P1

Упознавање са основним класама оптимизационих алгоритама који се користе у инжењерству и ИТ струци :

Систематизација. Поделе оптимизационих алгоритама

• Оптимизациони алгоритми:

– систематско претраживање (енглески: systematic search)

– случајно претраживање (енглески: random search)

– градијентна метода (енглески: gradient method)

– симплекс алгоритам (енглески: Nelder-Mead simplex)

– Данцигов симплекс алгоритам (енглески: Dantzig simplex)

– симулирано каљење (енглески: simulated annealing)

– генетички алгоритам (енглески: genetic algorithm)

– кретање јата (енглески: particle swarm optimization)

– диференцијална еволуција (енглески: differential evolution)

• Оптимизација са више критеријума. Парето фронт и његово одређивање

коришћењем оптимизационих алгоритама

Проблем који се решава помоћу рачуна (и рачунара):

• Проблем = скуп свих вредности улазних података + одговарајући

резултат за сваку вредност улазних података

– Скуп улазних података може бити коначан или бесконачан

Подела:

– Проблеми одлучивања: имају бинаран резултат

{да, не} или {1, 0} или {true, false}

– Проблеми оптимизације: имају генералан нумерички резултат

{реалан број, низ бита, скуп бројева, пермутација елемената скупа итд.}

Проблеми оптимизације могу се трансформисати у проблеме

одлучивања формулацијом питања

– Оптимизација: пронаћи најкраћу путању између задатих тачака

– Одлука: да ли постоји путања краћа од *x* [m] између задатих тачака?

– Последица: теорија рачунске сложености (енг: computational complexity

theory) важи и за оптимизационе проблеме!

Одзиви нису познати унапред за све могуће побуде (ако су сви одзиви познати, најбољи одзив је решење проблема).

Формално, сваки одзив је функција више променљивих (побуда).

Одзив, или нека функција одзива, је мера колико је решење добро или лоше.

Подела модела (поставки) према броју одзива:

– са једним одзивом (један критеријум оптимизације)

– са више одзива (више критеријума оптимизације)

Побуде ↔ оптимизационе променљиве

Одзив ↔ оптимизациона функција

Оптимизациона функција ≡ функција грешке ≡ cost function ≡ evaluation function ≡нумеричка мера квалитета разматраног решења

• Једно израчунавање оптимизационе функције ≡ итерација

• Оптимизационе променљиве ≡ улазни подаци за проблем

• Оптимизациони простор (*S*) је скуп свих могућих вредности оптимизационих променљивих (енг: optimization space, search space, set of candidate solutions…)

• Простор (смислених) решења (*F*) је скуп свих решења које има смисла разматрати (енг: feasible solutions,…)

• Број димензија оптимизационог простора (*D*) је број оптимизационих променљивих

• Свака тачка у оптимизационом простору представља један избор вредности улазних података

• Величина оптимизационог простора је од изузетне важности

• Избор оптимизационог алгоритма зависи од:

– величине оптимизационог простора

– домена оптимизационих променљивих

• Величина простора зависи од усвојеног записа!

– Бијективни записи имају исту величину

оптимизационог простора

– Најбољи запис је онај за који важи *F* = *S*

• Оптимизациони простор може бити

ограничен или неограничен

Основни типови оптимизационих проблема (TSP, SAT, NLP)

• Проблем трговачког путника (енглески: traveling salesman problem, TSP)

• Булова алгебра (дискретна стања) (енглески: Boolean satisfiability, SAT)

• Нелинеарни проблеми (енглески: Nonlinear programming, NLP)

• Домени основних типова проблема (дискретан или континуалан)

Сваки проблем који може да се сведе на проверу свих пермутација називаћемо проблем TSP класе:

• Енглески: TSP-encoded problems

• Поделе TSP:

– Симетричан (енг: symmetric TSP, STSP)

– Асиметричан (енг: asymmetric TSP, ATSP)

– Најкраћи Хамилтонов пут (енг: shortest Hamiltoninan path, SHP)

• Хамилтонова контура (енг: Hamiltonian cycle)

– Обилазак са условима (енг: Sequential ordering problem, SOP)

– Рутирање возила ограниченог капацитета (енг: Capacitated vehicle

routing problem, CVRP)

SAT класа проблема и поделе:

• Сваки проблем који се своди на проверу свих могућих вредности секвенце бита називаћемо проблем SAT класе

• Поделе (Karp's 21 NP-complete problems)

– проблем ранца (knapsack)

– подела посла (job-sequencing)

– directed/undirected Hamiltonian cycle

• Примери: свака манипулација битима може да претвори у проблем SAT класе

– Откључавање ZIP архиве

– Пробијање RSA кодова

NLP проблеми

• Сваки оптимизациони проблем са континуалним променљивима спада у NLP класу проблема

• Број могућих решења је (теоријски) бесконачан!

