

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

METAHEURISTIKY V JAZYKU R
BAKALÁRSKA PRÁCA

2023
JANA VIKTÓRIA KOVÁČIKOVÁ

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

METAHEURISTIKY V JAZYKU R
BAKALÁRSKA PRÁCA

Študijný program: Dátová veda
Študijný odbor: Informatika a Matematika
Školiace pracovisko: FMFI.KAMŠ - Katedra aplikovej matematiky a štatistiky
Školiteľ: doc. Mgr. Radoslav Harman, PhD.

Bratislava, 2023
Jana Viktória Kováčiková



50262626

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Jana Viktória Kováčiková
Študijný program: dátová veda (Medziodborové štúdium, bakalársky I. st., denná forma)
Študijné odbory: informatika
matematika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Metaheuristiky v jazyku R
Metaheuristics in R

Anotácia: Pod pojmom "metaheuristika" rozumieme triedu algoritmov na globálnu optimalizáciu zložitých mnohorozmerných funkcií. Väčšina metaheuristík je inšpirovaná prírodnými dejmi a zásadným spôsobom využíva náhodnosť. Typickými metaheuristikami sú simulované žihanie, genetické algoritmy a optimalizačné metódy založené na napodobňovaní správania sa spoločenstva živočíchov určitého druhu.

Cieľ: Prvým cieľom práce je urobiť stručný prehľad základnými typmi metaheuristík a ich implementácií v softvéri R. Druhým cieľom je identifikovať, ktoré z týchto metód sú najvhodnejšie pre riešenie istej triedy úloh týkajúcej sa problémov optimalizácie štatistického experimentu.

Vedúci: doc. Mgr. Radoslav Harman, PhD.

Katedra: FMFI.KAMŠ - Katedra aplikovanej matematiky a štatistiky

Vedúci katedry: prof. RNDr. Marek Fila, DrSc.

Dátum zadania: 27.09.2022

Dátum schválenia: 23.10.2022

doc. Mgr. Tomáš Vinař, PhD.

garant študijného programu

.....
študent

.....
vedúci práce

Pod'akovanie: Veľká vd'aka patrí môjmu školiteľovi doc. Mgr. Radoslavovi Harmannovi, PhD. za pravidelné konzultácie, vynikajúcu komunikáciu, cenné rady pri písaní práce, veľkú ochotu, trpežlivosť a podporu.

Abstrakt

Práca obsahuje stručný prehľad základných typov metaheuristik a ich implementácií v softvéri R. Pomocou vybraných metaheuristik, ktorými sú genetické algoritmy, differenciálna evolúcia, optimalizácia rojom častíc a umelá kolónia včiel, optimalizujeme viacrozmernú Rosenbrockovu funkciu a riešime dva problémy z istej triedy úloh z oblasti navrhovania štatistického experimentu, konkrétnejšie D-optimálny návrh pre logistickú regresiu a D-optimálny návrh pre Poissonovu regresiu. Práca zahŕňa porovnanie výkonu použitých metód na optimalizovaných problémoch. Ukazuje sa, že v kontexte zvolenej triedy úloh z oblasti optimálneho navrhovania experimentu by popri často používanej metóde optimalizácia rojom častíc mohla byť prekvapivo efektívna aj metaheuristika umelá kolónia včiel, ktorá sa v súčasnosti na optimalizáciu úloh z tejto oblasti typicky nepoužíva.

Kľúčové slová: metaheuristiky, jazyk R, optimálny návrh experimentu, umelá kolónia včiel

Abstract

In this work, we provide a brief overview of the basic types of metaheuristic algorithms and their implementations in the R software environment. We optimize the multidimensional Rosenbrock function, as well as two problems related to the field of optimal design of experiments. Specifically, we address a problem of finding D-optimal design for the logistic model, and another of finding D-optimal design for the Poisson model, using metaheuristic algorithms. These algorithms include genetic algorithms, differential evolution, particle swarm optimization, and artificial bee colony. We have included a comparative analysis of the performance of the selected methods on the optimized problems. It is noteworthy that – apart from particle swarm optimization, which is commonly used within the field of optimal experimental design – the method of artificial bee colony, albeit rarely employed for optimization in this area, demonstrates surprisingly efficient results.

Keywords: metaheuristic algorithms, R language, optimal design of experiments, artificial bee colony

Obsah

| | |
|---|-----------|
| Úvod | 1 |
| 1 Prehľad optimalizačných metód | 3 |
| 1.1 Optimalizácia | 3 |
| 1.2 Typy optimalizačných problémov | 4 |
| 1.3 Optimalizačné algoritmy | 4 |
| 1.3.1 Stochastické optimalizačné metódy | 5 |
| 1.3.1.1 Genetické algoritmy | 7 |
| 1.3.1.2 Diferenciálna evolúcia | 9 |
| 1.3.1.3 Optimalizácia rojom častíc | 10 |
| 1.3.1.4 Umelá kolónia včiel | 13 |
| 1.3.2 Deterministické optimalizačné metódy | 17 |
| 1.4 Úvod do optimálneho navrhovania experimentov | 19 |
| 2 Porovnanie výkonnosti vybraných optimalizačných metód v softvéri R | 25 |
| 2.1 Optimalizácia Rosenbrockovej funkcie | 28 |
| 2.1.1 Výsledky vybraných metód | 28 |
| 2.2 D-optimálny návrh experimentu pre logistickú regresiu | 32 |
| 2.3 D-optimálny návrh experimentu pre Poissonovu regresiu | 38 |
| Záver | 43 |
| Príloha A | 49 |

Zoznam obrázkov

| | | |
|------|--|----|
| 1.1 | Príklad rekombinácie v binárnych genetických algoritmoch | 8 |
| 1.2 | Príklad mutácie v binárnych genetických algoritmoch | 8 |
| 1.3 | Príklad mutácie v DE | 10 |
| 1.4 | Hviezdica susedných riešení v ABC algoritme | 17 |
| 1.5 | Príklad návrhu experimentu s dvoma faktormi, $N = 12$ | 20 |
| 2.1 | Porovnanie optimalizačných metód na 3D Rosenbrockovej funkcií . . . | 29 |
| 2.2 | Porovnanie optimalizačných metód na 10D Rosenbrockovej funkcií . . | 29 |
| 2.3 | Porovnanie optimalizačných metód na 20D Rosenbrockovej funkcií . . | 30 |
| 2.4 | Porovnanie optimalizačných metód na 50D Rosenbrockovej funkcií . . | 30 |
| 2.5 | Porovnanie optimalizačných metód na probléme A, veľkosť experimentu $N=6$ | 35 |
| 2.6 | Porovnanie optimalizačných metód na probléme A, veľkosť experimentu $N=24$ | 36 |
| 2.7 | Porovnanie optimalizačných metód na probléme A, veľkosť experimentu $N=96$ | 37 |
| 2.8 | Porovnanie optimalizačných metód na probléme B, veľkosť experimentu $N=10$ | 39 |
| 2.9 | Porovnanie optimalizačných metód na probléme B, veľkosť experimentu $N=50$ | 40 |
| 2.10 | Porovnanie optimalizačných metód na probléme B, veľkosť experimentu $N=100$ | 41 |
| 2.11 | Porovnanie optimalizačných metód na probléme B, veľkosť experimentu $N=200$ | 42 |

Zoznam tabuliek

| | |
|---|----|
| 2.1 Vybrané optimalizačné metódy podporujúce intervalové ohraničenia pre každú premennú | 26 |
|---|----|

Úvod

Metaheuristiky sú algoritmy slúžiace na nájdenie približne optimálneho riešenia pre optimalizačné úlohy so zložitými účelovými funkciami mnohých premenných. Často sa využívajú v praxi vďaka ich flexibilite a efektivite. Typickými metaheuristikami sú simulované žíhanie, genetické algoritmy a optimalizačné metódy založené na napodobňovaní správania sa spoločenstva živočíchov určitého druhu.

Prvým cieľom práce je spraviť prehľad základných typov metaheuristik a ich dostupnosti v knižniciach prostredia R. Druhým cieľom je identifikovať, ktoré z týchto metód sú najvhodnejšie na riešenie úloh typu D-optimálny návrh pre logistickú regresiu a D-optimálny návrh pre Poissonovu regresiu z oblasti optimálneho navrhovania štatistického experimentu.

Úlohy z oblasti optimálneho navrhovania štatistického experimentu, na ktoré sa v tejto práci sústredíme, sú zvyčajne problémy s mnohými premennými, častokrát navyše nevieme zaručiť analytické vlastnosti optimalizovanej funkcie. To je dôvodom, prečo je úlohy tohto typu výhodné optimalizovať pomocou metaheuristik. Identifikovanie vhodnej metaheuristiky by mohlo nielen urýchliť optimalizáciu týchto zložitých problémov, ale tiež skvalitniť výsledky v tejto oblasti. V súčasnosti sa na optimalizáciu daného typu problémov využívajú najmä metaheuristiky optimalizácia rojom častíc, diferenciálna evolúcia a genetické algoritmy.

V prvej kapitole práce summarizujeme prípisy známych metaheuristik, akými sú simulované žíhanie, genetické algoritmy, diferenciálna evolúcia, optimalizácia rojom častíc, a tiež princíp menej známej metaheuristiky umelá kolónia včiel. Taktiež uvádzame problematiku optimálneho navrhovania štatistického experimentu a špecifikujeme triedu úloh, na ktoré sa v tejto práci zameriavame.

V druhej kapitole porovnávame výkon vybraných stochastických metaheuristik a vybraných deterministických algoritmov. Medzi optimalizované problémy patria viac-rozumná Rosenbrockova funkcia a dva problémy z cieľovej triedy úloh z oblasti optimálneho navrhovania experimentu. Našou snahou je zistiť, ktorá z použitých metaheuristik je na optimalizáciu zvolenej triedy problémov najvýkonnejšia.

Kapitola 1

Prehľad optimalizačných metód

V tejto kapitole stručne vysvetlíme, čo je to optimalizácia, vysvetlíme pojemy metaheuristika a objasníme rozdiely medzi stochastickou a deterministickou optimalizáciou. Ďalej spravíme stručný prehľad základných typov stochastických a deterministických algoritmov, ako aj podrobnejší prehľad špecifických použitých metód, ktoré sa ukažujú byť efektívne na riešenie nami zvolených problémov - opíšeme, na akom princípe dané metódy fungujú, prípadne vysvetlíme, akými prírodnými javmi boli inšpirované. Dobre známe metódy vysvetlíme povrchnejšie, niektoré menej známe opíšeme podrobnejšie. Spomenieme aj základné informácie o existujúcich implementáciách vybraných optimalizačných metód v knižničiach prostredia R.

1.1 Optimalizácia

Pod pojmom optimalizácia rozumieme hľadanie „najlepšieho“ prvku istej množiny podľa určitého kritéria. Tento „najlepší“ prvak sa nazýva globálne optimum, respektíve optimálne riešenie. Množina (označme ju X), na ktorej hľadáme globálne optimum, sa nazýva množina prípustných riešení. Kritérium, ktoré určuje, ktoré prvky sú „najlepšie“, býva vyjadrené pomocou takzvanej účelovej funkcie (označme ju f). Typicky je X podmnožinou priestoru n -rozmerných reálnych vektorov \mathbb{R}^n a f je zobrazením z X do \mathbb{R} . Na garanciu existencie riešenia optimalizačného problému stačí, ak je splnená aspoň jedna z týchto základných podmienok:

- a) množina X prípustných riešení je konečná,
- b) množina X prípustných riešení je kompaktná a účelová funkcia f je spojitá na X .

Optimálne riešenie môže byť jedno, alebo ich môže byť viacero – či stačí nájsť jedno, alebo treba nájsť všetky, závisí od kontextu úlohy.

Typický optimalizačný problém hľadá extrém reálnej funkcie – globálne maximum alebo minimum. Maximalizačná úloha sa dá jednoducho preformulovať na minimalizačnú a naopak – všeobecne sa používa skôr úloha minimalizácie, formulovaná nasledovne:

$$\min_x f(x) \quad (1.1)$$

za podmienky $x \in X$.

Hľadáme teda také prípustné riešenie x^* , v ktorom je hodnota účelovej funkcie $f(x)$ minimálna. Čiže, hľadáme nejaký prvok množiny

$$\arg \min_{x \in X} f(x) = \{x^* \in X \mid f(x^*) \leq f(x) \forall x \in X\}.$$

1.2 Typy optimalizačných problémov

Ak má úloha tvar (1.1), pričom $\emptyset \neq X \subseteq \mathbb{R}^n$, $f : X \rightarrow \mathbb{R}$ a X je otvorená množina, potom ide o úlohu na voľný extrém (bližšie detaily sú napríklad v knihe [12]). Iným typom je úloha obsahujúca obmedzujúce podmienky (angl. *constraints*), pri ktorej je množina prípustných riešení X ohraničená a uzavretá, čiže kompaktná.

Podľa typu premenných sa optimalizácia rozdeľuje na diskrétnu a spojité. Pojmom konvexná optimalizácia sa označujú také prípady, kedy minimalizujeme konvexnú účelovú funkciu (alebo maximalizujeme konkávnu účelovú funkciu) nad konvexnou množinou X prípustných riešení.

V našej práci sa sústredíme na úlohy spojitej optimalizácie s intervalovými ohraňčeniami pre každú premennú (angl. *box constraints*). Na porovnávanie sme vybrali len tie procedúry nachádzajúce sa v knižničach pre prostredie R, ktoré tieto ohraňčenia umožňujú zadať ako vstupné parametre. Keďže budeme optimalizovať spojité funkcie na kompaktej množine, máme zaručené, že optimum existuje. Na druhej strane, optimalizačné problémy, na ktoré sa v tejto práci sústredíme, majú aj menej žiaducu vlastnosť, že sú nekonvexné, čiže pôjde o nekonvexnú optimalizáciu.

1.3 Optimalizačné algoritmy

Na hľadanie riešenia optimalizačných problémov sa využívajú rôzne algoritmy, ktoré môžu byť buď deterministické, alebo stochastické. Zatiaľ čo pri deterministickom prístupe je postup striktne definovaný a výsledné hodnoty sú pre rovnaké vstupné parametre zreprodukovaťelné, stochastický prístup využíva náhodnosť a rôzne spustenia stochastických algoritmov za rovnakých iniciálnych podmienok môžu produkovať rozdielne výsledky.

Pojmom metaheuristika v optimalizácii nazývame triedu algoritmov slúžiacich na nájdenie približne optimálneho riešenia pre optimalizačné úlohy so zložitými účelovými funkciami mnohých premenných. Toto slovo sa skladá z predpony *meta-*, ktorá zvykne označovať niečo, čo funguje na vyšej úrovni, a slova *heuristika* (z gr. *heuréka* = už to mám, objavil som), ktoré pomenúva praktickú stratégiu na riešenie problému na chádzajúcim akceptovateľné riešenia v reálnom čase aj pre komplexnejšie problémy, pri ktorých častokrát klasické metódy zlyhávajú ([40]). Metaheuristiky, alebo „heuristiky fungujúce na vyšej úrovni“, nepotrebujú o problémoch takmer nič predpokladať (na rozdiel od väčšiny exaktných metód, ktoré často vyžadujú diferencovateľnosť a iné analytické vlastnosti účelovej funkcie), bývajú tak aplikovateľné na širokú škálu problémov. Väčšina metaheuristik je inšpirovaná prírodnými dejmi a zásadným spôsobom využíva náhodnosť. Typickými metaheuristikami sú genetické algoritmy, simulované žíhanie, či optimalizačné metódy založené na napodobňovaní správania sa spoločenstva živočíchov určitého druhu. Niektoré z týchto metód popíšeme podrobnejšie neskôr.

Metaheuristiky sú nezriedka využívané v praxi vďaka ich flexibilite a efektivite. Pri reálnych problémoch v praxi je totiž častokrát výhodnejšie uspokojiť sa s „dostatočne dobrým“ riešením, ktorého funkčná hodnota je blízko tej v globálnom optime, než získať garantované skutočné globálne optimum, ktoré by mohlo mať za následok neakceptovateľne dlhé trvanie výpočtu. Ich využitie je teda obzvlášť výhodné na riešenie veľkých problémov, ale aj vtedy, keď nevieme určiť analytickej vlastnosti problému.

Je nemožné vybrať jeden optimalizačný algoritmus, ktorý by bol univerzálnie najlepší pre riešenie všetkých optimalizačných problémov. Algoritmus, ktorý produkuje najlepšie výsledky pri jednej konkrétnej úlohe, býva častokrát slabý v porovnaní s inými pri ďalšej, odlišnej úlohe (pozri [39]). Porovnávať efektivitu algoritmov má teda zmysel skôr v kontexte konkrétnej triedy optimalizačných úloh.

1.3.1 Stochastické optimalizačné metódy

Okrem náhodnosti kombinuje väčšina stochastických algoritmov dva základné princípy, ktorými sú globálne prehľadávanie (angl. *exploration*) a lokálne prehľadávanie (angl. *exploitation*). Globálne prehľadávanie slúži na prehľadanie rôznych častí priestoru prípustných riešení, zatiaľ čo lokálne prehľadávanie pomáha nájsť optimálne riešenie v okolí určitého prípustného riešenia.

Medzi metaheuristiky, ktoré sú zároveň stochastickými optimalizačnými metódami, patria napríklad genetické algoritmy, simulované žíhanie, diferenciálna evolúcia, optimalizácia rojom častíc, evolučná stratégia s adaptáciou kovariančnej matice, algoritmus umelej kolónie včiel, či optimalizácia kolóniou mravcov. Už priamo z názvov sa dá vytušiť, akými javmi boli inšpirované. Tieto metódy sa dajú ďalej klasifikovať na populačné a nepopulačné. Populačné metódy sa vyznačujú tým, že si po celý čas udržiavajú mno-

žinu kandidátskych riešení. Jedinci (kandidátske riešenia) súhrnne tvoria populáciu a sú vyhodnocovaní na základe „fitness“ funkcie, pričom v paralele so selekciou v prírode, preferovaní sú tí „najživotaschopnejší“ jedinci. Je dôležité, aby si populácia dostatočne dlho udržala diverzitu a vyhla sa tak predčasnej konvergencii k lokálnemu optimu. Dvomi dôležitými triedami populačných algoritmov sú evolučné algoritmy a algoritmy založené na kolektívnej inteligencii roja. Nepopulačné metódy si naopak udržiavajú iba jedno riešenie a ich cieľom je toto riešenie iteratívne vylepšovať ([4]).

Príkladom nepopulačnej metódy je simulované žíhanie, ktoré má svoj pôvod vo fyzike. Žíhanie je spôsob tepelného spracovania zliatin: Materiál sa najprv rozpáli na vysokú teplotu, následne sa ochladí na žihaciu teplotu, na ktorej sa podrží predpísaný čas, a nakoniec sa nechá vychladnúť. Týmto procesom vzniká kov s lepšími vlastnosťami. Algoritmus simulovaného žíhania napodobňuje tento proces. Podľa článku [21], potrebné sú tieto štyri zložky: vhodný popis konfigurácie (t.j. možného riešenia) systému, náhodný generátor susednej konfigurácie, účelová funkcia a žihací harmonogram (angl. *annealing schedule*) udávajúci, ako dlho má byť udržiavaná aká teplota. Na začiatku sa inicializuje počiatočná konfigurácia a vysoká teplota, a začne sa tento iteratívny proces: Zo súčasnej konfigurácie sa vygeneruje nová, susedná konfigurácia. Ak je hodnota účelovej funkcie novej konfigurácie lepšia ako hodnota tej súčasnej, nová konfigurácia sa akceptuje. V opačnom prípade sa akceptuje len s určitou pravdepodobnosťou (pre bližšie detaily pozri články [6, 21]). Teplota v algoritme slúži práve na reguláciu tejto pravdepodobnosti. Pri vysokej teplote je pravdepodobnosť akceptovania horšieho riešenia veľká, čo pomáha dobre prehľadať priestor riešení a nezaseknúť sa v lokálnom minime. S postupným znižovaním teploty sa algoritmus začne správať konzervatívnejšie. Existujú viaceré dôkazy konvergencie tohto algoritmu za určitých podmienok, napríklad v článku [11]. Použitie tejto optimalizačnej metódy je obzvlášť výhodné pri problémoch obsahujúcich veľa lokálnych extrémov. Dá sa využiť na optimalizáciu spojítých aj diskrétnych problémov.

