy

súvisiace s kryptografiou. V tejto časti budeme čerpať najmä z našej bakalárskej práce [7], keďže táto diplomová práca na ňu priamo nadväzuje.

Správa, ktorú posielame sa nazýva **čistý (plain) text** a utajená správa sa nazýva **šifrovaný (cipher) text**. Množinu všetkých čistých správ budeme označovať $\mathcal{P}$ a množinu všetkých šifrovaných správ označíme $\mathcal{C}$. Čistý aj šifrovaný text sú napísané v abecede, ktorá môže, ale nemusí byť pre oba texty rovnaká. Proces zmeny čistého textu na šifrovaný sa nazýva šifrovanie a opačný proces dešifrovanie.

Šifrovanie prebieha tak, že nezašifrovaná správa sa rozdelí na viacero častí, na ktoré sa aplikuje tzv. **šifrovacia transformácia**, ktorá mení čistý text na šifrovaný. Šifrovacia transformácia je funkcia $f : \mathcal{P} \to \mathcal{C}$. Budeme predpokladať, že táto funkcia je bijekcia, teda prosté zobrazenie na množinu $\mathcal{C}$. Takto ku každej zašifrovanej správe existuje práve jedna čistá správa. **Dešifrovacia transformácia** je teda inverzná funkcia f^{-1}, ktorá funguje spätne, teda zo šifrovaného textu dostane opäť čistý text. Schematicky môžeme toto zapísať diagramom

$$\mathcal{P} \xrightarrow{f} \mathcal{C} \xrightarrow{f^{-1}} \mathcal{P}.$$

Každý takýto systém šifrovacej a dešifrovacej transformácie nazývame **kryptosystém**.

Vo všeobecnosti predpokladáme, že spôsob šifrovania je verejne známy, teda je známa trieda použitých šifrovacích transformácií. Čo držíme v tajnosti sú konkrétne parametre šifrovacej transformácie. Tieto parametre nazývame **šifrovací kľúč**, ozn. K_E. Podobne pri dešifrovacej transformácii máme tajný **dešifrovací kľúč**, ozn. K_D.

V niektorých kryptosystémoch nie je potrebné bližšie špecifikovať dešifrovací kľúč, keď už poznáme šifrovací kľúč, pretože sa dá ľahko zistiť. To znamená, že ak je šifrovací kľúč prezradený tretej strane, tak tretia strana dokáže v dostatočne krátkom čase zistiť (vypočítať) aj dešifrovací kľúč. Šifra už teda prestáva byť bezpečná. Takéto kryptosystémy nazývame kryptosystémy **so symetrickým kľúčom**.

Existujú však aj tzv. kryptosystémy **s verejným kľúčom**, ktoré sú aj stále viac používané. Konkrétnemu kryptosystému s verejným kľúčom sa budeme venovať aj v tejto diplomovej práci. Princípom je, že ten, kto pozná šifrovací kľúč nevie bez časovo alebo inak náročného výpočtu zistiť dešifrovací kľúč. To znamená, že K_E môže byť verejný a nikto, okrem adresáta poznajúceho K_D, nemôže správu jednoducho dešifrovať. Medzi takéto kryptosystémy patrí aj kryptosystém RSA.

2 RSA

RSA je kryptosystém, ktorý je vôbec prvým realizovateľným kryptosystémom s verejným kľúčom. Aj v dnešnej dobe je široko používaný na prenos šifrovaných dát. Ako sme už spomínali, v takomto kryptosystéme sa dešifrovací kľúč, na rozdiel od šifrovacieho kľúča, drží v tajnosti. V RSA je táto asymetria založená na praktickej výpočtovej náročnosti rozkladania čísla, ktoré je súčinom dvoch veľkých prvočísel. Názov RSA vznikol ako skratka mien Rona Rivesta, Adiho Shamira a Leonarda Adlemana, ktorí ako prví verejne predstavili tento algoritmus na prelome rokov 1977 a 1978 (pozri [14], [15]). Používateľ RSA vytvorí a zverejní súčin dvoch veľkých prvočísel (ich súčin je rádovo 10^{617}), spolu s pomocným číslom, o ktorom si viac povieme neskôr. Táto dvojica čísel je používateľovým verejným kľúčom. Dve prvočísla sa musia držať v tajnosti. Ktokoľvek môže použiť verejný kľúč na to, aby zašifroval správu, ale iba ten, kto pozná prvočíselný rozklad verejne známeho čísla je schopný správu dešifrovať v prijateľnom čase. Problém rozlúštenia RSA šifry je známe ako RSA problém.

Základnou myšlienkou šifrovania pomocou RSA je teda výpočtová náročnosť rozkladania veľkých prirodzených čísel na prvočísla. Predpokladáme, že nejaké veľké číslo N má práve dva delitele väčšie ako 1, p a q; teda $N = p \cdot q$. Čísla p a q sú spravidla rádovo blízke prvočísla. Symbolom ϕ označíme hodnotu Eulerovej funkcie v bode N. Definícia Eulerovej funkcie je nasledovná:

Definícia 2.1 (Eulerova funkcia)**.** *Pre $n \geq 1, \varphi(n)$ označuje počet celých čísel v intervale $\langle 1, n\rangle$, ktoré sú nesúdeliteľné s n.*

Viac o Eulerovej funkcii a jej vlastnostiach sa dá dočítať aj v [1]. Jednou z vlastností tejto funkcie je, že ak poznáme rozklad čísla na prvočísla, $n = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$, tak $\varphi(n) = n(1-\frac{1}{p_1})(1-\frac{1}{p_2})\cdots(1-\frac{1}{p_k})$. Teda v našom prípade platí, že $\phi = \varphi(N) = N(1-\frac{1}{q})(1-\frac{1}{p}) = (p-1)(q-1)$.

V RSA kryptosystéme sa používa šifrovacia transformácia

$$f(P) = P^e \pmod{N}, \quad e \in \mathbb{N}. \tag{1}$$

K nej inverzná funkcia je

$$f^{-1}(C) = C^d \pmod{N}, \quad d \in \mathbb{N}, \tag{2}$$

kde d je inverzný prvok vzhľadom na násobenie čísla e modulo ϕ. To znamená, že e a d sú navzájom vo vzťahu

$$e \cdot d = 1 \pmod{\phi}. \tag{3}$$

Korektné definovanie inverznej transformácie (2) vyplýva zo známej Eulerovej vety, ktorú uvádzame bez dôkazu. Tento dôkaz je možné nájsť napríklad v knihe [17].

Veta 2.1 (Eulerova veta)**.** *Nech $n \geq 2$ je prirodzené číslo. Potom ak $NSD(a, n) = 1$, tak*

$$a^{\varphi(n)} \equiv 1 \pmod{n},$$

kde $\varphi(n)$ je Eulerova funkcia.

Chceme ukázať, že inverzná šifrovacia transformácia f^{-1} je korektne definovaná, teda že $f^{-1}(f(P)) \equiv P \pmod{N}$. Tento vzťah môžno zapísať ako

$$P^{ed} \equiv P \pmod{N},$$

kde $ed \equiv 1 \pmod{\phi}$, teda $ed = k\phi + 1$, pričom $\phi = \varphi(N)$ je hodnota Eulerovej funkcie v bode N. Podľa Eulerovej vety platí

$$P^{\phi} \equiv 1 \pmod{N}.$$

Potom platí aj

$$P^{k\phi} \equiv 1 \pmod{N}.$$

Prenásobením kongruencie číslom P dostávame

$$P^{k\phi+1} \equiv P \pmod{N},$$

z čoho je zrejmé, že

$$P^{ed} \equiv P \pmod{N}.$$

Z tohto dôkazu môžme vidieť, že vďaka špeciálnej voľbe čísla d, je možné zašifrovanú správu dešifrovať späť na pôvodný text. Ak by však nebolo d volené ako inverzný prvok vzhľadom na násobenie čísla e modulo ϕ, tak by dešifrovanie nebolo korektne definované.

Jediný prvok, ktorý v dešifrovacej transformácii (2) nepoznáme je práve číslo d. Problém rovnice (3) však vieme pomerne jednoducho vyriešiť pomocou tzv. rozšíreného Euklidovho algoritmu, ale iba vtedy, ak poznáme ϕ, resp. ak poznáme rozklad N na prvočísla p a q. Viac o Euklidovom algoritme aj o jeho aplikácii na tento konkrétny problém možno nájsť v našej bakalárskej práci [7].

Teda úlohu, ktorá robí z RSA efektívny šifrovací algoritmus môžme definovať nasledovne:

Definícia 2.2 (Problém RSA)**.** *Dané je prirodzené číslo N, ktoré je súčinom dvoch rôznych nepárnych prvočísel p a q. Ďalej nech pre $e \in \mathbb{N}$ platí $NSD(e, \phi) = 1$ a nech $C \in \mathbb{Z}$. Nájdite $P \in \mathbb{Z}$ také, že $P^e \equiv C \pmod{N}$.*

Nižšie popíšeme algoritmus, pomocou ktorého môžme vytvárať súkromné a verejné kľúče v RSA. Vychádzať budeme z knihy [1].

Algoritmus 1: Vytváranie kľúčov v RSA

1. vygenerujeme dve náhodné veľké rôzne prvočísla p a q
2. vypočítame $N = pq$ a $\phi = (p-1)(q-1)$
3. náhodne vyberieme celé číslo e, $1 < e < \phi$ také, že $NSD(e, \phi) = 1$
4. použitím rozšíreného Euklidovho algoritmu vypočítame číslo $d, 1 < d < \phi$ také, že $ed \equiv 1 \pmod{\phi}$
5. verejný kľúč je dvojica (N, e); súkromný kľúč je d

Ďalej uvádzame algoritmus samotného RSA kryptosystému, teda proces šifrovania a dešifrovania pomocou RSA. Kvôli zrozumiteľnosti algoritmu použijeme namiesto označenia A a B známe kryptografické aliasy Alica a Bob.

Algoritmus 2: Šifrovanie a dešifrovanie pomocou RSA

1. *Šifrovanie*: Bob musí spraviť nasledovné
2. zistí Alicin verejný kľúč, (N, e)
3. správu reprezentuje ako celé číslo P z intervalu $[0, N-1]$
4. vypočíta $C = P^e \pmod{N}$
5. šifrovaný text pošle Alici
6. *Dešifrovanie*: Alica vykoná
7. použije súkromný kľúč d na vypočítanie $P = C^d \pmod{N}$

Zopakujme teraz, že verejným kľúčom v RSA je dvojica čísel N a e, ktoré môžu byť zverejnené v akomsi "telefónnom zozname". Súkromný kľúč je číslo d, ktoré nedostaneme bez informácie o rozklade N na prvočísla p, q. Bezpečnosť RSA kryptosystému teda spočíva v tom, že tretia strana nevie bez p a q zistiť dešifrovací kľúč, resp. je to výpočtovo príliš náročné. Ak by sa však dalo efektívne rozkladať veľké čísla na prvočísla, RSA kryptosystém by prestal byť bezpečným. V nasledujúcich kapitolách sa budeme venovať práve rôznym spôsobom rozkladania čísel na prvočísla.

3 Motivácia

V predchádzajúcej kapitole sme spomenuli, že úlohou, ktorou sa budeme zaoberať, je rozkladanie čísla N na dve rôzne prvočísla. Aby sme však mohli uvažovať o spôsoboch riešenia tejto úlohy, potrebujeme ju sformulovať matematicky. Je zrejmé, že táto úloha je ekvivalentná s hľadaním netriviálneho deliteľa N. Základná myšlienka metód, z ktorej vychádzame, pochádza z knihy [1].

Nech $x, y \in \mathbb{Z}_N^*$ také, že

$$x^2 \equiv y^2 \pmod{N},$$

ale zároveň

$$x \not\equiv \pm y \pmod{N}. \tag{4}$$

Potom N delí

$$x^2 - y^2 = (x + y)(x - y), \tag{5}$$

pričom nedelí $(x+y)$ ani $(x-y)$. Keďže $N = p \cdot q$, tak $NSD(x-y, N) = p$ alebo q, teda sme našli netriviálny deliteľ čísla N.