• Посебни случајеви

– диференцијабилне и недиференцијабилне *f*

– линеарно програмирање

– квадратно програмирање

– конвексни/конкавни проблеми

– са ограниченим или неограниченим доменом

Класификација је према запису улазних података (побуда) и природи оптимизационог проблема.Често је довољно наћи задовољавајуће решење. Трагање за најбољим (оптималним) решењем је можда занимљиво али најчешће неприхватљиво

Тада се примењују оптимизациони алгоритми и хеуристике

ХЕУРИСТИКА (енг: heuristic) грчки корен речи “Εὑρίσκω“ − пронаћи или открити

• IEEE: All engineering is heuristic

• Технике решавања проблема засноване на искуству, учењу и откривању које доводе до решења (не мора нужно бити оптимално али је довољно добро)

• Када год је немогуће или непрактично потпуно претраживање простора, користе се хеуристике (интуиција, стереотипи, здрав разум, енг: rule-of-thumb, educated guess, …)

• State-of-the-art проблеми се увек решавају хеуристички

P2

Како проценити особине и перформансе опт. алгоритма?

• Ситуација #1 (инжењерска пракса): постављен је оптимизациони проблем и потребно је решити га

– бирамо алгоритам на основу претходног искуства и информација које имамо о датом проблему

– пробамо све доступне оптимизационе алгоритме и користимо оне који имају најбоље перформансе

• Ситуација #2 (истраживање / наука): потребно је сагледати особине оптимизационог алгоритма

– тестирамо алгоритме на познатим оптмизационим функцијама (проблемима) и поредимо перформансе оптимизационих алгоритма

• Потребне су нам познате оптимизационе функције (познати оптимизациони проблеми) са различитим особинама, када разматрамо алгоритме

Формирање оптмизационе функције и норме

• Оптмизациона функција је нека функција разлике величина које описују решења

• Примери:

– разлика између перформанси жељеног и пројектованог електричног уређаја

– разлика између мерених резултата и функције која их фитује

– разлика између жељене и добијене цене производа

– разлика између жељеног и оствареног управљања

• Да ли узети апсолутну вредност разлике, квадрат разлике или неки други степен?

• У оптимзационој функцији у пракси често се појављује норма разлике жељеног и оствареног

Локално решење: формална дефиниција

• Епсилон околина решења x је

• За NLP класу проблема:

Еуклидовско растојање

• За SAT класу проблема:

Хамингово растојање

• За TSP класу проблема:

број различитих прелазака

• *dist*(x,y) може да се дефинише на много начина!

• Локални минимум x, у околини *N*(x):

NFL теорема

• NO-FREE-LUNCH theorem – Wolpert D.H.; Macready W.G. "No free lunch theorems for optimization " *Evolutionary* *Computation IEEE Transactions on* vol.1 no.1 pp. 67-82 Apr 1997

– уколико су све оптимизационе функције једнако вероватне сви опт. алгоритми су једнако (не)ефикасни

P3

Партиција (подела/разбијање) природног броја *n* је низ позитивних природних бројева

*a*1 ≥ *a*2 ≥ … ≥ *a*m ≥ 1 тако да је

*n* = *a*1 + *a*2 + …+ *a*m

• При томе је: 1 ≤ *m* ≤ *n* и *n* ≥ 1

• Лексикографски поредак

– прва партиција је *a*1 = *n*

– последња партиција је

1 + 1 + … + 1 = *n*

Партиција скупа *S* је подела *S* на подскупове *Sm*, 1 ≤ *m* ≤ |*S*| тако да важи:

Партиција скупа са редоследом подскупова: генерисати партицију скупа + пермутација свих подскупова

Претраживање по свим стаблима је сложенији проблем од TSP!

Постоји једнозначно пресликавање између оваквих секвенци и стабала комплетног графа са *n* чворова (Prüfer).