Predbežne sme skúšali použiť implementáciu algoritmu simulovaného žíhania *optim_sa* z balíka *optimization* [15] prostredia R, ktorá umožňuje zadať intervalové ohraničenia. Keďže sa však čas výpočtu tejto metódy pomerne ľahko reguluje (je potrebné vhodne nastaviť kombináciu viacerých parametrov), a predbežné výsledky tejto metódy na nami zvolených problémoch sa neukazovali byť výrazne dobré, nakoniec sme sa rozhodli túto metódu z porovnania v kapitole 2 vynechať.

Robili sme predbežný prieskum efektivity mnohých metaheuristik pre nami zvolenú triedu úloh viažucu sa s optimálnym navrhovaním experimentu. Zvažovali sme tiež dostupnosť implementácie príslušnej metódy v prostredí R. V tomto prieskume sme ako pre nás zaujímavé identifikovali nasledovné metaheuristiky: genetické algoritmy (skr. GA z angl. *genetic algorithms*), diferenciálna evolúcia (skr. DE z angl. *differential evo-*

lution), optimalizácia rojom častíc (skr. PSO z angl. *particle swarm optimization*), umelá kolónia včiel (skr. ABC z angl. *artificial bee colony*). V ďalšom stručnejšie po- píšeme známejšie metódy GA, DE a PSO a podrobnejšie sa budeme venovať menej známej metóde ABC, ktorá sa ukázala ako prekvapivo efektívna pri riešení nami zvole- ných problémov z optimálneho navrhovania experimentov, najmä ak sú tieto problémy väčšieho rozsahu.

1.3.1.1 Genetické algoritmy

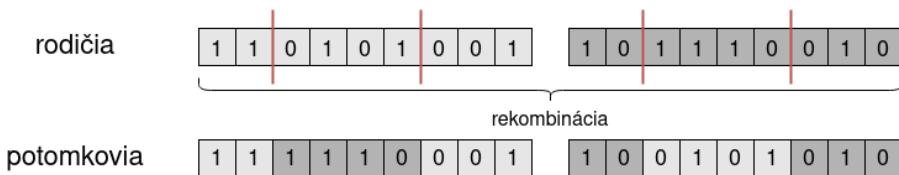
Genetické algoritmy sú abstrakciou evolúcie a prirodzenej selekcie biologických systé- mov. Prípustné riešenia sú nazývané chromozómy, a tradične bývajú reprezentované binárne ako postupnosť 0 a 1 (pomocou bitových polí/reťazcov), ale aj iné typy kó- dovania sú možné. Na chromozómy sú aplikované operácie ako kríženie (nazývané aj rekombinácia - výmena genetickej informácie medzi chromozómami), mutácia a selek- cia. Ako pri biologickej evolúcii, aplikovaný je princíp prežitia najschopnejších (angl. *survival of the fittest*). Účelová funkcia sa v kontexte genetických algoritmov nazýva fit- ness funkcia ([14]). Chromozómy tvoria populáciu, v rámci každej iterácie sa populácia nazýva novou generáciou. V nasledujúcom texte vysvetlíme základné kroky algoritmu, ale opíšeme operátory rekombinácie a mutácie len pre prípad binárnych genetických algoritmov (v ktorých sú prípustné riešenia kódované binárnym vektorom).

Evolúcia začne z iniciálnej populácie (zvyčajne náhodne vygenerovaných chromo- zómov) a aplikuje sa iteratívny proces:

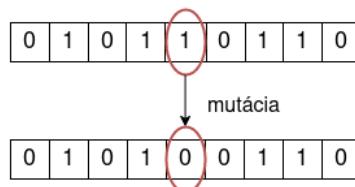
1. Výber rodičovských chromozómov: Z populácie sa vyberú chromozómy, ktoré sa budú krížiť. Tento výber môže prebiehať rôznymi spôsobmi: rovnomerne ná- hodne, proporcionalne (chromozómom s vyššou fitness hodnotou je priradená väčšia pravdepodobnosť, že budú zvolené), princípm turnajového výberu (po- rovnávajú sa dvojice chromozómov a víťaz turnaja bude zvolený), a.i. (viac v knihe [14]). Zvyčajne sa volí taký spôsob, pri ktorom majú väčšiu pravdepodob- nosť kríženia chromozómy s vyššou fitness hodnotou.
2. Rekombinácia: Prebehne kríženie dvojíc rodičovských chromozómov a vznikne množina potomkov. V závislosti od implementácie môže kríženie nastať na jed- nom alebo viacerých náhodných miestach, a každá dvojica rodičovských chromo- zómov typicky zanechá jedného alebo dvoch potomkov. Na obrázku 1.1 možno vidieť konkrétny príklad kríženia na dvoch miestach, z ktorého vzniknú dvaja potomkovia.
3. Mutácia: Potomkovia prejdú mutáciou, to znamená, že každý bit chromozómu sa so stanovenou pravdepodobnosťou obráti na opačný. Na obrázku 1.2 možno vidieť konkrétny príklad mutácie chromozómu na piatom bite.

4. Evaluácia: Vyhodnotí sa „životaschopnosť“ (hodnota fitness funkcie) všetkých chromozómov súčasnej generácie.
5. Selekcia: Vyberú sa najživotaschopnejšie chromozómy, ktoré budú tvoriť nasledovnú generáciu.

Iteratívny proces sa opakuje, až kým nie je dosiahnuté zastavovacie kritérium, napríklad maximálny povolený počet iterácií alebo určitá hodnota fitness funkcie. V závislosti od konkrétnej implementácie sa môžu kroky mierne lísiť.



Obr. 1.1: Príklad rekombinácie v binárnych genetických algoritmoch



Obr. 1.2: Príklad mutácie v binárnych genetických algoritmoch

Genetický algoritmus je citlivý na výber hodnôt parametrov. Veľkosť populácie, spôsob kríženia a pravdepodobnosť mutácie - to všetko sú hyperparametre, ktoré treba vhodne vybrať, inak genetický algoritmus nemusí konvergovať ku globálnemu optimu ([40]). Vo väčšine implementácií je veľkosť populácie fixná pre všetky generácie. Príliš malá populácia predstavuje riziko, že niektorý chromozóm a jeho potomkovia sa budú krížiť tak často, že prehlušia genetickú informáciu ostatných chromozómov, a algoritmus predčasne skonverguje do lokálneho optima ([40]). Príliš veľká populácia zase spôsobí výpočtovú náročnosť algoritmu, pretože v každej iterácii sa vyhodnocuje fitness hodnota všetkých chromozómov populácie. Pravdepodobnosť mutácie by sa mala opatrne zvoliť na nízku hodnotu, príliš veľa mutácií totiž stáruje konvergenciu systému, alebo dokonca posiela systém k zlým riešeniam.

Jednou z výhod genetických algoritmov je, že chromozómy v populácii sa správajú ako nezávislé agenti, populácia môže teda prehľadávať viacero smerov priestoru riešení simultánne a algoritmus sa dá implementačne paraleлизovať ([14]).

Existuje aj reálna verzia GA, kde vektory reprezentujúce chromozómy neobsahujú binárne hodnoty, ale reálne čísla ([14]). V tomto prípade sa pri rekombinácii môže

využiť napríklad konvexná kombinácia alebo extrapolácia komponentov rodičovských chromozómov a mutácia prebieha za pomoci náhodného generátora.

V prostredí R je dostupná knižnica *GA* [33], ktorá poskytuje rôzne funkcie na optimalizáciu pomocou genetických algoritmov. Balík zahŕňa implementácie genetických algoritmov pre spojité aj diskrétné problémy. V práci využívame metódu *ga* z tohto balíka, ktorá na vstupe očakáva maximalizačný problém zadaný pomocou fitness funkcie, dolného a horného ohraničenia a je potrebné upresniť, či ide o spojity alebo diskrétny problém. Metóda umožňuje zadať ďalšie parametre ako napríklad veľkosť populácie, R funkcie na selekciu, kríženie a mutáciu, veľkosť populácie, pravdepodobnosť kríženia, pravdepodobnosť mutácie a mnohé ďalšie. Predvolená veľkosť populácie je 50, pravdepodobnosť kríženia 0.8, pravdepodobnosť mutácie 0.1.

1.3.1.2 Diferenciálna evolúcia

Diferenciálna evolúcia je populačná metóda, ktorá podobne ako GA využíva operátory mutácie a kríženia, avšak pri tejto metóde sú definované iným spôsobom, založeným na vektorovej diferencii a posunoch. Prípustné riešenia v DE sú reprezentované m -rozmernými vektormi reálnych čísel. Hyperparametrami algoritmu sú veľkosť populácie n , diferenciálna váha F a pravdepodobnosť kríženia CR , a ich voľba zásadne ovplyvňuje výkonnosť optimalizácie. Diferenciálna váha F je reálny konštantný faktor z intervalu $[0, 2]$, ktorý kontroluje intenzitu diferenciálnej variácie. Mutácia a kríženie si vyžadujú 4 rôzne vektory z populácie - pre veľkosť populácie musí teda platiť $n \geq 4$ ([36]).

Algoritmus vychádza z iniciálnej populácie n vektorov a kým nie je splnené zastavovacie kritérium, opakuje sa iteračný proces, každou iteráciou produkujúc novú generáciu vektorov. V rámci každej iterácie sa pre $i = 1, 2, \dots, n$ zopakujú tieto kroky:

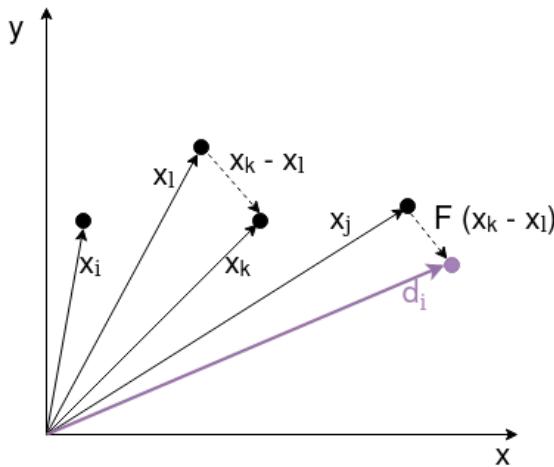
1. Mutácia: Vytvorí sa tzv. darcovský (angl. *donor, mutant*) vektor d_i ako kombinácia troch náhodne vybraných vektorov z aktuálnej generácie podľa vzorca ([36]):

$$d_i = x_j + F \cdot (x_k - x_l), \quad (1.2)$$

kde i, j, k, l sú 4 rôzne prirodzené čísla z množiny $\{1, 2, \dots, n\}$. Darcovský vektor môže byť chápáný ako mutácia vektora x_j pomocou diferencie vektorov x_k a x_l .

Príklad mutácie v DE je pre dvojrozmerný priestor riešení ilustrovaný na obrázku 1.3.

2. Kríženie: Z i -teho vektora x_i aktuálnej generácie a darcovského vektora d_i sa nižšie popísaným spôsobom vytvorí nový, tzv. vyzývateľský (angl. *challenger*) vektor v_i . Aby sa zabezpečilo, že sa mutácia určite prenesie do vyzývateľského vektora, najprv sa náhodne vyberie jedna pozícia j_d , ktorej hodnotu vyzývateľský



Obr. 1.3: Príklad mutácie v DE

vektor automaticky prevezme z darcovského vektora. Zvyšné zložky vyzývateľského vektora s pravdepodobnosťou CR zdedia hodnotu z darcovského vektora d_i , s doplnkovou pravdepodobnosťou $(1 - CR)$ zdedia hodnotu z vektora x_i . To znamená, ak je r_j číslo vygenerované rovnomerne náhodne na intervale $[0, 1]$ a j_d náhodne zvolené číslo z množiny $\{1, \dots, m\}$, potom j -ta zložka vyzývateľského vektora v_i je definovaná nasledovne ([36]):

$$v_{ij} = \begin{cases} d_{ij}, & \text{ak } r_j \leq CR \text{ alebo } j = j_m, \\ x_{ij}, & \text{inak.} \end{cases} \quad (1.3)$$

3. Selekcia: Prebehne „súboj“ vektora x_i aktuálnej populácie a vyzývateľského vektora v_i - vypočíta a porovná sa hodnota účelovej funkcie oboch vektorov. Do novej generácie postupuje vektor s lepšou funkčnou hodnotou (t.j. menšou funkčnou hodnotou pri minimalizačnej úlohe), čiže algoritmus DE sa rozhoduje pažravo.

Za riešenie sa považuje najlepší člen poslednej generácie.

V prostredí R je na diferenciálne evolúciu zameraný balík *DEoptim* [23], ktorý pre väčšiu rýchlosť využíva rozhranie C. Z tohto balíka využívame rovnomennú metódu *DEoptim*, do ktorej vstupujú povinné parametre účelová funkcia, dolné a horné ohraňičenie. Ďalej sú prednastavené parametre pravdepodobnosť kríženia 0.5, diferenciálna váha 0.8 a veľkosť populácie je 10-krát dĺžka zadaného vektora dolného ohraňičenia.

1.3.1.3 Optimalizácia rojom častíc

Optimalizácia rojom častíc je metóda inšpirovaná kolektívou inteligenciou roja, napríklad húfu rýb, či kŕdľa vtákov. Hlavnou výhodou kolektívu je, že jedinci okrem svojej vlastnej skúsenosti môžu ľažiť aj zo skúsenosti iných, vďaka čomu je kolektív schopný riešiť omnoho zložitejšie úlohy ako jedinec sám ([4]).

Algoritmus PSO si udržiava populáciu riešení, nazývanú roj, zadanej veľkosti n . Roj sa skladá z častíc a je umiestnený v m -rozmernom priestore prípustných riešení, v rámci ktorého sa pohybuje s cieľom nájsť globálne optimum. Aktuálne polohy častíc zodpovedajú jednotlivým riešeniam. Každá častica si okrem svojej aktuálnej polohy pamäta aj svoju historicky najlepšiu polohu a aktuálnu rýchlosť. Tieto tri vlastnosti častic sa reprezentujú m -rozmernými vektormi x_i , p_i a v_i .

Interakcie v rámci roja prebiehajú na základe štruktúry vzájomných vzťahov, respektíve známostí častíc. Táto štruktúra sa nazýva topológia roja, predstavuje sociálnu sieť a dá sa reprezentovať grafom, v ktorom vrcholy predstavujú časticu a hrany ich vzájomné známosti. Množinu známostí častic nazývame jej okolím. Interakcia prebieha tým spôsobom, že častica so svojím okolím zdieľa informáciu o svojej doposiaľ najlepšej nájdenej polohe a čerpá informáciu o tom, aká je najlepšia poloha nájdená okolitými časticami. Pri vzniku PSO autori používali topológiu založenú na euklidovských vzdialenosťach častíc, avšak využitie tejto topológie sa ukázalo ako výpočtovo náročné a dokonca malo nežiadúce konvergenčné správanie ([25]). Výber vhodnej topológie je pre výkon PSO mimoriadne dôležitý, no nič nenaznačuje, že by niektorá bola univerzálne najvhodnejšia ([25]). V súčasnosti sa často využívajú tieto statické topológie:

- úplne prepojená topológia „gbest“ (každý vrchol má $n - 1$ susedov),
- kružnicová topológia „lbest“ (každý vrchol má práve 2 susedov).

Výhodou „gbest“ prístupu je, že okolím každej z častíc je celý roj a stačí si preto pamätať jednu globálne najlepšiu polohu nájdenú v rámci celého roja, na rozdiel od „lbest“ prístupu, pri ktorom treba poznáť lokálne najlepšiu polohu nájdenú okolím každej častice. Na druhej strane, „lbest“ topológia umožňuje paraleлизáciu ([25]). PSO využívajúci „gbest“ topológiu konverguje rýchlejšie, PSO s „lbest“ topológiou podľa článku [25] zase lepšie prehľadáva priestor riešení.

V algoritme PSO je potrebné nastaviť tieto parametre: počet častíc (veľkosť populácie) n , koeficienty zrýchlenia ϕ_1 , ϕ_2 a váhu zotrvačnosti (angl. *inertia weight*) w . Pohyb roja v PSO prebieha tak, že sa v každej iterácii aktualizujú polohy a rýchlosťi všetkých častíc. Nová poloha častice sa vypočíta ako súčasná poloha posunutá o vektor rýchlosťi, rýchlosť je teda v princípe veľkosťou kroku. Na výpočet rýchlosťi existujú rozličné predpisy, my si predstavíme základnú verziu. Rýchlosť častice je kombináciou týchto troch zložiek: zotrvačnosť (informácia o rýchlosti v predošej iterácii), kognitívna zložka (informácie o vlastnej histórii častice) a sociálna zložka (informácie o najlepších polohách častíc z okolia). Koeficienty zrýchlenia ϕ_1 , ϕ_2 určujú mieru vplyvu náhodných súčin na pohyb častice v smere najlepšej polohy častice a najlepšej polohy nájdenej jej okolím, typicky sa volia na intervale $[0, 4]$. Voľba príliš veľkých koeficientov zrýchlenia v kombinácii s určitými hodnotami váhy zotrvačnosti môže mať za následok nekontrolované zrýchľovanie pohybu častíc ([25]). Váha zotrvačnosti w je tlmiaci faktor, kontrolujúci veľkosť zotrvačnej rýchlosťi z predchádzajúcej iterácie. Vyššie hod-

noty w podporujú prehľadávanie rozličných častí priestoru, nižšie hodnoty umožňujú lokálne prehľadávanie. Preto môže byť výhodné použiť dynamickú váhu zotrvačnosti w , ktorá bude zo začiatku nastavená na relatívne vysokú hodnotu (napr. 0.9) a postupne sa bude znižovať na malú hodnotu (napr. 0.4), avšak dajú sa použiť aj iné stratégie ([25]).