Poďme sa teraz bližšie pozrieť, ako táto úloha funguje. Keďže N delí súčin (5), tak platí

$$(x - y)(x + y) = kN,$$

pre nejaké $k \in \mathbb{N}$. Keďže predpokladáme, že N má práve dva delitele p a q, tak môžu nastať dva prípady:

1. p aj q sú deliteľmi čísla $x - y$ alebo čísla $x + y$,

2. p je deliteľom čísla $x - y$ a q je deliteľom čísla $x + y$, alebo naopak.

V prvom prípade nezískame žiadnu užitočnú informáciu, pretože to znamená, že $NSD(x-y, N) = 1$ alebo N. Avšak v druhom prípade $NSD(x - y, N) = p$ alebo q, teda nájdeme

netriviálneho deliteľa čísla N. Preto musíme v definícii úlohy zahrnúť podmienky (4), keďže tie vylúčia neužitočnú prvú možnosť.

Zhrňme teda, formuláciu úlohy faktorizácie čísla $N \in \mathbb{N}$ ako:

nájdite $x, y \in \mathbb{Z}_N^*$, ktoré spĺňajú

$$x^2 \equiv y^2 \pmod{N}; \quad x \not\equiv \pm y \pmod{N}. \tag{U1}$$

3.1 Prístup teórie čísel

Z teórie čísel sú známe viacerá algoritmy na riešenie tejto úlohy. Najznámejšími algoritmami sú **faktorizácia eliptickými krivkami** (elliptic curve factoring), **kvadratické sito** (quadratic sieve factoring) a **číselné sito** (number sieve factoring). Popis všetkých týchto algoritmov je mimo rozsahu tejto práce, podrobnejšie sú popísané napríklad v knihe [1]. V skratke len spomenieme, že vo faktorizácii eliptickými krivkami sa namiesto množiny $\mathbb{Z}_n^*$ pracuje s náhodnými eliptickými krivkami nad zvyškovou triedou $\mathbb{Z}_n$. Kvadratické sito na rozkladanie čísel využíva kvadratické zvyšky (pozri Kapitola 4), pričom hľadá čísla hladké čísla (pozri Definícia 4.1) rádovo $\sqrt{N}$. Napokon číselné sito je vylepšením kvadratického sita, pretože dokáže hľadať hladké čísla vzhľadom na menšiu medzu hladkosti (viď Definícia 4.1).

Zaujímavé je pre nás porovnanie rýchlosti jednotlivých algoritmov. V knihe [1] sa píše, že všeobecne je kvadratické sito efektívnejšie ako faktorizácia eliptickými krivkami, iba v špeciálnom prípade, keď rozkladáme číslo, ktoré je súčinom dvoch prvočísel rovnakej dĺžky, je ich rýchlosť asymptoticky rovnaká. Preto sme sa faktorizáciou eliptickými krivkami v práci nezaoberali. Číselné sito je však momentálne najrýchlejším známym prostriedkom na faktorizáciu čísel. Menezes a kol. v knihe [1] uvádzajú, že efektívnosť číselného sita sa prejavuje až pri rádovo 110 – 120-ciferných číslach. Pre účely nášho skúmania sme teda zvolili prístup kvadratického sita.

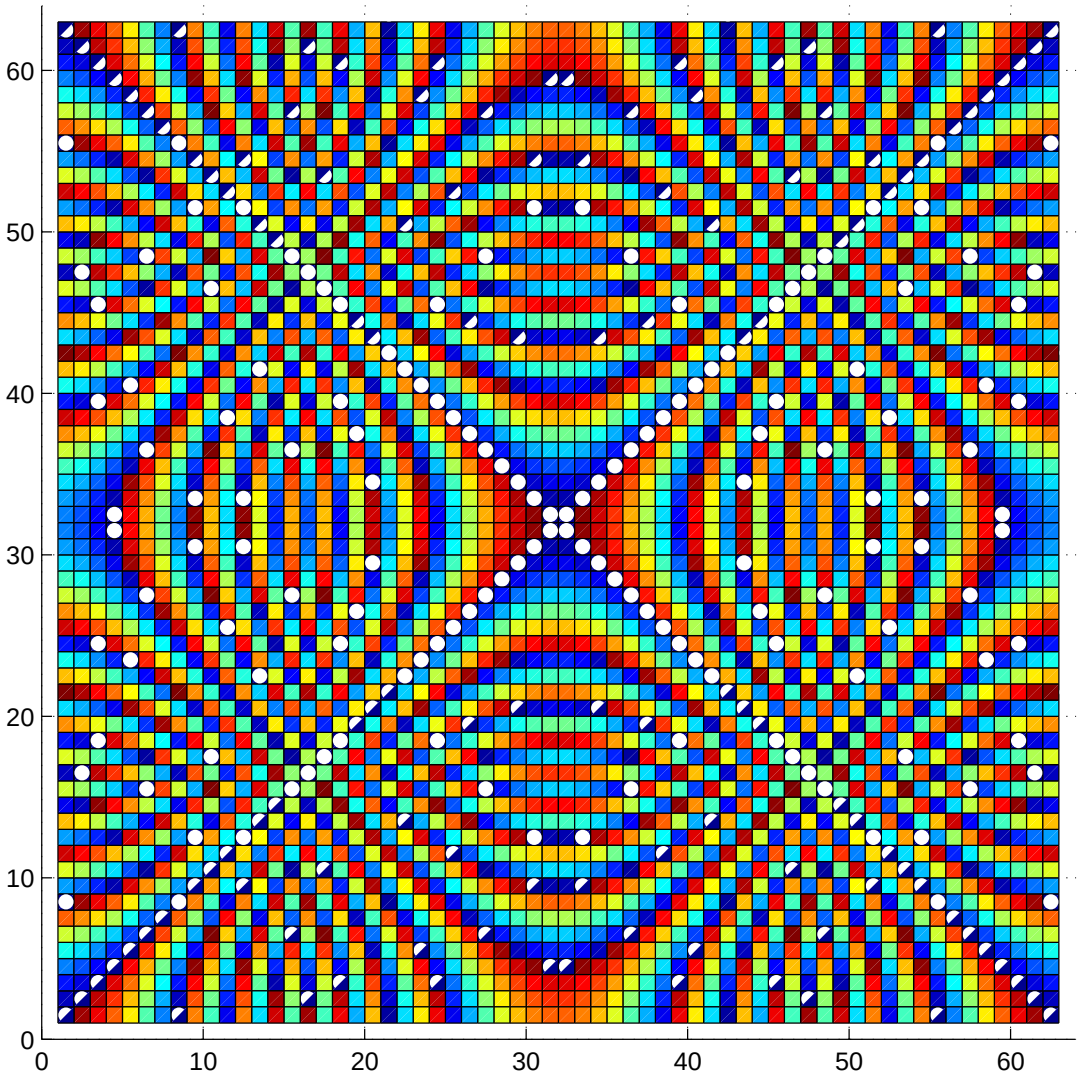
3.2 Faktorizácia ako úloha diskrétnej optimalizácie

V článku [2] z roku 2005 sa páni Laskari, Meletiou a Vrahatis na úlohu (U1) pozreli ako na úlohu diskrétnej optimalizácie. Na riešenie optimalizačného problému navrhli použiť tzv. evolučné algoritmy, ktoré potom na danú úlohu aj aplikovali. Tento článok bol motiváciou aj na našu prácu. V spomínanom článku autori skúmali efektívnosť algoritmov Particle Swarm Optimization (skrátene PSO) a Differential Evolution vzhľadom na problém faktorizácie čísel, ktoré porovnávali s náhodným prehľadávaním. Keďže ako najvhodnejší (s výnimkou náhodného prehľadávania) sa ukázal práve algoritmus PSO, rozhodli sme sa ho použiť.

Existuje však obrovské množstvo rôznych evolučných algoritmov. Za zmienku stoja napríklad simulované žíhanie (Simulated Annealing – SANN), genetické algoritmy (Genetic Algorithms – GA), či pomerne nový (2005) algoritmus svätojánskych mušiek (Firefly Algorithm – FA). V publikácii [8] bol však predstavený algoritmus Self-Organizing Migrating Algorithm (skrátene SOMA). V práci [4] bol tento algoritmus porovnaný s viacerými evolučnými algoritmami (vrátane SANN), a výsledky, ktoré dosahoval na testovacích funkciách často porážal aj práve simulované žíhanie. Preto sme sa rozhodli ako druhý algoritmus použiť práve algoritmus SOMA.

Na Obr. 1 môžno vidieť funkčné hodnoty funkcie, ktorú budeme v našej úlohe optimalizovať. Modrou farbou sú vyznačené hodnoty blízke 0, červenou zasa hodnoty blízke $N-1$. Biele krúžky navyše označujú minimá funkcie. V článku [2] autori konštatovali, že minimalizácia tejto funkcie má dynamiku, ktorá evolučným prehľadávacím algoritmom len málo vyhovuje. My sme sa rozhodli ich postupy vylepšiť a vyskúšať aj iné evolučné algoritmy. Zároveň sme chceli testovať aj klasický prístup teórie čísel.

Okrem toho sme skúsili tieto dva prístupy spojiť, keď sme využili poznatky teórie čísel v spojení s diskrétnou optimalizáciou. Viac o tomto prístupe možno nájsť v kapitole 6.



Obr. 1: Úrovňové hodnoty účelovej funkcie v úlohe diskrétnej optimalizácie

4 Kvadratické sito

Kvadratické sito je jedným z rozšírených algoritmov na faktorizáciu čísel. Úlohu (U1) rieši algoritmus pomocou metód lineárnej algebry. V našej práci sme pri popise tohto algoritmu vychádzali zo zdrojov [11] a [1].

Predtým ako predstavíme samotný algoritmus kvadratického sita, zadefinujeme pojem hladkosti, ktorý sa v ňom využíva.

Definícia 4.1 (Hladkosť)**.** *Nech B je prirodzené číslo. Číslo $n \in \mathbb{Z}$ sa nazýva B-**hladké**, ak všetky jeho prvočíselné delitele sú menšie alebo rovné ako B. Číslo B sa nazýva **medza hladkosti**.*

Pracovať budeme nad špeciálnou množinou čísel, ktorá sa nazýva multiplikatívna grupa.

Definícia 4.2 (Multiplikatívna grupa)**.** ***Multiplikatívnou grupou zvyškovej triedy*** *$\mathbb{Z}_n$ nazývame množinu*

$$\mathbb{Z}_n^* = \{a \in \mathbb{Z}_n;\ NSD(a, n) = 1\}.$$

Kvadratické sito faktorizuje čísla pomocou tzv. kvadratických zvyškov. Uvádzame preto definíciu tohto pojmu.

Definícia 4.3 (Kvadratický zvyšok)**.** *Nech $a \in \mathbb{Z}_n^*$. Číslo a sa nazýva **kvadratický zvyšok modulo** n, ak existuje $x \in \mathbb{Z}_n^*$ také, že*

$$x^2 \equiv a \pmod{n}. \tag{6}$$

*Ak také x neexistuje, tak a sa nazýva **kvadratický nezvyšok modulo** n. Množina všetkých kvadratických zvyškov modulo n sa označuje Q_n, množina všetkých kvadratických nezvyškov sa označuje $\overline{Q}_n$.*
Pre $a \in Q_n$ sa x, ktoré spĺňa (6)*, nazýva **druhá odmocnina** a **modulo** n.*

Poznámka: Táto odmocnina nemusí byť pre dané a a n jediná.

Algoritmus kvadratického sita prebieha v nasledovných krokoch:

1. Vyberieme faktorovú bázu $S = \{p_1, \ldots, p_t\}$, teda množinu prvých t prvočísel. Viac o voľbe čísla t uvedieme neskôr.

