Plan dela – optimizacija s kompleksnim odzivom za potrebe projekta COBIK

Igor Grešovnik

Revision 0.2, May 2011
(revision 0: May 2011)

Vsebina:

1 PLAN DELA – OPTIMIZACIJSKI ALGORITMI

1.1 Optimizacija s kompleksnim odzivom

1.1.1 Predstavitev problema

Optimizacijske probleme definiramo v obliki problema minimizacije funkcije optimizacijskih parametrov, kjer so lahko parametri podvrženi določenim omejitvam:

$$\begin{aligned} &\text{minimize} \quad f(\mathbf{p}) \\ &\text{subject to} \begin{cases} c_i(\mathbf{p}) = 0;\ i \in E \\ c_j(\mathbf{p}) \geq 0;\ j \in I \end{cases} \end{aligned}$$

Pri tem je f namenska funkcija, s katero definiramo cilj optimizacije (npr. ocena akumulirane obrabe v delu orodja), c_i in c_j pa so omejitvene funkcije, ki določajo dovoljeno območje parametrov (skupno ime odzivne funkcije). Omejitve so lahko enakostne – npr. pozicija skrajne točke preoblikovanca po preoblikovanju je natančno določena – ali neenakostne, npr. volumen kanala po preoblikovanju mora biti večji od določene meje. Pri reševanju problema iščemo nabor parametrov $\mathbf{p}_0$, pri katerem je vrednost namenske funkcije najnižja na tistem območju v prostoru parametrov, kjer so izpolnjene vse omejitve.

Za reševanje optimizacijskih problemov obstaja zelo veliko število algoritmov, ki so spcializirani za različne posebne primere: minimizacijo brez omejitev, linearno programiranje (namenska in omejitvene funkcije so linearne), kvadratično programiranje (kvadratična namenska funkcija, linearne omejitvene funkcije), konveksno programiranje (konveksna namenska funkcija) itd.

Struktura algoritmov je lahko dokaj kompleksna, ker navadno reševanje bolj splošnih problemov poteka preko rešitve podproblemov s posebno strukturo – npr. reševanje kvadratičnega problema vključuje rešitev linearnega problema za določitev sprejemljive točke, rešitev splošnega problema vključuje zaporedno reševanje kvadratišnih problemov, reševanje omejenega problema lahko prevedemo na zaporedno reševanje problemov brez omejitev itd.

Metode reševanja so lahko zelo prilagojene naravi problemov, obstajajo na primer posebne metode za primere z velikim številom spremenljivk ali posebno strukturo matrike drugih odvodov,

metode za probleme, kjer se da razdvojiti problem na šibko sklopljene probleme (nekakšen "multiscale" za optimizacijo), ter "multigrid" metode, ki so zelo vezane na specifične probleme in pri katerih med reševanjem prilagajamo natančnost (npr. št. prostostnih stopenj, št. parametrov oblike) numerične analize. Nekatere metode delujejo samo na podlagi izračunov namenskih in omejitvenih funkcij, druge uporabljajo gradiente in te so navadno veliko bolj efektivne. Pri večini deterministčnih metod lahko zagotovimo le konvergenco k lokalnemu minimumu. Poleg teh se včasih uporabljajo nedeterministične metode (simulated annealing, genetski algoritmi, particle swarm, differential evolution...), s katerimi lahko z večjo verjetnostjo zadenemo globalni minimum ali določimo več lokalnih minimumov hkrati. Pri teh metodah navadno potrebujemo zelo veliko število izračunov, da pridemo do rešitve, poleg tega niso sposobne dobro upoštevati odvodov.

V naših primerih večinoma nastopajo splošni problemi, ko so namenske in omejitvene funkcije niso linearne ali kvadratične. V redkih primerih bi morda lahko shajali brez omejitev ali pa so predpisane le eksplicitne meje po parametrih. Poleg tega izračunavamo namensko in omejitvene funkcije preko numerične simulacije sistema, ki ga natančno definirajo šele optimizacijski parametri. Sama simulacija je navadno časovno zelo zahtevna, tako da lahko v večini primerov kot merilo za učinkovitost algoritmov upoštevamo skoraj izključno potrebno število izračunov namenske in omejitvenih funkcij ter njihovih gradientov. Notranja kompleksnost samih algoritmov (t.j. reševanje delnih problemov, npr. sistemov enačb ali kvadratičnega problema) bo verjetno relativno malo prispevala k časovni zahtevnosti. Pri funkcijah lahko včasih iz fizikalne narave problema sklepamo na nekatere splošne značilnosti (npr. monotona odvisnost od danega parametra), v veliko primerih pa lahko o tem bolj malo uganemo. Zaradi tega tudi o enoličnosti rešitev ne moremo zanesljivo sklepati. Zaradi časovne zahtevnosti navadno ne pridejo v poštev bolj robustni algoritmi (npr. genetski). V praksi se lahko zadovoljimo z dobrim lokalnim minimumom, ki pomeni bistveno izboljšavo glede na izhodiščni načrt, poskusimo z zelo različnimi začetnimi približki in uporabimo najboljšo rešitev, ali pa na grobo pretipamo prostor parametrov okrog rešitve. Velikokrat ni tako pomembno, da natančno izračunamo minimum, ampak želimo predvsem doseči znatno izboljšanje glede na začetni dizajn. To bo nekoliko drugače pri inverznih problemih, kjer bo dovolj natančen izračun minimuma zelo pomemben, problem pa bomo poskusili že konstruirati tako, da se bo dalo obiti težave z veliko lokalnimi minimumi.