Samotný algoritmus PSO je podľa článku [25] definovaný nasledovne: Na začiatku sa náhodne vygenerujú polohy častíc x_i^0 a iniciálne rýchlosť v_i^0 ($i = 1, \dots, n$). Zároveň sa inicializujú premenné p_i^0 a fp_i^0 predstavujúce doteraz najlepšiu polohu i -tej častice a hodnotu účelovej funkcie v tomto bode. V cykle cez $t = 0, 1, 2, \dots$ sa až po dosiahnutie zastavovacieho kritéria budú opakovať tieto kroky, v rámci každej iterácie t pre všetky častice $i = 1, \dots, n$:

1. Vypočíta sa hodnota fx_i účelovej funkcie v aktuálnom bode x_i^t . Ak je lepšia ako hodnota fp_i^t , tak sa hodnoty premenných doteraz najlepšej polohy a jej fitness hodnoty aktualizujú: $p_i^{t+1} = x_i^t$ a $fp_i^{t+1} = fx_i^t$. V opačnom prípade hodnoty týchto premenných zostávajú nezmenené, t.j. $p_i^{t+1} = p_i^t$ a $fp_i^{t+1} = fp_i^t$.
2. Identifikuje sa historicky najlepšia poloha l_i^t nájdená okolím častice i .
3. Vygeneruje sa m -rozmerný vektor u_{i1} čísel rozdelených rovnomerne náhodne na $[0, \phi_1]$ a vektor u_{i2} čísel rozdelených rovnomerne náhodne na $[0, \phi_2]$.
4. Rýchlosť sa aktualizuje podľa predpisu

$$v_i^{t+1} = wv_i^t + u_{i1} \odot (p_i^t - x_i^t) + u_{i2} \odot (l_i^t - x_i^t), \quad (1.4)$$

kde symbol \odot označuje násobenie po zložkách.

5. Poloha sa aktualizuje podľa predpisu

$$x_i^{t+1} = x_i^t + v_i^{t+1}. \quad (1.5)$$

Predstavili sme si základnú verziu algoritmu PSO. Existujú aj rôzne modifikácie tohto algoritmu, napríklad PSO s koeficientami ohraničenia (angl. *constriction coefficients*), hybridný PSO, či PSO využívajúci predátora a korist' ([4, 25]).

V prostredí R existujú balíky *psotim*, *psoptim* a *ppso* na optimalizáciu rojom častíc. V práci používame implementáciu *psoptim* z balíka *psotim* [2]. Predvolená veľkosť populácie je $\lfloor 10 + 2\sqrt{\text{length}(\text{par})} \rfloor$, pričom par je štartovací bod zadaný vo forme vektora.

1.3.1.4 Umelá kolónia včiel

Algoritmus umelej kolónie včiel, podobne ako metóda PSO, je založený na abstrakcii kolektívnej inteligencie roja. Presnejšie, simuluje správanie sa roja včiel pri hľadaní potravy. Je najmladším spomedzi metaheuristik, ktoré využívame v rámci porovnania v tejto práci – vznikol v roku 2005 ([19]). Keďže články [19, 18, 17, 20] opisujúce ABC algoritmus obširne vysvetľujú najmä prírodnú motiváciu, priblížime ju v našej práci tiež.

Kľúčovou vlastnosťou procesu hľadania zdrojov potravy kolóniou včiel medonosných je časová efektivita. Okrem toho, že je nevyhnutné stihnúť využiť kvetnaté plochy počas ich neraz krátkeho obdobia kvitnutia, roj včiel sa musí postarať o to, aby bohaté zdroje potravy objavil a vyčerpal skôr, než tak spravia konkurenčné roje včiel z iných úľov. Robotnice prehľadávajú okolie úľa a monitorujú rôzne vzdialené kvitnúce plochy. Informáciu o polohe a kvalite zdroja si odovzdávajú prostredníctvom včelieho tanca, ktorým mobilizujú ďalšie včely z úľa k návštive tohto zdroja s cieľom vyťažiť z neho čo najviac v krátkom čase. Čím vyšší je obsah nektáru a peľu v objavenom zdroji, tým silnejšie je včela stimulovaná k tancom. Zvyšné včely majú tendenciu nasledovať tie včely, ktoré tancujú najintenzívnejšie – roj teda v princípe selektívne využíva čo najlukratívnejšie zdroje potravy ([34]).

V algoritme umelej kolónie včiel sú pomyselnými zdrojmi potravy konkrétnie prípustné riešenia optimalizačnej úlohy a množstvo nektáru v zdroji zodpovedá fitness hodnote ([18]). Ku každému zdroju potravy je ďalej pridelená práve jedna zamestnaná včela (angl. *employed bee*), ktorá si pamätá polohu zdroja a náhodným spôsobom prezerá a vyhodnocuje kvalitu susedných zdrojov. Ak nájde v okolí aktuálneho zdroja kvalitnejší zdroj, zapamätá si jeho polohu a polohu pôvodného zdroja zabudne. Zamestnané včely odovzdajú informáciu o polohe a kvalite zdroja pozorovateľským včelám (angl. *onlooker bees*), ktoré sa následne rozhodujú, na ktoré z nájdených zdrojov sa presunú. Každá pozorovateľská včela si vyberá jeden z nájdených zdrojov, pričom pravdepodobnosť, s akou bude zdroj zvolený, je úmerná množstvu obsiahnutého nektáru. Rôzne pozorovateľské včely si môžu zvoliť aj ten istý zdroj, na ktorý sa presunú a následne vyhodnocujú kvalitu zdrojov v jeho okolí. Ak bol niektorý zdroj opakovane navštívený stanovený počet ráz a v jeho okolí sa nepodarilo nájsť kvalitnejší zdroj, zdroj sa považuje za vyčerpaný a bude opustený. V takom prípade je vyslaná skautská včela (angl. *scout bee*), ktorá si namiesto vyčerpaného zdroja náhodne vyberie ľubovoľný zdroj na ďalšie skúmanie. Je dôležité poznamenať, že v algoritme ABC sa historicky najlepšie nájdené riešenie udržiava v pamäti – nie je totiž zaručené, že opustený zdroj nie je globálnym optimom. V základnom modeli z článkov [19, 18] je počet zamestnaných včiel rovnaký ako počet pozorovateľských včiel a používa sa iba jedna skautská včela, to znamená, že počas jednej iterácie je opustený maximálne jeden zdroj potravy.

Algoritmus 1: pseudoalgoritmus ABC podľa článku [17]

```

1 Inicializácia populácie
2 while nie je dosiahnuté zastavovacie kritérium do
3   Umiestni zamestnané včely na ich zdroje potravy
4   Umiestni pozorovateľské včely na vybrané zdroje potravy v závislosti od
      množstva nektáru
5   Pošli skautské včely, aby objavili nové zdroje potravy
6   Zapamätaj si doteraz najlepší nájdený zdroj potravy
7 end

```

V nasledujúcom opíšeme jednotlivé kroky algoritmu ABC precíznejšie, bez použitia biologického kontextu. Keďže v článkoch [19, 17, 18, 20] nie je postup týmto spôsobom explicitne opísaný, opierame sa do istej miery o implementáciu ABC algoritmu v knižnici *ABCoptim* [37] prostredia R.

Do algoritmu vstupujú tieto hyperparametre: veľkosť populácie riešení n , maximálny počet cyklov *max_cycle* (zastavovacie kritérium) a maximálny povolený počet navštívení riešenia *limit*. Cyklus v ABC algoritme pomenúva jednu iteráciu. V implementácii ABC algoritmu v knižnici prostredia R [37] existuje ďalší parameter *criter*, ktorý predstavuje ďalšie zastavovacie kritérium – algoritmus vráti výsledok, ak sa najlepšie nájdené riešenie nezlepší po dobu *criter* cyklov. V popise základného ABC algoritmu ([18]) sa však hovorí iba o 3 parametroch a tento dodatočný parameter sa nespomína. My sme ho nastavovali na maximálnu možnú hodnotu *Machine\$integer.max*, aby nemal žiadny efekt na výpočet a aby bolo vždy použité zastavovacie kritérium maximálneho počtu cyklov *max_cycle*.

V inicializačnej fáze sa vygeneruje náhodná populácia zadanej veľkosti. Jednotlivé riešenia v populácii sú reprezentované m -rozmernými vektorami x_1, x_2, \dots, x_n , kde m je dimenzia priestoru riešení. Inicializuje sa n počítadiel navštívení (pre každé riešenie z populácie jedno) a všetky sa nastavia na nulu. Vytvoria sa premenné *best_solution* a *best_value* na udržiavanie historicky najlepšieho nájdeného riešenia a hodnoty účelovej funkcie v tomto bode, a uloží sa do nich najlepšie riešenie z iniciálnej populácie a jeho funkčná hodnota. Potom sa začne iteračný proces, ktorý sa opakuje po dosiahnutie zastavovacieho kritéria:

1. Pre každé riešenie x_i z aktuálnej populácie sa vygeneruje susedné riešenie s_i modifikáciou súčasného riešenia x_i . Susedné riešenie s_i vznikne tak, že sa prevezmú všetky zložky vektora x_i okrem jednej náhodne zvolenej zložky (označme jej index k). Táto k -ta zložka vektora s_i vznikne kombináciou k -tej zložky vektora x_i a k -tej zložky jedného iného náhodne vybraného vektora x_l z populácie. Presnejšie,

k -ta zložka vektora s_i vznikne posunutím k -tej zložky vektora x_i smerom ku k -tej zložke vektora x_l , alebo naopak smerom od k -tej zložky vektora x_l .

Formálne, ak x_i je i -ty vektor aktuálnej populácie ($i = 1, \dots, n$), k je náhodné číslo z množiny $\{1, \dots, m\}$, ϕ_i náhodné číslo z intervalu $[-1, 1]$ a x_l je náhodné riešenie z populácie ($l \neq i$), potom j -ta zložka susedného riešenia s_i je definovaná ako

$$s_{ij} = \begin{cases} x_{ij} + \phi_i(x_{ij} - x_{lj}), & \text{ak } j = k, \\ x_{ij}, & \text{inak.} \end{cases} \quad (1.6)$$

Môže sa však stať, že týmto spôsobom vznikne riešenie, ktoré nepatrí do množiny prípustných riešení – v takom prípade treba prestaviť s_{ik} na nejakú akceptovateľnú hodnotu, napríklad hranicu ([18]).

Následne sa vypočíta fitness hodnota riešenia s_i a porovná sa s fitness hodnotou riešenia x_i . Pri minimalizačnom probléme môže byť použitá napríklad takáto fitness funkcia¹ ([16]):

$$fit(x_i) = \begin{cases} \frac{1}{1+f(x_i)}, & \text{ak } f(x_i) \geq 0, \\ 1 + |f(x_i)|, & \text{ak } f(x_i) \leq 0, \end{cases} \quad (1.7)$$

kde $f(x_i)$ je hodnota účelovej funkcie v bode x_i . Uplatní sa pažravý prístup. To znamená, že ak je fitness hodnota riešenia s_i lepšia ako fitness hodnota riešenia x_i , riešenie s_i sa akceptuje ako nový člen populácie namiesto x_i a riešenie x_i sa zabudne. V opačnom prípade sa v populácii ponechá riešenie x_i . Ak bolo akceptované riešenie s_i , i -te počítadlo navštívení sa reštartuje. Inak sa hodnota i -teho počítadla zväčší o 1.

2. Vypočítajú a uložia sa pravdepodobnosti výberu pre všetky riešenia z populácie. Jednotlivé pravdepodobnosti sú proporcionálne vzhľadom na fitness hodnotu – riešeniam s lepšou fitness hodnotou je priradená väčšia pravdepodobnosť. Typický predpis pre i -tu pravdepodobnosť je podľa článku ([18])

$$p_i = \frac{fit(x_i)}{\sum_{j=1}^n fit(x_j)}. \quad (1.8)$$

3. Nasledujúca procedúra sa zopakuje n -krát:

Proporcionálnym výberom s návratom (s použitím pravdepodobností vypočítaných v kroku 2) sa vyberie jedno riešenie z populácie, označme jeho index v . K

¹Fitness funkcia sa neskôr používa na výpočet pravdepodobnosti. Z tohto dôvodu je neadekvátnie použiť v algoritme priamo účelovú funkciu ako fitness funkciu, keďže nie je garantované, že účelová funkcia, ktorú užívateľ zadá, bude nadobúdať iba nezáporné hodnoty. Namiesto toho sa volí vhodná transformácia účelovej funkcie.

tomuto v -temu riešeniu x_v , sa vygeneruje susedné riešenie s_v podľa vzorca (1.6), ďalej sa vyhodnotí fitness hodnota riešenia s_v a porovná sa s fitness hodnotou riešenia x_v a napokon sa pažravo vyberie lepšie z týchto dvoch riešení. To znamená, že ak je susedné riešenie s_v lepšie ako súčasné v -te riešenie v populácii x_v , súčasné riešenie sa zabudne a v -tym riešením v populácii bude odteraz riešenie s_v . Taktiež sa aktualizuje hodnota počítadla rovnako ako v kroku 1 – ak pôvodné riešenie x_v zostalo v populácii, počítadlo sa zvýši o 1, inak sa resetuje na nulu.

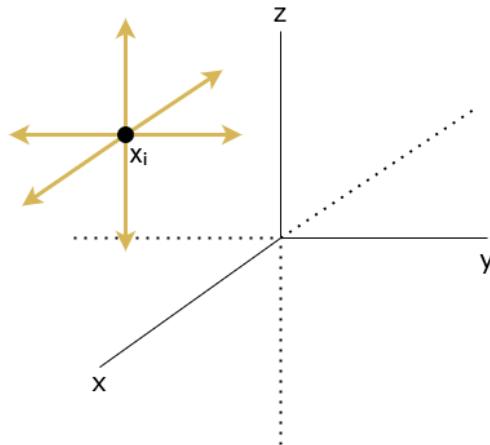
4. Ak je v aktuálnej populácii riešenie s lepšou hodnotou účelovej funkcie ako zapamätaná historicky najlepšia hodnota $best_value$, toto nové najlepšie riešenie a jeho funkčná hodnota nahradia staré zapamätané hodnoty $best_solution$ a $best_value$.
5. Zo všetkých počítadiel navštívení sa vyberie to s najväčšou hodnotou, označme jeho index k . Ak je hodnota tohto k -teho počítadla väčšia ako maximálny povolený počet navštívení riešenia (hodnota parametra $limit$), k -te riešenie z populácie bude nahradené novým náhodne vygenerovaným riešením z priestoru prípustných riešení a k -te počítadlo sa resetuje na nulu.

Po ukončení iteračného procesu sa ako výsledok optimalizácie vráti zapamätané historicky najlepšie riešenie $best_solution$ spolu s hodnotou účelovej funkcie v tomto bode $best_value$.

Algoritmus umeľej kolónie včiel riešenia z populácie iteratívne vylepšuje pomocou dvoch hlavných stratégií, ktorými sú prehľadávanie susedných riešení (lokálne prehľadávanie) a opustenie slabých riešení alebo takých riešení, ktorých okolie bolo dostatočne prehľadané, a ich následné nahradenie novými náhodnými riešeniami (globálne prehľadávanie). Ďalej si môžeme všimnúť, že v kroku 3 sa nahradzajú iba také riešenia, ktoré boli vybrané na základe ich pravdepodobnosti z kroku 2 – riešenia, ktoré vôbec neboli vybrané, sa v populácii ponechávajú napriek tomu, že môžu mať nízku fitness hodnotu, čo zabezpečuje variabilitu populácie. Iba ak dlhodobo nedôjde k zlepšeniu riešenia (teda k jeho nahradeniu lepším susedným riešením), dané riešenie bude zahodené.

Zamyslime sa nad geometriou prehľadávania priestoru riešení v tomto algoritme. Generovanie susedného riešenia podľa vzorca (1.6) je vlastne posunom riešenia x_i v smere jednej zo súradníc. Predstavme si trojrozmerný prípad, v ktorom je priestorom riešení \mathbb{R}^3 : vytvorenie susedného riešenia s_i k riešeniu $x_i = (x_{i1}, x_{i2}, x_{i3})^T$ zodpovedá zmene niektornej z jeho troch súradníc, čo je vlastne posunom riešenia x_i v smere alebo protismere niektornej z osí x, y, z . Susedné riešenie s_i sa teda vyberá na akejsi „hviezdič“ okolo riešenia x_i , ktorá je pre trojrozmerný prípad ilustrovaná na obrázku 1.4.

Náhodný index k určuje, po ktorej z týchto osí sa x_i posunie. To, aký maximálny rozsah bude mať tento posun, je udané k -tou súradnicou náhodne vybraného iného



Obr. 1.4: Hviezdica susedných riešení v ABC algoritme

vektora x_l z populácie. Ak sú riešenia x_i a x_l navzájom veľmi vzdialené v zmysle k -tej súradnicovej osi (ak je rozdiel $|x_{ik} - x_{lk}|$ veľký), maximálna možná dĺžka kroku (maximálna „mutácia“ riešenia x_i) bude veľká. Naopak, ak je rozdiel k -tych zložiek týchto riešení malý, aj maximálny možný posun bude malý. Skutočná dĺžka kroku je však ešte znáhodnená číslom ϕ_{ik} z intervalu $[-1, 1]$. Pri veľkej variabilite populácie sa dajú očakávať väčšie kroky a s postupným približovaním populácie k optimálnemu riešeniu sa dĺžka kroku adaptívne zmenšuje ([18]).

V softvéri R je algoritmus ABC implementovaný v knižnici *ABCOptim* [37]. V práci používame metódu *abc_optim* z tohto balíka. Predvolená je veľkosť populácie (počet zdrojov potravy) 20 a limit počtu navštívení riešenia je 50. Zvažovali sme tiež použitie metódy *abc_cpp*, ktorá využíva jazyk C++, avšak pri predbežnom teste metóda zlyhávala na chybe „*Location index is out of bounds*“.

1.3.2 Deterministické optimalizačné metódy

Deterministické metódy sa vyznačujú tým, že ich výsledky sú zreprodukovaťné ([40]). Postup výpočtu ako aj výstup je plne určený vstupujúcimi parametrami a podmienkami. Niektoré deterministické metódy využívajú analytické vlastnosti problému, prípadne vyžadujú splnenie rôznych predpokladov, ako napríklad spojitosť alebo diferencovateľnosť účelovej funkcie.

V tejto práci sa sústredíme najmä na metódy stochastickej optimalizácie, no chceli sme urobiť aj porovnanie s deterministickými metódami. Preto vybrané deterministické algoritmy spomenieme len veľmi stručne, s odkazom na literatúru. Do porovania sme sa snažili vybrať také algoritmy, ktoré nepoužívajú derivácie (s výnimkou algoritmu L-BFGS-B, ktorý je modifikáciou známej metódy BFGS). Vzhľadom na to, že v úlohách optimálneho navrhovania experimentov, na ktoré sa v tejto práci sústredíme, často nevieme zaručiť analytické vlastnosti optimalizovanej funkcie, je vhodné použiť také

metódy, ktoré kladú čo najmenej požiadaviek na optimalizovaný problém.

Metóda BFGS s obmedzenou pamäťou (skr. L-BFGS-B z angl. *limited-memory BFGS*) je kvázineutronovská metóda, ktorá je založená na metóde gradientového prie- metu a využíva aproximáciu hesiánu účelovej funkcie pomocou BFGS matice s obme- dzenou pamäťou. Algoritmus nevyžaduje druhé derivácie, dá sa teda využiť na optimali- záciu problémov, pre ktoré je výpočet hesiánu nepraktický. Je vhodný na optimalizáciu problémov s veľkým počtom premenných n . Pamäťová zložitosť aproximácie hesiánu je $O(n)$ ([5]).

Subplexová metóda je údajne efektívnejším a robustnejším variantom známeho al- goritmu Nelder-Mead. Keďže dizertačná práca [31], ktorá je pôvodným zdrojom tejto metódy, je momentálne nedostupná, nevieme získať podrobnejšie teoretické informácie o tejto metóde. Napriek tomu, implementácia tejto metódy pre prostredie R sa ukazuje byť veľmi efektívna. Navyše, túto metódu môžeme považovať za efektívneho reprezen- tanta celej triedy často používaných metód typu Nelder-Mead, preto sme považovali za vhodné ju do porovnania zaradiť.