2. Potrebujeme nájsť $t + 1$ dvojíc (x_i, y_i), ktoré spĺňajú

$$y_i \equiv x_i^2 \pmod{N}.$$

   Uvažujme čísla y_i v tvare

$$y_i = \prod_{j=1}^{t} p_j^{e_{ij}}; e_{ij} \in \mathbb{N} \cup \{0\}.$$

   Takto definované čísla y_i sú zjavne p_t-hladké. Zároveň chceme z takýchto čísel y_i súčinom vyskladať štvorec $\pmod{N}$. Na to stačí, aby mocnina každého prvočísla v ich súčine bola párna. Preto budeme uvažovať iba paritu e_{ij}. Pre $\forall i = 1, 2, \ldots, t+1$ vytvoríme binárny vektor $v_i = (v_{i1}, \ldots, v_{it})$ asociovaný s vektorom exponentov $(e_{i1}, \ldots, e_{it})$ nasledovným spôsobom:

$$v_{ij} \equiv e_{ij} \pmod{2}.$$

3. Získame $t + 1$ dvojíc (x_i, y_i) a k nim prislúchajúce t-rozmerné vektory v_i nad $\mathbb{Z}_2$. Tieto vektory musia byť lineárne závislé (nad $\mathbb{Z}_2$), teda existuje neprázdna indexová množina $T \subseteq \{1, 2, \ldots, t + 1\}$ taká, že

$$\sum_{i \in T} v_i = 0 \pmod{2}.$$

Indexovú množinu T možno nájsť riešením systému nad $\mathbb{Z}_2$:

$$Vz = 0,$$

resp. nájdením nulového priestoru $Null(V)$, kde

$$V = \begin{pmatrix} | & | & & | \\ v_1 & v_2 & \cdots & v_{t+1} \\ | & | & & | \end{pmatrix}.$$

Potom $T = \{i | Vz = 0; z_i = 1\}$.

4. Keďže platí $\sum_{i \in T} v_i = 0$, tak $\sum_{i \in T} e_{ij}$ je párne $\forall j$ a teda číslo $\prod_{i \in T} y_i$ je štvorec (mod N). Zrejme číslo $\prod_{i \in T} x_i^2$ je tiež štvorec (mod N). Čísla

$$x := \prod_{i \in T} x_i \text{ a } y := \sqrt{\prod_{i \in T} y_i}$$

spĺňajú požadovanú vlastnosť

$$y^2 \equiv x^2 \pmod{N}.$$

5. Vypočítame $NSD(x - y, N)$. Ak $NSD(x - y, N) = 1$, tak opakujeme proces s inou kombináciou T. Ak $NSD(x - y, N) \neq 1$, tak sme našli netriviálny deliteľ a označíme ho p.

Algoritmus zapísaný pomocou pseudokódu uvádzame na nasledujúcej strane.

Rýchlosť kvadratického sita spočíva vo voľbe y_i ako $q(z) = (z + m)^2 - N$. Keďže $m = \lfloor \sqrt{N} \rfloor$, tak

$$y_i = q(z) = z^2 + 2mz + m^2 - N \approx z^2 + 2mz,$$

Algoritmus 3: Kvadratické sito

Input: zložené číslo N
Output: netriviálny deliteľ čísla N

1 vyberieme faktorovú bázu $S = \{p_1, \ldots, p_t\}$
2 vypočítame $m = \lfloor \sqrt{N} \rfloor$
3 *(nájdeme $t+1$ párov (x_i, y_i), hodnoty z volíme postupne v poradí $1, 2, \ldots$)*
4 $i \leftarrow 1$ **while** $i \leq t+1$ **do**
5 | vypočítame $y_i = q(z) = (z+m)^2 - N$
6 | **if** y_i *nie je* p_t*-hladké* **then**
7 | | opakujeme krok 5 s ďalším z
8 | **end**
9 | $x_i \leftarrow (z+m)$, $y_i \leftarrow y_i$, $v_i = (v_{i1}, \ldots, v_{it})$, kde $v_{ij} = e_{ij} \pmod 2, 1 < j < t$
10 | $i \leftarrow i+1$
11 **end**
12 nájdeme neprázdnu indexovú množinu $T \subseteq \{1, 2, \ldots, t+1\}$, takú že $\sum_{i \in T} v_i = 0 \pmod 2$
13 vypočítame $x = \prod_{i \in T} x_i \pmod N$
14 $\forall j, 1 \leq j \leq t$, vypočítame $l_j = (\sum_{i \in T} e_{ij})/2$
15 vypočítame $y = \prod_{j=1}^{t} p_j^{l_j} \pmod N$
16 ak $x \equiv \pm y \pmod N$, tak nájdeme inú neprázdnu množinu T, že $\sum_{i \in T} v_i = 0 \pmod 2$, a vrátime sa do kroku 13
17 vypočítame $p = NSD(x - y, N)$ a vrátime p

čo je v porovnaní s N malé číslo, ak z je v absolútnej hodnote malé. Keďže algoritmus sa zakladá na nachádzaní y_i, ktoré sú p_t-hladké, práve vďaka tejto špeciálnej voľbe je veľká šanca, že faktorizácia y_i bude mať všetky prvočísla menšie ako p_t. Tým sa algoritmus zrýchľuje, pretože podmienka v kroku 6 v Algoritme 3 často nie je splnená.

Napokon si predstavíme optimálnu voľbu čísla t tak, aby algoritmus dosahoval čo najkratší výpočtový čas. Ak je totiž t príliš malé, tak algoritmus len ťažko nájde p_t-hladké čísla. Ak však t zvolíme príliš veľké, tak p_t-hladkých čísel bude veľmi veľa a kvadratické sito bude ťažko hľadať lineárnu závislosť (viď. [18]). Heuristickou analýzou v práci [19] sa ukázalo, že nasledujúci vzorec dáva optimálnu hodnotu t

$$L(N) = \left\lceil \left(e^{\frac{1}{2} \ln(N) \ln(\ln(N))} \right)^{1/2} \right\rceil .$$

5 Globálna optimalizácia

V tejto kapitole predstavíme čitateľovi alternatívny prístup k problému rozkladania čísla na prvočísla. Problém, ktorý sme v predošlej kapitole riešili pomocou kvadratického sita, predstavíme ako úlohu globálnej diskrétnej optimalizácie. Úlohu (U1) najprv prevedieme na minimalizačnú úlohu, ktorú budeme na určitom obore celých čísel riešiť pomocou tzv. evolučných algoritmov, ktorým sa v ďalších kapitolách budeme venovať. Ako sme už spomínali v kapitole 2, problém faktorizácie čísel tvorí jadro RSA kryptosystému. Preto optimalizačné metódy, ktoré by efektívne túto úlohu riešili v prijateľnom výpočtovom čase, by boli pokladané za užitočný prostriedok kryptoanalýzy.

Všeobecná úloha globálnej optimalizácie je nasledovná [4]. Pre danú účelovú funkciu $f : D \to \mathbb{R}, D \subset \mathbb{R}^d$ hľadáme bod x^* taký, že

$$x^* = \arg\min_{x \in D} f(x).$$

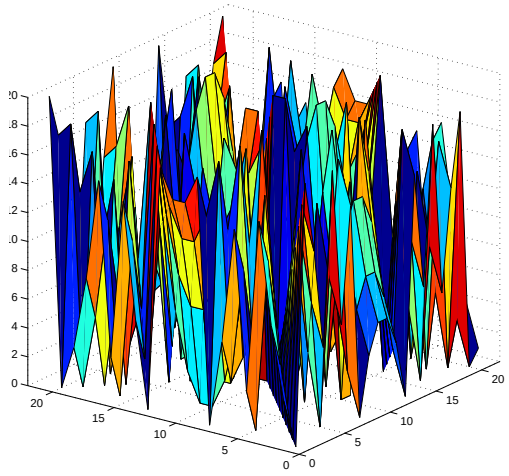
Bod x^* sa potom nazýva globálne minimum funkcie f. Potrebujeme teda nájsť vhodnú účelovú funkciu a zadefinovať množinu D.

5.1 Formulácia úlohy

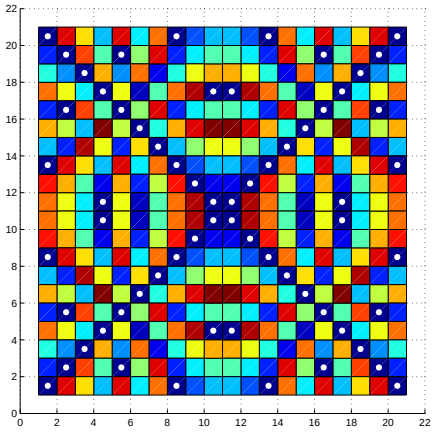
Pri formulácii minimalizačnej úlohy sme vychádzali z článku [2]. Úloha (U1) definovaná v kapitole 2 pomocou kongruencií sa dá formulovať ako úloha diskrétnej optimalizácie definovaním účelovej funkcie $f : \{1, \ldots, N-1\} \times \{1, \ldots, N-1\} \to \{0, \ldots, N-1\}$ s predpisom

$$f(x, y) = (x^2 - y^2) \pmod{N},$$

za podmienok $x \not\equiv \pm y \pmod{N}$. Vieme, že globálne minimum tejto funkcie je 0, ktoré nastáva práve vtedy, ak nájdeme (triviálneho alebo netriviálneho) deliteľa čísla N. Na Obr. 2 je vykreslená funkcia f pre $N = 21$ a na Obr. 3 sa nachádza pohľad na funkciu



Obr. 2: Graf funkcie f



Obr. 3: Úrovňové hodnoty funkcie f

zhora, pričom rôznymi farbami sú zvýraznené jej úrovne, s modrou približujúcou sa k 0. Bielymi krúžkami sú označené body, v ktorých $f = 0$. Táto funkcia je symetrická podľa osí $x = (N-1)/2$ a $y = (N-1)/2$, čo môžeme vidieť aj na Obr. 3. Stačí nám teda ako množinu prípustných riešení brať iba časť z celkového priestoru, čím sa aj zbavíme podmienky $x \not\equiv -y \pmod N$. Zmeníme teda definičný obor funkcie f, čim sa úloha zredukuje na minimalizáciu funkcie $g : \{1, 2, \ldots, (N-1)/2\} \times \{1, 2, \ldots, (N-1)/2\} \to \{0, \ldots, N-1\}$ s predpisom

$$g(x, y) = (x^2 - y^2) \pmod N, \text{ za podmienky } x \not\equiv y \pmod N. \tag{U2}$$

Z grafu funkcie na Obr. 2 môžeme vidieť, že klasické spádové metódy zrejme nebudú veľmi účinné, pretože s veľkou pravdepodobnosťou sa zaseknú v lokálnom minime. Preto na riešenie budeme používať rôzne heuristické evolučné algoritmy, ktoré na prehľadávanie priestoru nevyužívajú spád, ale napríklad sociálne správanie rojov, kolektívnu inteligenciu a iné poznatky z biológie, či z fyziky.

5.2 Evolučné algoritmy

Evolučné algoritmy sú široká trieda heuristických prehľadávacích algoritmov založených na vývoji populácii. Populácia sa vyvíja buď pohybom v prehľadávacom priestore alebo krížením jedincov medzi sebou. Jedinci sa potom testujú pomocou účelovej funkcie a do novej populácie sa vyberie nová sada jedincov podľa hodnoty ich účelovej funkcie. Jednotlivé evolučné algoritmy sa od seba výrazne líšia, či už spôsobom pohybu po priestore, alebo krížením jedincov medzi sebou. Často sa v týchto algoritmoch vyskytuje nejakým spôsobom náhodnosť – napríklad pohyb častíc (jedincov) môže byť z istej časti náhodný, prípadne do kríženia vstupuje mutácia.