Poleg časovne zahtevnosti izračuna namenske in omejitvenih funkcij je pri problemih, s katerimi bomo imeli opravka, stalen in velikokrat prevladujoč problem numeričen šum. Neizogibna posledica napak numeričnega modela, ki ga uporabimo v optimizaciji, je nenatančen izračun namenske in omejitvenih funkcij, zaradi česar je nenatančen tudi rezultat optimizacije. Dodatna praktična težava povezana s tem je negladkost izračunanih namenskih in omejitvenih funkcij, kar vpliva na delovanje optimizacijskih algoritmov. Najbolj učinkoviti klasični algoritmi temeljijo na privzetku o zvezni odvedljivosti funkcij in so navadno zelo občutljivi na gladkost funkcij. Težave zaradi negladkosti odziva so lahko različne. Če ima šum veliko amplitudo, se lahko algoritem neurejeno in brez konca lovi v prostoru parametrov, ker ne uspe zadovoljiti konvergenčnih kriterijev. V boljšem primeru lahko algoritem detektira, da odzivne funkcije niso konsistentne s privzetki algoritma in se ustavi z obvestilom o napaki. Če je amplituda šuma dovolj mala, lahko algoritem na začetku kaže zadovoljive performance, v bližini minimuma pa se obnaša na prej opisan način in ne konvergira. V primeru, da ima šum dovolj malo amplitudo, lahko uporabimo običajne optimizacijske algoritme, pri tem pa je treba paziti, da tolerance ne postavimo preveč na tesno. To v večini primerov ni tako trivialno, kot se sliši, ker je to, kaj je zadosti tesna toleranca

odvisna od tega, kako so parametri skalirani (v primeru, ko imajo optimizacijski parametri različen pomen, težko računamo na dobro skaliranje) ter od lokalnih lastnosti odzivnih funkcij v bližini rešitve, česar ne poznamo vnaprej. Subjektivne ocene o tem, kaj je primerna toleranca, ko poženemo optimizacijski algoritem, so lahko hitro v razponu nekaj velikostnih redov. V praksi navadno postavimo tolerance tako, da so "zelo verjetno dovolj na tesno", vendar to dobro deluje le, če je odziv dovolj gladek in imamo algoritem, ki lokalno konvergira bolje kot linearno.

Ovira za učinkovito optimizacijo je pogosto tudi v tem, da nimamo na voljo učinkovitih postopkov za izračun gradientov odzivnih funkcij po optimizacijskih parametrih. Če je optimizacijski problem definiran z analitičnimi odzivnimi funkcijami, lahko gradiente računamo numerično. To zahteva po en dodaten izračun odzivnih funkcij za vsak parameter v vsaki točki, kjer računamo odziv in metoda z nekaj manjšimi modifikacijami navadno deluje, je pa v številnih primerih možno konstruirati učinkovitejše metode, ki temeljijo na aproksimacijah odziva. V primeru, ko odzivne funkcije računamo z zahtevnimi numeričnimi simulacijami, tak način skoraj nikoli ne pride v poštev.

1.1.2 Cilji

Grobo rečeno potrebujemo algoritme, ki lahko ob prisotnosti šuma v numeričnem odzivu najdejo parametre, ki so blizu optimalnim ali ki vsaj predstavljajo znatno izboljšanje glede na začetni približek, in to s čim manjšim številom izračunov namenske in omejitvenih funkcij. Zelo pomembno je, da takšni algoritmi delujejo dobro pri različnih nivojih šuma (tudi pri gladkih odzivih), ker pri reševanju problemov navadno ne bomo imeli časa za preizkušanje, kako se na danem problemu obnesejo različni algoritmi. Tipično pričakujemo precej nelinearne odzivne funkcije, kar pomeni, da se bo vpliv šuma med potekom optimizacijskega postopka spreminjal in potrebujemo dovolj prožne adaptivne postopke.

Takšna definicija je nekoliko ohlapna. Postavljanje kriterijev za ocenjevanje performanc algoritmov za funkcije z negladkim odzivom je na splošno težje kot pri gladkem odzivu. Pri algoritmih za gladke funkcije je enostavno merljiv performančni kriterij vezan na red in hitrost lokalne konvergence. Performance algoritmov daleč od rešitve navadno primerjamo preko dovolj velikega nabora različnih testnih primerov (na osnovi tega je na primer postavljena trditev, da ima BFGS boljše performance od DFP) in na podlagi izkušenj.