Ohraničená optimalizácia pomocou lineárnej aproximácie (skr. COBYLA z angl. *constrained optimization by linear approximation*) je metóda, ktorá používa lineárne approximácie účelovej a obmedzujúcej funkcie pomocou interpolácie vo vrcholoch sim- plexu. Jednou z výhod tejto metódy je, že nepoužíva derivácie. Na druhej strane, výpočet lineárnych approximácií môže byť veľmi neefektívny, a to najmä pri veľkom počte premenných ([26]).

Viazaná optimalizácia pomocou kvadratickej approximácie (skr. BOBYQA z angl. *bound optimization by quadratic approximation*) je metóda, ktorá má rovnakého autora ako metóda COBYLA a využíva podobný princíp. Rovnako ako metóda COBYLA, ani BOBYQA nevyžaduje derivácie účelovej funkcie a používa approximácie, ale interpolácia v metóde BOBYQA je kvadratická ([27]).

Spomenuli sme hlavnú myšlienku každej z deterministických metód, ktoré sú zahr- nuté v porovnaní v kapitole 2. Keďže však deterministické metódy nie sú nosnou téhou tejto práce a teória, na ktorej sú tieto metódy založené, vyžaduje hlbšie porozumenie, odporúčame prečítať si priamo zdrojové články [5, 31, 26, 27].

V práci využívame implementáciu algoritmu *L-BFGS-B* zabudovaného v metóde *optim* zo základnej knižnice *stats* [29] prostredia R. Metódy *sbplx*, *cobyla* a *bobyqa* importujeme z knižnice *nloptr* [41] slúžiacej na nelineárnu optimalizáciu.

1.4 Úvod do optimálneho navrhovania experimentov

V tejto bakalárskej práci sa zameriavame na optimalizáciu funkcií z oblasti optimálneho navrhovania experimentov. Uvedieme preto stručný úvod do problematiky, ktorý sa opiera o poznatky z literatúry [1, 10, 24, 28].

Navrhovanie experimentov je dôležitým odvetvím matematickej štatistiky. Za základný pilier možno považovať monografiu [8] z roku 1935, ktorú napísal anglický štatistik Ronald Fisher. V práci sa rozoberajú všeobecné koncepty navrhovania experimentov, predovšetkým princípy randomizácie, zaslepenia, blokovania, kontroly placebom, a iné.

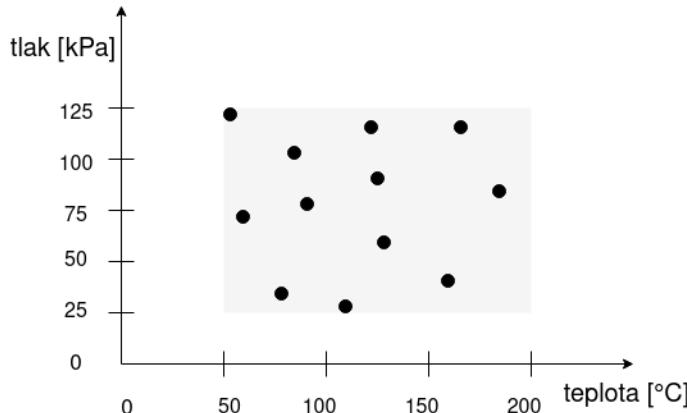
Podoblastou navrhovania experimentov je optimálne navrhovanie experimentov. Vysvetlíme exaktný návrh experimentu. Pod pojmom experiment sa rozumie postupnosť pokusov. Každý pokus je možné vykonať za určitých podmienok – podmienky pokusu sú obvykle kombináciou k faktorov. To znamená, pokus je určený k -rozmerným vektorom, respektíve bodom v k -rozmernom priestore. Množina X všetkých možných podmienok, za ktorých je možné vykonať pokus, sa nazýva priestor bodov návrhu. Typický priestor bodov návrhu je kartézsky súčin reálnych intervalov:

$$X = [a_1, b_1] \times \dots \times [a_k, b_k],$$

kde každý interval $[a_i, b_i]$ reprezentuje možný rozsah i -teho faktoru ($i = 1, \dots, k$).

Počet pokusov N , z ktorých experiment pozostáva, sa nazýva veľkosť experimentu. N môže byť určené napríklad časovým obmedzením na experiment (každý pokus má určité trvanie, pokusy je nutné vykonávať sekvenčne a je predpísaná doba, za akú je nutné vykonať celý experiment) alebo finančným obmedzením (na každý pokus je nutné vynaložiť finančné náklady a experimentu je pridelený istý rozpočet). Štandardne je však počet pokusov N zadaný vopred.

Uvedieme ilustratívny príklad. Majme experiment, v ktorom budeme opakovane N -krát vykonávať istú chemickú reakciu. Dá sa kontrolovať teplota a tlak, za akých sa bude reakcia vykonávať, pričom je predpísaný rozsah, v ktorom sa môžu hodnoty pohybovať – teplota medzi 50 a 200°C, tlak medzi 25 a 125 kPa. Pokusom je teda chemická reakcia a nastavujú sa úrovne 2 faktorov. Priestor bodov návrhu je v tomto prípade $X = [50, 200] \times [25, 125]$, čo je obdĺžnik v rovine. Možné voľby pokusov sú všetky body ležiace v obdĺžniku X . Úlohou je navoliť kombinácie hodnôt faktorov pre N pokusov, čo geometricky zodpovedá zvoleniu N bodov v obdĺžniku X . Príklad možného návrhu pre $N = 12$ je ilustrovaný na obrázku 1.5.



Obr. 1.5: Príklad návrhu experimentu s dvoma faktormi, $N = 12$

Experiment ξ je formálne definovaný ako postupnosť

$$\xi := (x_1, \dots, x_N), \quad x_1, \dots, x_N \in X.$$

Aj keď je experiment definovaný ako postupnosť pokusov (bodov x_i v priestore), obvykle na poradí vykonania pokusov nezáleží, zvlášť ak sú jednotlivé pokusy navzájom nezávislé.

Inou reprezentáciou experimentu je matica $N \times k$, v ktorej každý riadok zodpovedá jednému pokusu:

$$\xi = \begin{pmatrix} -x_1^T- \\ \vdots \\ -x_N^T- \end{pmatrix}, \quad x_1, \dots, x_N \in X.$$

Kedže optimalizačné metódy bývajú spravidla vyvinuté tak, že podporujú len riešenia v tvare vektorov (a nie postupnosti vektorov ani matice), pre účely optimalizácie je vhodné postupnosť vektorizovať na (Nk) -rozmerný vektor tak, že sa jednoducho stĺpcové vektorov x_i umiestnia pod seba:

$$\xi = \begin{pmatrix} x_1 \\ \vdots \\ x_N \end{pmatrix}.$$

Pri úlohe optimálneho navrhovania experimentu je cieľom vhodným spôsobom zoštaviť experiment, čiže vybrať čo najvhodnejšiu kombináciu N pokusov z priestoru bodov návrhu. Kvalita návrhu experimentu sa vyhodnocuje pomocou takzvaného kritéria optimality ϕ , čo je funkcia, ktorá každému návrhu priraďuje nejaké reálne číslo:

$$\phi : \Xi_N \rightarrow \mathbb{R},$$

kde Ξ_N označuje množinu všetkých návrhov experimentu veľkosti N . V súlade s uvedenými alternatívnymi definíciami návrhu experimentu, Ξ_N je možné formalizovať ako

množinu všetkých postupností N bodov množiny X , alebo ako množinu všetkých matíc rozmeru $N \times k$, ktorých každý riadok je bod z X , alebo ako množinu všetkých Nk -rozmerných reálnych vektorov vzniknutých vektorizáciou takýchto matíc. Formalizáciu volíme tak, aby sa najlepšie hodila pre aktuálne potreby. Napríklad, ak na hľadanie vhodného experimentu používame štandardné optimalizačné procedúry, je najvhodnejšie použiť poslednú uvedenú formalizáciu. Otázkou zostáva, čo je to dobrý návrh, teda ako by mala funkcia ϕ vyzerať, čo objasníme nižšie.

Vysvetlíme si klasický prístup merania kvality návrhu. Výstup pokusov sa zvykne modelovať štatistickým modelom, napríklad lineárhou či logistickou regresiou. Každý pokus totiž generuje výsledok y_i , ktorý je náhodný a pochádza z nejakého rozdelenia π . Toto rozdelenie je závislé od bodu x_i , v ktorom sa pokus vykonáva, a od neznámeho parametra θ :

$$y_i \sim \pi(\theta, x_i).$$

V rámci experimentu sa vykoná nezávislých N pokusov, pričom pre rozdelenie $\pi(\theta, x_i)$ výsledku y_i pokusu je známe x_i , ktoré reprezentuje nami zvolený bod pokusu, avšak parameter θ je neznámy. Pri klasickom prístupe je cieľom navrhnúť experiment ξ tak, aby sa dal čo najpresnejšie odhadnúť parameter θ . Odhad prametra θ nás obvykle zaujíma preto, lebo jeho jednotlivé zložky majú často priamu interpretáciu. Vďaka presnejšiemu odhadu θ však vieme robiť aj presnejšiu predikciu výsledkov pokusov vykonaných za iných podmienok. Návrh je tým lepší, čím lepšie umožňuje odhadnúť parameter θ – optimalizuje sa množstvo informácie získanej z experimentu.

Uvedieme jednoduchý príklad. Máme experiment, v ktorom sme 10-krát podať pacientovi liečivo a následne odmerať krvný tlak. Sú vopred určené hranice a_1 (najnižšia zmysluplná dávka liečiva) a b_1 (najvyššia bezpečná dávka liečiva), pričom dávka podávaného liečiva v každom pokuse je jediný faktor, ktorý možno kontrolovať. Výsledok (krvný tlak pacienta) modelujeme na intervale $[a_1, b_1]$ lineárhou regresiou²:

$$y_i \sim ax_i + b + \epsilon_i, \quad i = 1, \dots, N.$$

kde pokus x_i je reálne číslo z intervalu $[a_1, b_1]$ zodpovedajúce veľkosti dávky podanej v i -tom pokuse, ϵ_i je normálne rozdelená chyba ($\epsilon_i \sim N(0, \sigma^2)$) a a, b sú parametre lineárnej regresie. Potom platí:

$$y_i \sim N(ax_i + b, \sigma^2)$$

²V reálnych aplikáciách sa na modelovanie reakcie organizmu na liečivo používajú nelineárne modely. Tento zjednodušený model používame len pre ilustráciu. Aj tento model by však bol približne správny, ak by dĺžka intervalu $[a_1, b_1]$ bola malá.

a neznámy parameter θ je v tomto prípade

$$\theta = \begin{pmatrix} a \\ b \end{pmatrix}.$$

Z pozorovaní chceme odhadnúť, ako sa mení tlak v závislosti od dávky. Pritom hodnoty a, b je možné interpretovať, takže môže byť pre nás zaujímavý ich presný odhad (a znamená nárast tlaku na jednotku dávky liečiva a b by sme mohli interpretovať ako placebo efekt). Pri lineárnej regresii sa dajú parametre a, b odhadnúť na základe nameraných dát. Pri navrhovaní experimentu však dáta namerané ešte nemáme, máme aktívnu rolu – našou úlohou je práve navrhnúť, v ktorých bodech treba dáta namerať tak, aby sme parameter $\theta = (a, b)^T$ vedeli odhadnúť čo najlepšie. Na základe bežných postupov by v prípade tejto lineárnej regresie bol ideálny taký experiment, v ktorom by sa vykonal po 5 pokusov na oboch hraniciach intervalu, teda experiment

$$\xi = (x_1, \dots, x_{10}), \quad x_1 = x_2 = \dots = x_5 = a_1, x_6 = x_7 = \dots = x_{10} = b_1.$$

Vo všeobecnosti je postup nasledovný: Na základe toho, aký model je použitý na modelovanie výstupu pokusov, spočítame Fisherovu informačnú maticu pre parameter θ , ktorá závisí od x_1, \dots, x_N . Fisherova informačná matica (FIM) nesie informáciu o miere odhadnuteľnosti parametra θ a vzorec na jej výpočet pre jednotlivé modely sa dá dohľadať v literatúre. Ak je parameter θ m -rozmerný vektor, FIM bude mať rozmer $m \times m$. Želaný výsledok výpočtu je ten, aby FIM vyšla v aditívnom tvare:

$$M(x_1, \dots, x_N) = \sigma^2 \cdot \sum_{i=1}^N f(x_i) f^T(x_i), \quad (1.9)$$

kde σ^2 je konštantný reálny parameter a $f : X \rightarrow \mathbb{R}^m$ je známe zobrazenie určené použitým modelom. Konkrétna funkcia f pre jednotlivé modely sa dá tiež dohľadať v literatúre, napríklad pri lineárnej regresii použitej v príklade vyššie $f(x) = \begin{pmatrix} 1 \\ x \end{pmatrix}$. V nasledujúcom uvedieme dva ďalšie príklady, ktoré sa v našej práci budú vyskytovať aj neskôr v kapitole 2.

Model A. Uvažujme model logistickej regresie, v ktorej je $X = [-1, 1] \times [-1, 1]$ a pozorovanie y_i má Bernoulliho rozdelenie s parametrom $g^{-1}(h^T(x_i)\theta)$, kde

$$g(\eta) = \ln\left(\frac{\eta}{1-\eta}\right), \quad h(x) = (1, x_{(1)}, x_{(1)}^2, x_{(2)}, x_{(2)}^2, x_{(1)}x_{(2)})^T$$

a $x_{(1)}$ a $x_{(2)}$ sú komponenty vektora $x \in X$. Takýto model bol uvažovaný napríklad v článku [22]. Potom Fisherova informačná matica je približne³ rovná (1.9), kde

$$\sigma^2 = 1 \quad \text{a} \quad f(x) = \frac{\sqrt{e^{h^T(x)\theta_0}}}{1 + e^{h^T(x)\theta_0}} h(x), \quad (1.10)$$

pričom θ_0 je takzvaný nominálny parameter, vo vyššie uvedenom článku stanovený ako

$$\theta_0 = (-1, 2, 0.5, 2, 0.1, 0.01)^T.$$

Model B. Uvažujme model Poissonovej regresie, v ktorej je $X = [-1, 1] \times [-1, 1] \times [-1, 1]$ a pozorovanie y_i má Poissonovo rozdelenie s parametrom $g^{-1}(h^T(x_i)\theta)$, kde

$$g(\eta) = \ln(\eta), \quad h(x) = (1, x_{(1)}, x_{(1)}^2, x_{(2)}, x_{(2)}^2, x_{(3)}, x_{(3)}^2, x_{(1)}x_{(2)}, x_{(2)}x_{(3)}, x_{(1)}x_{(3)})^T$$

a $x_{(1)}$, $x_{(2)}$ a $x_{(3)}$ sú komponenty vektora $x \in X$. Tento model bol použitý napríklad v článku [7]. V tomto prípade je Fisherova informačná matica je približne rovná (1.8), kde opäť

$$\sigma^2 = 1 \quad \text{a} \quad f(x) = \sqrt{e^{h^T(x)\theta_0}} h(x), \quad (1.11)$$

pričom nominálny parameter bol v uvedenom článku stanovený na

$$\theta_0 = (0.5, -0.2, 0.5, -0.2, -0.1, 0.2, -0.1, 0.2, -0.1, 0.2)^T.$$

Ak vyjde FIM v aditívnom tvare (1.9), kvalitu návrhu budeme vyhodnocovať na základe tejto matice. Pre účel porovnania matíc však treba vybrať vhodné zobrazenie, ktoré každej matici priradí číselnú hodnotu. Existujú rôzne spôsoby, napríklad výhodnotenie FIM na základe jej najmenšej vlastnej hodnoty, alebo na základe stopy jej inverzie. Pri takzvanom D-optimálnom návrhu sa FIM výhodnocuje na základe jej determinantu, a hľadá sa experiment ξ určený bodmi x_1, \dots, x_N tak, aby sa maximizovala hodnota $\det(M(x_1, \dots, x_N))$. Pri D-optimálnom návrhu teda po dosadení do vzorca (1.9) hľadáme

$$\arg \max_{x_1, \dots, x_N \in X} \det \left(\sum_{i=1}^N f(x_i) f^T(x_i) \right), \quad (1.12)$$

čo zodpovedá maximalizácii objemu elipsoidu spoľahlivosti pre parameter θ ([28]).

³Použili sme slovo „približne“, pretože v nelineárnych modeloch, akým je model logistickej regresie, potrebujeme urobiť isté aproximácie, aby sme vedeli riešiť problém optimálneho návrhu prevodom informačnej matice na tvar (1.9). Prvou aproximáciou je to, že použijeme len takzvanú asymptotickú informačnú maticu. Druhou aproximáciou je to, že stanovíme takzvanú nominálnu hodnotu parametra, ktorá je „dostatočne blízko“ skutočnej, neznámej, hodnote parametra. Podrobnosti viď napríklad knihu [28].

Optimalizácia tejto funkcie Nk reálnych premenných je často výpočtovo náročná, v špeciálnych prípadoch sa dá vyriešiť analyticky. Uvedená účelová funkcia je zvyčajne nekonvexná. Z hľadiska numerickej zložitosti to môže byť NP-ťažký problém. Pri takzvanom approximativnom návrhu sa dá problém približne vyriešiť konvexnou relaxáciou. Problému approximativného návrhu sa však v tejto práci nevenujeme, čiže budeme riešiť priamo problém maximalizácie determinantu uvedený vyššie, vzhľadom na voľbu bodov x_1, \dots, x_N v množine X , a to pomocou heuristík opísaných v prechádzajúcej časti práce. Výsledným cieľom je zostaviť odporúčania, ktoré v R implementované procedúry sú vhodné na riešenie úloh optimálneho navrhovania experimentov.

Kapitola 2

Porovnanie výkonnosti vybraných optimalizačných metód v softvéri R

V tejto kapitole porovnáme výkon vybraných stochastických metaheuristik a deterministických algoritmov knižníc prostredia R na zvolených optimalizačných problémoch. Medzi optimalizované problémy patria Rosenbrockova funkcia rôznych dimenzií, problém optimálneho návrhu pre model A z časti 1.4 (D-optimálny návrh experimentu pre logistickú regresiu z článku [22]) a problém optimálneho návrhu pre model B z časti 1.4 (D-optimálny návrh experimentu pre Poissonovu regresiu z článku [7]).

Vyskúšali sme rôzne optimalizačné metódy z knižníc softvéru R, z ktorých sme následne spravili užší výber niekoľkých metód, ktoré podporujú intervalové ohraničenia pre každú premennú (angl. *box constraints*). Takýto výber metód je obzvlášť výhodný pre optimalizáciu ľahších úloh z oblasti navrhovania štatistického experimentu, pretože úlohy tohto typu spravidla bývajú ohraničené. Metódy, ktoré sme do základného porovnania vybrali, sú vypísané v tabuľke 2.1 spolu s informáciou o tom, z akej knižnice prostredia R pochádzajú a či podporujú zadanie štartovacieho bodu. Metódy, ktoré štartovací bod nepodporujú, sú v tomto prípade populárne algoritmy, ktoré si iniciálnu populáciu generujú automaticky z množiny zadanej pomocou intervalových ohraničení pre každú premennú.