Evolučné algoritmy majú väčšinou svoj pôvod v javoch vypozorovaných z prírody, prípadne sa inšpirujú poznatkami z biológie, genetiky, fyziky či sociálneho správania zvierat. Hlavnou výhodou evolučných algoritmov je to, že na hľadanie optimálneho riešenia zväčša nepotrebujú o účelovej funkcii poznať nič iné okrem jej predpisu. Sú charakteristické svojou robustnosťou voči zaseknutiu v lokálnom optime – sú teda multimodálnym a multikriteriálnym nástrojom na hľadanie globálneho optima funkcie [4]. Vhodné sú preto na optimalizáciu nediferencovateľných, nespojitých aj vyslovene diskrétnych funkcií, či funkcií s "divokým"priebehom. Viac o evolučných algoritmoch sa dá dočítať napríklad v publikáciách [12] a [13].

Naopak ich nevýhody sú, že počas behu algoritmu sa nedá určiť vzdialenosť aktuálneho riešenia od globálneho optima, ani s akou pravdepodobnosťou toto optimum môžme dosiahnuť. Taktiež zbehnutie takéhoto algoritmu môže trvať veľmi dlho, pričom aj tak nezaručujú nájdenie optimálneho riešenia a sú častokrát veľmi závislé na voľbe parametrov. Záujem o evolučné algoritmy však v poslednej dobre veľmi narastá, medzi najznámejšie patria zrejme genetický algoritmus, simulované žíhanie alebo Particle Swarm Optimization. V našej práci sa budeme venovať algoritmu Particle Swarm Optimization, pretože sa ukázal ako najlepší z testovaných evolučných algoritmov v článku [2] a predstavíme aj pomerne nový algoritmus SOMA, ktorým sa pokúsime prekonať výsledky dosiahnuté

algoritmom PSO.

5.2.1 Particle swarm optimization – PSO

Particle swarm optimization (skrátene PSO) je algoritmus, ktorý ako princíp využíva správanie roja (swarm). PSO ako prví publikovali Eberhart a Kennedy v roku 1995. Neskôr bolo PSO aplikované a upravené na mnohých úlohách, okrem iného aj v článku [2] na riešenie RSA problému.

Tento algoritmus je, podobne ako genetické algoritmy, založený na vývoji populácií. Populácia ale svojich jedincov (v PSO sa nazývajú častice) nemení, len sa hýbe po množine prípustných riešení X. Jednotlivé častice populácie sa pohybujú určitou rýchlosťou a navzájom sa ovplyvňujú. Každá častica má svoje okolie $O_i \subset X$. Štruktúra okolí sa nazýva topológia a môže byť reprezentovaná pomocou grafu. Najčastejšie topológie sú topológia na kompletnom grafe (všetky častice sú susedné) a kruhová topológia.

Časticu $i \in \{1, \ldots, Npart\}$ v čase $t = 1, 2, \ldots$ (resp. počas jednej iterácie) charakterizuje:

- poloha $x_i^{(t)}$,
- dosiaľ najlepšia poloha $p_i^{(t)}$ častice i (najlepšou polohou sa myslí taká poloha, kde je hodnota účelovej funkcie najnižšia),
- dosiaľ najlepšia poloha susedov $l_i^{(t)}$,
- rýchlosť $v_i^{(t)}$.

Rýchlosť častice v nasledujúcom kroku sa vypočíta pomocou vzorca:

$$v_i^{(t+1)} = w^{(t)} v_i^{(t)} + \phi_1 u_1 \left(p_i^{(t)} - x_i^{(t)} \right) + \phi_2 u_2 \left(l_i^{(t)} - x_i^{(t)} \right),$$

kde prvý sčítanec je tzv. zotrvačnosť (inertia), druhý sa nazýva kognitívny komponent (cognitive component) a tretí predstavuje spoločenský komponent (social component).

$w^{(t)}$ sa nazýva inertia weight (akási váha zotrvačnosti) a je to parameter z intervalu $[0, 1]$, ktorý sa môže s časom meniť. ϕ_1 a ϕ_2 sú parametre zrýchlenia, sú to konštanty. A nakoniec u_1 a u_2 sú náhodné premenné z rovnomerného rozdelenia na intervale $(0, 1)$. Poloha častice v čase $t + 1$ sa potom určí vzorcom

$$x_i^{(t+1)} = x_i^{(t)} + v_i^{(t+1)}.$$

Znamená to teda, že častica sa sčasti hýbe v svojom pôvodnom smere, sčasti sa hýbe k svojej doterajšej najlepšej polohe a sčasti ku globálnej najlepšej hodnote (resp. k najlepšej hodnote svojho okolia). Pohyb tiež ovplyvňuje náhodnosť.

Počas pohybu častice sa môže stať, že vyjde z prehľadávacieho priestoru. Toto sa dá ošetriť viacerými spôsobmi. Jedným z prístupov je reštrikcia, teda časticu, ktorá vyšla mimo priestor zrkadlovo prevrátime smerom dovnútra. Iný možný prístup je tzv. penalizácia. K účelovej funkcii sa pričíta penalizačná funkcia, ktorá je v prehľadávacom priestore nulová a mimo neho nadobúda určité hodnoty, ktoré sa zväčšujú tým viac, čím ďalej je daný bod od hranice. Algoritmus potom bude automaticky zlé body zamietať a bude sa pohybovať hlavne v priestore X. Existuje ešte viacero iných možností, avšak v praktickej časti našej práce sme skúšali iba tieto dva prístupy.

Keďže náš problém je diskrétna úloha, potrebujeme algoritmus upraviť. Najjednoduchším spôsobom je výsledné polohy zaokrúhliť. Môže sa však stať, že táto metóda spôsobí zníženie efektívnosti algoritmu. My sme však zaokrúhľované PSO použili, pretože sa ukázala ako pomerne dobrá metóda na úlohu (U2) (viď Kapitola 7.3). Kennedy a Eberhart však vo svojom článku [5] predstavili aj binárnu verziu PSO, ktorá sa dá aplikovať priamo na diskrétne úlohy.

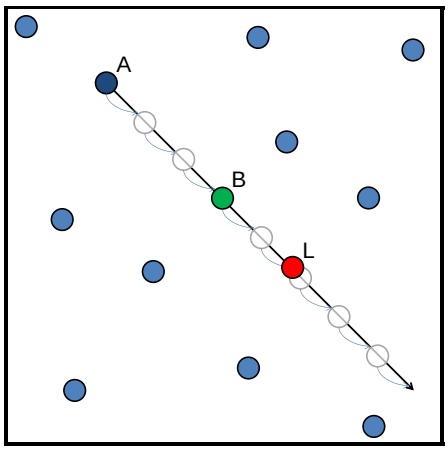
5.2.2 Self-Organizing Migrating Algorithm – SOMA

SOMA, alebo inak Self-Organizing Migrating algorithm, je pomerne novým objavom na poli evolučných algoritmov, jeho autorom je Ivan Zelinka [8]. Tento algoritmus, podobne ako PSO, nevytvára nových jedincov, ale jedinci sa pohybujú po množine prípustných riešení. Jednotliví jedinci navzájom ovplyvňujú svoj pohyb.

Jednu časticu (jedinca) počas jednej iterácie k charakterizuje:

- poloha $x_i^{(k)}$ – n-rozmerný vektor, kde n je rozmer úlohy
- fitness – skalárna hodnota vypočítaná pomocou účelovej funkcie

Algoritmus sa inicializuje populáciou náhodných jedincov a určí sa ich fitness. Jedinec s najlepšou fitness hodnotou sa vyberie ako líder. Nová populácia sa vytvorí tak, že všetci jedinci okrem lídra sa pokúsia nájsť si vhodnejšiu polohu. Skokmi určitej dĺžky sa budú približovať po priamke k lídrovi a spomedzi týchto polôh si vyberú tú s najlepšou hodnotou fitness.



Obr. 4: Migrácia jedinca (A) k lídrovi (L) v SOMA

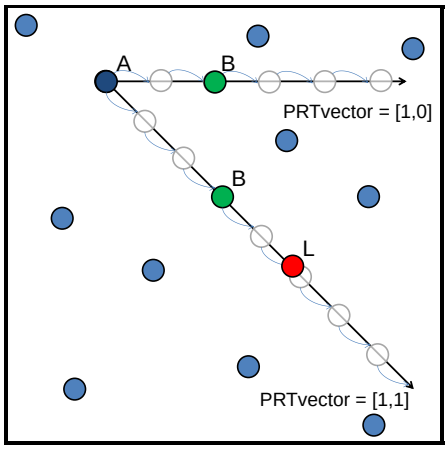
Jedinec (A) sa pohybuje skokmi danej dĺžky smerom k lídrovi (L) a každú novú polohu ohodnocuje účelovou funkciou – na Obr. 4 reprezentované ako prázdne krúžky.

Po prejdení celej trajektórie si vyberie polohu s najlepšou fitness – na príslušnom obrázku zelený bod (B). Pôvodnú populáciu tvoria modré body a líder, novú populáciu vytvoria líder spolu so zmigrovanými ostatnými časticami – budúce zelené body.

Do algoritmu prináša náhodnosť parameter **perturbácie**, ozn. PRT, definovaný v rozmedzí $(0, 1\rangle$. Pomocou tohto parametru sa vytvorí vektor perturbácie, $PRTvector$, nasledovným spôsobom:

$$PRTvector_i = \begin{cases} 0 & \text{s pravdepodobnosťou } 1 - PRT \\ 1 & \text{s pravdepodobnosťou } PRT \end{cases}, \text{ pre } i = 1, \ldots, n,$$

kde n je rozmer úlohy. Hodnota 0 na i-tom mieste vektora perturbácie znamená, že v danom smere sa častica nebude hýbať. Na nasledovnom obrázku môžeme vidieť, ako vplýva na pohyb častice perturbácia pre rozmer úlohy $n = 2$.



Obr. 5: Náhodnosť v SOMA

Budeme predpokladať, že vektor perturbácie nemôže byť nulový vektor, pretože vtedy by častica neprehľadala žiadnu časť priestoru. Teda ak sa vygeneruje $PRTvector = \mathbf{0}_n$, tak na náhodnom mieste vygenerujeme jednotku.

Opäť sa môže stať, že niektorí jedinci vyjdú z priestoru prípustných riešení. Aj v

tomto prípade môžme postupovať rôznym spôsobom. V práci sme podobne ako v PSO využili reštrikciu a penalizáciu.

Aj algoritmus SOMA má rôzne verzie. V práci [4] sú uvedené varianty:

- All-to-One – jedinci z populácie sa hýbu k jednému pevnému lídrovi,
- All-to-All – jedinci nemigrujú k určenému lídrovi, ale ku všetkým ostatným jedincom, čo výrazne zvyšuje výpočtovú náročnosť, ale aj veľkosť priestoru, ktorý algoritmus prehľadá, čím sa zvyšuje pravdepodobnosť nájdenia optima,
- All-to-One-Random – všetci jedinci sa hýbu k inému náhodne vybratému jedincovi,

a iné. V našej práci sme použili základný prístup All-to-One.

Algoritmus zapísaný pomocou pseudokódu uvádzame na nasledujúcej strane.

Algoritmus 4: SOMA

Input: $maxiter, popsize, PRT, krok, cesta$

vygenerujeme náhodnú populáciu veľkosti *popsize*
vypočítame fitness populácie
do L priradíme index jedinca s najmenšou hodnotou fitness
for $k = 1, \ldots, maxiter$ **do**
  for $i = 1, \ldots, popsize; i \neq L$ **do**
    for $j = 1, \ldots, n$ **do**
      if $rnd_j < PRT$ **then**
        $PRTvector_j = 1$
      else
        $PRTvector_j = 0$
      end
    end
    aktualizácia polohy: $x_{i,start}^{(k)} + (x_L^{(k)} - x_{i,start}^{(k)}) t PRTvector$, kde $t \in \langle \text{od } 0, \text{po } krok, \text{do } cesta \rangle$ a do $x_i^{(k+1)}$ priradíme hodnotu s najlepšou fitness
  end
  vyberieme nového lídra a jeho index priradíme do L
  if $\min fitness = 0$ **then**
    return *aktuálna populácia*
  end
end

6 1D prístup

Z vlastností kvadratických zvyškov sa dá ukázať, že je možné riešiť úlohu (U2) ako jednorozmernú, teda pre fixné y minimalizovať funkciu

$$g(x) = x^2 - y^2 \pmod{N}.$$

V tejto časti ukážeme, že toto je možné pre ľubovoľné $y \in \mathbb{Z}_n^*$. V našich experimentoch sme neskôr volili hodnotu $y = 1$.