Pri optimizaciji s šumom ne moremo govoriti o redu konvergence v klasičnem pomenu besede, saj problem v resnici sploh nima rešitve, kjer bi bili izpolnjeni potrebni in zadostni pogoji za lokalni minimum. Performance algoritmov za optimizacijo s šumom lahko primerjamo na testnih primerih, pri katerih za osnovo vzamemo gladek odziv, ki mu je dodan šum z določenimi karakteristikami. Pri tem lahko opazujemo, kako blizu je rešitev, ki jo izračuna tak algoritem, rešitvi problema brez šuma, v odvisnosti od amplitude šuma, ter za kakšno ceno lahko rešitev bolj približamo rešitvi gladkega problema. Lahko tudi statistično primerjamo napredovanje algoritma v primerjavi s klasičnimi algoritmi na gladkem odzivu (npr. merimo število potrebnih izračunov, da se algoritem rešitvi gladkega problema približa pod določeno razdaljo).

Na podlagi opisanega lahko bolj konkretno definiramo željene lastnosti algoritmov:

- Performance naj bi se v limiti, ko je odziv gladek, približale performancam učinkovitih klasičnih algoritmov. Algoritem naj bi dosegel takšne performance, ne da bi podali

informacijo o tem, da so odzivne funkcije gladke, ker mora biti algoritem uporaben brez vnaprejšnjega znanja o naravi odziva.

- Rezultat algiritma mora v primeru, ko ni dodanega šuma, sovpadati (v okviru predpisanih toleranc) z rešitvijo gladkega problema.
- Pri postopnem večanju amplitude šuma mora natančnost rezultata »zvezno« padati. To pomeni, da algoritem ne sme naenkrat odpovedati, ko raven šuma preseže nek prag, ki je v dovolj zmernem območju.
- Dodatek zmerne količine šuma v območju dovolj daleč od rešitve gladkega problema ne sme drastično poslabšati performanc.
- Algoritem mora avtomatično detektirati, kakšna je največja možna natančnost rezultata glede na nivo šuma v odzivu. Če predpisane tolerance presegajo to natančnost, mora algoritem končati, ko ni več možno z zmernim naporom bistveno povečati natančnosti, ter dati indikacijo, da je natančnost rezultata nižja od zahtevane.
- Vrniti mora oceno natančnosti rezultata, temelječo na predpostavki, da je odziv sestavljen iz natančnega gladkega dela in nekoreliranega šuma. Če je amplituda šuma prevelika za kakršnokoli zanesljivo izboljšavo, mora vrniti indikacijo o tem.
- Največja dosežena natančnost mora biti primerljiva z največjo "možno" natančnostjo, ki jo lahko dosežemo z zelo robustnimi (in časovno zelo zahtevnimi) metodami.
- Zaželjeno je, da obstaja možnost vpliva na kompromis med večjo natančnostjo in manjšim računskim naporom.

Nekatere od naštetih zahtev je težko bolj precizno definirati, predvsem tiste povezane s performancami. Pri oceni performanc v limitnem primeru brez šuma se lahko opremo na najboljše znane klasične algoritme. Pri znatni ravni šuma je manj jasno, kaj vzeti za referenco, in lako primerjamo samo en algoritem z drugim. Zelo pomembno je, da obstaja ta možnost, ker lahko pri različnih algoritmih pričakujemo velike performančne razlike, poleg tega lahko možnost objektivne primerjave izkoristimo za fino nastavitev parametrov in strategije algoritmov. Primerjalni testi bodo morali biti standardizirani statistični testi, ki jih lahko natančno definiramo.

Poleg zgornjih zahtev, ki so bistvene za uporabnost algoritma, lahko definiramo še **seznam želja** glede koristnih lastnosti za praktično rabo:

- Zelo koristna je možnost shranjevanja vmesnih rezultatov v podobni obliki kot pri končnih rezultatih. Tako lahko v primeru, da je izračun končnega rezultata preveč časovno zahteve, uporabimo vmesen rezultat. Pri optimizaciji z omejitvami je to težje doseči kot pri optimizaciji brez omejitev in potegne za sabo generacijo približkov, ki so v dovoljenem območju glede na parametre.
- Koristna bi bila možnost uporabniške interakcije med potekom – spremljanje vmesnih rezultatov in nastavljanje parametrov, s katerimi določimo, na kateri strani kompromisa med natančnostjo in manjšo časovno zahtevnostjo smo.
- Paralelizacija. Predvideni algoritmi so zelo primerni za paralelizacijo.
- Možnost izogibanja lokalnim minimumom daleč od rešitve s težnjo povečevanja območja vzorčenja.
- Morda integracija s stohastičnimi algoritmi; v to se lahko spustimo, če bo dovolj časa in če se pokažejo indikacije, da bi bilo to koristno.

1.1.3 Osnovna ideja, poznavanje problemov in obstoječe stanje

Večji del klasičnih optimizacijskih algoritmov temelji vsaj posredno na postopni izgradnji aproksimacije odziva z enostavnim modelom ter na zaporednem reševanju enostavnejšega problema, ki ga dobimo z zamenjavo odzivnih funkcij s funkcijami modela. Postopek izgradnje modela velikokrat temelji na razvoju v Taylorjevo vrsto, model pa z zajemanjem novih podatkov sproti izboljšujemo in prilagajamo lokalnim lastnostim odziva. Bistvene komponente so še strategija, kako s čim manj računskimi operacijami in dodatnimi izračuni odzivnih funkcij zgradimo čim boljšo lokalno aproksimacijo odziva ter različni varnostni mehanizmi za zagotovitev globalne konvergence. Med te mehanizme spadata osnovna prototipna algoritma – »line search«[1] in » restricted step«[2].