Sledovali sme, ako blízko k optimálnemu riešeniu sú metódy schopné zájsť v stanovenom čase. Špecifické parametre každej optimalizačnej procedúry nechávame na vopred nastavených hodnotách, okrem počtu iterácií alebo evaluácií úcelovej funkcie a tolerancie pre zastavovacie pravidlo (podrobnejšie neskôr). Iniciálne riešenie, pokiaľ ho umožňuje procedúra zadať, vyberáme pri problémoch z oblasti navrhovania experimentu rovnomerne na množine všetkých prípustných riešení. Pri Rosenbrockovej funkcií iniciálne riešenie generujeme na mnohorozmernom intervale $[-5, 5] \times \dots \times [-5, 5]$. Zvolíme si orientačné časové body na logaritmickej škále (napr. 0.01s, 0.1s, 1s, 10s), pričom šírku tohto časového intervalu pre každú metódu nastavujeme zmysluplné tak,

| názov metódy | anglický názov | knižnica | metóda | typ | štartovací bod |
|--|--|----------|----------------|-----------------|----------------|
| Genetický algoritmus | Genetic algorithm | GA | ga | stochastická | nie |
| Diferenciálna evolúcia | Differential evolution | DEoptim | DEoptim | stochastická | nie |
| Umelá kolónia včiel | Artificial bee colony algorithm | ABCOptim | abc_optim | stochastická | áno |
| Optimalizácia rojom častíc | Particle swarm optimization | pso | psoptim | stochastická | áno |
| BFGS s obmedzenou pamäťou | Limited-memory BFGS | stats | optim L-BFGS-B | deterministická | áno |
| Subplexová metóda | Subplex algorithm | nloptr | sbplx | deterministická | áno |
| Ohraničená optimalizácia pomocou lineárnej aproximácie | Constrained optimization by linear approximation | nloptr | cobyla | deterministická | áno |
| Viazaná optimalizácia pomocou kvadratickej aproximácie | Bound optimization by quadratic approximation | nloptr | bobyqa | deterministická | áno |

Tabuľka 2.1: Vybrané optimalizačné metódy podporujúce intervalové ohraničenia pre každú premennú

aby bolo možné pozorovať, či a ako rýchlo metóda nachádza globálne optimum. Tieto časy teda predstavujú želanú dĺžku výpočtu. Keďže jednotlivé metódy neumožňujú nastaviť čas výpočtu ako parameter, je potrebné nájsť a nastaviť maximálny počet iterácií alebo evaluácií tak, aby metóda skončila približne po želanom čase. Je dôležité poznamenať, že pre rôzne začiatočné body procedúra za rovnakých nastavení parametrov skončí po rôznych časoch, čiže čas výpočtu sa dá nastaviť len veľmi približne. Taktiež niektoré metódy sa po nájdení riešenia vypnú a navýšenie maximálneho počtu iterácií alebo evaluácií už nemá na čas výpočtu vplyv, to znamená, že metódy často krát skončia skôr, než je želaný čas výpočtu. Aby sa metódy nevypli zbytočne skoro a aby produkovali čo možno najlepšie výsledky, nastavujeme zastavovacie kritériá jednotlivých metód na také hodnoty, aby mali čo najmenší efekt. Napríklad, nastavujeme relatívnu toleranciu na hodnotu 0 pri všetkých metódach, ktoré tento parameter podporujú (t.j. pri metódach z balíkov *DEoptim*, *pso*, *nloptr*)¹. Pri metóde *L-BFGS-B*

¹Metóda *cobyla* s nastavením *xtol_rel* = 0 pri veľkom počte evaluácií zlyhávala na chybu *Error in fn(unlist(x), ...)* : Assertion on 'x' failed: Contains missing values (element 1). Pokial sme však použili nastavenie *xtol_rel* = 10^{-15} , metóda vrátila výsledok bez problému, preto pri problémoch Rosenbrock a Problém A je pre zhluk bodov metódy *cobyla* so želaným časom 100 sekúnd použité toto nastavenie *xtol_rel* = 10^{-15} .

nastavujeme na 0 zastavovacie kritérium *factr*, čo je kritérium, ktoré prehlási konvergenciu, ak sa hodnota účelovej funkcie zníži o menej ako hodnotu tohto parametra. Pri metóde *abc_optim* nastavujeme zastavovacie kritérium *criter*, ktoré sme spomínali v časti 1.3.1.4., na maximálnu možnú hodnotu *.Machine\$integer.max*. Zastavovacie kritériá sú teda pri každom spustení v našom porovnaní fixne nastavené na tieto hodnoty. Do metód ďalej posielame parametre: účelová funkcia, dolné a horné intervalové ohraničenie, a do niektorých metód štartovací bod. Parameter, s ktorým pri porovnaní hýbeme, je:

- a) maximálny počet iterácií (pri metódach optim *L-BFGS-B*, *ga*, *DEoptim*, *psoptim*, *abc_optim*),
- b) maximálny počet evaluácií (pri metódach z balíka *nloptr*).

Kedže v niektorých prípadoch je nájdené iba lokálne minimum, a naviac, čas výpočtu môže oscilovať, jedno spustenie metódy z jedného štartovacieho bodu by nemuselo mať veľkú výpovednú hodnotu. Preto sme pre každý želaný čas spustili každú metódu viackrát (20-krát).

Aby sme výsledky nemuseli ukladať manuálne, napísali sme v jazyku R skript, ktorý výsledky spolu s dĺžkou výpočtu zapisuje do .csv súboru. Rôzne metódy vracajú výsledky v rôznej forme, prostredníctvom skriptu preto tiež upravujeme výsledky optimalizácie do jednotnej formy.

Výsledky sme vizualizovali. Pre každý zo želaných cieľových časov a pre každú metódu sa v grafe nachádza 20 zodpovedajúcich dátových bodov (vykreslené bodkami), okrem prípadov, kedy nie je možné dosiahnuť želaný čas (keď metóda vracia výsledok oveľa skôr aj za veľkého povoleného počtu evaluácií, alebo ak spustenie jednej iterácie/evaluácie trvá výrazne dlhšie ako želaný čas). Body vizuálne tvoria akési „zhluky“ a niektoré body sa navzájom prekrývajú. Pomocou trojuholníkových symbolov sú zobrazené body, ktoré zodpovedajú mediánu zo skutočných časov výpočtu a mediánovej nájdenej minimálnej hodnote v rámci každého zhluku. Tieto mediány zhlukov sú poprepájané čiarami, aby bolo možné sledovať, či a ako rýchlo sa s rastúcim povoleným časom výpočtu výsledky metód približujú ku globálnemu minimu.

Je potrebné spomenúť, že rýchlosť výpočtu nezávisí len od teoretickej efektivity metódy, konkrétnej implementácie v jazyku R a vhodnosti použitia metódy na konkrétny problém, ale taktiež od rôznych iných skrytých faktorov zahŕňajúcich napríklad typ procesora, spôsob ukladania do cache a poradie, v akom sa úlohy spúšťajú. Výpočty prezentované v tejto bakalárskej práci prebiehali na našom súkromnom počítači značky Acer, model Swift 3, s operačným systémom Ubuntu 20.04 LTS, s procesorom intel Core i5-8250U so 4 jadrami, s veľkosťou cache 6144 KB a s celkovou pamäťou RAM 8 GB.

2.1 Optimalizácia Rosenbrockovej funkcie

Rosenbrockova funkcia je nekonvexná matematická funkcia, ktorá je často využívaná na testovanie výkonnosti (angl. *performance testing*) optimalizačných metód. Nesie meno po Howardovi Harrymu Rosenbrockovi, anglickom odborníkovi na teóriu riadenia ([9]), ktorý túto funkciu použil vo svojom článku [30] v roku 1960. Funkcia je definovaná ako

$$f : \mathbb{R}^2 \rightarrow \mathbb{R}$$

$$f(x, y) = (a - x)^2 + b(y - x^2)^2,$$

pričom zvyčajne sa volia koeficienty $a = 1$ a $b = 100$. Táto dvojrozmerná funkcia má práve jedno globálne minimum ([35]) v bode $(x, y) = (a, a^2)$ s funkčnou hodnotou 0.

Existujú rozšírenia tejto funkcie do viacerých rozmerov ([35]). Pod viacozmernou Rosenbrockovou funkciou sa zvyčajne rozumie funkcia $f : \mathbb{R}^n \rightarrow \mathbb{R}$ s predpisom

$$f(x) = \sum_{i=1}^{n-1} 100 \cdot (x_{(i+1)} - x_{(i)}^2)^2 + (1 - x_{(i)})^2. \quad (2.1)$$

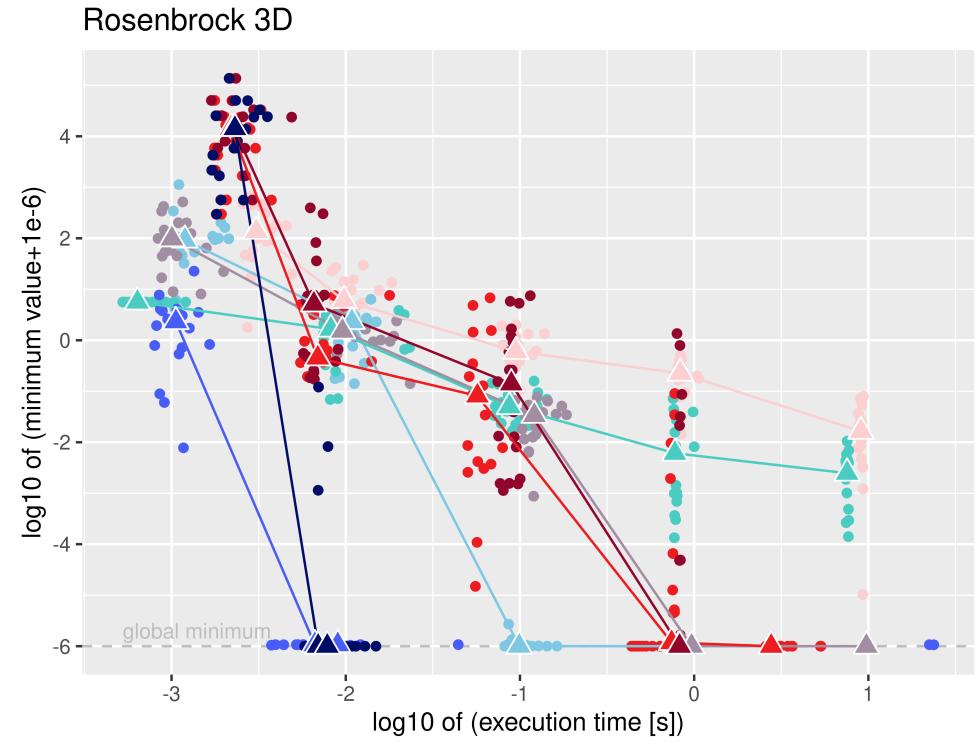
Táto funkcia má globálne minimum $f(x^*) = 0$ v bode $x^* = (1, \dots, 1)$.

V našej práci využívame implementáciu Rosenbrockovej funkcie *makeRosenbrockFunction* z knižnice *smoof* [3] softvéru R.

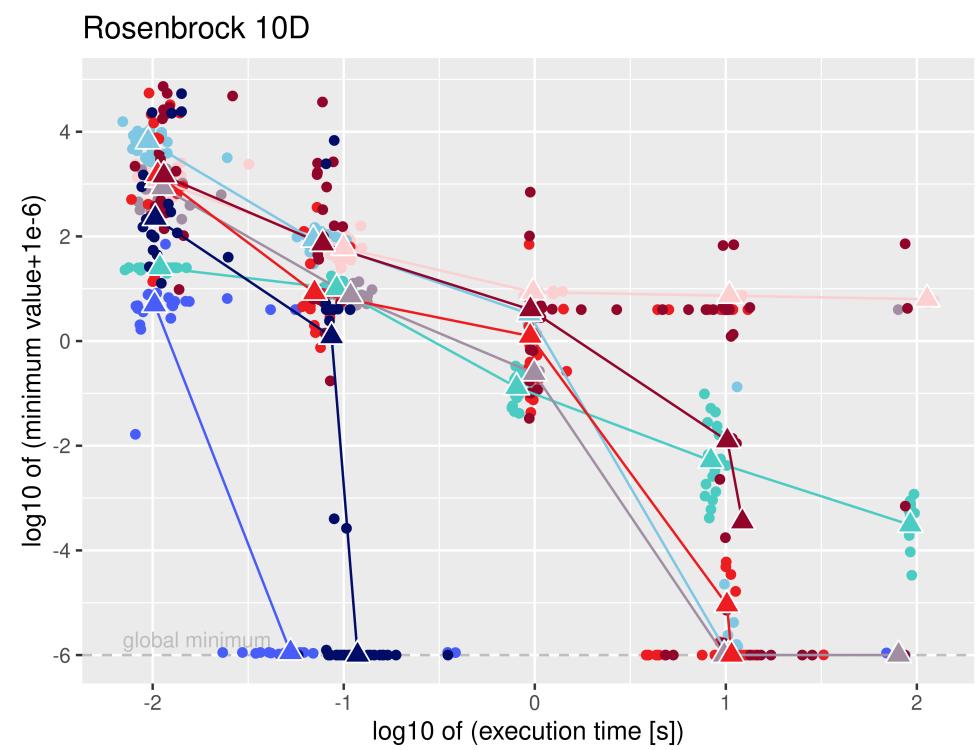
2.1.1 Výsledky vybraných metód

Pomocou metód vypísaných v tabuľke 2.1 sme optimalizovali Rosenbrockovu funkciu (2.1) dimenzií 3, 10, 20 a 50. Pri dimenzií 3 sme zvolili cieľové želané časy 0.001, 0.01, 0.1, 1 a 10 sekúnd, pri väčších dimenziách časy 0.01, 0.1, 1, 10 a 100 sekúnd. Pri metódach, do ktorých vstupuje ako parameter štartovací bod, sme štartovacie body generovali rovnomerne náhodne na mnohorozmernom intervale (angl. *box*) $[-5, -5, \dots], [5, 5, \dots]$. Riešenia sme hľadali na tom istom intervale, čiže pri všetkých metódach sme zvolili ako dolné ohraničenie bod $(-5, -5, \dots)$ a horné ohraničenie bod $(5, 5, \dots)$.

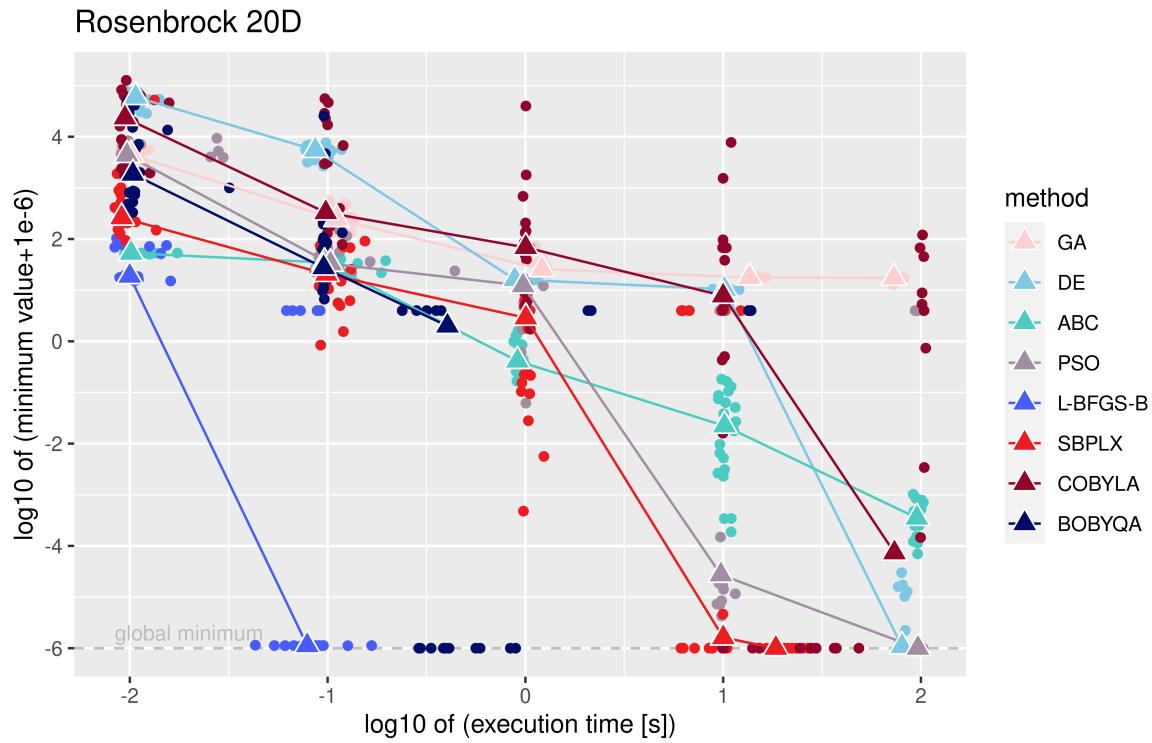
Výsledky optimalizácie možno vidieť na obrázkoch 2.1, 2.2, 2.3, 2.4. Na osi x je zobrazený čas výpočtu, na osi y sme zobrazili vypočítanú minimálnu hodnotu. Pre lepšiu vizualizáciu sme na obe osi aplikovali dekadický logaritmus. Skutočné globálne optimum Rosenbrockovej funkcie má funkčnú hodnotu 0, ale $\log_{10}(0)$ nie je definovaný. Aby sa teda logaritmus dal aplikovať na os y, pripočítali sme ku všetkým nájdeným minimálnym hodnotám konštantu 10^{-6} . V grafoch tak skutočnej minimálnej hodnote zodpovedá hodnota -6 . Ako bolo spomenuté vyššie, pomocou trojuholníkového symbolu zobrazujeme medián zo skutočných časov výpočtu a nájdených minimálnych hodnôt vrámcí každého zhluku bodov pre želaný čas výpočtu a danú optimalizačnú metódu.