Систематско претраживање: особине и употреба

• Уколико желимо да докажемо да смо нашли глобални оптимум (најбоље могуће решење) једини начин је да систематски претражимо читав оптимизациони простор у општем случају

• Једно решење (једна тачка оптимизационог простора) се једном и само једном проверава

• Континуалне променљиве (NLP):

– систематско претраживање подразумева коначан корак за сваку димензију (тачност)

– решење смо нашли са тачношћу која је пропорционална кораку

• Дискретне променљиве (SAT и TSP):

– сва решења су проверена

Други називи и особине систематског претраживања

• Други називи за систематско претраживање

– Grid search

– Brute-force search

– Parameter sweep

– Exhaustive search

– Enumeration

– Generate & test

• Најбољи могући приступ уколико можемо да сачекамо да се претрага заврши

– оптимизациони простор је мали у односу на расположиве ресурсе

• Уколико претрага траје недопустиво дуго није од користи

– оптмизациони простор је велики у односу на рачунарске ресурсе

• Погодан за извршавање у паралели на рачунарима са више процесора (језгара)

Случајно претраживање

• На случајан начин генеришу се тачке у којима се рачуна функција грешке

• Најчешће се користи генератор са униформном расподелом

• Неефикасан начин оптимизације јер су претходно и наредно израчунавање оптимизационе функције независни (иста тачка може да се испита више пута)

• Добар начин за грубу претрагу простора

• Сложеност *O*(*N*) *N* број одбирака оптимизационог простора

P4

Случајно генерисана варијација са понављањем

• По један случајно изабран елемент из скупа од *n* елемената постављамо на

свако од *k* места

– Понављање елемената је дозвољено

• Имплементација: за свако од *k* места генеришемо један случајан цео број из скупа од *n* елемената

• Случајан вектор од *n* реалних бројева из опсега [*x*min, *x*max]

– На формално идентичан начин генерише се случајно изабран вектор реалних бројева

– Једина разлика је што се користи генератор случајних реалних бројева из датог опсега

Случајно генерисане партиције и стабла графа

• Случајно генерисана партиција броја *n*:

– Пронаћи укупан број партиција броја, генерисати случајан број од 1 до *n* и генерисати одговарајућу партицију алгоритмом за генерисање свих партиција

– Једноставан али рачунарски захтеван приступ

• Случајно генерисана партиција скупа *S*

– Генерисати случајан RGS (restricted growth string), сваки елемент је између нуле и највећи пре њега +1

• Случајно генерисано стабло потпуног графа K*n*

– Генерисати низ од *n* − 2 елемента на чијим местима може да буде 1,2,… *n* (видети пресликавање таквих низова у стабла графа)

– Ако граф није потпун, недозвољена стабла се изостављају

• Постоје рачунарски ефикасније методе

Локални оптимизациони алгоритми

• Енг: hill-climbing, down-hill, greedy, local optimization…

• Претражују околину полазног решења

• Заснивају се на итеративном поправљању полазног решења

• Генерализовани блок дијаграм је приказан десно

Слабости Hill-Climbing

Алгоритама

• По правилу проналазе само локални оптимум

• Нема информације о томе колико смо далеко или близу глобалног оптимума

• Крајње решење зависи од полазног

• Генерално, није могуће предвидети максималан број потребних итерација (зависи од оптимизационе функције)

SAT локално претраживање

• Дефинисати околину која се претражује

• Хамингово растојање (број бита, *k*, колико сме да се промени)

• Ако је број димензија *D* генерисати све могуће промене

– број избора *k* бита од *D*

– пута број распореда бита на *k* места

• Памтити решења и не проверавати више пута исто решење

Класични методи

(NLP: континуалне функције)

• Метод половљења интервала (енглески: bisection/bracketing method)

• Метод сечице (regula-falsi)

• Њутнов метод (апроксимација тангентама)

• Брентов метод (апроксимација параболама)

• Лагранжови мултипликатори

• ККТ услови

• Градијентни метод

• Хесијан

– BFGS (Broyden–Fletcher–Goldfarb–Shanno) метод

Половљење интервала

• Услов:постоји интервал *f*(*a*) *f*(*b*) < 0

• *x*1 = 0,5 ( *a* + *b* )

• Нови интервал: [*a*, *x*1] ако *f*(*a*) *f*(*x*1) < 0 [*x*1,*b*] ако *f*(*x*1) *f*(*b*) < 0

• Поновити процедуру док се нула не одреди са задатом тачношћу

Половљење интервала брзо конвергира

• У кораку *n* интервал је

• Свака итерација смањује интервал 2 пута

• Изузетно робустан

• Једноставан за програмирање

• Не захтева познавање извода функције!