Takšni algoritmi temeljijo na določenih predpostavkah o odzivnih funkcijah (zvezna odvedljivost in nekatere druge). Strategija izgradnje modela in varnostni mehanizmi so zasnovani na predpostavki, da odziv sicer odstopa od modela, vendar ga lahko z modelom lokalno aproksimiramo. Treba je poudariti, da pri učinkovitih algoritmih ne težimo vedno k čimbolj natančni lokalni aproksimaciji odziva, ampak gre za kompromis med natančnostjo in čimmanjšim potrebnim številom ovrednotenj odziva. Natančnejša lokalna aproksimacija odziva se zgradi šele blizu rešitve.

Pri optimizaciji s šumom predpostavljamo, da je odziv sestavljen iz »gladkega dela«, ki mu je dodana naključna komponenta. Predstavljamo si lahko, da gladek del sovpada z resničnim odzivom fizikalnega sistema in zanj veljajo podobne predpostavke, kot pri konstrukciji klasičnih algoritmov, vendar je ta »pravi« odziv zabrisan s šumom. Algoritem mora biti zasnovan tako, da čim bolj učinkovito in natančno ugane rešitev problema, ki ga določa »pravi odziv«, pri tem pa lahko zajema le zabrisan odziv, ki vsebuje šum.

Algoritmi za optimizacijo s šumom bodo zato temeljili na podobnih osnovnih idejah kot klasični, le da bo potrebno upoštevati pomembno razliko. Lokalno polinomsko aproksimacijo gladke funkcije lahko konstruiramo tako, da vzorčimo funkcijo v končni množici točk v okolici točke, okrog katere aproksimiramo funkcijo, ter z dobljenimi podatkih izračunamo aproksimacijo po metodi najmanjših kvadratov. Če vzorčne točke krčimo proti točki, v kateri aproksimiramo funkcijo, dobljene aproksimacije konvergirajo k Taylorjevemu razvoju funkcije v tej točki. Če vzorčimo funkcijo, ki ji je dodan šum, to ni več res, z vzorčenjem v okolici točke ne moremo konstruirati poljubno dobre lokalne aproksimacije. Če vzorčimo daleč od točke, v kateri aproksimiramo funkcijo, bo napaka zaradi dodanega šuma manjša, večja pa bo napaka zaradi odstopanja gladkega dela od Taylorjevega razvoja. Za dano število predpisano razporejenih vzorcev obstaja optimalna velikost območja vzorčenja, pri kateri dobimo aproksimacijo, ki je najbljižja

[1] Zaporedoma izvajamo minimizacijo v eni dimenziji, kjer lahko omejimo območje (interval), na katerem je zanesljivo vsebovan minimum – podobno kot pri reševanju enačbe z bisekcijo, le da so potrebne tri točke za dokaz, da je minimum vsebovan na intervalu. Kvadratični model gradimo na podlagi vrednosti in odvodov v točkah dobljenih po linijski minimizaciji, uporabimo pa ga za določitev smeri linijskih minimizacij.

[2] Omejimo območje veljavnosti modela; zaporedoma rešujemo problem, kjer dejanski odziv nadomestimo z modelom, z dodatno omejitvijo na velikost koraka. Omejitev velikosti koraka prilagajamo glede na dosežen učinek in razliko med vzorčenimi vrednostmi pravega odziva ter napovedmi modela. Podproblem, ki ga zaporedoma rešujemo, je QP (kvadratična namenska funkcija z linearnimi omejitvenimi funkcijami) in je za konveksno namensko funkcijo eksaktno rešljiv v končnem številu korakov.

Taylorjevemu razvoju. Opis je sicer malo poenostavljen, ker se v praksi uporabljajo tudi aproksimacije, ki niso neposredno izpeljane iz Taylorjevega razvoja.

Shematski primer konstrukcije algoritma je nasleden. Osnovna je v lokalni aproksimaciji odziva na podlagi zajema končnega števila vzorcev. V poštev pridejo npr. linearne ali kvadratne aproksimacije. Funkcijo vzorčimo v končnem številu točk in potem kot model v vsaki točki prostora parametrov uporabimo lokalizirano aproksimacijo, kjer vzorčene vrednosti utežimo z utežmi, ki padajo z razdaljo od vzorčnih točk (»moving least squares« ali več »weighted least squares«). S funkcijsko odvisnostjo uteži od razdalje lahko spreminjamo efektivni radij gruče vzorčnih točk, ki bistveno prispevajo k aproksimaciji. Pri večji amplitudi šuma izberemo večji efektivni radij, kar pomeni boljše glajenje oscilacij, ki so posledice šuma, na račun manj lokalizirane aproksimacije. Cilj je skonstruirati algoritem, ki sam sproti določa in prilagaja optimalen efektivni radij. Pri večjem številu parametrov ne moremo vzorčiti po celotnem vnaprej določenem območju, edini sprejemljiv pristop je sprotno vzorčenje okrog vsakokratnega pribilžka optimuma.