Pripomeňme, že množinou $\mathbb{Z}_n^*$ sa myslia tie čísla zo zvyškovej triedy $\mathbb{Z}_n$, pre ktoré platí $NSD(a, n) = 1$ (pozri Definícia 4.2). Je zrejmé, že počet prvkov množiny $\mathbb{Z}_N^*$ pre $N = p \cdot q$ je práve hodnota Eulerovej funkcie $\varphi(N)$, t. j. $(p-1)(q-1)$, keďže Eulerova funkcia udáva počet nesúdeliteľných čísel menších ako N. Pre lepšie pochopenie čitateľa uvádzame nasledujúci príklad.

Príklad 6.1. Nech $N = 3 \cdot 5 = 15$. Potom množina $\mathbb{Z}_{15}^* = \{1, 2, 4, 7, 8, 11, 13, 14\}$ má 8 prvkov a hodnota Eulerovej funkcie pre N je $\varphi(N) = (3-1) \cdot (5-1) = 8$.

Množinu $\mathbb{Z}_n^*$ možno rozdeliť na dve disjunktné podmnožiny – kvadratické zvyšky a kvadratické nezvyšky. Nás by však zaujímalo, koľko je kvadratických zvyškov a koľko nezvyškov.

Dá sa odvodiť (pozri knihu [16]), že pre prvočíslo p platí $|Q_p| = \frac{p-1}{2}$ a $|\overline{Q}_p| = \frac{p-1}{2}$ (pripomeňme, že Q_p je množina všetkých kvadratických zvyškov a $\overline{Q}_p$ množina všetkých kvadratických nezvyškov), teda že polovica členov množiny $\mathbb{Z}_p^*$ sú kvadratické zvyšky a druhá polovica sú kvadratické nezvyšky. Uvažujme teraz $N = pq$. Platí, že $a \in \mathbb{Z}_N^*$ je kvadratický zvyšok modulo N práve vtedy, keď $a \in Q_p$ a zároveň $a \in Q_q$. Z toho vyplýva, že

$$|Q_N| = |Q_p| \cdot |Q_q| = \frac{(p-1)(q-1)}{4} \text{ a } |\overline{Q}_N| = \frac{3(p-1)(q-1)}{4}.$$

Pre objasnenie uvádzame nasledovný príklad.

Príklad 6.2. Nech $N = 3 \cdot 5 = 15$. Potom $Q_{15} = \{1, 4\}$ a $\overline{Q}_{15} = \{2, 7, 8, 11, 13, 14\}$. Vidíme, že $|Q_{15}| = |Q_3| \cdot |Q_5| = \frac{(3-1)(5-1)}{4} = 2$ a $|\overline{Q}_{15}| = \frac{3(3-1)(5-1)}{4} = 6$.

Nasledujúce tvrdenie a jeho dôsledok sú pre nás kľúčové.

Tvrdenie 6.1 (Počet odmocnín modulo n)**.** *Nech $n = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$, kde p_i sú rôzne nepárne prvočísla a $e_i \geq 1$. Ak $a \in Q_n$, tak a má práve 2^k rôznych odmocnín modulo n.*

Dôkaz. Pre potreby našej práce uvádzame bez dôkazu, viac v [1] a v [16]. □

Dôsledok 6.1. Špeciálne pre $N = pq$ platí, že ak $a \in Q_N$, tak a má práve 4 rôzne odmocniny modulo N.

Pre lepšie pochopenie dôsledku uvádzame príklad.

Príklad 6.3. Nech $N = 3 \cdot 5 = 15$. Pre kvadratický zvyšok 1 existujú 4 odmocniny modulo N – 1, 4, 11 a 14. Pre kvadratický zvyšok 4 máme tiež 4 odmocniny modulo N – 2, 7, 8 a 13.

Takto možno celú množinu $\mathbb{Z}_N^*$ rozdeliť na triedy ekvivalencie podľa toho, ku ktorému kvadratickému zvyšku prvky ako odmocniny prináležia. V každej triede sú potom 4 čísla a tried je presne toľko, koľko je kvadratických zvyškov $\pmod N$.

Teda pre zvyšok $a \in Q_N$ máme 4 odmocniny x_1, x_2, x_3 a x_4, pre ktoré platí

$$x_1^2 \equiv x_2^2 \equiv x_3^2 \equiv x_4^2 \equiv a \pmod N.$$

Preto pre ľubovoľné $y \in \mathbb{Z}_N^*$ máme 4 rôzne $x \in \mathbb{Z}_N^*$ také, že $x^2 \equiv y^2 \pmod N$. Z toho dve sú triviálne riešenia $x = \pm y$.

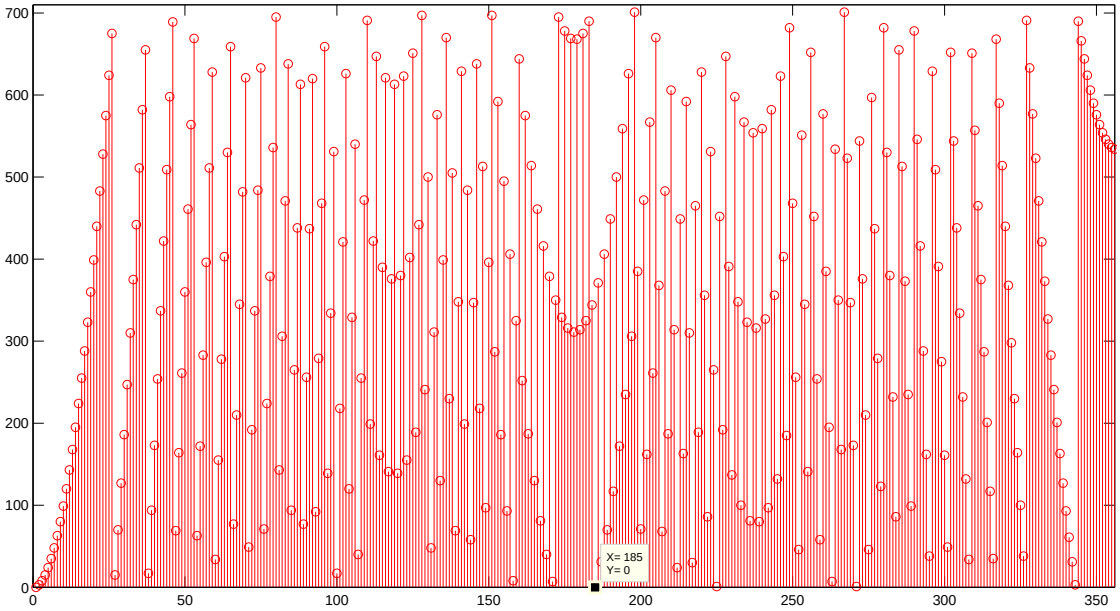
V praktickej časti sme konkrétne zvolili $y = 1$. Potom rovnica $x^2 \equiv 1 \pmod N$ má práve 4 rôzne riešenia v $\mathbb{Z}_N^*$. Dve riešenia sú triviálne $x = 1$ a $x = N - 1$. Tieto

však nevyhovujú podmienke $x \not\equiv \pm y \pmod{N}$ v našej úlohe (U2). Ostatné dve budú symetrické okolo bodu $(N-1)/2$. Máme teda zaručené práve jedno riešenie x úlohy (U2) pre fixné $y = 1$ na množine $X = \{2, \ldots, (N-1)/2\}$.

Na nasledujúcom obrázku možno vidieť funkciu $h : \{1, 2, \ldots, (N-1)/2\} \to \{0, \ldots, N-1\}$ s predpisom

$$h(x) = (x^2 - 1) \pmod{N}, \tag{7}$$

pre $N = 23 \cdot 31$. Na obrázku je vyznačené aj minimum funkcie s hodnotou 0 v bode $x = 185$.



Obr. 6: Graf funkcie h pre $N = 23 \cdot 31$

Z vlastností funkcie je zrejmé, že deterministické metódy globálnej optimalizácie nám nepomôžu. Preto sa túto funkciu pokúsime optimalizovať upraveným SOMA algoritmom. Účinnosť algoritmu potom porovnáme s náhodným prehľadávaním na množine X.

7 Praktická časť

V nasledujúcich kapitolách si ukážeme fungovanie spomenutých prístupov v praxi. V programovacom prostredí Python sme napísali jednotlivé procedúry, ktoré prikladáme v Prílohe. Každý z týchto algoritmov sme 100-krát spustili pre tri rôzne hodnoty N a výsledky porovnáme. Konkrétne sme používali hodnoty $N = 133 \cdot 211$, $N = 691 \cdot 701$, $N = 1777 \cdot 2777$.

Všetky výpočty sme vykonávali na počítači Lenovo Y570 so špecifikáciou:

- Pamäť RAM: 8GB
- Procesor: Intel(R) Core(TM) i7-2670QM CPU @ 2.20GHz (8 CPUs), ~2.2GHz
- Operačný systém: Windows 7 Home Premium 64-bit

7.1 Kvadratické sito

Prvú uvedieme klasickú metódu rozkladania N na prvočísla p a q – kvadratické sito. Algoritmus sme spustili pre spomenuté tri hodnoty N, ktoré sme použili aj pri ostatných metódach pre porovnanie. Parameter t sme volili sp;sobom uvedeným v kapitole 4. Nakoniec sme sa pokúsili faktorizovať čo najväčšie číslo. Najväčšie, ktoré sme úspešne rozložili bolo $N = 112272535095293 \cdot 794757777698303$, čo je 29 miestne číslo. V tabuľke nižšie uvádzame čas trvania behu algoritmu pre jednotlivé čísla N v sekundách (T).

N	**T [s]**
$199 \cdot 211$	0.08
$691 \cdot 701$	3.90
$1777 \cdot 2777$	17.71
$1517265216 \cdot 2663357$	74.33
$112272535095293 \cdot 794757777698303$	5011.28

Tabuľka 2: Kvadratické sito – výsledky

Môžme vidieť, že síce algoritmus trvá pomerne dlho, dokázal v prijateľnom čase úspešne faktorizovať relatívne veľké číslo. Pre malé N je algoritmus pri optimálnej voľbe t extrémne rýchly. Výpočtový čas však s rastúcim N veľmi narastá, preto čísla rádovo 10^{617}, aké sa pri RSA používajú, nedokážeme týmto spôsobom rozložiť v prijateľnom čase. V práci budeme skúmať, či sú prístupy globálnej optimalizácie rovnocenným súperom pre kvadratické sito, teda nie je našim cieľom kvadratické sito priamo poraziť. V tomto smere sa budeme snažiť, aby nami navrhnuté algoritmy porazili náhodné prehľadávanie priestoru.

7.2 Náhodné prehľadávanie (Random search)

V náhodnom prehľadávaní sme jednoducho vyberali náhodné dvojice x a y z priestoru prípustných riešení a testovali sme, či vyhovujú našej rovnici (U2). V tabuľke nižšie je uvedená percentuálna úspešnosť algoritmu na 100 spustení (Usp), priemerný výpočtový čas v sekundách (T) a priemerný počet iterácií, ktoré algoritmus bežal (Iter). Maximálny počet iterácií sme nastavili na 1000000.