• Може да се примени и ако је функција са шумом или са целим бројевима

• Захтева познавање почетног интервала

• Генерализација на вишедимензионе проблеме?

• Нула се тражи у пресеку линеарне апроксимације функције и апсцисе

Услов: постоји интервал *f*(*a*) *f*(*b*) < 0

• Следећа апроксимација

• Нови интервал: [*a*, *x*1] ако *f*(*a*) *f*(*x*1) < 0 или [*x*1,*b*] ако *f*(*x*1) *f*(*b*) < 0

• Поновити док се нула не одреди са задатом тачношћу

Regula falsi: употреба

• Потребно је знати почетни интервал у којем постоји једна нула, а функција мења знак

• Конвергенција је нешто бржа од метода половљења интервала

• Мало сложенији за програмирање од половљења интервала (постоји формула која се програмира)

• Генерализација на вишедимензионе проблеме?

Њутнов метод: основна идеја и скица извођења

• Такође је познат под именом Newton-Raphson

• Позната функција *f*(*x*) и њен први извод *f’*(*x*)

• Једначина тангенте у тачки *xn* је

Њутнов метод (апроксимација тангентом)

• Следећа апроксимација:

• Захтева добро полазно решење (тангента “води” ка нули, није тако у општем случају)

• Потребно је израчунати извод функције, што није увек једноставно

Њутнов метод: употреба

• Изузетно брза конвергенција

• Тачно или апроксимативно израчунавање извода ограничава употребу

• Уколико функција садржи локални минимум, алгоритам може да дивергира

• Функције са нумеричким шумом нису погодне за овај метод због израчунавања извода

Брентов метод (квадратна интерполација)

• Услов: функција има само један минимум (или максимум) у посматраном интервалу

• Квадратна апроксимација на основу 3 тачке

• Конструисање параболе

• Рачунање минимума параболе

• Понављање метода док се не постигне потребна тачност

• Брентов метод обједињује

– половљење интервала,

– метод сечице и

– квадратну интерполацију

Лагранжови мултипликатори

• Решавамо проблем

• *f*(x) и *gk*(x) су непрекидна и имају све парцијалне изводе

• Формирамо помоћну (нову опт.) функцију

• Коефицијенти λ*k* су Лагранжови мултипликатори

• Опт. проблем са условима сведен на проблем без услова компактан запис:

• Тачке које задовољавају овај систем једначина су екстремуми (минимуми, максимуми или седласте тачке)

• Проверити вредности свих добијених екстремума

Пример примене Лагранжових мултипликатора

• У једном чвору електричног кола струја *I* дели се у *D* паралелно везаних грана

• У свакој грани је један отпорник познате отпорности *R*1, *R*2,…*RD*

• Пронаћи струје грана *I*1, *I*2,…*ID.* тако да је електрична енергија (и снага) овог кола

минимална

• Формални запис опт. Проблема

Karush–Kuhn–Tucker (генерализација Лагранжових мул.)

• Решавамо проблем:

• *f*, *g*, *h* су континуалне диференцијабилне функције, а *f* је конвексна функција

• Помоћна функција

• x0 je оптимално решење:

Ограничења класичних метода

• Оптимизациона функција не мора имати нулу (ако је потребно пронаћи минимум, онда се тражи нула извода)

• Изводи оптимизационе функције не морају нужно да буду познати, а могуће је и да не постоје!

• Генерализација у случају више променљивих није једноставна за све наведене алгоритме

• За све наведене, сем метода половљења, постоје cитуације када методи дивергирају или стану

P5

Предности и мане градијентног метода

• Локални оптимизациони алгоритам

• Изузетно брза конвергенција

• Додатно повећање брзине конвергенције: уколико је добро процењен правац повећати корак у следећој итерацији

• Проналази само најстрмији минимум у околини полазног решења (то није нужно и најдубљи минимум!)

