ELEKTROTEHNIŠKI VESTNIK 79(5): 231–236, 2012
EXISTING SEPARATE ENGLISH EDITION

Metaoptimizacija z visokozmogljivim računskim sestavom

Árpád Bűrmen, Tadej Tuma, Iztok Fajfar

University of Ljubljana, Faculty of Electrical Engineering, Tržaška cesta 25, 1000 Ljubljana, Slovenija
E-pošta: arpad.buermen@fe.uni-lj.si

Povzetek. Vsi optimizacijski poostopki imajo parametre, ki določajo hitrost in zanesljivost postopka. Po navadi so njihove optimalne vrednosti odvisne od problema oziroma skupine problemov, ki jih s tem postopkom rešujemo. Ne glede na to pa se v praksi skoraj vedno uporabljajo vrednosti, ki so objavljene v literaturi. Določanje optimalnih vrednosti parametrov je računsko zahteven optimizacijski problem, ki ga imenujemo tudi metaoptimizacija. Računska zahtevnost izhaja iz dejstva, da je vsak izračun kriterijske funkcije v metaoptimizaciji sestavljen iz enega ali več tekov optimizacijskega postopka, ki ovrednosti njegovo obnašanje pri danih vrednostih parametrov. Metaoptimizacijo običajno izvedemo s pomočjo vzporednega računanja. V prispevku prikažemo postopek za sestavljanje kriterijske funkcije v metaoptimizaciji direktnih optimizacijskih postopkov. Kot primer izvedemo optimizacijo Nelder-Meadovega simpleksnega postopka na uveljavljenem naboru testnih problemov s pomočjo vzporednega računalnika s 100 procesorji. Rezultati metaoptimizacije so presenetljivi, saj optimalne vrednosti močno odstopajo od tistih, ki so bile objavljene pred skoraj petdesetimi leti in se kljub temu še vedno uporabljajo v praksi.

Ključne besede: metaoptimizacija, globalna optimizacija, simpleksni postopek

Meta-Optimisation on a High-Performance Computing System

All optimisation algorithms have parameters that affect their performance and reliability. Usually the default values of these parameters are problem-dependent. Regardless of this fact it is common practice to use some default values that are provided with the optimization algorithm. Finding the optimal values of these parameters is a computationaly expensive optimization problem also known as meta-optimization. The computational complexity comes from the fact that every cost function evaluation in meta-optimization involves several runs of an optimization algorithm that evaluate its behavior for given values of algorithm parameters. The most common approach to making meta-optimisation feasible is the use of parallel computing. The paper presents the construction of the cost function for meta-optimisation of direct search optimisation algorithms. We demonstrate the approach by optimizing the parameters of the Nelder-Mead simplex algorithm using a high-performance computing system comprising 100 processing units. The results of the meta-optimisation are surprising because the obtained values of parameters greatly differ from the values that were published 50 years ago, but are still used despite their suboptimality.

1 UVOD

Postopkom za iskanje minimuma realne funkcije n spremenljivk pravimo tudi optimizacijski postopki, funkciji, katere minimum iščemo, pa kriterijska funkcija (KF). Ključni merili za kakovost optimizacijskega postopka sta njegova hitrost in končna vrednost KF, ki jo dosežemo, ko se postopek ustavi. Hitrost postopka po navadi izražamo s številom vrednotenj kriterijske funkcije (N). Hitrejši postopki pridejo do enake končne vrednosti KF z manjšim N. Vsak optimizacijski postopek ima tudi parametre, od katerih sta odvisna hitrost postopka in končna vrednost KF. Vektor teh parametrov imenujemo tudi strategija optimizacijskega postopka.

Optimalna strategija je na splošno odvisna od problema, ki ga rešujemo z optimizacijskim postopkom. V literaturi je ob vsakem objavljenem postopku skoraj vedno objavljena tudi priporočena strategija, ki je pogosto dobljena s pomočjo omejenega števila poskusov, ali pa je plod nekega (običajno preveč) poenostavljenega sklepanja. Kljub temu se objavljene strategije v praksi uporabljajo brez kritične presoje in po ključu ”avtor je že vedel zakaj”. Posledica takega nekritičnega prevzemanja priporočenih vrednosti je skoraj vedno neučinkovitost postopka pri reševanju optimizacijskih problemov.

Zaradi odvisnosti optimalne strategije od narave optimizacijskega problema (ali družine problemov) jo moramo po navadi določiti sami. Naloga je sama po sebi spet optimizacijski problem, ki ga imenujemo tudi metaoptimizacijski problem, postopek njegovega reševanja pa metaoptimizacija. Optimizacijski postopek, za katerega iščemo strategijo, imenujemo tudi osnovni (optimizacijski) postopek. Tudi pri metaoptimizaciji definiramo KF, ki odraža kakovost strategije. Spremenljivke v metaoptimizaciji so komponente vektorja, ki pomeni strategijo. Določanje vrednosti KF za izbrano strategijo vključuje enega ali več tekov osnovnega postopka, s katerimi

Prejet 19. november, 2012
Odobren 9. januar, 2013

skušamo zajeti njegovo obnašanje na dani družini problemov. Metaoptimizacija je zaradi tega računsko zelo zahteven postopek, saj lahko že en sam izračun KF traja tudi po več ur.