N	**Usp [%]**	**T [s]**	**Iter**
$199 \cdot 211$	100	0.16	20272.29
$691 \cdot 701$	97	2.05	278006.07
$1777 \cdot 2777$	32	7.26	860169.46

Tabuľka 3: Random search – výsledky

Môžme vidieť, že kým je N ešte pomerne malé, tak algoritmus beží veľmi úspešne a hlavne veľmi rýchlo. Napríklad $N = 199 \cdot 211$ rozložíme náhodným prehľadávaním v priemere za menej ako sekundu. Ako sa však N zväčšuje, tak výrazne klesá úspešnosť algoritmu. Čas sa tiež zväčšuje, ale vidíme, že stále je výrazne menší ako čas kvadratického sita. To súvisí aj s nastavením parametra počtu maximálnych iterácií. Ak by sme zvýšili tento parameter, zvýšila by sa aj úspešnosť, aj výpočtový čas.

7.3 PSO

Algoritmus PSO vyšiel v článku [2], ktorý slúžil ako inšpirácia našej práce, ako najvhodnejší z evolučných algoritmov (síce nie lepšie ako Random search). Preto sme sa rozhodli venovať sa mu aj v tejto práci. Skúšali sme tri rôzne varianty PSO. V prvých dvoch sme použili klasickú spojitú verziu PSO so zaokrúhľovaním. Líšia sa od seba spôsobom, akým zabezpečujeme, že kandidáti na riešenie ostávajú v priestore prípustných riešení. V Tabuľke 4 uvádzame výsledky pre variant, v ktorom sme použili reštrikciu. Ak častica vyšla z priestoru, tak sme najprv skúsili jej rýchlosť zmenšiť na polovicu. Ak aj tak presiahla hranice, tak sme ju otočili smerom dovnútra s polovičnou rýchlosťou. Rýchlosť tiež bola obmedzená na maximálne $\lfloor (N-1)/4 \rfloor$.

Parametre sme nastavovali počas viacerých spustení, pričom najlepšie výsledky dosahoval algoritmus pri nasledovných parametroch:

- $Npart = 15$ – počet častíc, obvykle sa volí v intervale $\langle 10, 20 \rangle$
- $v_0 = 3$ – počiatočná rýchlosť, volí sa náhodne v intervale $\langle -v_0, v_0 \rangle$
- $maxiter = 50000$ – maximálny počet iterácií
- $w = 0.9$ – parameter zotrvačnosti (inertia weight)
- $\phi_1 = 3$ – personal component, volí sa zvyčajne medzi 0 a 4
- $\phi_2 = 1$ – social component, volí sa zvyčajne medzi 0 a 4

Zotrvačnosť sme volili 0.9, pretože čím je nižšia, tým skôr by sa mali častice ustáliť a my chceme, aby prehľadali čo najväčší priestor. Keďže účelová funkcia nemá okolo minima oblasti s nízkou funkčnou hodnotou, tak nemôžme očakávať, že pri doteraz najlepšom lokálnom optime, sa bude vyskytovať globálne optimum. Preto sme volili social komponent rovný 1. Na druhej strane môžme povedať, že ak sa bude častica pohybovať k zatiaľ známemu najlepšiemu riešeniu, môže náhodou natrafiť na globálne

minimum, skúsili sme preto zvoliť $\phi_1 = 3$. Výsledky získané zo 100 behov programu sú uvedené v Tabuľke 4.

N	**Usp [%]**	**T [s]**	**Iter**
$199 \cdot 211$	100	1.33	1732.02
$691 \cdot 701$	82	17.45	22682.00
$1777 \cdot 2777$	14	35.96	46348.77

Tabuľka 4: PSO s reštrikciou – výsledky

Pri menších číslach je PSO pomerne účinný, problém nastáva pri $N = 1777 \cdot 2777$, kedy je úspešnosť už len 14 zo 100. Zmeníme teda parametre w, ϕ_1 a ϕ_2 na

- $w = 0.6$, $\phi_1 = 1$, $\phi_2 = 3$.

Výsledkom je Tabuľka 5, v ktorej môžme vidieť, že tento algoritmus je naozaj citlivý na výber parametrov. Výsledky nám vyšli podstatne lepšie, dokonca porovnateľne s náhodným prehľadávaním.

N	**Usp [%]**	**T [s]**	**Iter**
$199 \cdot 211$	100	1.04	1539.10
$691 \cdot 701$	96	10.02	15055.12
$1777 \cdot 2777$	30	29.55	42353.30

Tabuľka 5: PSO s reštrikciou – výsledky s novými parametrami

Ďalší variant PSO používal na vyradenie nevhodných riešení penalizáciu. Teda k účelovej funkcii sme pridali funkciu, ktorá je v prehľadávacom priestore nulová a mimo neho narastá kvadraticky so zväčšujúcim sa presahom. Pridali sme ešte penalizáciu bodov, v ktorých $x = y$, keďže tie nevyhovujú podmienke. V tomto prípade sme hodnotu účelovej funkcie nastavili na $+\infty$. Parametre $Npart, v0, maxiter, w, \phi 1$ a $\phi 2$ sme nastavili rovnako ako v prístupe s reštrikciou, aby sme ich mohli navzájom dobre porovnať.

N	**Usp [%]**	**T [s]**	**Iter**
$199 \cdot 211$	100	0.78	1553.66
$691 \cdot 701$	93	9.45	19169.00
$1777 \cdot 2777$	24	21.21	43281.12

Tabuľka 6: PSO s penalizáciou – výsledky

Vidíme, že penalizácia dopadla veľmi podobne ako reštrikcia. Síce je algoritmus o niečo menej úspešný, časovo trval kratšie. To by sa ale dalo vysvetliť aj nerovnakým zaťažením počítača, preto môžme konštatovať, že penalizácia je vzhľadom na úspešnosť mierne horšia ako reštrikcia. Ako posledné sme skúsili úlohu previesť na spojitú, čím by sme sa zbavili zaokrúhľovania. To sme dosiahli obmenou účelovej funkcie úlohy (U2). Najprv sme definovali spojitú funkciu, ktorá nadobúda nulové hodnoty iba pre celé čísla. Takou funkciou je napríklad:

$$F(z) = \begin{cases} z - \lfloor z \rfloor, & \text{ak } z - \lfloor z \rfloor \leq 1/2, \\ \lceil z \rceil - z, & \text{inak.} \end{cases}$$

Ďalej sme vzali funkciu

$$f(x, y) = \frac{|x^2 - y^2|}{N},$$

ktorej hodnoty sú celé čísla práve vtedy, keď $(x^2 - y^2) \pmod N \equiv 0$. Zložením F a f dostaneme funkciu, ktorá má minimum 0 vtedy, ak sme našli riešenie rovnice z úlohy (U2). V tomto prípade sa ale môže stať (a v praxi sa to dialo pomerne často), že $f(x, y)$ dáva celočíselné hodnoty, aj keď samotné x a y nie sú celé čísla. Preto ešte k funkcii pripočítame $F(x)$ a $F(y)$. Naša nová účelová funkcia teda bude

$$G(x, y) = F(f(x, y)) + F(x) + F(y),$$

s definičným oborom $\langle 1, (N-1)/2\rangle \times \langle 1, (N-1)/2\rangle \subset \mathbb{R}^2$. Súradnice bodu s najnižšou hodnotou fitness sme potom zaokrúhlili na celé čísla.

N	**Usp [%]**	**T [s]**	**Iter**
$199 \cdot 211$	0	36.93	50000

Tabuľka 7: spojitá verzia PSO s reštrikciou – výsledky

Keďže pre $N = 199 \cdot 211$ bol tento algoritmus úplne neúspešný, tak nemalo zmysel ho spúšťať pre väčšie čísla. Problém spojitých evolučných algoritmov totižto je, že nevedia hľadať presné, iba približné výsledky, pričom naša úloha vyžaduje presný výsledok. Pomôcť v úspešnosti by mohlo využitie klasickej optimalizácie v rámci tohto algoritmu. Jedna možnosť je aplikovať klasické metódy na výsledok tejto PSO, čo by však nemalo veľký význam, pretože hodnota G blízka 0 ešte neznamená, že bod je blízko globálneho minima. Mohli by sme však používať spádové metódy priamo v PSO. Okolie dosiaľ najlepšieho riešenia by sme prehľadali spádovou metódou a ak nenájdeme bod s $G(x, y) = 0$, tak by sme z tohto okolia s algoritmom úplne odišli. Takýto postup by ale ešte výrazne zvýšil výpočtovú náročnosť algoritmu, takže by nemohol konkurovať ostatným metódam. Preto sme sa radšej venovali ďalším evolučným algoritmom, ktoré prinášali sľubnejšie výsledky.

7.4 SOMA

V práci, ako sme už spomenuli, sme využívali All-to-One variant algoritmu SOMA. Tiež sme ale skúsili využiť penalizáciu aj reštrikciu. Tretí prístup popíšeme neskôr.

Parametre pre algoritmy sme nastavovali nasledovne:

- $popsize = 20$ – veľkosť populácie
- $PathLen = 4$ – dĺžka cesty, znamená, že častica prehľadá 4-násobok vzdialenosti od lídra, číslo 4 sme zvolili, aby sa častice, ktoré sa dostanú blízko k sebe, vedeli

opäť vzdialiť

- $stepcount = 1000$ – počet krokov, ktoré častica počas cesty urobí, znamená to, že jeden krok má dĺžku $\frac{PathLen \cdot (\text{vzdialenosť od lídra})}{stepcount}$
- $PRT = 0.5$ – parameter perturbácie sa volí v intervale $\langle 0, 1 \rangle$, čím je vyšší, tým menšia je náhodnosť
- $maxiter = 25$ – maximálny počet iterácií, volí sa podstatne menší ako pri PSO, keďže v každej iterácií sa vykonáva $stepcount \cdot (popsize - 1)$ operácií

Algoritmus sme spustili najprv s reštrikciou, teda riešenia, ktoré vyšli z prehľadávacieho priestoru sme zamietli. Namiesto nich sme ale testovali náhodné body z priestoru. Výsledky tejto metódy je možno vidieť v tabuľke nižšie.

N	**Usp [%]**	**T [s]**	**Iter**
$199 \cdot 211$	100	4.54	0.49
$691 \cdot 701$	67	44.26	13.00
$1777 \cdot 2777$	15	77.83	22.00

Tabuľka 8: SOMA s reštrikciou – výsledky

Tiež sme skúsili prístup s penalizáciou, ktorú sme robili analogicky ako pri penalizácii v algoritme PSO – mimo priestoru sme k účelovej funkcii pridali funkciu, ktorá narastá kvadraticky so zväčšujúcim sa presahom a body, kde $x = y$ sme penalizovali hodnotou $+\infty$.

N	**Usp [%]**	**T [s]**	**Iter**
$199 \cdot 211$	99	7.12	1.70
$691 \cdot 701$	42	42.83	16.00
$1777 \cdot 2777$	4	82.88	23.00

Tabuľka 9: SOMA s penalizáciou – výsledky

Zdá sa, že použitie reštrikcie je pre tento algoritmus vhodnejšie, keďže algoritmus bol úspešnejší bez penalizácie. Rozdiel medzi týmito prístupmi však nie je veľmi veľký. Problémom je skôr nízka úspešnosť SOMA pre $N = 1777 \cdot 2777$ v porovnaní s PSO, ktoré dopadlo oveľa lepšie. Skúmali sme teda, čo sa v SOMA dialo pre veľké N a zistili sme, že algoritmus sa často zasekol v lokálnom minime, z ktorého sa už nevedel dostať preč. Implementovali sme preto vlastnú verziu algoritmu SOMA s resetovaním. Keď sa v dvoch iteráciách po sebe najlepšie riešenie nezmenilo, tak sme populáciu resetovali tak, že všetky jedince sme opäť náhodne inicializovali. Túto metódu sme použili aj pre variant s reštrikciou aj pre penalizáciu. Výsledky možno vidieť v tabuľkách 10 a 11.