• Ефикасан уколико знамо добро полазно решење

• У *D*-димензионом простору захтева *D*+1 итерација за процену градијента

• Корак за процену градијента Δ*xi* зависи од проблема!

Предности и мане оптимизације засноване на Хесијану

• У претходном примеру даје решење у једном кораку за произвољну полазну тачку

• Уколико оптимизациона функција има непрекидан први и други извод, оптимизација заснована на Хесијану је ефикаснија од градијентне методе

• Које се решење добија уколико оптимизациона функција има више од једног минимума?

• Израчунавање двоструких парцијалних извода може да буде проблематично код практичних оптимизационих проблема

• …или први и други извод не морају да постоје

Практична употреба Хесијана

• Оптимизациона функција са више променљивих

• Тејлоров развој

• Проблем је рачунање Хесијана

• Најпознатији је Broyden–Fletcher–Goldfarb–Shanno Алгоритам

Broyden–Fletcher–Goldfarb– Shanno (BFGS) алгоритам

• x0 је полазна тачка, B0 је нулта ест. Хес., типично I

• У x*k* одреди се правац претраге p*k*

• Промени се позиција

• Нека је

• Естимација Хесијана у наредној тачки је

• Понавља се за наредну тачку у простору

Комплетно и парцијално решење

• Алгоритми који раде са комплетним решењем

– потпуно дефинисани проблем

– заустављање алгоритма даје до тада најбоље решење

– примери: систематско претраживање, случајно претраживање, hillclimbing, градијентни метод…

• Алгоритми који раде са парцијалним решењем

– апроксимативно решење (surrogate models, fitness fitting, fitnessapproximation, space mapping, Kriging, response surface methodology… )

– заустављени алгоритам не мора да има корисно решење (случајно претраживање без памћења најбољег пронађеног…)

• Инжењерски употребљиви су алгоритми који дају најбоље решење до тада

Constrained vs. Unconstrained optimization

• Како алгоритам који решава оптимизационе проблеме без ограничења искористити за решавање опт. проблема са ограничењима?

• Најједноставнији приступ: уколико ограничења нису задовољена додати “пенал” на оптимизациону функцију *F*(x) = *f*(x) + *f*p(*g*(x)) где је *g*(x) услов који није испуњен, а *f*p(*g*(x)) додатна функција (“пенал”) који се додаје

Nelder-Mead Симплекс Алгоритам

• Основне референце

– J. A. Nelder, R. Mead “A Simplex Method for Function Minimization”,*The Computing Journal* 7, pp. 308-313, 1965.

– Lagarias, J. C., J. A. Reeds, M. H. Wright, and P. E. Wright.

“Convergence Properties of the Nelder-Mead Simplex Method in Low

Dimensions.” *SIAM Journal of Optimization*. Vol. 9, Number 1, 1998,

pp. 112–147.

– Gao, F. and Han, L. “Implementing the Nelder-Mead simplex algorithm with adaptive parameters,” *Computational Optimization and* *Applications*, 51:1, 2012, pp. 259-277.

• Jедан од најчешће коришћених алгоритама за локалну NLP оптимизацију

• Често се референцира само под именом симплекс или Nelder-Mead

• Понекад се назива и “amoeba” алгоритам

Почетак симплекс алгоритма

• Алгоритам почиње формирањем *D*+1 тачке симплекса (обавезно ненулте запремине) у простору и израчунавањем оптимизационе функције у тим тачкама

• На почетку *k*-тог корака алгоритма, задато је *D*+1 тачака које заједно формирају симплекс

• Сваки корак алгоритма почиње сортирањем и обележавањем ових тачака x1(*k*), x2(*k*), ... x*D*+1(*k*)

Најбоље и најгоре тачке и функције у оквиру алгоритма

• У сваком кораку алгоритма формира се нови скуп тачака (симплекс) који је различит од симплекса у претходном кораку

• Циљ алгоритма је проналажење минимума функције *f*

– тачка x1(*k*) се назива најбоља тачка,

– x*D*+1(*k*) је најгора тачка, а

– x*D*(*k*) је друга најгора тачка

• Најбоља функција грешке је *f*1 (*k*), а најгора функција грешке је *fD+*1 (*k*)