Zadeve dodatno zaplete še dejstvo, da je KF pri metaoptimizaciji pogosto nezvezna, ima več minimumov in je več ali manj popačena z numeričnim šumom. Ravno yato nam pogosto ne preostane nič drugega, kot da za metaoptimizacijo uporabimo globalne optimizacijske postopke. Njihova glavna slabost je veliko število izračunov KF (10000 in več), ki so potrebni, da postopek najde rešitev metaoptimizacijskega problema. Na srečo lahko omenjeno slabost močno omilimo z uporabo vzporednega računanja na več procesorskih jedrih. V tem smislu nam gre v prid tudi dejstvo, da za globalne optimizacijske postopke ponavadi obstajajo tudi učinkovite vzporedne izvedbe, ki izkoriščajo računsko moč več procesorjev.

V nadaljevanju se bomo omejili na metaoptimizacijo lokalnih optimizacijskih postopkov. To so postopki, ki nehajo iskati, ko najdejo lokalni minimum KF. Po navadi zahtevajo precej manjše število izračunov KF kot globalni optimizacijski postopki. Čeprav lokalni minimum ponavadi ne pomeni najboljše rešitve nekega optimzacijskega problema, pa se lokalnih optimizacijskih postopkov pogosto poslužujemo v inženirski praksi. Če je optimizacijski problem zelo zahteven (en izračun kriterijske funkcije traja dolgo), računska moč pa omejena, je lokalni optimizacijski postopek edina uporabna možnost, ki jo imamo. Tudi ko smo zadovoljni že samo z zmanjšanjem vrednosti KF, je lokalni optimizacijski postopek ustrezen.

2 KRITERIJSKA FUNKCIJA V METAOPTIMIZACIJI

Pred vsako metaoptimizacijo se moramo vprašati, za kakšne optimizacijske probleme iščemo optimalno strategijo. Realni optimizacijski problemi iz inženirske prakse so pogosto prezahtevni, da bi jih lahko uporabili pri določanju učinkovitosti osnovnega optimizacijskega postopka. Zato se raje poslužujemo naborov matematičnih testnih funkcij. Danes se za preizkušanje lokalnih optimizacijskih postopkov uporabljata dva nabora: nekoliko starejši Moré-Garbow-Hillstromov nabor [1] in nabor CUTEr [2]. Prvi obsega 35 neomejenih optimizacijskih problemov, drugi pa več 1000 problemov, od katerih so nekateri omejeni, drugi pa neomejeni. Po navadi iz nabora problemov (kriterijskih funkcij) izberemo podmnožico $\{f_1, f_2, ..., f_m\}$. Ker lokalni optimizacijski postopki potrebujejo tudi začetno točko, bomo le-te označevali z $x_1^0, x_2^0, ..., x_m^0$.

Pri določanju izraza za KF v metaoptimizaciji po navadi izhajamo iz rezultatov, ki jih dobimo, ko izbrane probleme rešimo z uporabo osnovnega optimizacijskega postopka in trenutno najboljših znane (referenčne) strategije. Pri tem nam dobljeno število izračunov kriterijske funkcije ($N_1', N_2', ..., N_m'$) podaja referenčne vrednosti za hitrost osnovnega postopka. Kakovost rezultata osnovnega postopka lahko ocenimo z ℓ_2 normo gradienta KF v točki, ki jo postopek vrne kot rešitev optimizacijskega problema. Nižja vrednost norme gradienta ustreza boljšemu približku lokalnega minimuma optimizacijskega problema. Označimo z $x_1', x_2', ..., x_m'$ približke rešitev izbranih optimizacijskih problemov, ki jih dobimo z uporabo referenčne strategije, z $g_1', g_2', ..., g_m'$ pa pripadajoče vrednosti gradienta.

Recimo, da za neko strategijo dobimo kot približke rešitev izbranih optimizacijskih problemov točke x_1, x_2, ..., x_m, ki jim pripadajo vrednosti gradienta $g_1, g_2, ..., g_m$. Število vrednotenj KF, ki pripadajo posameznim optimizacijskim tekom osnovnega postopka, označimo z N_1, N_2, ..., N_m. Ker se lahko primeri, da optimizacijski postopek pri tej strategiji deluje slabše, kot pri referenčnih strategiji, lahko postanejo pripadajoči optimizacijski teki zelo dolgi. Zato število izračunov f_i omejimo na

$$N_i^{\max} = \max\left(5N_i', 10000\right). \tag{1}$$

Končni cilj metaoptimizacije je najti strategijo, pri kateri se optimizacijski postopek obnaša vsaj tako dobro kot pri referenčni strategiji. Omejitve te vrste lahko izrazimo s kriterijskimi fukcijami, ki jih sestavimo iz kazenskih prispevkov (C_i). [3] Pri tem obnašanje osnovnega postopka, ki je slabše od tistega pri referenčni strategiji, kaznujemo z velikim pozitivnim prispevkom C_i. Obnašanje, ki je boljše od referenčnega nagradimo z majhnim negativnim prispevkom C_i. Če se osnovni postopek obnaša enako kot pri referenčnih strategiji, velja $C_i = 0$. KF metaoptimizacije je enaka vsoti prispevkov, ki pripadajo m testnim problemom.

$$C = \sum_{i=1}^{m} C_i. \tag{2}$$

Vrednost $C = 0$ tako ustreza enaki učinkovitosti postopka, kot jo dobimo pri referenčni strategiji (gledano prek nabora m izbranih optimizacijskih problemov). Prispevek i-tega optimizacijskega problema k vrednosti KF lahko zapišemo kot

$$C_i = \begin{cases} (N_i/N_i' - 1)\cdot 10; & N_i > N_i' \\ (N_i/N_i' - 1)/10; & N_i \leq N_i' \end{cases} \tag{3}$$