V literaturi so že opisane metode, ki delujejo po tem načelu. Obstoječe metode imajo dve večji pomanjkljivosti. Prva je v tem, da se celotno vzorčenje za aproksimacijo okrog trenutnega približka izvede naenkrat. Druga pomanjkljivost je v tem, da niso dovolj dobro razvite strategije prilagajanja parametrov vzorčenja in aproksimacije (velikost območja vzorčenja, število vzorcev, ki se upoštavajo pri aproksimaciji, ter meje uporabe aproksimacije). Pri malo večjem številu parametrov lahko verjetno z boljšo strategijo hitro nekajkrat izboljšamo efektivnost algoritma. Efektivnost je ključnega pomena, ker po eni strani določa mejo kompleksnosti problemov, pri kateri si še lahko privoščimo optimizacijo, pod to mejo pa določa, koliko časa imamo za »igranje s problemom« - korekcijo optimizacijskega problema, ko vidimo, kakšne so lastnosit rešitve in podobno.

Nekaj tehničnih stvari, ki jih bo potrebno rešiti pri konstrukciji algoritmov:

- Takšna strategija vzorčenja, da je aproksimacijski problem dobro definiran; varnostni mehanizmi v zvezi s tem, regularizacija.
- Uporaba informacij iz prejšnjih iteracij (predvsem v bližini minimuma)
- Optimalna strategija vzorčenja (dovolj enakomerno razporejeni vzorci, pogojenost aproksimacijskega problema)
- Prepreševanje osciliranja. Regularizacija z aproksimacijami iz prejšnjih iteracij. Primerna strategija uteževanja med vztrajnostjo in prožnostjo pri prilagajanju lokalnim lastnostim odziva.
- Osnovna strategija uteževanja vzorcev pri konstrukciji aproksimacij v odvisnosti od razdalje od »težišča« aproksimacije, razdalje od »najboljše točke«, vrednosti funkcije itd.
- Možnost filtriranja napačnih vzorcev. To je pomembno v primerih, ko je možno, da pri določenih parametrih numerična analiza odpove. V praksi lahko skoraj zanesljivo računamo s tem.
- Optimalna strategija uravnavanja pomembnih parametrov algoritma:
  - Velikost območja zaupanja aproksimacije, na katerem rešimo modelni podproblem
  - Velikost območja vzorčenja
  - Število presežnih vzorcev
  - ??

- Mehanizmi za preverjanje, če so zgornji parametri dobro nastavljeni (npr. detekcija oscilatornega obnašanja) s spremljanjem delovanja algoritma čez več iteracij.
- Ustavitveni kriteriji. Temeljili bodo na oceni natančnosti rešitve in možnosti nadaljnje izboljšave glede na količino šuma.
- Razdelitev na različne faze s spremenjenim delovanjem: začetek, normalen potek in zaključni del s konvergenco ali ustavitvijo zaradi doseganja najboljše možne natančnosti glede na šum in dopusten napor.
- Strategija za primer, ko so na voljo gradienti odzivnih funkcij (ko imamo senzitivnostno analizo).
- Uravnavanje napora potrebnega za preverjanje in optimalno nastavitev parametrov (spomin in procesorski čas). Pri klasičnih algoritmih je v večini primerov, ki so nam v interesu, poraba virov za izvedbo algoritmov neznatna v primerjavi s porabo virov za izračun odzivnih funkcij. Pri predvidenih algoritmih za optimizacijo s šumom morda v nekaterih primerih to ne bo več res. Strategije za optimalno prilagajanje parametrov, s katerimi želimo do skrajnosti zmanjšati potrebno število izračunov odzivnih funkcij za dano natančnost, bo vključevala časovno zahtevne operacije, vsaj pri večjem številu parametrov (~ nad 50).
- Učenje o lastnostih odziva z vplivom na strategijo uravnavanja parametrov – vpeljava »inteligence«.

Velikega dela tehničnih vidikov naštetih zgoraj ni potrebno rešiti v naprej in tudi ne bodo prišli v poštev pri vseh problemih, ki jih bomo reševali. Zelo pomembno pa je, da se algoritemska baza zgradi na način, ki bo omogočal enostavno širitev in dopolnjevanje. Pomembno je tudi, da se pravočasno in intenzivno začne graditi in preizkušati osnovo.

Glede na zgornje postavke je morda videti konstrukcija takšnih algoritmov zelo zahtevna stvar, vendar je najverjetneje večina od opisanih problemov zadovoljivo rešljiva z uporabo trenutnega znanja in dobrega numeričnega občutka. Optimalna strategija uravnavanja parametrov ni tako eksaktna in kritična stvar, da bi morali tu pričakovati nerešljive probleme. V prid temu govorijo izkušnje pri uravnavanju območja zaupanja pri klasičnih »restricted step« algoritmih, kjer se izkaže, da so algoritmi relativno neobčutljivi na faktor prilagajanja območja zaupanja.

Groba delitev metod metod:
- Optimizacija brez omejitev / z omejitvami
- Optimizacija brez uporabe gradientov / z uporabo gradientov

1.1.4 Plan dela

Vodi: I.G.
Izvajalec: pretežno I.G. Morda bi lahko kdo pomagal pri kakšnih komponentah.
Predviden začetek: ? Konec poletja?