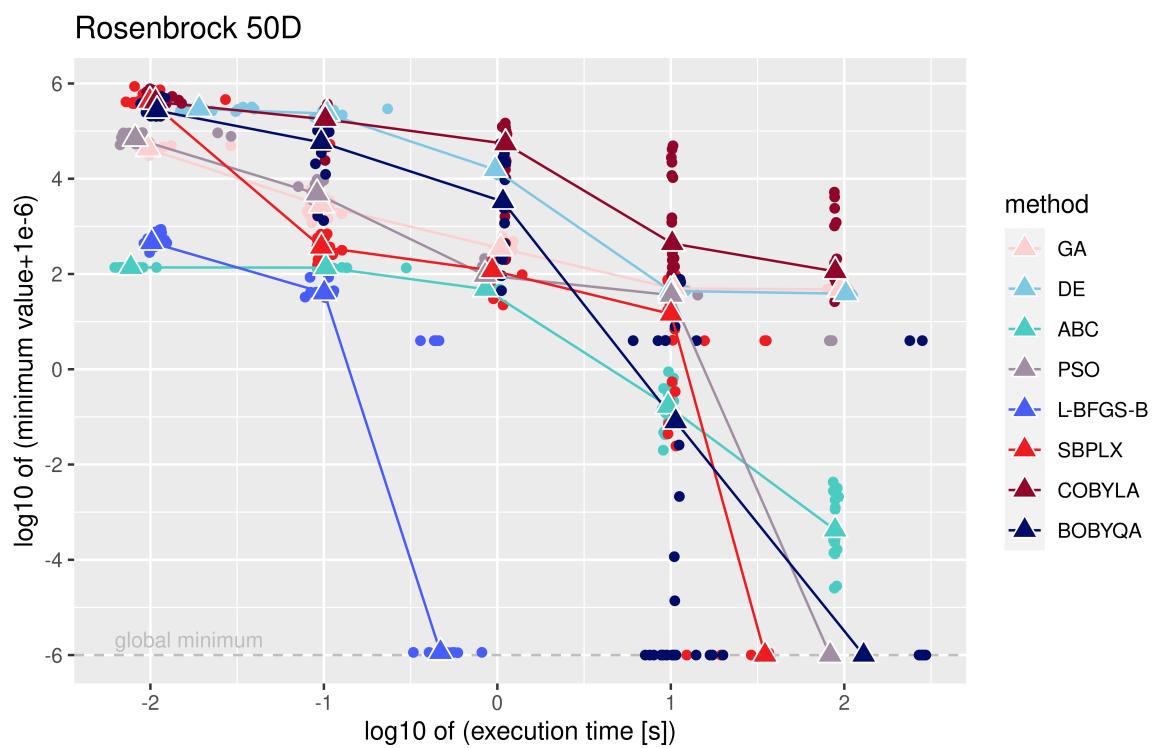
Obr. 2.1: Porovnanie optimalizačných metód na 3D Rosenbrockovej funkcií



Obr. 2.2: Porovnanie optimalizačných metód na 10D Rosenbrockovej funkcií



Obr. 2.3: Porovnanie optimalizačných metód na 20D Rosenbrockovej funkcií



Obr. 2.4: Porovnanie optimalizačných metód na 50D Rosenbrockovej funkcií

Z obrázkov 2.1, 2.2, 2.3, 2.4 je zrejmé, že Rosenbrockovu funkciu dimenzií 3, 10, 20 a 50 najrýchlejšie spomedzi vybraných metód optimalizuje metóda *optim* využívajúca algoritmus *L-BFGS-B*. Ďalej pomerne rýchle boli aj metódy *bobyqa*, *sbplx* a *psoptim*, ktoré dokázali mediánovo nájsť globálne minimum (resp. bod s funkčnou hodnotou veľmi blízkou globálnemu minimu) do 100 sekúnd aj pri 50-rozmernej Rosenbrockovej funkcií. Metódy *DEoptim* a *cobyla* nachádzali dobré riešenie pomerne rýchlo pri menších dimenziách, no pri dimenzii 20 už začali byť pomalšie a pri dimenzii 50 dokonca výrazne pomalšie oproti metódam spomenutým v predchádzajúcej vete. Metóda *abc_optim* bola druhá najpomalšia pri menších dimenziách 3 a 10, no s väčšou dimenziou sa rozdiely v jej rýchlosti oproti ostatným metódam začali stierať. Metóda *ga* bola pri všetkých dimenziách veľmi pomalá, pri všetkých dimenziách okrem dimenzie 50 dokonca signifikantne najpomalšia.

Na obrázkoch 2.2, 2.3, 2.4 si môžeme všimnúť, že niektoré z bodov vytvárajú akési horizontálne čiary. Napríklad, v grafe pre 20-rozmernú Rosenbrockovu funkciu 2.3 leží viacero bodov približne na priamke $y = 0.6$. Skutočná funkčná hodnota týchto bodov (bez aplikovania dekadického logaritmu) je približne 3.9866, a riešenie (nájdené „optimum“) je približne bod

$$(-0.9932681, 0.99665107, 0.99833032, 0.99916774, 0.99958520, 0.99979328, 0.99989698, 0.99994866, 0.99997441, 0.99998724, 0.99999363, 0.99999680, 0.99999835, 0.99999908, 0.99999933, 0.99999925, 0.99999879, 0.99999772, 0.99999550, 0.99999101),$$

čo je podľa článku [35] bod, v ktorom má 20-rozmerná Rosenbrockova funkcia lokálne minimum. Podobné lokálne minimum má viacrozmerná Rosenbrockova funkcia aj pri iných dimenziách, s výnimkou 3-rozmerného prípadu ([35]). Približne v lokálnom minime mali tendenciu častokrát zostávať všetky z deterministických metód, zriedkavejšie aj stochastická metóda *psoptim*. Metóda *bobyqa* pri dimenzii 20 za veľkého povoleného počtu evaluácií vracala hodnotu približne 3.9866 až v polovici prípadov, čo sa prejavilo aj na polohe najlepšej mediánovej hodnoty tejto metódy v obrázku 2.3. Metódy *DEoptim* a *abc_optim* zvykli túto prekážku prekonať. Metóda *ga* sa žiaľ ani pri jednom zo spustení pre dimenzie 10, 20 a 50 počas sledovaného času nedopracovala k funkčnej hodnote lepšej ako 4.09, takže ani nemala šancu uviaznúť v tomto lokálnom minime.

Hoci optimalizácia Rosenbrockovej funkcie nie je naším primárnym cieľom, naše porovnanie naznačuje, že najlepšie na tento problém funguje deterministická metóda L-BFGS-B, ale aj iné deterministické metódy, ktoré sme analyzovali. Zo stochastických metód možno odporučiť PSO. Metóda DE je vhodná skôr pre menšie dimenzie Rosenbrockovej funkcie a metóda ABC skôr pre väčšie dimenzie. Metóda GA, aspoň teda v implementácii, ktorú sme použili, je na optimalizáciu Rosenbrockovej funkcie výrazne najhoršia.

2.2 D-optimálny návrh experimentu pre logistickú regresiu

Optimalizujeme problém D-optimálneho návrhu pre model z článku [22], ktorý sme predstavili v časti 1.4 ako Model A. Úlohou, nazvime ju Problém A, je maximalizovať determinant

$$\det \left(\sum_{i=1}^N f(x_i) f^T(x_i) \right),$$

pričom f je dané vzorcom (1.10). Vzhľadom na to, že väčšina optimalizačných procedúr je prednastavená na optimalizáciu minimalizačných úloh, problém sme previedli na ekvivalentnú úlohu nájst'

$$\arg \min_{x_1, \dots, x_N \in X} -\det \left(\sum_{i=1}^N f(x_i) f^T(x_i) \right).$$

Táto úloha sa však ukázala ako numericky nestabilná kvôli výpočtom s veľmi malými hodnotami blízkymi nule, preto sme úlohu previedli na úlohu nájst'

$$\arg \min_{x_1, \dots, x_N \in X} g(x_1, \dots, x_N),$$

s účelovou funkciou

$$g(x_1, \dots, x_N) := -\ln(t(x_1, \dots, x_N)), \quad (2.2)$$

kde

$$t(x_1, \dots, x_N) := \max \left(\det \left(\sum_{i=1}^N f(x_i) f^T(x_i) \right), 10^{-100} \right).$$

Všimnime si, že touto (nevyhnutnou) numerickou stabilizáciou sme účelovú funkciu urobili nediferencovateľnou, takže vhodnejšie je použiť optimalizačné metódy, ktoré si nevyžadujú silné analytické predpoklady účelovej funkcie.

Výsledky optimalizácie problému A s účelovou funkciou (2.2) pre veľkosti experimentu $N = 6, 24, 96$ možno vidieť na obrázkoch 2.5a, 2.6a a 2.7a. Keďže ide o numericky veľmi náročný nekonvexný optimalizačný problém, nie je možné s istotou vedieť, či nami nájdené najlepšie riešenie je dokonalé globálne optimum. Z tohto dôvodu vedúci bakalárskej práce použil kombináciu pokročilých metód ([13], [32], s následným doladením pomocou lokálnej metódy L-BFGS-B) na získanie riešení problémov A a B, pričom tieto riešenia môžeme s dostatočnou spoľahlivosťou považovať za presné optimum². Takto vypočítanú hodnotu globálneho minima pre každú z veľkostí experimentov 6, 24 a 96 používame v grafoch 2.5b, 2.6b a 2.7b, v ktorých zobrazujeme

²Toto tvrdenie je podporené aj faktom, že priamočiare metódy, ktoré v tejto práci používame, nikdy nedosiahli lepšie riešenie, až na prípady zanedbateľných odchyliek spôsobených numerickými nepresnosťami jazyka R.

rozdiel nájdených hodnôt od globálneho minima, pričom k tomuto rozdielu bola ešte pripočítaná konštanta 10^{-6} a hodnota bola následne zlogaritmovaná (dekadickým logaritmom). V grafoch 2.5b, 2.6b a 2.7b tak skutočnému globálnemu minimu zodpovedá hodnota $y = -6$, a hodnoty na osi y tiež dávajú hrubú³ predstavu o tom, v ktorom desatinnom mieste sa nájdené výsledky líšia od globálneho minima. Tento alternatívny spôsob zobrazenia výsledkov tiež lepšie diferencuje metódy po dlhšom čase výpočtu (ked' už z absolútneho hľadiska dosahujú všetky procedúry podobnú hodnotu účelovej funkcie).

Na obrázku 2.5 pozorujeme, že pre veľkosť experimentu $N = 6$ globálne optimum našli za menej ako 1 sekundu všetky z deterministických metód a stochastická metóda *psoptim*. Nielen, že ho tieto metódy našli viackrát, pre dostatočne veľký povolený čas výpočtu ho nachádzali dokonca mediánovo. Metóda *abc_optim* sa do 1 sekundy dopracovala k riešeniu, ktorého funkčná hodnota bola vzdialenosť od globálneho minima približne o 10^{-5} . Problém sa za krátky sledovaný čas 1 sekundu nedarilo optimalizovať metódam *DEoptim* a *ga*, ktoré neboli schopné dodať globálne minimum ani len s presnosťou na jednu desatinu.

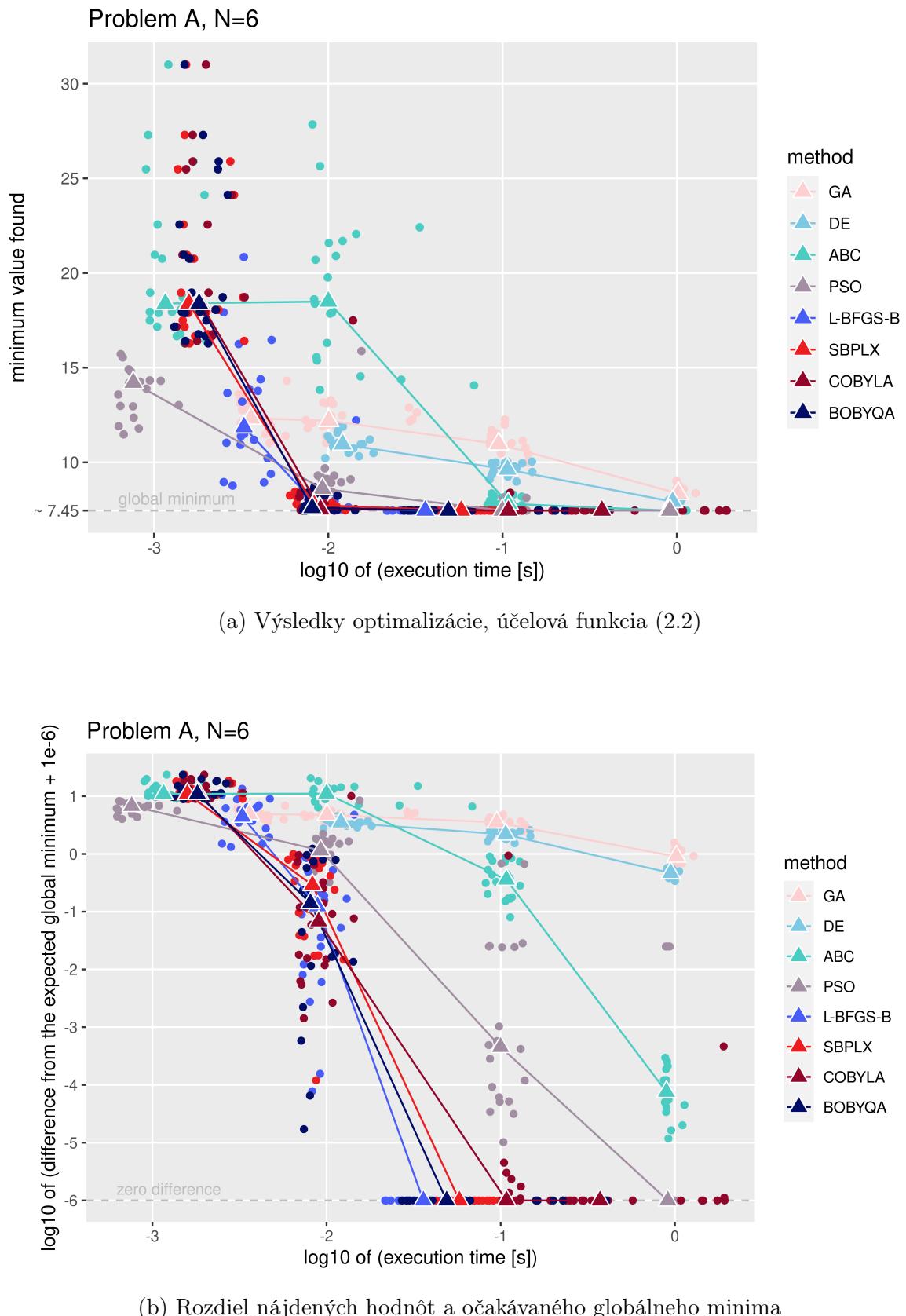
Pri veľkosti experimentu $N = 24$ sa už metódam nedarilo optimalizovať problém A s takou ľahkosťou, ako to bolo pri veľkosti $N = 6$. Na obrázku 2.6b vidíme, že globálne minimum s vysokou presnosťou v čase do 100 sekúnd našla jedine metóda *psoptim*. Aj táto metóda však vo väčšine prípadov vracala slabší výsledok, čo sa prejavilo na polohe jej mediánu. Metóda *abc_optim* konzistentne so zväčšujúcim povoleným počtom iterácií vracala aj presnejšie výsledky, pri povolenom čase výpočtu 100 sekúnd dokázala vrátiť riešenie s funkčnou hodnotou zhodujúcou sa s globálnym minimom na 3 desatinných miestach. Deterministické metódy *L-BFGS-B*, *sbplx* a *bobyqa* pri veľkom počte povolených iterácií/evaluácií odmietaли problém optimalizovať dlhšie ako 10 sekúnd napriek tomu, že ich výsledky boli pomerne slabé, s tým, že metóda *bobyqa* mala problémy so zaokrúhľovaním, metóda *sbplx* hlásila, že dosiahla maximálnu toleranciu a metóda *L-BFGS-B* v niektorých prípadoch vracala status *ERROR: ABNORMAL_TERMINATION_IN_LNSRCH*, zatiaľ čo v iných prípadoch vracala status konvergencie. Deterministická metóda *cobyla* sice počítala dlhšie ako metódy spomenné v predchádzajúcej vete, avšak za sledovaný čas sa jej nedarilo nájsť výsledky výrazne lepšie oproti zvyšným deterministickým metódam. Tieto problémy deterministických procedúr možno pripísaať tomu, že ide už o problém relatívne veľkej dimenzie (celkový počet reálnych premenných je tu 48), silnej nekonvexnosti účelovej funkcie a zrejme aj faktu, že účelová funkcia nie je všade diferencovateľná. Stochastické metódy *ga* a *DEoptim* boli opäť veľmi pomalé, no ich trend v grafe 2.6 naznačuje, že s

³Hodnoty na osi y by presne zodpovedali tomu, v ktorom desatinnom mieste sa nájdený výsledok líší od globálneho minima v prípade, ak by sme nepričítavali k rozdielu konštantu 10^{-6} . To si však minimálne pri $N = 6$ a $N = 24$ nemôžeme dovoliť, keďže $\log(0)$ nie je definovaný.

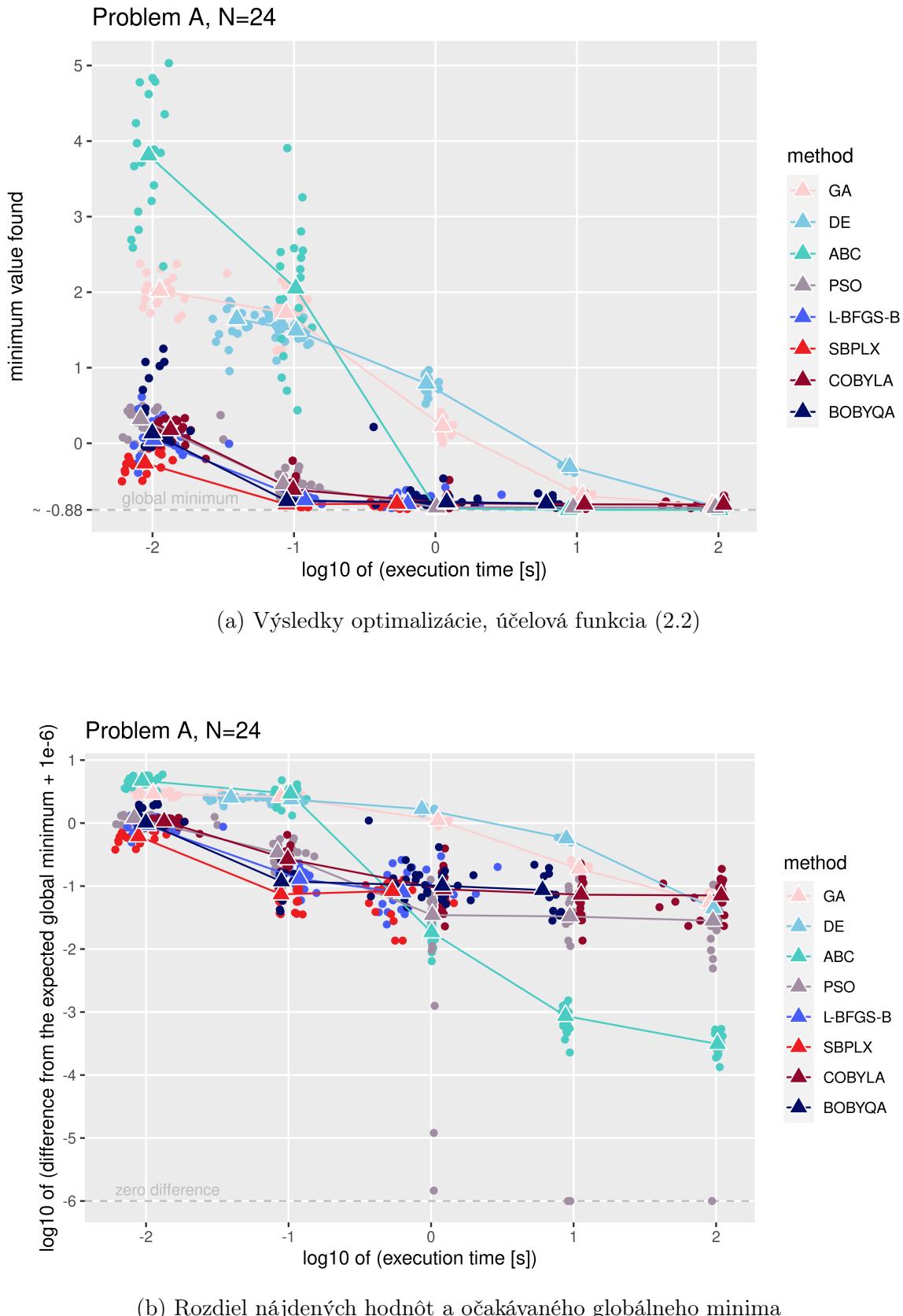
väčším povoleným časom výpočtu by možno vracali presnejšie výsledky ako sledované deterministické metódy, ktoré optimalizáciu predčasne ukončili.