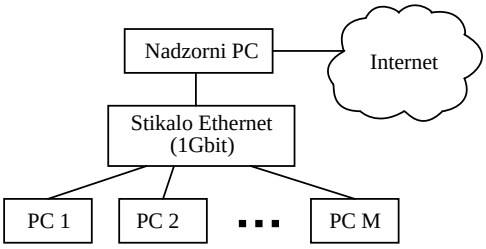
$$+ \begin{cases} \log_{10}(||g_i||/||g_i'||)\cdot 10; & ||g_i|| > ||g_i'|| \\ \log_{10}(||g_i||/||g_i'||)/10; & ||g_i|| \leq ||g_i'|| \end{cases} \tag{4}$$

Namesto vrednosti 10 lahko uporabimo poljubno konstanto $P > 1$. Pri tem je absolutna vrednost razmerja med kazenskim prispevkom in nagrado enaka P^2. En izračun kriterijske funkcije (2) v metaoptimizaciji obsega m tekov osnovnega optimizacijskega postopka.

3 STROJNA OPREMA

Metaoptimizacija je računsko zelo zahteven postopek. Da bi jo lahko sploh izvedli, potrebujemo ve-

liko računsko moč, ki jo danes dajejo viskozmogljivi računski sestavi (ang. High Performance Computing, HPC). Ti so po navadi sestavljeni iz velikega števila procesorskih enot, v vlogi katerih lahko nastopajo tudi namizni računalniki. Cena ene procesorske enote je danes relativno nizka, saj za 600 evrov že dobimo zmogljiv namizni računalnik s štirijedrnim procesorjem, kar znaša 150 evrov na procesorsko jedro. Ker so namizni računalniki večinoma že opremljeni z mrežnim vmesnikom Ethernet hitrosti $1\mathrm{Gbit/s}$, je edina dodatna investicija, ki jo potrebujemo za vzpostavitev visokozmogljivega računskega sestava, mrežno stikalo, katerega cena je ponavadi nižja od cene enega namiznega računalnika. Cena sistema s 100 procesorskimi jedri se tako giblje okrog 15000 evrov.



Slika 1: Zgradba visokozmogljivega računskega sestava

Ozko grlo tako zgrajenega računskega sestava je mrežna povezava. Kljub veliki hitrosti prenosa, so zakasnitve pri komunikaciji med dvema računalnikoma v sestavu reda velikosti $20\mu\mathrm{s}$. Zakasnitev je v veliki meri posledica narave mrežnega vmesnika, delno pa jo lahko pripišemo tudi neučinkovitim gonilnikom (pribl. $10\mu\mathrm{s}$, [4]). To je precej več kot v primerljivih sistemih, ki uporabljajo povezave InfiniBand (kot npr. [5]). Te zlahka dosegajo zakasnitve reda velikosti $1\mu\mathrm{s}$. Tudi hitrost prenosa podatkov je pri InfiniBandu boljša, saj za najosnovnejšo povezavo SDR znaša $2\mathrm{Gbit/s}$. Seveda uporaba InfiniBanda precej zviša ceno sestava.

Ne glede na nekoliko nižjo hitrost prenosa podatkov in večjo zakasnitev pa je računski sestav s povezavami Ethernet dovolj zmogljiv za izvajanje vzporednih optimizacijskih postopkov. Označimo hitrost prenosa podatkov z R, zakasnitev pri prenosu pa s τ. Da počasnost mrežne povezave ne postane omejilni faktor sistema, morata biti izponjena dva pogoja. Čas od sprejema naloge do oddaje rezultatov (čas T, ki ga procesorsko jedro potrebuje za reševanje dodeljene naloge) mora biti precej daljši od zakasnitve pri prenosu. Celotna količina podatkov (D), ki jih procesorsko jedro prejme pred računanjem oziroma odda po končanem računanju, mora biti dovolj majhna, da se podatki prenesejo v precej krajšem času, kot pa traja samo računanje. Oba pogoja lahko zapišemo matematično kot

$$T \gg \tau, \tag{5}$$

$$D/R \ll T. \tag{6}$$

Metaoptimizacija, pa tudi večina optimizacij inženirskih problemov, ki vključujejo takšne ali drugačne simulacije, zlahka izpolni oba kriterija. Pogoj (5) je izpolnjen, ker je čas T reda velikosti sekunde ali več, medtem ko znaša τ le nekaj $10\mu\mathrm{s}$. Vsako nalogo lahko podamo z naborom n' realnih števil (n' je dimenzija metaoptimizacijskega problema oz. število parametrov osnovnega postopka), njen rezultat pa z enim realnim številom. Če uporabljamo 64-bitni zapis števil s plavajočo vejico, je količina prenesenih podatkov (D) enaka $64 \cdot (n' + 1)$ bitov in velja $D/R = (n' + 1) \cdot 64\mathrm{ns}$. Če traja reševanje ene naloge zgolj $T = 1\mathrm{ms}$, je pogoj (6) izpolnjen za $n' < 15624$.