Predvideno trajanje: Prva etapa okrog 2-3 leta (odvisno od intenzivnosti dela in zahtev, ki se bodo pokazale, ko bomo imeli narejen numeični modle in definiranih nekaj konkretnih problemov. Rezultat: uporabna algoritemska osnova, ki bo uporabna za industrijske primere. V nadaljevanju izboljšave glede na izkušnje ter potrebe pri projektih, možno nadaljevanje, če se bo to izkazalo za pomembno (implementacija stvari z liste želja). Nadaljevanje razvoja je lahko priložnostno ob industrijskih projektih, ali pa se za to namensko pridobi kak projekt.

Možnosti za pridobitev dodatnih namenskih sredstev za razvoj algoritmov: MZT, evropski projekti, iskanje zainteresirane tretje strani (močnejši industrijski partner).

1.1.4.1 Grobi plan dela za prvo fazo od začetka naprej[1] (razdelitev osnovnih nalog, grobe ocene glede trajanja)

1. **Pregled in študij literature**. Obseg **4-6 PM**, začetek M1, trajanje do konca, skoncentrirano v M1 (do 1 **PM**).
2. **Izbira in implementacija testnih primerov**. Obseg **3-4 PM**, začetek M3, trajanje do konca prve faze. Zahteve: dovolj težak gladek odziv + enostaven gladek odziv, modeli z omejitvami in brez njih, izbira poljubnega števila optimizacijskih parametrov, možnost spreminja skaliranja parametrov, sukanja odzivnih funkcij in izbire nastavitve pogojenosti matrike drugih odvodov, možnost izračuna minimuma z gladkim odzivom, testi za omejeno in neomejeno optimizacijo, možnost nastavitve amplitude šuma na dva načina: absolutno in relativno (glede na razliko med vrednostjo gladke namenske funkcije v vzorčni točki in vrednostjo v optimumu). Možnost koreliranega šuma?
   2.1. **Izdelava statističnih testov za primerjavo algoritmov**. Obseg **2 PM**, začetek ob testiranju 1. algoritma, trajanje čez celotno 1. fazo (dopolnjevanje pri vgraditvi novih algoritmov)
3. **Pregled dosegljivih obstoječih algoritmov primernih za naše zahteve**. Obseg **4-6 PM**, začetek M1, trajanje skozi celotno 1. fazo.
   3.1. Odločanje o uporabi algoritmov, ki jih dobimo od zunaj.
   3.2. Vgradnja algoritmov, ki jih dobimo od drugje. Tu lahko pomaga Tadej Kodelja, ker se to prekriva z njegovim delom.
   3.3. Testiranje algoitmov, ki jih dobomo od zunaj.
4. Algoritmi za reševanje vmesnih problemov. Trajanje od M3, potencialno do malo pred koncem 1. faze.
   4.1. Izračun aproksimacije z regularizacijo. Obseg **2-3 PM**, Trajanje od M1 do M4, manjši kosi morda še pozneje.
   4.2. Algoritmi za izbiro vzorčnih točk. Obseg **do 2 PM**, trajanje od M2 do M5, manjši kosi morda še pozneje.
   4.3. Optimizacijski algoritmi. Obseg **3-5 PM**, trajanje od M1 do M6.
      4.3.1. QP s konveksno namensko funkcijo
      4.3.2. QQP – splošen (konveksne ali nekonveksne funkcije)
      4.3.3. Nelinearno programiranje
      4.3.4. Genetski algoritmi?
   4.4. Ostalo. Obseg **2-3 PM**.

[1] Zaenkrat je to samo groba predstava, plan se bo dopolnjeval glede na izkušnje.

5. **Enostaven prototipni algoritem – brez odvodov, brez omejitev**. Obseg **2 PM**, začetek M3, trajanje do M7
6. **Enostaven prototipni algoritem - brez odvodov, z omejitvami**. Obseg **2-3 PM**, začetek M3, trajanje do M11.
7. **Izboljšava adaptivne strategije**. Obseg **2-3 PM**, trajanje M6-M12.
8. **Vključitev gradientne informacije**. Obseg **1-2 PM**, trajanje M8-M12. Morda lahko to zaenkrat izpustimo.
9. **Testiranje algoritmov**. Obseg **3-4 PM**, začetek ob prototipni izvedbi 1. algoritma, trajanje skozi celotno 1. fazo.
10. **Iskanje sredstev za nadaljevanje dela**. Obseg **3 PM**, začetek pred iztekom prve faze in pred M12, trajanje do uspeha ali opustitve cijla. Izvedba Igor G., Božidar Š. in ostali.
11. **Druga faza – nadaljnja izboljšave algoritmov**. Obseg **0-12 PM**, začetek po izteku prve faze.

Legenda:

- **PM**= mesec dela (person month)
- M1 = prvi mesec od začetka dela na projektu, M2 drugi mesec itd.

Celoten obseg 1. faze (seštevek): **23-29 PM**. Minimum sem postavil tako, da bi lahko prišli do delujočih orodij za naše delo. Ob normalnem razvoju dogodkov je maksimum naravnan tako, da morda ne bo potrebna druga faza, vsaj v okviru trenutne rabe znotraj projekta COBIK.

Plan se bo prilagajal glede na situacijo. V tem trenutku je preveč neznanih stvari, da bi ga lahko jemali kot dokončnega.