N	**Usp [%]**	**T [s]**	**Iter**
$199 \cdot 211$	100	3.52	2.55
$691 \cdot 701$	60	22.72	16.00
$1777 \cdot 2777$	9	33.15	24.00

Tabuľka 10: SOMA s penalizáciou a resetovaním – výsledky

N	**Usp [%]**	**T [s]**	**Iter**
$199 \cdot 211$	100	3.50	2.00
$691 \cdot 701$	82	22.32	13.00
$1777 \cdot 2777$	15	39.39	24.00

Tabuľka 11: SOMA s reštrikciou a resetovaním – výsledky

Môžme vidieť, že nami navrhnuté riešenie v oboch variantoch algoritmu SOMA prinieslo zlepšenie. Najlepšou verziou sa teda ukázalo použitie reštrikcie v kombinácii s resetovaním. Z tabuliek 8 a 11 však vidíme, že zlepšenie je výrazné iba pre $N = 691 \cdot 701$, pre $N = 1777 \cdot 2777$ dokonca resetovanie žiadne zlepšenie neprinieslo.

Celkovo môžme konštatovať, že varianty algoritmu SOMA boli na testovaných číslach

menej úspešné ako diskrétne PSO. V kombinácii so zvýšenou časovou náročnosťou sa algoritmus SOMA ukázal ako nie veľmi vhodným na tento typ úlohy.

7.5 1D prístup

Nakoniec sme sa pokúsili úlohu riešiť v jednom rozmere, ako sme uviedli v Kapitole 6. Najprv sme použili náhodné prehľadávanie na množine $\{2, 3, \ldots (N-1)/2\}$ a potom algoritmus, ktorý sme nazvali 1D SOMA (bude popísaný neskôr). Skúšali sme aplikovať aj PSO v jednom rozmere, avšak nedosahoval výsledky dosiahnuté ostatnými prístupmi, preto sme ho ďalej neuvádzali. Výstup z tejto procedúry však možno nájsť v prílohe. Výsledky pre jednorozmerné náhodné prehľadávanie uvádzame v tabuľke nižšie. Táto metóda má len jeden parameter, ktorým je maximálny počet iterácií. Ten sme zvolili kvôli konzistencii pre všetky N rovnaký, a to 1000000 iterácií. Výsledky sa nachádzajú v nasledovnej tabuľke.

N	**Usp** [%]	**T** [s]	**Iter**
$199 \cdot 211$	100	0.08	20346.38
$691 \cdot 701$	100	0.96	260467.77
$1777 \cdot 2777$	34	3.47	821526.00

Tabuľka 12: random search 1D – výsledky

Tak isto ako v algoritme SOMA pracujeme s náhodnou počiatočnou populáciou jedincov, z ktorých najlepší sa vyberie ako líder. Ostatní jedinci sa potom pohybujú smerom k lídrovi po určitých krokoch, v ktorých testujú funkčnú hodnotu a zastavia sa tam, kde bola hodnota najmenšia. Rozdielom oproti klasickému algoritmu SOMA je, že jedinci nemajú možnosť výberu smeru, keďže pracujeme iba v jednom rozmere. 1D SOMA je teda algoritmus, ktorý sa náhodne inicializuje, ale potom už funguje deterministicky. Keďže sa ukázalo, že zaseknutie v lokálnom minime je veľmi časté (aj preto, že algoritmus nie je náhodný), implementovali sme aj resetovanie. Jedinci sa pohybujú po krokoch defino-

vaných parametrom *step*, pričom dĺžka cesty je určená parametrom *path* ako násobok vzdialenosti od lídra. Celkový počet krokov jedného jedinca je obmedzený parametrom *maxsteps*. Napokon celkový počet iterácií sa riadi parametrom *maxiter*.

Parametre algoritmu sme po viacerých pokusoch nastavili nasledovne:

- $Npop = 15$
- $maxiter = 100$
- $maxsteps = 1000$
- $path = 3$
- $step = 2$

V Tabuľke 13 možno vidieť, že výsledky sú porovnateľné s náhodným prehľadávaním, pre $N = 1777 \cdot 2777$ bolo dokonca 1D SOMA úspešnejšie, avšak aj priemerný výpočtový čas bol vyšší. Môžme teda konštatovať, že tento prístup je použiteľný rovnako ako náhodné prehľadávanie.

N	**Usp [%]**	**T [s]**	**Iter**
$199 \cdot 211$	100	0.14	2.84
$691 \cdot 701$	99	1.74	21.23
$1777 \cdot 2777$	44	4.92	78.32

Tabuľka 13: 1D SOMA – výsledky

Z výsledkov sa dokonca dá usúdiť, že 1D variant je vo všeobecnosti lepší ako pôvodný prístup globálnej optimalizácie. S rastúcim N sa však šanca na nájdenie globálneho minima podstatne zmenšuje, keďže ako sme ukázali v Kapitole 6 je toto riešenie na množine $X = \{2, 3, \ldots, (N-1)/2\}$ jediné. Potrebovali by sme teda algoritmus, ktorý by bol účinnejší ako náhodné prehľadávanie.

Skúsili sme preto ešte jednu obmenu algoritmu, tentokrát sme zmenili účelovú funkciu. Keďže aj evolučné prehľadávacie algoritmy mali problém s mnohými lokálnymi minimami funkcie $h(x) = (x^2 - 1) \pmod N$ (funkcia (7) zobrazená na Obrázku 6), tak sme sa rozhodli zrezať funkciu h tak, aby sme časť lokálnych miním odstránili. Zostrojili sme teda funkciu $H(x)$, $H : \{1, 2, \ldots, (N-1)/2\} \to \{0, \ldots, (N-1)/2\}$ ako minimum $h(x)$ a kvadratickej funkcie $q_h(x) = A(x - B)^2 + C$, kde

$$A = \frac{\frac{N-1}{2} - 10}{\left(1 - \left\lfloor \frac{N-1}{4} \right\rfloor\right)^2},\ B = \left\lfloor \frac{N-1}{4} \right\rfloor \text{ a } C = 10.$$

Takto zvolené parametre nám zabezpečia, že vrchol paraboly bude v bode $\left[\left\lfloor \frac{N-1}{4} \right\rfloor, 10\right]$ a v bodoch 1 a $\frac{N-1}{2}$ bude funkčná hodnota $q_h(x) = \frac{N-1}{2}$. Funkcia H je teda definovaná ako

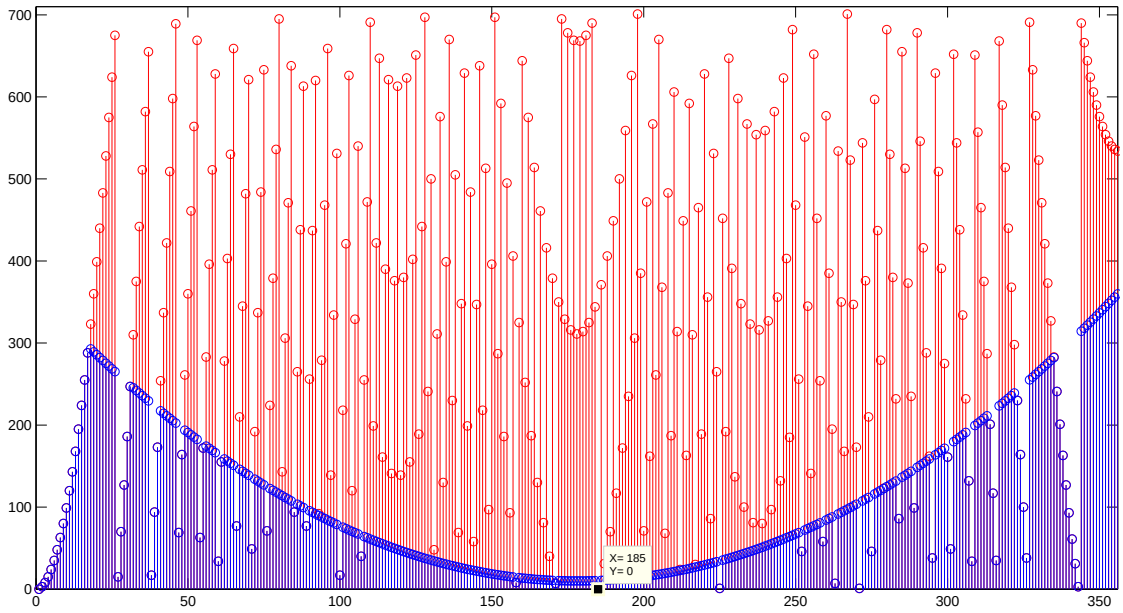
$$H(x) = \min\left(h(x), q_h(x)\right).$$

Na Obrázku 7 je červenou farbou vykreslená pôvodná funkcia h a modrou upravená funkcia H. Je vidieť, že priebeh funkcie h sme transformáciou vyhladili pri zachovaní rovnakého minima. Na túto funkciu sme potom aplikovali algoritmus 1D SOMA s rovnakým nastavením parametrov. Výsledky sú uvedené v Tabuľke 14.

N	**Usp [%]**	**T [s]**	**Iter**
$199 \cdot 211$	100	0.22	0.62
$691 \cdot 701$	99	3.89	24.16
$1777 \cdot 2777$	44	11.80	72.55

Tabuľka 14: 1D SOMA pre funkciu H – výsledky

Môžme si všimnúť, že úspešnosť algoritmu sa vôbec nezlepšila, naopak výrazne sa zvýšil priemerný čas, za ktorý algoritmus našiel riešenie. Tiež sme si pri spúšťaní algoritmu všimli, že v porovnaní s predošlými prípadmi použitia resetovania sa algorit-



Obr. 7: Graf funkcie h a H pre $N = 23 \cdot 31$

mus značne častejšie resetoval. Je to zrejme dôsledok toho, že algoritmus 1D SOMA sa oveľa častejšie dokonvergoval do lokálneho minima vo vrchole paraboly q_h, do bodu $\left[\left\lfloor \frac{N-1}{4} \right\rfloor, 10\right]$. Najlepšou metódou pre jednorozmerný prístup teda ostal algoritmus 1D SOMA aplikovaný na pôvodnú funkciu.

7.6 Zhrnutie

V predchádzajúcich podkapitolách sme predstavili rôzne metódy rozkladania čísel na prvočísla. Teraz sa pozrieme na to, ktorý z prístupov sa ukázal ako najefektívnejší.

V prvom rade musíme konštatovať, že žiaden z navrhnutých evolučných algoritmov neprekonal doteraz známu metódu rozkladania na prvočísla. Algoritmus kvadratického sita sa naozaj ukázal ako najlepšia alternatíva na faktorizáciu čísel. Tento výsledok však nie je veľmi prekvapivý, nakoľko kvadratické sito je známe a skúmané už od roku 1981. Avšak použitie evolučných algoritmov v tejto oblasti sa začalo skúmať až v 21. storočí, kedy sa výrazne zvýšil záujem o tieto metódy. V porovnaní sa teda budeme sústrediť na

jednotlivé evolučné algoritmy a kvadratické sito z porovnania vynecháme, keďže výrazne predčí všetky ostatné algoritmy. Vynecháme tiež neúspešnú metódu spojitého PSO. Ďalej z algoritmov SOMA uvádzame iba SOMA s resetovaním, keďže tento prístup úspešnosť zlepšoval.

Nižšie uvádzame tabuľku, v ktorej môžme vidieť porovnanie jednotlivých metód pre $N = 1777 \cdot 1777$, keďže pri tejto voľbe sa najviac ukázali rozdiely v prístupoch.