4 PROGRAMSKA OPREMA ZA OPTIMIZACIJO

Reševanje problemov v sodobnih visokozmogljivih računskih sestavih poteka tako, da vsako procesorsko jedro izvaja po en program, ki rešuje del problema. Programi komunicirajo in se tako medsebojno usklajujejo. Ponavadi poteka komunikacija v obliki sporočil, na katere se programi odzovejo z računanjem, ki mu sledi pošiljanje povratnih sporočil z rezultati. Programiranje se močno poenostavi, če uporabimo katero od knjižnic, ki pošiljanje in sprejemanje sporočil izvedeta s preprostim programskim vmesnikom (ang. Application Programming Interface, API). Ta skrije podrobnosti protokolov, prek katerih komunikacija dejansko poteka. Trenutno sta na voljo dve rešitvi: PVM [6] in MPI [7]. PVM je nekoliko starejša knjižnica, MPI pa je pravzaprav le specifikacija, ki predpisuje API za komunikacijo. Samih izvedb knjižnice MPI je veliko.

Za razvoj optimizacijskih postopkov je najprimernejše okolje, ki omogoča 1) preprosto programiranje zapletenih matematičnih postopkov in 2) hitro iskanje napak v programih. Prvemu pogoju ustreza precejšnje število programskih jezikov v kombinaciji z ustreznimi matematičnimi knjižnicami. Drugi pogoj ponavadi izpolnjujejo različni skriptni jeziki. Zaradi njegove enostavnosti, splošnosti in velikega števila knjižnic smo se odločili za Python [8] v kombinaciji z matematičnima knjižnicama NumPy/SciPy [9].

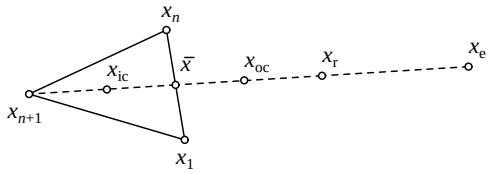
Metaoptimizacijo lahko pospešimo na dva načina. Pri prvem vsako procesorsko jedro izvaja enega od m tekov osnovne optimizacijske metode. Ko je vseh m tekov končanih, iz dobljenih rezultatov izračunamo vrednost kriterijske funkcije (2). Metaoptimizacijski postopek nato določi novo strategijo, ki jo spet ovrednotimo z m teki osnovnega optimizacijskega postopka. Teoretično lahko tako dosežemo do m-kratno pospešitev računanja. V praksi je ta precej manjša, saj se ne končajo vsi teki hkrati. Procesorska jedra, ki končajo prej, morajo čakati, da računanje končajo vsa jedra, preden jim je dodeljena nova naloga. Posledica je manjša pospešitev računanja. Pojavu pravimo sinhronizacijska kazen (ang. synchronization penalty).

Da se izognemo temu, v metaoptimizaciji uporabimo asinhroni vzporedni optimizacijski postopek, ki poskrbi, da je vsako procesorsko jedro ves čas zasedeno z računanjem. Tak postopek ne razdeli izračuna vrednosti KF med več procesorjev. To ima dodatno prednost, da je čas T zelo dolg in sta pogoja (5) in (6) izpolnjena tudi pri počasnih mrežnih povezavah. Primer takega postopka je opisan v [10].

5 PRIMER

Simpleksni postopek Nelderja in Meada [11] pri iskanju minimuma funkcije n spremenljivk premika množico $n+1$ točk (simpleks). Predpostavimo, da so točke urejene po pripadajoči vrednosti funkcije tako, da je ta najvišja pri x_{n+1} oziroma najnižja pri x_1. Večinoma se simpleks premika tako, da postopek zamenja najslabšo točko x_{n+1} z eno od točk, ki ležijo na poltraku z izhodiščem v x_{n+1}, ki je usmerjen proti centroidu $\overline{x}$ težišču (centroidu) n najboljših točk $\overline{x} = (x_1 + ... + x_n)/n$. Lego točk določata parameter γ in enačba

$$x = \overline{x} + \gamma(\overline{x} - x_{n+1}). \tag{7}$$



Slika 2: Koraki zrcaljenja, odmika, zunanjega primika in notranjega primika v postopku Nelderja in Meada

Za vrednosti parametra γ, ki jih označimo z γ_{r}, γ_{e}, γ_{oc} in γ_{ic}, dobimo štiri točke (x_{r}, x_{e}, x_{oc} in x_{ic}), ki nastopajo kot zamenjava za x_{n+1} (slika 2). Pripadajoče korake imenujemo tudi zrcaljenje, razteg, zunanji primik in notranji primik. Če nobena od teh točk ni boljša od x_{n+1}, skrčimo simpleks s faktorjem γ_{s} proti najboljši točki.

$$x_i \rightarrow x_1 + \gamma_{\mathrm{s}}(x_i - x_1), \quad i = 2,...,n+1 \tag{8}$$

Parametri γ morajo izpolnjevati pogoja $0 < \gamma_{\mathrm{r}} < \gamma_{\mathrm{e}}$ in $0 < \gamma_{\mathrm{oc}}, -\gamma_{\mathrm{ic}}, \gamma_{\mathrm{s}} < 1$. Priporočene vrednosti parametrov v [11] pomenijo strategijo

$$[\gamma_{\mathrm{r}}, \gamma_{\mathrm{e}}, \gamma_{\mathrm{oc}}, \gamma_{\mathrm{ic}}, \gamma_{\mathrm{s}}] = [1, 2, 1/2, -1/2, 1/2]. \tag{9}$$

Postopek lahko povzamemo z naslednjimi koraki:

1) Uredi točke, da velja $f(x_1) \leq f(x_2) \leq ... \leq f(x_{n+1})$.
2) Izračunaj $f_{\mathrm{r}} = f(\vec{x}_{\mathrm{r}})$.
   Če je $f_{\mathrm{r}} < f_1$, izračunaj $f_{\mathrm{e}} = f(x_{\mathrm{e}})$.
   Če je $f_{\mathrm{e}} < f_{\mathrm{r}}$, zamenjaj x_{n+1} z x_{e}, če ne pa z x_{r}.
3) Če je $f_1 \leq f_{\mathrm{r}} < f_n$, zamenjaj x_{n+1} z x_{r}.
4) Če je $f_n \leq f_{\mathrm{r}} < f_{n+1}$, izračunaj $f_{\mathrm{oc}} = f(x_{\mathrm{oc}})$.
   Če je $f_{\mathrm{oc}} < f_{\mathrm{n+1}}$, zamenjaj x_{n+1} z x_{oc},
5) Če je $f_{n+1} \leq f_{\mathrm{r}}$, izračunaj $f_{\mathrm{ic}} = f(x_{\mathrm{ic}})$.
   Če je $f_{\mathrm{ic}} < f_{\mathrm{n+1}}$, zamenjaj x_{n+1} z x_{ic}.
6) Če x_{n+1} ni bila zamenjana v korakih 2-5, skrči simpleks.
7) Če ustavitveni pogoji niso izpolnjeni, pojdi na 1).

Opis postopka v [11] ne opredeljuje podrobneje urejanja točk za primer, ko ima več točk enako vrednost f. Tedaj točke uredimo tako, da se od točk z enako vrednostjo f kot zadnja uvrsti tista, ki je doživela najmanj zamenjav.

V literaturi je znanih kar nekaj primerov (npr. [12]), za katere postopek ne konvergira proti minimumu funkcije. Kljub temu pa je precej priljubljen, saj je zelo preprost in nazoren. V postopku metaoptimizacije smo iskali optimalno strategijo, ki jo podaja $n' = 5$ parametrov postopka.

Kot nabor testnih funkcij smo uporabili funkcije iz [13], katerih večina je podanih v [1]. Po priporočilih [1] smo vsakega od 39 problemov preizkusili s tremi začetnimi točkami: x^0, $10x^0$ in $100x^0$. Pri tem x^0 označuje začetno točko problema, ki je podana v [1] oziroma [13]. Izjema je bila funkcija Jennricha in Sampsona, ki pri $100x^0$ ni definirana. Tako smo dobili $m = 3 \cdot 39 - 1 = 116$ testnih problemov. Kot referenčne vrednosti smo uporabili število izračunov KF in končne vrednosti gradienta KF, ki jih dobimo s konvergentno različico postopka [13] in neskaliranimi začetnimi točkami (x^0). Začetni simpleks in ustavitveni pogoji so bili enaki kot v [13].

Uporabili smo naslednje omejitve parametrov (strategije)

$$0{,}001 \leq \quad \gamma_{\mathrm{r}} \quad \leq 2, \tag{10}$$
$$0 \leq \quad \gamma_{\mathrm{e}} - \gamma_{\mathrm{r}} \quad \leq 4, \tag{11}$$
$$0{,}001 \leq \quad \gamma_{\mathrm{oc}}/\gamma_{\mathrm{r}} \quad \leq 0{,}999, \tag{12}$$
$$0{,}001 \leq \quad -\gamma_{\mathrm{ic}} \quad \leq 0{,}999, \tag{13}$$
$$0{,}001 \leq \quad \gamma_{\mathrm{s}} \quad \leq 0{,}999. \tag{14}$$

Metaoptimizacijo smo izvajali na sestavu, ki ga je tvorilo $M = 25$ računalnikov s procesorjem Intel Core i5 750 hitrosti $2{,}66\mathrm{GHz}$ in en nadzorni računalnik, ki je skrbel za povezavo z internetom (slika 1). Teoretična računska moč takega sestava je $1\mathrm{Tflops}$ (ob predpostavki, da je vsako procesorsko jedro sposobno izvesti do 4 operacije s plavajočo vejico na urni cikel). Seveda smo pri tem omejeni na probleme, ki izpolnjujejo pogoja (5)-(6).

Kot metaoptimizacijski postopek smo uporabili [10], ki je kombinacija simuliranega ohlajanja in diferencialne optimizacije. Zaključili smo ga po 200000 izračunih kriterijske funckije. Rezultat smo dobili po dveh dneh

računanja. Optimalno strategijo podajajo

$$\gamma_{\mathrm{r}} = 0{,}983, \tag{15}$$
$$\gamma_{\mathrm{e}} = 1{,}27, \tag{16}$$
$$\gamma_{\mathrm{oc}} = 0{,}734, \tag{17}$$
$$\gamma_{\mathrm{ic}} = -0{,}609, \tag{18}$$
$$\gamma_{\mathrm{s}} = 0{,}385. \tag{19}$$

Parametra γ_{r} in γ_{ic} sta zelo blizu vrednostma, ki sta ju predlagala Nelder in Mead. Drugi parametri se precej razlikujejo od privzetih vrednosti. Zanimivo je, da je optimalna vrednost $\gamma_{\mathrm{e}} = 1{,}27$ blizu vrednosti $1{,}2$, ki je predlagana v [14]. Parametra γ_{oc} in γ_{s} se razlikujeta od privzete vrednosti $0{,}5$.