2 PLAN DELA – OPTIMIZACIJSKO OKOLJE

Časa za kakšno zelo sistematično narejeno optimizacijsko okolje verjetno v okviru projekta ne bo. Poudarek bi bil na razvoj orodij, modulov in standardov, ki bodo omogočili dovolj fleksibilno nastalvjanje in reševanje različnih problemov. Minimalne zahteve so naslednje:

- Standard za vključevanje tazličnih modulov za analizo
  - Oblika vhodnih in izhodnih podatkov določena
  - Implementirane osnovne oblike interakcije (npr. preko datotek)
- Standard za vključevanje optimizacijskih algoritmov
  - Vključeni morajo biti tako, da delajo s standardno obliko analize, potem so avtomatično povezlljivi z različnimi oblikami definicije analize.
- Sistem za definicijo optimizacijskega problema in načina eševanja
  - Hard-coaded – vnaprej definirane možnosti v kodi, najbolj primitiven način, trenutno se uporablja v projektu štore, ker ni bilo možno najti konsenza of bolj naprednem načinu in še ni bilo možno izvesti
  - Hard coaded, vendar tako, da se problemi nastavijo v projektih, ki uporabljajo sistem kot knjižnico. Projekt v C# je potem del projekta za

rešitev problema, poskuša se vzdrževati določena stopnja kompatibilnosti za nazaj.

- Fleksibilen način, definicija preko interpreterja, v katerem so preko inštaliranih metod vgrajena potrebna orodja za interakcijo z direktnimi analizami, optimizacijski algoritmi itd.

Dobro bi bilo imeti še (»wish list«):

- Nastavke za paralelno poganjanje analiz (nekaj osnovnih načinov)
- Orodja za za preverjanj rešitev (neodvisna od algoritma, vendar tako, da se lahko uporabi zadnje izračune pri algoritmu)
- Orodja za tipanje rešitev
- V primeru na aproksimaciji temelječih algoritmov – standard za »restart« in orodja za analizo poteka optimizacije

3 PLAN DELA – OSTALO

3.1 Tadej Kodelja

Za tadeja je treba definirati, kaj bo delal na projektu. Ena od stvari bo aproksimcija z nevronskimi mrežami. Smiselno je, da se ostale stvari določi tako, da bodo s tam povezane in da bo čimbolje dopolil delo mene in Katarine na projektu.

Predlog:

Rdeča nit bi bila manipulacija z numeričnim modelom za industrijske potrebe. Poudarek je lahko na softverskem konceptu in uporabi numeričnega modela ter povezavi med numeričnim modelom in optimizacijo. Možna področja, ki jih pokrije:

- Modul za globalno aproksimacijo odzivov na podlagi izmerjenih podatkov iz industrijskega procesa ali na podlagi numeričnega modela
- Aproksimacija z nevronskimi mrežami
  - Dopolnitve mojega modula (npr. omogočenje uporabe ločenega vzorčenja za vsak izhodni parameter posebej)
  - Vgradnja dodatnih osnovnih knjižnic, primerjava med njimi, izboljšanje algoritmov
  - Implemetacija algoritmov za ravnanje z nekvalitetnimi podatki (filtriranje, preizkušanje ustreznosti aproksimacije)

        - Algoritmi za preizkušanje ustreznosti postopka aproksimcije (parametri mreže, postopek učenja...)
        - Pomožna orodja za ocenjevnje kvalitete aproksimacij, npr. primerjava z drugimi tipi aproksimacije, ki so matematično bolj obvladnljive.
    - Druge vrste aproksimacije, npr. MLS, Krieging
        - Orodja za povezavo numeričnega modela z aproksimacijskimi in optimizacijskimi moduli
        - Orodja za procesiranje rezultatov numeričnega modela (branje, prikazovanje, morda izračun izpeljanih količin)
        - Orodja za parametrizacijo, če bo poterba za to in bo čas to dopuščal
        - Obdelava merskih podatkov, priprava za inverzne analize itd.
        - Vgrajevanje in testiranje algoritmov za optimizacijo, ki jih lahko dobimo drugje
    - Vključevanje v sistem po standardih, ki jih definiram za optimizacijsko okolje
        - Poenoten način za zapis vhodnih podatkov za optimizacijo (začetni približki, standardni parametri kot so tolerance, specifični parametri)
        - Implementacija orodij za interakcijo z algoritmi in procesiranje rezultatov (npr. grafi konvergence itd.)
        - Inplementacija testnih problemov za optimizacijo (dopolnitev mojega dela, kjer bo to potrebno)
    - Testiranje algoritmov
    - Implementacija standardnih testov
        - Sodelovanje pri dogovarjanju za projekte z industrijo

3.2 Vključevanje ostalih v C#

- Quingguo ne bi začel učiti C#, preden inštaliramo sisteme za skupne servise. To bo kakšna 2 tedna po tem, ko dobimo strežnik za skupne storitve (kar bo morda čez 2 ali 3 tedne, morda pozneje - zelo se je zavleklo zaradi komplikacij z izborom, saj ponudba iz supporta ni bila več veljavna in so se morali pogajazi za obnovitev rezervacije - sistem je bil namreč zelo specifičen)
- Vsi naj bi začeli prevzemati konkretne naloge (kot npr. razvoj grafike, module za zapis in branje vhodnih in izhodnih podatkov, definicijo geometrije, itd., ko bodo pridobili dovolj znanja programiranja.
    - Ciljali bi recimo na to, da septembra – oktobra začnejo in potem delajo na tem eno etapo bolj skoncentrirano, npr. do konca leta ali do začetka naslednjega leta.