Pri veľkosti experimentu $N = 96$ už presné globálne optimum nenašla počas sledovaného času približne 100 sekúnd ani jedna optimalizačná metóda, čo je zrejmé z obrázku 2.7b. Hodnotu (približne -9.21799) najbližšie ku globálnemu minimu (približne -9.21855) vrátila metóda *abc_optim*, a opäť pri rôznych spusteniach vracala vcelku konzistentné výsledky. Za ňou nasleduje metóda *psoptim*, ktorá do 100 sekúnd v najlepšom prípade vrátila hodnotu približne -9.21083 . Mediánové hodnoty tejto metódy pri časoch 10s a 100s ležia nad metódou *abc_optim* ako druhé v poradí. Pri zvyšných metódach pozorujeme podobný trend ako v prípade veľkosti experimentu $N = 24$.

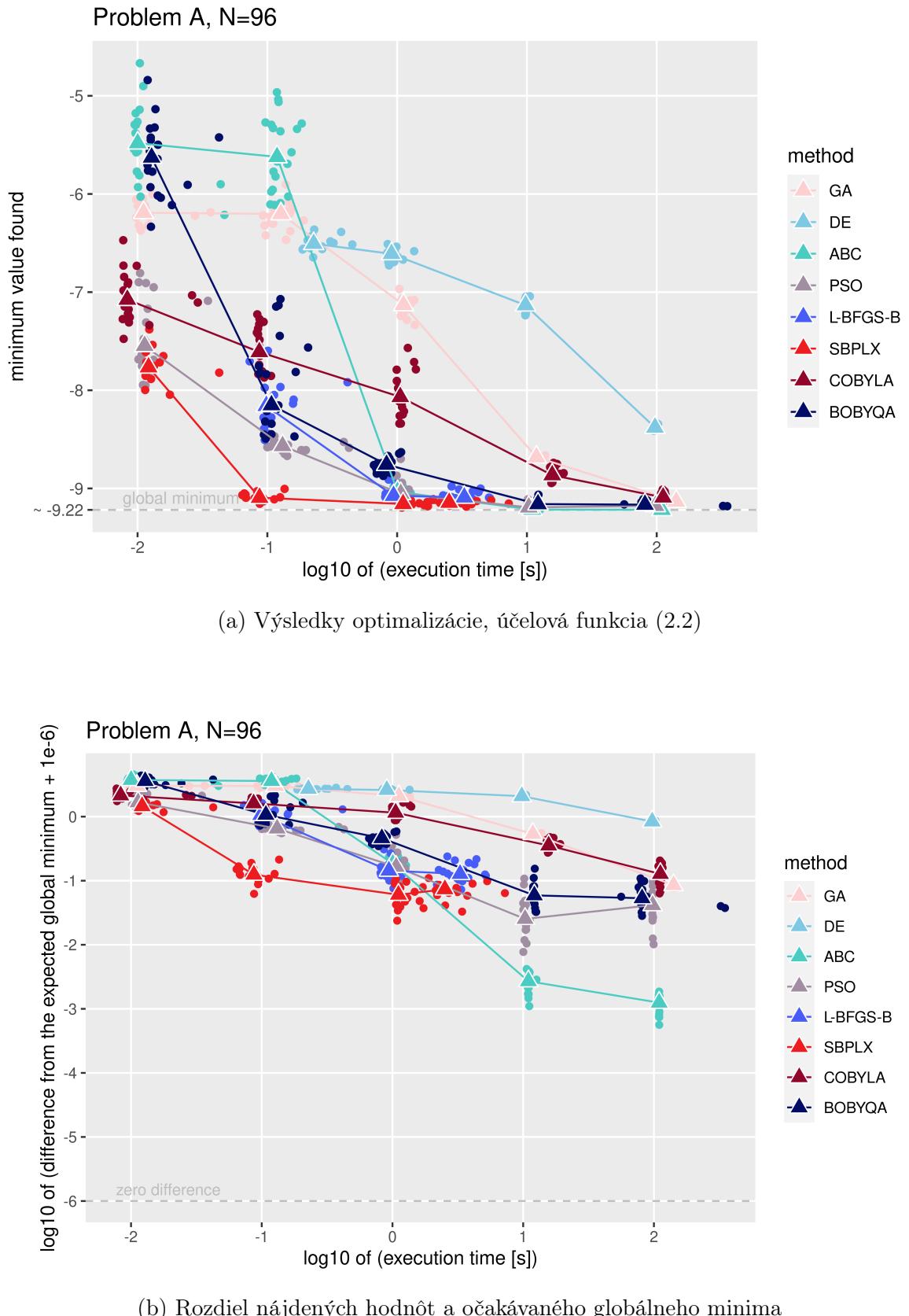
Bolo by zaujímavé pozrieť si, ako by si metódy poradili s problémom A s veľkosťou $N = 96$ pri povolenom čase výpočtu 1000 sekúnd. Lenže takýto výpočet s použitím rovnakej metodiky ako v predošlých grafoch (spustiť každú metódu 20-krát, okrem metód *L-BFGS-B* a *sbplx*, pri ktorých tento čas 1000s nevieme dosiahnuť) by nás stál 33 hodín 20 minút, takže sme sa rozhodli z tohto plánu upustiť.



Obr. 2.5: Porovnanie optimalizačných metód na probléme A, veľkosť experimentu N=6



Obr. 2.6: Porovnanie optimalizačných metód na probléme A, veľkosť experimentu N=24



Obr. 2.7: Porovnanie optimalizačných metód na probléme A, veľkosť experimentu N=96

2.3 D-optimálny návrh experimentu pre Poissonovu regresiu

Optimalizujeme návrh pre model z článku [7], ktorý sme predstavili v časti 1.4 ako Model B. Daný optimalizačný problém nazveme Problém B. S použitím rovnakých úvah ako v časti 2.2, úlohou je minimalizovať účelovú funkciu

$$g(x_1, \dots, x_N) := -\ln(t(x_1, \dots, x_N)),$$

kde

$$t(x_1, \dots, x_N) := \max \left(\det \left(\sum_{i=1}^N f(x_i) f^T(x_i) \right), 10^{-100} \right),$$

pričom f je určené predpisom (1.11).

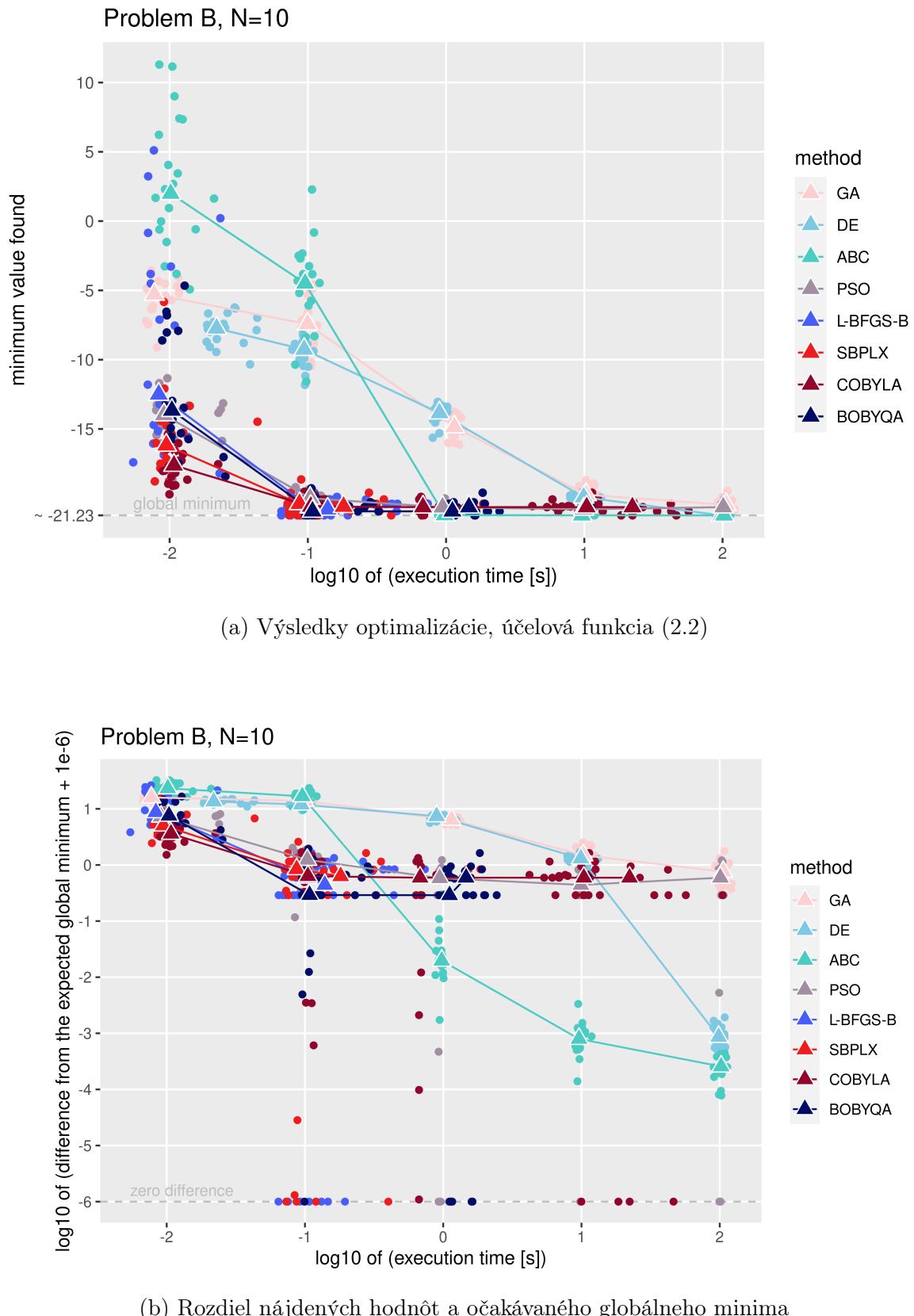
Výsledky optimalizácie tohto problému pre veľkosti experimentu $N = 10, 50, 100, 200$ sú zobrazené na obrázkoch 2.8, 2.9 a 2.10. Na výpočet predpokladaného globálneho minima a na zobrazenie výsledkov boli použité rovnaké metodiky ako pri Probléme A.

Pri veľkosti experimentu $N = 10$ bolo skutočné globálne optimum počas sledovaného času nájdené niekoľkokrát metódami *L-BFGS-B*, *sbplx*, *psoptim*, *bobyqa* a *cobyla*, pričom najrýchlejšie takýto bod našla metóda *L-BFGS-B*. Žiadnej zo spomenutých metód sa nedarilo globálne minimum ani hodnotu blízku globálnemu minimu nájsť mediánovo, to znamená, že metódy vracali častokrát aj pomerne zlé výsledky. Pri povolenom čase 100 sekúnd vracali mediánovo najlepšie hodnoty stochastické metódy *abc_optim* a *DEoptim*.

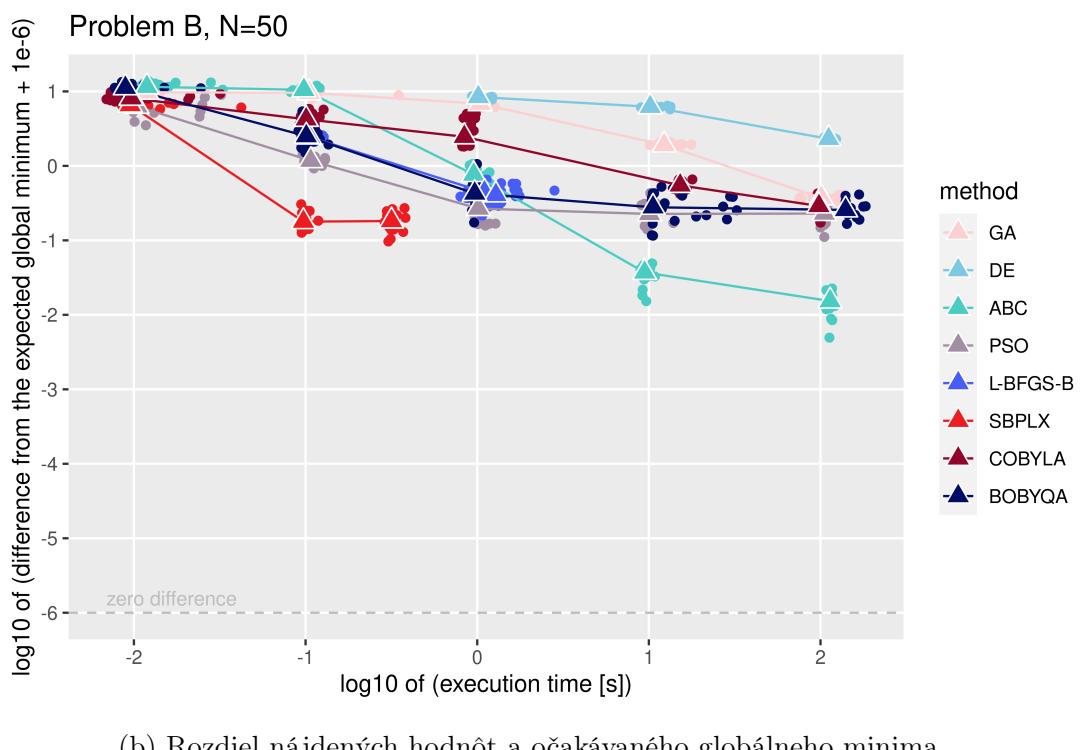
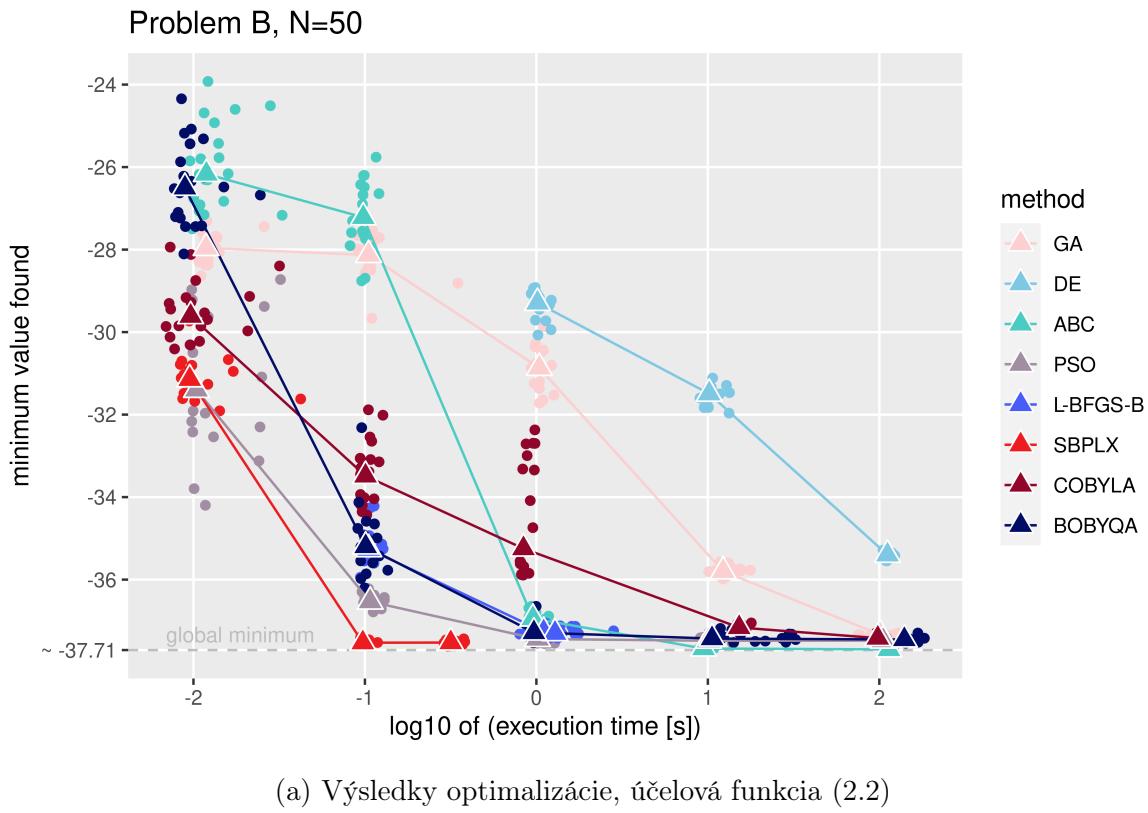
Pri väčších experimentoch s veľkosťami $N = 50, 100, 200$ úlohu najlepšie optimalizovala metóda *abc_optim*, ktorá sa počas sledovaného času dokázala priblížiť ku globálnemu minimu najviac spomedzi všetkých metód nielen jednorazovo, ale aj mediánovo, čo pozorujeme na obrázkoch 2.9, 2.10 a 2.11. Spomedzi sledovaných metaheuristických metód sa na druhom mieste darilo metóde *psoptim*, za ňou nasledovala metóda *ga*, a najhoršie výsledky nachádzala metóda *DEoptim*. Spomedzi deterministických metód do 100 sekúnd nachádzala najrýchlejšie a najkvalitnejšie výsledky metóda *sbplx*, ktorá pre veľkosti experimentu 50 a 100 vrátila výsledok pri veľkom povolenom počte evalúácií približne do jednej sekundy (ďalej nechcela pokračovať z dôvodu dosiahnutia tolerancie⁴), o niečo horšie výsledky za dlhší čas nachádzala metóda *bobyqa* a metóda *L-BFGS-B*. Z deterministických metód bola najmenej vhodná na použitie pre veľké návrhy metóda *cobyla*.

Z porovnania vyplýva, že zvolené problémy A a B z oblasti optimálneho narhovania experimentu najlepšie optimalizovala metóda *abc_optim*, ktorej sa darilo výnimcočne dobre najmä pri veľkých návrhoch.