Da bi preverili učinkovitost dobljene strategije, smo pognali osnovni optimizacijski postopek na 39 testnih problemih z neskaliranimi začetnimi točkami (x^0). Število izračunov KF smo omejili na 100000. Rezultati so podani v tabeli 1. Prvotni postopek je v dveh primerih porabil vse razpoložljive izračune KF, ne da bi izpolnil ustavitvene pogoje. Z optimalnimi vrednostmi parametrov se to ne zgodi za nobeno od uporabljenih kriterijskih funkcij. Z zvezdicami so označeni primeri, ko ena strategija prekaša drugo glede števila izračunov KF oziroma končne vrednosti norme njenega gradienta. V številu izračunov KF privzeta strategija prekaša optimalno v 23 primerih, medtem ko slednja prekaša privzeto v 16 primerih. Kar se tiče končne vrednosti gradienta je priveza strategija boljša v 14 primerih, optimalna pa v 25 primerih. Če preštejemo še primere, ko je ena strategija boljša od druge, tako glede na število izračunov KF kot tudi glede na končno vrednost norme gradienta, dobimo, da optimalna strategija prekaša privzeto v 10 primerih. Nasprotno se zgodi v 8 primerih.

Stolpci #r, #e, #oc, #ic in #s navajajo delež uspešnih korakov zrcaljenja, odmika, notranjega in zunanjega primika ter število korakov skrčenja. Delež korakov zrcaljenja se nekoliko zmanjša, zato pa se poveča delež uspešnih korakov odmika. Predvidevamo, da je to delno tudi razlog za uspešnost optimalne strategije. Pogostnost primerov, ko je optimalna strategija boljša od privzete, se veča z dimenzijo problema. To nakazuje, da je optimalna strategija odvisna od dimenzije problema. Število korakov skrčenja je majhno in se še dodatno zmanjša z uporabo optimalne strategije. Dobljena vrednost parametra $\gamma_{\mathrm{oc}} = 0{,}758$ ni zanesljiva, saj na korake zunanjega primika v večini primerov odpade le nekaj odstotkov vseh izračunov KF. Po drugi strani koraki notranjega primika pomenijo $10\% - 20\%$ vseh izračunov KF, kar daje verodostojnost dobljeni vrednosti $\gamma_{\mathrm{ic}} = 0{,}559$.

6 SKLEP

Izbira optimalnih vrednosti parametrov optimizacijskega postopka (strategije), ki določajo njegovo obnašanje, je računsko zahteven postopek. Avtomatiziramo ga lahko z uporabo zunanje optimizacijske zanke, ki išče njihove optimalne vrednosti. Postopek, ki teče v tej zanki, imenujemo tudi metaoptimizacija. Podali smo primer kriterijske funkcije za metaoptimizacijo. Sestavljena je iz prispevkov večjega števila testnih funkcij, na katerih preizkušamo osnovni optimizacijski postopek, za katerega iščemo optimalno strategijo.

Metaoptimizacija je računsko zahteven postopek, ki ga skoraj ne moremo izvesti brez uporabe visokozmogljivih računskih sestavov. Predstavili smo primer takega sestava s 100 procesorskimi jedri, ki je cenovno dokaj dostopen. Sestav temelji na standardnih namiznih računalnikih, ki jih povežemo z gigabitnim omrežjem Ethernet. Dobljeni sistem je primeren za vzporedno računanje, če je posamezna naloga, ki jo mora opraviti procesorsko jedro, dovolj obsežna, da so časi prenosa podatkov in zakasnitve pri prenosu v primerjavi z njo zanemarljivi.

Postopek smo ilustrirali z metaoptimizacijo petih parametrov (strategije) simpleksnega postopka Nelderja in Meada. Rezultati kažejo, da optimalna strategija še zdaleč ni enaka tisti, ki je bila objavljena skupaj s postopkom. Predvsem močno odstopa vrednost parametra, ki določa dolžino koraka odmika. Zanimivo je, da se dobljena vrednost dobro ujema s tisto, ki je bila predlagana za konvergentno različico simpleksnega postopka iskanja po mreži [14]. Rezultati tudi nakazujejo, da so optimalne vrednosti parametrov postopka odvisne od dimenzije optimizacijskega problema.

V nadaljevanju bi bilo zanimivo poiskati odvisnost optimalnih vrednosti parametrov postopka od dimenzije problema in ugotoviti, ali jo je mogoče razložiti tudi teoretično.

ZAHVALA

Raziskavo je omogočilo Ministrstvo za izobraževanje, znanost, kulturo in šport Republike Slovenije v okviru programa P2-0246 - Algoritmi in optimizacijski postopki v telekomunikacijah.

LITERATURA

[1] J. J. Moré, B. S. Garbow, K. E. Hillstrom, "Testing Unconstrained Optimization Software," *ACM Transactions on Mathematical Software*, vol. 7, no. 1, pp. 17–41, 1981.

[2] N. I. M. Gould, D. Orban, P.L. Toint, "CUTEr and SifDec: a constrained and unconstrained testing environment, revisited," *ACM Transactions on Mathematical Software*, vol. 29, no. 4, pp. 373–394, 2003.

[3] Á. Bűrmen et al., "Automated robust design and optimization of integrated circuits by means of penalty functions," *AEÜ, International Journal of Electronics and Communications*, vol. 57, no. 1, pp. 47–56, 2003.