  - V končni fazi naj vsak pri razvoju pokriva bolj svoja področja pri razvoju, uporaba pa mora biti omogočena vsem. Stvari morajo biti narejeno tako, da bodo drugi zlahka prevzeli razvoj – kar pomeni modularno in objektno.
  - Vsak bo zadolžen za to, da pregledno uvede druge v svoje stvari, jaz bom vse skupaj spremljal.
  - Delo bo urejeno timsko.
  - Preden bodo lahko začeli, moram pripraviti nekja osnov na naslednjih področjih:
    - Parserji podatkov iz datotek
    - 2D grafika
    - 3D grafika
- Tadej in katarina naj prevzameta vsak svoj del pri administraciji novih računalnikov.
  - Katarina bo poskrbela za delujoč Linux z razvojnimi orodji na močnejšem računalniku. Tadej bo moj pomočnik pri skupni administraciji. Oba bosta poskrbela vsak za en sistem za task management (recimo Bugzilla in Trac), od katerih bomo potem izbrali najbolj primernega.

3.3 Oprema, potovanja itd.

- 1 konferenca?
- 1 obisk? – preveriti, ali smo to res načrtovali za letos
- Knjige? Ali so bile predvidene knjige nabavljene? Kje so obtičale?
- Kdaj bodo urejeni prostori?
- Oprema:

  - Switch 16 ali 24 portov, 1Gbit/s? – 260 EUR (se lahko nabavi preko univerze?)
  - UPS? Si lahko to privoščimo?
  -
  - Mathematica
  - ActiViz.Net (2 licenci?)
  -

4 KAJ JE TREBA REŠITI

1. Ali bom delal na optimizaciji
   1. To je možno samo, če lahko takoj začnem z delom in dorečemo naslednje stvari

- Da bom lahko intenzivno delal na optimizaciji, kar je predpogoj za uspeh, čisto enako kot pri vseh drugih področjih
- Bom imel možnost aplicirati svoje stvari v povezavi s tem, kar delajo ostali v skupini (Robert in drugi). To je enako , kot smo rekli pri nevronskih modelih (ko smo tudi dobili podatke iz Štor), le da je za optimizacijo to še bolj pomembno, ker gre za bolj kompleksne stvari in so potrebe po možnosti sprotne praktične verifikacije večje
- Če se odločimo, da bom delal na kakšni drugi stvari, je treba takoj definirati drugo področje dela in zagotoviti, da lahko takoj grem v intenzivno uvajalno fazo, da dohitim tempo in nadoknadim čas zaradi spremembe načrta

2. Kako je z uporabo mojih knjižnic
   - To naj bi rešil neposredno Rebeko
   - Načeloma s tem ne bi smelo biti drugače kot z uporabo drugih knjižic, kar se na veliko uporablja v vseh razvojnih ustanovah. S tem, kar bomo delali v COBIK, bomo uporabniki knjižnice s partnerskim sodelovanjem (knjižnico konsistentno širimo z dopolnitvami za specifične potrebe COBIKA).

4.1 Konkretni optimizacijski problemi

Primer tipičnega optimizacijskega problema povezanega s proizvodno celico je naslednji:

- Find values of process parameters:
  - Pressure within the cell
  - Temperature of cell walls
  - Electric current
  - Gap between electrodes
- that give maximal yield of different types of carbon nanostructures

Zelo očitno je, da problem vsebuje vse elemente, ki so opisani v poglavju 1.1 (Optimizacija s kompleksnim odzivom).

Poleg zgornjega primera je v povezavi z numeričnim modeliranjem pomembna še družina optimizacijskih problemov, ki nastopajo pri inverzni identifikaciji modelskih parametrov. Ta ima skupno z zgornjim problemom to, da je uporabljen podoben simulator in bodo zato nekatere lastnosti odziva po pričakovanjih nekoliko podobne.

Kar se tiče optimiranja celice, je veliko variant optimizacijskih problemov sorodnih zgornjemu glede na to, na kaj ciljamo, kje so problemi v procesu, katere parametre je ekonomično uravnavati itd. Spodaj je naštetih nekaj tipičnih parametrov modela proizvodne celice, za tem je naštetih nekaj izhodnih vrednosti modela, ki so odvisne od teh parametrov.

Input parameters:

- anode diameter
- cathode diameter
- anode length
- anode hole diameter
- anode hole depth
- catode length
- reactor dimensions (length and diameter)
- current and current intensity
- presure
- anode cathode distance
- anode temperature (side and tip)
- cathode temperatute (side and tip)
- wall temperature
- mass density of the inert gas
- anode surface area
- pre-exponential factor
- carbon mass fraction
    a. initial C, C2, C3, Ni mole fraction

Output parameters:

- deposit rate
- anode gas velocity
- electric power dissipation
- dilution factor at the anode
- erosion rate
- estimated electron density
- temperature
- He, Ni, Y ... number density
- growth rate