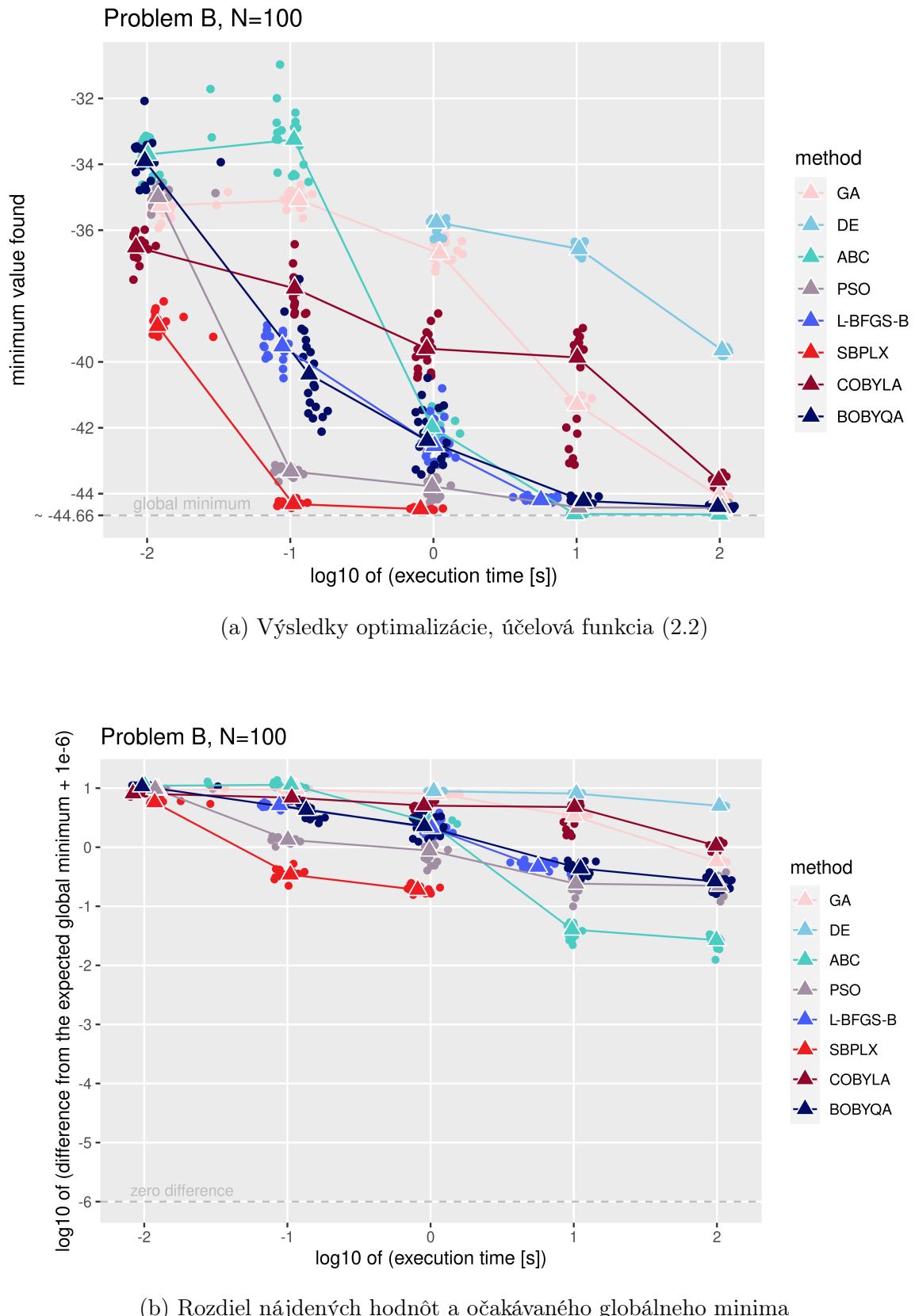
⁴Kedže sme relatívnu toleranciu nastavili na hodnotu 0 a absolútna tolerancia nevstupuje ako parameter do tejto metódy, ide pravdepodobne o nastavenie, ktoré nevieme ovplyvniť.



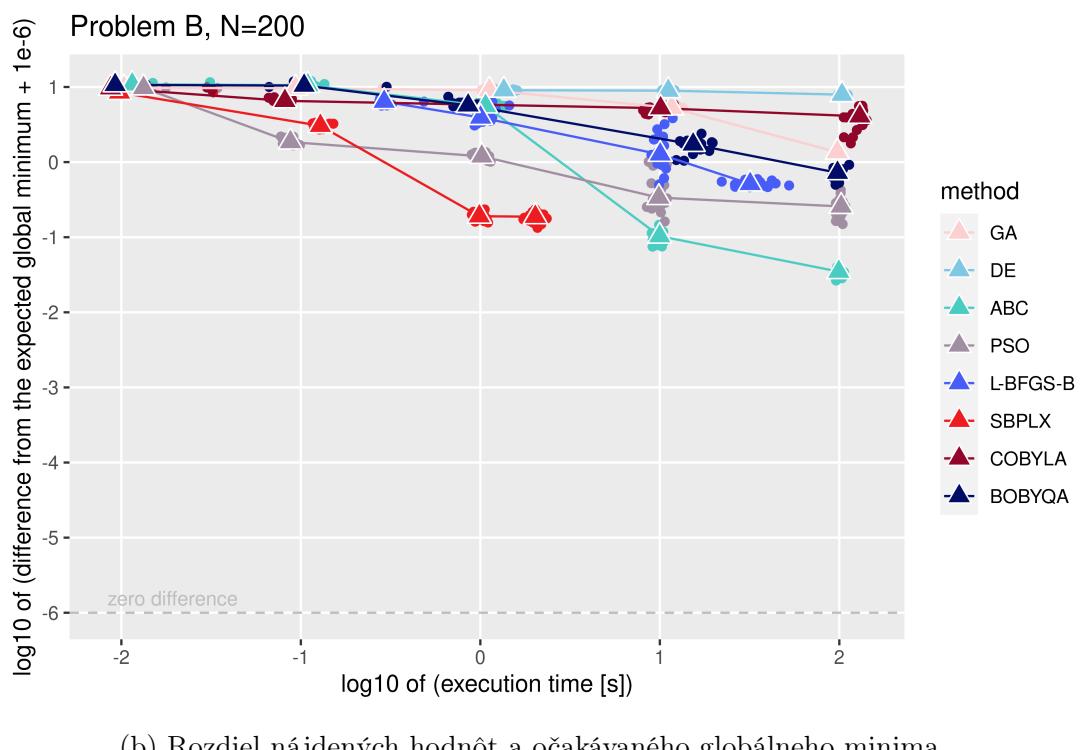
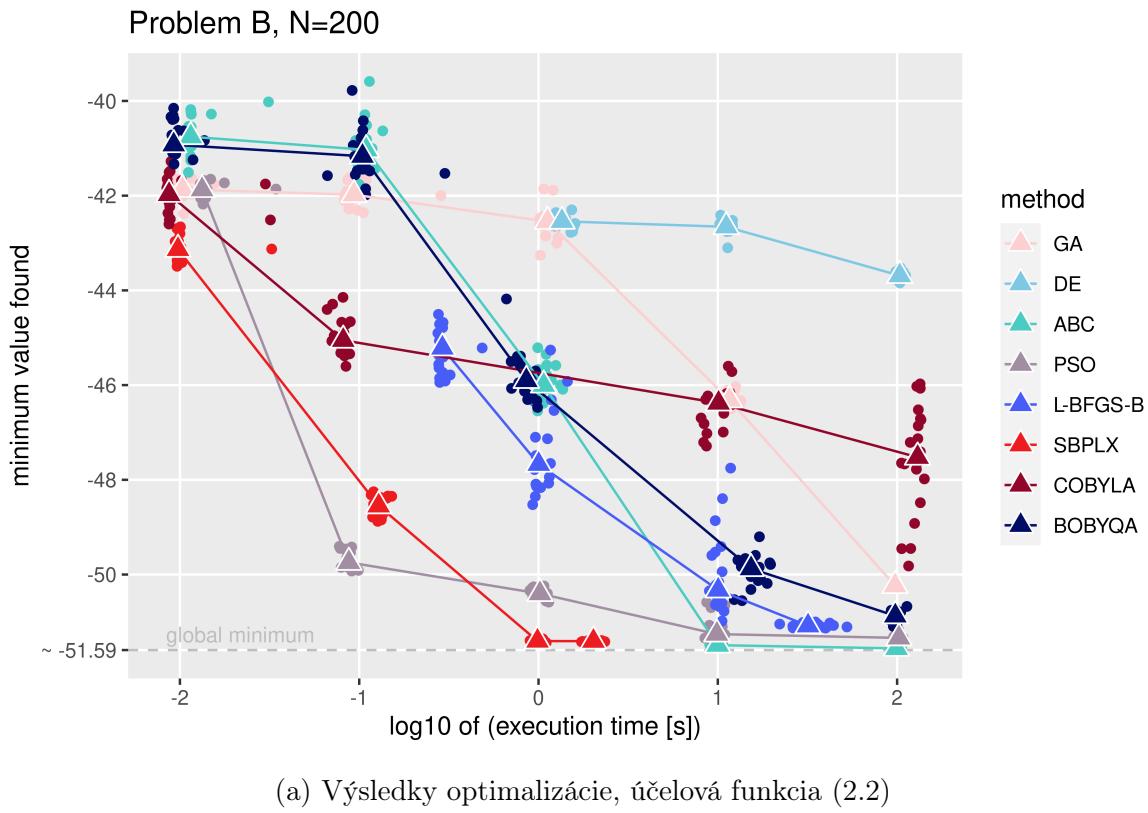
Obr. 2.8: Porovnanie optimalizačných metód na probléme B, veľkosť experimentu $N=10$



Obr. 2.9: Porovnanie optimalizačných metód na probléme B, veľkosť experimentu N=50



Obr. 2.10: Porovnanie optimalizačných metód na probléme B, veľkosť experimentu $N=100$



Obr. 2.11: Porovnanie optimalizačných metód na probléme B, veľkosť experimentu N=200

Záver

Spravili sme stručný prehľad základných typov metaheuristik a ich implementácií v softvéri R. Vysvetlili sme princíp stochastických metaheuristickej algoritmov, akými sú simulované žíhanie, genetické algoritmy, diferenciálna evolúcia, optimalizácia rojom častíc a umelá kolónia včiel. Ďalej sme predstavili problematiku optimálneho navrhovania experimentu a opísali istú triedu úloh z tejto oblasti, ktorú sme v rámci tejto bakalárskej práce optimalizovali. Cieľom bolo identifikovať metaheuristiky, ktoré sú najvhodnejšie pre riešenie optimalizačných úloh typu D-optimálny návrh pre logistickú regresiu a D-optimálny návrh pre Poissonovu regresiu.

Porovnávali sme výkon vybraných stochastických aj deterministických algoritmov implementovaných v knižničach prostredia R. Do porovnania sme zahrnuli stochastické metódy *ga* z knižnice *GA*, *DEoptim* z knižnice *DEoptim*, *psoptim* z knižnice *pso*, *abc_optim* z knižnice *ABCOptim* a deterministické metódy *optim* využívajúcu algoritmus *L-BFGS-B* zo základnej knižnice *stats*, *sbplx* z knižnice *nloptr*, *cobyla* z knižnice *nloptr* a *bobyqa* z knižnice *nloptr*. Menili sme počty iterácií a evaluácií tak, aby tieto metódy vrátili výsledky približne po želanom čase. Každú metódu sme pre každý sledovaný čas spustili 20-krát a ak metóda umožňovala zadať štartovací bod, pri každom spustení sme ho generovali náhodne z priestoru prípustných riešení. Optimalizovali sme viacrozmernú Rosenbrockovu funkciu dimenzií 3, 10, 20 a 50, ktorá sa často používa na testovanie výkonnosti optimalizačných metód. Okrem toho sme optimalizovali dva problémy z oblasti navrhovania štatistického experimentu (problémy D-optimálny návrh experimentu pre logistickú regresiu a D-optimálny návrh experimentu pre Poissonovu regresiu prevzaté z článkov [22] a [7]) pre rôzne veľkosti experimentu. Výsledky optimalizácie sme vizualizovali.

Výsledky získané v tejto práci naznačujú, že na optimalizáciu danej triedy úloh z oblasti optimálneho navrhovania štatistického experimentu by popri v tejto oblasti častokrát odporúčanej ([38]) a dobre známej metaheuristickej metóde optimalizácia rojom častíc (PSO) mohla byť vhodná aj menej známa metaheuristika umelá kolónia včiel (ABC). Implementácia tejto metódy *abc_optim* z balíka *ABCOptim* [37] prostredia R umožňuje zadať intervalové ohraničenia pre každú premennú a metóda zvládala optimalizovať pomerne dobre oba zvolené problémy z článkov [22] a [7], pričom boli použité predvolené nastavenia tejto metódy, okrem parametrov *criter* a *maxCycle*, ktoré

sme vhodne prestavili. Metóda *abc_optim* vracala výsledky blízko globálnemu optimu aj pre veľké návrhy, okrem toho sa pre rôzne spustenia z rôznych štartovacích bodov správala konzistentne, to znamená, že multištart by nemusel byť potrebný a aj jedno spustenie metódy by mohlo byť postačujúce na nájdenie uspokojivo dobrého výsledku.

Vzhľadom na to, že v oblasti optimálneho návrhu experimentu sa dosiaľ nezachytil fakt, že optimalizácia metódou ABC funguje v mnohých prípadoch výrazne lepšie než tradične používané metaheuristiké metódy ako GA, PSO a DE, mohlo by byť vhodné toto pozorovanie otestovať na širšej triede probémov z navrhovania experimentov, urobiť vlastnú, efektívnejšiu implementáciu procedúry ABC s využitím paralelizácie a výsledky publikovať.

Literatúra

- [1] Anthony Atkinson, Alexander Donev, and Randall Tobias. *Optimum Experimental Designs, with SAS*, volume 34. OUP Oxford, 1st edition, 2007.
- [2] Claus Bendtsen. *pso: Particle Swarm Optimization*, 2022. R package version 1.0.4. <https://CRAN.R-project.org/package=pso>. Dátum prístupu: 20.05.2023.
- [3] Jakob Bossek. smoof: Single- and multi-objective optimization test functions. *The R Journal*, 9(1):103–113, 2017.
- [4] Anthony Brabazon, Michael O’Neill, and Seán McGarragh. *Natural Computing Algorithms*, volume 554. Springer, 2015.
- [5] Richard H. Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on scientific computing*, 16(5):1190–1208, 1995.
- [6] Vladimír Černý. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of optimization theory and applications*, 45:41–51, 1985.
- [7] Belmiro P.M. Duarte, Weng Kee Wong, and Holger Dette. Adaptive grid semidefinite programming for finding optimal designs. *Statistics and Computing*, 28:441–460, 2018.
- [8] Ronald A. Fisher. *The Design of Experiments*. Oliver and Boyd, 1st edition, 1935.
- [9] Keith Glover. Howard Harry Rosenbrock. 16 December 1920—21 October 2010. *Biographical Memoirs of Fellows of the Royal Society*, 68:353–369, 2020.
- [10] Peter Goos and Bradley Jones. *Optimal Design of Experiments: A Case Study Approach*. Wiley, 1st edition, 2011.
- [11] Vincent Granville, Mirko Krivánek, and Jean-Paul Rasson. Simulated annealing: A proof of convergence. *IEEE transactions on pattern analysis and machine intelligence*, 16(6):652–656, 1994.

- [12] Milan Hamala and Mária Trnovská. *Nelineárne programovanie*. Epos, 1st edition, 2012.
- [13] Radoslav Harman, Lenka Filová, and Samuel Rosa. Optimal design of multifactor experiments via grid exploration. *Statistics and Computing*, 31(70):1–13, 2021.
- [14] Randy L. Haupt and Sue Ellen Haupt. *Practical Genetic Algorithms*. John Wiley & Sons, 2nd edition, 2004.
- [15] Kai Husmann. *The R package optimization: Flexible Optimization with Simulated Annealing*, 2017. R package version 1.0-9. https://search.r-project.org/CRAN/refmans/optimization/html/optim_sa.html. Dátum prístupu: 19.05.2023.
- [16] Dervis Karaboga. Artificial bee colony algorithm. *Scholarpedia*, 5(3):6915, 2010. Revision #91003. Dátum prístupu: 5.4.2023.
- [17] Dervis Karaboga and Bahriye Akay. A comparative study of artificial bee colony algorithm. *Applied Mathematics and Computation*, 214(1):108–132, 2009.
- [18] Dervis Karaboga and Bahriye Basturk. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (abc) algorithm. *Journal of global optimization*, 39:459–471, 2007.
- [19] Dervis Karaboga et al. An idea based on honey bee swarm for numerical optimization. Technical report, Erciyes university, engineering faculty, computer engineering department, 2005.
- [20] Dervis Karaboga, Beyza Gorkemli, Celal Ozturk, and Nurhan Karaboga. A comprehensive survey: artificial bee colony (abc) algorithm and applications. *Artificial Intelligence Review*, 42:21–57, 2014.
- [21] Scott Kirkpatrick, C. Daniel Gelatt Jr., and Mario P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.
- [22] Shwetank Lall, Seema Jaggi, Eldho Varghese, Cini Varghese, and Arpan Bhowmik. An algorithmic approach to construct D-optimal saturated designs for logistic model. *Journal of Statistical Computation and Simulation*, 88(6):1191–1199, 2018.
- [23] Katharine Mullen, David Ardia, David L. Gil, Donald Windover, and James Cline. DEoptim: An R package for global optimization by differential evolution. *Journal of Statistical Software*, 40(6):1–26, 2011.
- [24] Andrej Pázman. *Foundations of Optimum Experimental Design*. Springer, 1st edition, 1986.

- [25] Riccardo Poli, James Kennedy, and Tim Blackwell. Particle swarm optimization: An overview. *Swarm intelligence*, 1:33–57, 2007.
- [26] Michael J.D. Powell. 'A Direct Search Optimization Method That Models the Objective and Constraint Functions by Linear Interpolation' in *Advances in Optimization and Numerical Analysis*, eds. S. Gomez and J.-P. Hennart, pages 51–67. Kluwer Academic, Dordrecht, 1994.
- [27] Michael J.D. Powell. The BOBYQA algorithm for bound constrained optimization without derivatives. Technical report, Department of Applied Mathematics and Theoretical Physics, University of Cambridge, 06 2009.
- [28] Andrej Pázman and Vladimír Lacko. *Prednášky z regresných modelov*. Univerzita Komenského v Bratislave, 1st edition, 2012.
- [29] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2013. <http://www.R-project.org/>. Dátum prístupu: 20.5.2023.
- [30] Howard H. Rosenbrock. An Automatic Method for Finding the Greatest or Least Value of a Function. *The Computer Journal*, 3(3):175–184, 01 1960.
- [31] Thomas H. Rowan. *Functional stability analysis of numerical algorithms*. PhD thesis, Department of Computer Sciences, University of Texas at Austin, 1990.
- [32] Guillaume Sagnol and Radoslav Harman. Computing exact D -optimal designs by mixed integer second-order cone programming. *The Annals of Statistics*, 43(5):2198 – 2224, 2015.
- [33] Luca Scrucca. GA: A package for genetic algorithms in R. *Journal of Statistical Software*, 53(4):1–37, 2013.
- [34] Thomas D. Seeley. *The Wisdom of the Hive: The Social Physiology of Honey Bee Colonies*. Harvard University Press, 1995.
- [35] Yun-Wei Shang and Yu-Huang Qiu. A Note on the Extended Rosenbrock Function. *Evolutionary Computation*, 14(1):119–126, 03 2006.
- [36] Rainer Storn and Kenneth Price. Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341, 1997.
- [37] George Vega Yon and Enyelbert Muñoz. *ABCoptim: An implementation of the Artificial Bee Colony (ABC) Algorithm*, 2017. R package version 0.15.0. <https://CRAN.R-project.org/package=ABCoptim>. Dátum prístupu: 30.04.2023.

- [38] Stephen J. Walsh and John J. Borkowski. Generating exact optimal designs via particle swarm optimization: Assessing efficacy and efficiency via case study. *Quality Engineering*, 35(2):304–323, 2023.
- [39] David H. Wolpert and William G. Macready. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.
- [40] Xin-She Yang. *Engineering Optimization: An Introduction with Metaheuristic Applications*. John Wiley & Sons, 2010.
- [41] Jelmer Ypma and Steven G. Johnson. *The NLOpt nonlinear-optimization package*, 2022. R package version 2.0.3. <https://CRAN.R-project.org/package=nloptr>. Dátum prístupu: 20.05.2023.

Príloha A: obsah elektronickej prílohy

V elektronickej prílohe priloženej k práci sa nachádza zdrojový kód v jazyku R a súbory s výsledkami optimalizácie.