[4] "Myrinet Express over Generic Ethernet Hardware," http://open-mx.gforge.inria.fr, april 2012.

[5] "Visokozmogljivi računski sestav HPCFS," http://hpc.fs.uni-lj.si, april 2012.

[6] V. S. Sunderam, "PVM: A Framework for Parallel Distributed Computing," *Concurrency: Practice and Experience*, vol. 2, no. 4, pp. 315–339, 1990.

Tabela 1: Primerjava privzete strategije in optimalne strategije za postopek Nelderja in Meada.

		privzeta strategija							optimalna strategija						
funkcija	dim.	N	$\|\|g\|\|$	#r	#e	#oc	#ic	#s	N	$\|\|g\|\|$	#r	#e	#oc	#ic	#s
Rosenbrock	2	* 231	4.8e-09	0.13	0.08	0.07	0.25	0.00	366	* 1.4e-09	0.15	0.10	0.06	0.23	0.00
Freudenstein&Roth	2	* 174	*7.1e-07	0.09	0.07	0.05	0.27	3.45	315	1.3e-06	0.16	0.10	0.05	0.20	1.90
Powell	2	* 752	*2.7e-08	0.24	0.11	0.02	0.18	0.00	1073	5.4e-08	0.22	0.16	0.03	0.14	0.00
Brown	2	* 320	*7.0e-02	0.09	0.12	0.06	0.25	0.00	713	1.1e-01	0.18	0.16	0.03	0.16	0.00
Beale	2	* 170	1.7e-09	0.11	0.05	0.08	0.28	0.00	241	* 3.8e-10	0.09	0.07	0.05	0.30	0.00
Jennrich&Sampson	2	* 154	*1.2e-05	0.10	0.03	0.05	0.31	2.60	191	1.8e-05	0.07	0.04	0.06	0.34	1.05
McKinnon	2	* 195	*1.7e-08	0.12	0.10	0.04	0.24	2.05	318	2.4e-08	0.12	0.17	0.02	0.21	1.26
McKinnon (alt)	2	247	1.0e+00	0.00	0.00	0.00	0.49	0.00	* 217	* 3.3e-08	0.11	0.01	0.05	0.34	1.84
Helical Valley	3	* 369	9.6e-09	0.15	0.11	0.04	0.25	0.00	561	* 8.5e-09	0.11	0.16	0.04	0.22	0.00
Bard	3	* 333	4.6e-09	0.14	0.08	0.08	0.24	1.80	439	* 4.0e-09	0.10	0.11	0.05	0.24	2.73
Gaussian	3	* 245	*1.3e-10	0.12	0.06	0.07	0.29	0.00	250	2.9e-10	0.11	0.00	0.05	0.36	0.00
Meyer	3	100002	*2.2e+01	0.22	0.00	0.11	0.00	32.8	* 2525	8.9e+01	0.26	0.17	0.03	0.10	2.61
Gulf R&D	3	3759	*9.8e-15	0.27	0.14	0.03	0.13	0.00	* 3317	1.8e-14	0.25	0.20	0.02	0.11	0.00
Box 3D	3	* 514	4.7e-09	0.13	0.14	0.03	0.23	2.33	656	* 3.2e-11	0.12	0.16	0.04	0.22	0.00
Powell Singular	4	* 1070	6.3e-16	0.27	0.09	0.04	0.19	0.00	1444	* 4.4e-16	0.20	0.14	0.04	0.18	0.00
Wood	4	* 683	5.4e-08	0.27	0.08	0.05	0.20	0.00	945	* 1.2e-08	0.25	0.11	0.05	0.19	0.00
Kowalik&Osborne	4	* 431	9.7e-10	0.24	0.05	0.06	0.21	3.71	551	* 9.1e-10	0.19	0.08	0.07	0.22	2.18
Brown&Dennis	4	* 490	9.8e-04	0.21	0.07	0.04	0.21	5.71	573	* 6.1e-04	0.20	0.08	0.06	0.21	4.19
Quadratic	4	* 359	*5.0e-10	0.17	0.05	0.06	0.28	0.00	510	9.0e-10	0.15	0.07	0.05	0.29	0.00
Penalty I	4	1404	5.4e-11	0.28	0.12	0.04	0.15	1.71	* 1227	* 9.5e-12	0.28	0.14	0.03	0.15	1.30
Penalty II	4	3768	1.5e-10	0.33	0.13	0.03	0.13	0.64	* 2924	* 9.2e-11	0.30	0.17	0.02	0.11	0.55
Osborne 1	5	* 1136	*2.5e-08	0.34	0.07	0.03	0.15	3.08	1277	2.9e-07	0.31	0.11	0.03	0.16	1.96
Brown	5	820	*2.2e-10	0.27	0.08	0.05	0.20	0.00	* 791	1.2e-09	0.23	0.08	0.03	0.25	0.00
Biggs EXP6	6	* 1127	6.6e-08	0.34	0.07	0.04	0.16	3.19	1184	* 1.1e-08	0.31	0.10	0.03	0.17	2.53
Rosenbrock	6	4321	*6.8e-09	0.38	0.11	0.02	0.13	0.00	* 2342	7.0e-09	0.36	0.13	0.02	0.14	0.00
Brown	7	1872	*7.6e-10	0.41	0.07	0.03	0.15	0.00	* 1248	1.2e-09	0.29	0.07	0.04	0.21	0.00
Quadratic	8	1783	1.8e-09	0.46	0.05	0.03	0.14	0.00	* 1166	* 1.7e-09	0.25	0.07	0.04	0.25	0.00
Rosenbrock	8	6181	7.0e-01	0.46	0.09	0.02	0.11	0.65	* 4046	* 1.0e-08	0.41	0.13	0.02	0.12	0.00
Var. Dim.	8	4044	6.6e-09	0.46	0.09	0.02	0.12	0.00	* 2368	* 5.6e-09	0.28	0.15	0.02	0.16	0.00
Powell	8	* 2579	1.3e-04	0.42	0.07	0.03	0.14	0.00	4770	* 6.9e-16	0.40	0.12	0.02	0.12	0.00
Watson	9	2953	*3.3e-08	0.35	0.13	0.03	0.13	1.22	* 2183	7.0e-08	0.23	0.20	0.02	0.14	1.37
Rosenbrock	10	* 6786	3.0e+00	0.51	0.08	0.02	0.10	0.74	11656	* 8.7e-09	0.47	0.14	0.01	0.08	0.00
Penalty I	10	* 5535	4.5e-05	0.50	0.08	0.02	0.11	1.08	6786	* 1.2e-10	0.46	0.13	0.01	0.09	0.88
Penalty II	10	* 6461	1.5e-04	0.47	0.08	0.02	0.13	0.62	9869	* 1.5e-05	0.45	0.15	0.01	0.08	0.61
Trigonometric	10	3196	1.7e-08	0.52	0.06	0.02	0.10	0.94	* 1457	* 3.8e-09	0.37	0.05	0.03	0.19	2.06
Osborne 2	11	4945	6.5e-08	0.55	0.05	0.02	0.09	0.89	* 2870	* 4.8e-08	0.44	0.08	0.02	0.14	1.53
Powell	12	* 7300	7.5e-04	0.55	0.06	0.02	0.10	0.00	12253	* 1.1e-15	0.50	0.12	0.01	0.08	0.00
Quadratic	16	9243	7.5e-09	0.66	0.03	0.01	0.06	0.00	* 3040	* 2.5e-09	0.42	0.05	0.02	0.19	0.00
Quadratic	24	100000	1.4e+00	0.76	0.03	0.01	0.02	0.00	* 6795	* 3.5e-09	0.60	0.04	0.02	0.11	0.00