Metóda		**Usp [%]**	**T [s]**	**Iter**
Random search		32	7.26	860169
PSO	reštrikcia	30	29.55	42353
	penalizácia	24	21.21	43281
SOMA reset	reštrikcia	15	39.39	24
	penalizácia	9	33.15	24
1D	Random search	34	3.47	821526
	SOMA 1D s funkciou h	44	4.92	78
	SOMA 1D s funkciou H	44	11.80	73

Tabuľka 15: Porovnanie metód pre $N = 1777 \cdot 2777$

Z tabuľky môžme vidieť, že ako najefektívnejšie prístupy sa pri rovnakom nastavení parametrov ukázali algoritmy v jednom rozmere. Tu náš upravený algoritmus SOMA 1D dokonca úspešnosťou poráža náhodné prehľadávanie. Časovo je však náročnejší. Výpočtová náročnosť stúpla aj transformáciou $h \to H$, preto môžme konštatovať, že takáto úprava algoritmu nepomohla.

Pozrime sa teraz na ostatné prístupy. Celkovo vhodnejší sa ukázal prístup reštrikcie v porovnaní s penalizáciou. Porovnaním PSO a SOMA sa ako lepší ukázal PSO, ktorý úspešnosťou konkuruje náhodnému prehľadávaniu. Časovo sú však oba algoritmy PSO aj SOMA výrazným zhoršením oproti náhodnému prehľadávaniu, preto môžme konštatovať,

že táto úloha pre ne nie je práve vhodná. Tým sa potvrdil záver autorov článku [2]. Pozitívne ale je, že naša implementácia algoritmu PSO dosahovala lepšie výsledky ako v spomínanom článku. To môže byť spôsobené napríklad nastavením parametrov, čo v článku autori presne nepopisujú.

Na záver môžme zhodnotiť, že našim výskumom sa podarilo zlepšiť výsledky z článku, ktorý bol našou motiváciou. Úplne nový prístup v jednom rozmere sa tiež ukázal ako veľmi úspešný, a taktiež efektivita nami navrhnutého jednorozmerného algoritmu SOMA sa ukázala ako veľmi priaznivá. Pri týchto výsledkoch však treba mať na zreteli, že klasický prístup teórie čísel ostáva neporazený. V každom prípade je prístup globálnej optimalizácie otvorený ďalšiemu výskumu. Naskytá sa mnoho rôznych nových evolučných algoritmov, ktorými by sa mohla úloha riešiť.

Ďalšou možnosťou urýchlenia algoritmov je paralelizácia. Dalo by sa riešiť viacero úloh $h_i(x) = x^2 - y_i^2 \pmod N$ pre akýkoľvek výber $y_i \in \mathbb{Z}_N^*$, keďže v kapitole 6 sme zaručili riešenie pre každú z týchto kongruencií. Tým by sa mohol výpočtový čas algoritmu výrazne skrátiť. Tomuto sme sa však už v našej diplomovej práci nevenovali, kvôli nedostatku priestoru a technickej vybavenosti. Táto oblasť teda ostáva otvorená budúcemu skúmaniu.

Záver

V práci sme skúmali známe aj nové prístupy na rozkladanie čísel na prvočísla. Čitateľa sme najprv oboznámili s kryptografickou podstatou tejto úlohy a so základnými princípmi a pojmami v kryptografii. Venovali sme sa aj konkrétnemu kryptosystému RSA, v ktorom sa na šifrovanie využíva práve náročnosť faktorizácie čísel. Ďalej sme predstavili prístup teórie čísel, ktorý je doteraz najefektívnejším známym prístupom na danú úlohu. V Kapitole 5 sme sa potom na úlohu pozreli cez diskrétnu globálnu optimalizáciu. Metódy, ktoré sme využili na jej riešenie patria medzi evolučné algoritmy. Motiváciu k tomuto prístupu sme čerpali z článku [2]. Úlohu predstavenú v tomto článku sme potom s využitím poznatkov z teórie čísel previedli do jedného rozmeru. Takto upravenú úlohu sme opäť optimalizovali použitím evolučných algoritmov.

V praktickej časti našej práce sme sa venovali testovaniu jednotlivých prístupov. V programe Python 3.4.3 sme vytvorili procedúry algoritmov kvadratického sita, náhodného prehľadávania, PSO, SOMA a jednorozmerný variant náhodného prehľadávania a algoritmu SOMA. Každý algoritmus sme stokrát spustili pre tri rôzne čísla, ktoré sme faktorizovali. Výstupy jednotlivých spustení sa nachádzajú v prílohe, úspešnosť algoritmu, priemerný čas a priemerný počet iterácií sú uvedené v tabuľkách v práci. Účelom nášho skúmania bolo zistiť, ktorý z prístupov globálnej optimalizácie je na úlohu faktorizácie najvhodnejší, respektíve či nejaký algoritmus bude úspešnejší ako náhodné prehľadávanie.

Zaujímavým výsledkom našej práce bolo, že algoritmus SOMA bol v porovnaní s PSO pri prístupe dvojrozmernej globálnej optimalizácie výrazne horší. Avšak v jednom rozmere sa ukázal SOMA algoritmus práve opačne – v porovnaní s PSO bol výrazne úspešnejší, dokonca predčil aj jednorozmerný variant náhodného prehľadávania.

Aplikovanie globálnej optimalizácie na problém RSA sa teda síce neukázalo ako úspešnejšie voči kvadratickému situ, avšak je možné túto myšlienku ďalej rozvíjať. Či už

použitím rôznych iných evolučných algoritmov, alebo prípadnou paralelizáciou úlohy, kedy by sa napríklad rozdelil prehľadávací priestor a na viacerých počítačoch naraz počítali jednotlivé úlohy. Ak by sa potom našlo optimum jednej z úloh, našli by sme celkové riešenie.

Celkovo môžme konštatovať, že cieľ práce sa nám podarilo splniť. Počas práce sme naštudovali a prakticky vyskúšali známe prístupy k problému RSA. Vybrané metódy sa nám podarilo úspešne aplikovať. Taktiež sme dokázali navrhnúť alternatívnu metódu na riešenie tohto problému, ktorá v praxi porážala aj náhodné prehľadávanie.

Prácu sme písali v prvom rade pre tých, ktorí sa zaujímajú o kryptosystém RSA a o matematický problém, ktorý využíva. Čitateľ sa oboznámi s inovačnou metódou riešenia tohto problému globálnou optimalizáciou a aplikovaním evolučných algoritmov. V neposlednom rade bola práca prínosom aj pre samotného autora, ktorý sa oboznámil s veľkým množstvom evolučných algoritmov a získal rozhľad v rôznych aspektoch teórie čísel v súvislosti s rozkladaním čísel na prvočísla. Tiež sa zoznámil s programovacím prostredím jazyku Python a úspešne naprogramoval jednotlivé metódy.

Literatúra

[1] **A. Menezes, P. van Oorschot, S. Vanstone**: *Handbook of Applied Cryptography*, CRC Press, 1996, `http://cacr.uwaterloo.ca/hac/`

[2] **E. C. Laskari, G. C. Meletiou, M. N. Vrahatis**: *Problems of cryptography as discrete optimization tasks*, Nonlinear Analysis vol. 63, p. e831 – e837, 2005

[3] **A. D. Martin, K. M. Quinn**: *A review of discrete optimization algorithms*, The Political Methodologist, vol. 7, 2003

[4] **Lucie Koděrová**: *Heuristiky*, bakalárska práca, Univerzita Palackého, Olomouc, Prírodovedecká fakulta, Katedra matematickej analýzy a aplikácií matematiky, 2008

[5] **J. Kennedy, R. C. Eberhart**: *A discrete binary version of the particle swarm algorithm*, IEEE, International Conference on Systems, Man, and Cybernetics, Orlando, FL, vol. 5, p. 4104 – 4108, 1997

[6] **S. Singh**: *The Science of Secrecy from Ancient Egypt to Quantum Cryptography*, Doubleday, New York, 1999

[7] **Kristína Ciesarová**: *Načo sú dobré prvočísla?*, bakalárska práca, Univerzita Komenského, FMFI, 2012

[8] **Ivan Zelinka**: *SOMA – Self-Organizing Migrating Algorithm*, New Optimization Techniques in Engineering, Springer-Verlag, 2004

[9] **Cypher Research Laboratories**: *A Brief History of Cryptography*, Marec 2014, `http://www.cypher.com.au/crypto_history.htm`

[10] **Marek Obitko**: *Introduction to Genetic Algorithms*, Marec 2014, `http://www.obitko.com/tutorials/genetic-algorithms/crossover-mutation.php`

[11] **Eric Landquist**: *The Quadratic Sieve Factoring Algorithm*, MATH 488: Cryptographic Algorithms, 14. December, 2001, `http://www.cs.virginia.edu/crab/QFS_Simple.pdf`

[12] **Marián Mach**: *Evolučné algoritmy: Prvky a princípy*, Technická univerzita, Košice, 2009, `http://neuron-ai.tuke.sk/machm/publications/kniha-eapp.pdf`

[13] **H.-P. Schwefel, G. Rudolph**: *Contemporary Evolution Strategies*, V: F. Morán, A. Moreno, P. Chacón, Advances in Artificial Life, Third International Conference on Artificial Life, Lecture Notes in Artificial Intelligence, Vol. 929, Springer Berlin p. 893 -– 907, 1998.

[14] **Ron R. Rivest, Adi Shamir, Leonard M. Adleman**: *Cryptographic communications system and method*, Massachusetts Institute Of Technology, U.S. Patent, US 4405829 A, 1977, `http://www.google.com/patents/US4405829`

[15] **Ron R. Rivest, Adi Shamir, Leonard M. Adleman**: *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*, Communications of the ACM 21 (2): p. 120 -– 126, 1978

[16] **M. Kolibiar, A. Legéň, T. Šalát, Š. Znám**: *Algebra a príbuzné disciplíny*, 4. vyd., Alfa, Bratislava, 1990

[17] **T. Šalát**: *Algebra a teoretická aritmetika (2)*, 1. vyd., Alfa, Bratislava, 1986

[18] **Chad Seibert**: *Integer Factorization using the Quadratic Sieve*, University of Minnesotta, March 16, 2011, `http://micsymposium.org/mics_2011_proceedings/mics2011_submission_28.pdf`

[19] **Stephani Lee Garrett**: *On the quadratic sieve*, Diplomová práca, The University of North Carolina at Greensboro, 2008, `http://libres.uncg.edu/ir/uncg/f/umi-uncg-1581.pdf`.

Dodatok

Programy v prostredí Python

K našej diplomovej práci prikladáme súbory s kódmi jednotlivých naprogramovaných procedúr v prostredí Python 3.3.4. Pracovali sme s vlastným modulom na prácu s maticami a vektormi, oba prikladáme. Tieto moduly a programy kvadratického sita boli vytvorené autorom počas online kurzu Coding the Matrix `https://www.coursera.org/course/matrix`. Moduly je možné nájsť aj na stránke `http://grading.codingthematrix.com/edition1`. Súčasťou tlačenej verzie je CD so všetkými programami.

- vec.txt
- mat.txt

Ďalej prikladáme programy na spustenie kvadratického sita:

- factoring_lab.txt
- factoring_support.txt
- quadratic_sieve.txt

Nakoniec vkladáme jednotlivé algoritmy globálnej optimalizácie. Na niektoré je potrebné rozšírenie jazyka Python pre matematické operácie – Numerical Python (Numpy):

- **náhodné prehľadávanie**: random_search.txt
- **PSO s penalizáciou**: PSO_penal.txt
- **PSO s reštrikciou**: PSO_restr.txt
- **spojitá verzia PSO**: PSO_spoj.txt
- **SOMA s penalizáciou**: SOMA_penal.txt

- **SOMA s reštrikciou**: SOMA_restr.txt
- **SOMA s resetovaním a penalizáciou**: SOMA_reset_penal.txt
- **SOMA s resetovaním a reštrikciou**: SOMA_reset_restr.txt
- **náhodné prehľadávanie 1D**: random_search_1D.txt
- **1D SOMA** vrátane funkcií s orezaním: SOMA_1D.txt
- **1D PSO** vrátane funkcií s orezaním: PSO_1D.txt

Tabuľky s výsledkami

Prikladáme tiež tabuľky s výsledkami jednotlivých 100 spustení algoritmov v súbore programu Excel:

- Results.xlsx