[7] "MPI: A Message-Passing Interface Standard, Version 2.2, September 4, 2009", http://www.mpi-forum.org/docs/mpi-2.2/mpi22-report.pdf, april 2012. Version 2.2

[8] "Python Programming Language", http://www.python.org/, april 2012.

[9] "SciPy", http://www.scipy.org/, april 2012.

[10] J. Olenšek, T. Tuma, J. Puhan, Á. Bűrmen, "A new asynchronous parallel global optimization method based on simulated annealing and differential evolution," *Applied Soft Computing Journal*, vol. 11, no. 1, pp. 1481–1489, 2011.

[11] J. A. Nelder, R. Mead, "A simplex method for function minimization," *The computer journal*, vol. 7, no. 4, pp. 308–313, 1965.

[12] K. I. M. McKinnon, "Convergence of the Nelder–Mead simplex method to a non-stationary point,", *SIAM Journal in Optimization*, vol. 9, no. 1, pp. 148–158, 1999.

[13] Á. Bűrmen, T. Tuma, "Unconstrained derivative-free optimization by successive approximation," *Journal of computational and applied mathematics*, vol. 223, no. 1, pp. 62–74, 2009.

[14] Á. Bűrmen, j. Puhan, T. Tuma, "Grid restrained Nelder-Mead algorithm," *Computational optimization and applications*, vol. 72, no. 5, pp. 359–375, 2006.

Árpád Bűrmen se je rodil leta 1976 v Murski Soboti. Leta 2003 je na Univerzi v Ljubljani doktoriral s področja elektrotehnike. Zaposlen je kot izredni profesor na Fakulteti za elektrotehniko. Njegovo raziskovalno področje vključuje zvezno in dogodkovno simulacijo vezij in sistemov, optimizacijske metode, njihovo konvergenco in uporabo ter algoritme za vzporedno in porazdeljeno računanje. Je eden vodilnih razvijalcev simulatorja vezij SPICE OPUS. Objavil je več kot 20 člankov v recenziranih revijah.

Tadej Tuma je diplomiral (1988), magistriral (1991) in doktoriral (1995) na Fakulteti za elektrotehniko Univerze v Ljubljani. Na isti fakulteti je redni profesor, kjer poučuje štiri dodiplomske in tri podiplomske predmete. Njegovi raziskovalni interesi so predvsem na področju računalniške analize in načrtovanja vezij.

Iztok Fajfar je diplomiral (1991), magistriral (1994) in doktoriral (1997) s področja elektrotehnike na Fakulteti za elektrotehniko Univerze v Ljubljani. V letu 1991 je bil raziskovalec na Inštitutu Jožef Stefan v Ljubljani, nakar je konec leta zasedel raziskovalno mesto na Fakulteti za elektrotehniko Univerze v Ljubljani. Trenutno na fakulteti zaseda mesto izrednega profesorja. Poučuje več uvodnih in nadaljevalnih predmetov s področja računalniškega programiranja. Pri razvoju programske opreme je sodeloval pri več industrijskih projektih s podjetji Adacta in ICE Telecom. Njegovo področje raziskovanja vključuje načrtovanje in optimizacijo elektronskih vezij.