Један корак симплекс алгоритма

• Један корак алгоритма састоји се из следећих пет операција

(1) сортирање

(2) рефлексија

(3) експанзија

(4) контракција

(5) сажимање

• Само неке операције се извршавају у оквиру једног корака

• Која ће се операција (или операције) извршити зависи од текућег стања

1. Сортирање

(1) Сортирање: сортирати темена симплекса тако да је испуњен услов

Алгоритам за сортирање није критичан, јер симплекс по правилу има до највише неколико стотина тачака

2. Рефлексија (2) Рефлексија: Израчунати центроид *D* најбољих

тачака, и у односу на њега израчунати тачку рефлексије хr према формули

Израчунати *f*r Уколико је изоставити најгору тачку, а уместо ње уврстити xr у симплекс и

завршити корак

3. Експанзија Експанзија: ако је *f*r < *f*1 израчунати тачку експанзије хе према формули

Израчунати *f*e Ако је *f*e < *f*r изоставити најгору тачку и уместо ње уврстити xе у симплекс и завршити корак

У супротном, ако је *f*e ≥ *f*r уврстити xr и завршити корак

4. Контракција Контракција: Ако је *f*r ≥ *fD* доделити израчунати тачку контракције хc као

Израчунати *f*c, уврстити xc и завршити корак ако је *f*c ≤ min(*f*D+1,*f*r),

(тј. тачка контракције је боља од x*D+*1 и xr) У супротном извршити сажимање

5. Сажимање

Сажимање: заменити све тачке осим најбоље са израчунати функцију грешке у свим новим тачкама и завршити корак

\* По завршетку једног корака, а пре преласка у наредни корак потребно је извршити проверу да ли је алгоритам испунио услове за завршетак оптимизације

Формирање полазног симплекса

• Типично је позната једна тачка x0 (почетно решење), осталих *D* је потребно формирати

• Најједноставнији приступ (хипер-правоугаоник):

x1 = x0+ (Δ*x*1,0,…0)

x2 = x0+ (0, Δ*x*2,…0)

x*D* = x0+ (0,0,… Δ*xD*)

• Проблеми:

– Како изабрати Δ*xk* ?

– Неке тачке могу изаћи из оптимизационог простора

• Провера да ли је тачка у оптимизационом простору и ако није промена знака Δ*xk*

• Могуће је користити и друге *D* димензионе фигуре (сфера, коцка, итд.)

Адаптивна промена параметара

• Параметри алгоритма могу се подешавати у односу на број димензија оптимизационог простора *D*

• Један могући приступ

• Повољно у случајевима “великог” *D* (од ~10 до ~30)

• Python имплементација адаптира параметре у зависности од *D*

• Оптимизација параметара оптимизационог алгоритма

(енглески: *metaheurstics*)

Колика је сложеност симплекс алгоритма?

• У пракси алгоритам типично конвергира за *О*(*D*2) итерација тј. ~*D*2 израчунавања оптимизационе функције

• У литератури постоје теоријска разматрања само за мањи број димензија оптимизационог простора, типично до 10

• У општем случају, није позната сложеност овог алгоритма

Практични закључци о симплекс алгоритму

• Спада у групу локалних оптимизационих метода

• Поседује способност да, у одређеним ситуацијама, прескочи из једног локалног минимума у други уколико се тиме добија боље решење

• Ефекат “прескакања” се не дешава увек и зависи од

– конкретног проблема и

– полазног симплекса

• Ова особина даје предност симплекс алгоритму у односу на (све) друге локалне оптимизационе алгоритме

• Постоје напори да се строго математички докаже због чега симплекс алгоритам постиже врло добре резултате у пракси,међутим генералног доказа нема

• Симплекс се може посматрати и као генерализација метода половљења интервала

P6

Dantzig simplex алгоритам

• Најпознатији алгоритам за решавање LP

• G. B. Dantzig, *Linear Programming and Extensions*, Princeton University Press, 1959.

• George B. Dantzig and Mukund N. Thapa. 1997. *Linear programming 1: Introduction*. Springer-Verlag.

• George B. Dantzig and Mukund N. Thapa. 2003. *Linear Programming 2: Theory and Extensions*. Springer-Verlag.

• 1975. Нобелова награда за LP програмирање у економији L. Kantorovich и T.C. Koopmans

Канонични облик

• Циљ: максимизирати *Z*

• Канонични облик LP (simplex tableau)

• У општем случају може се свести на овај облик (све једнакости и све променљиве ненегативне)

Dantzig simplex

• Поћи из једне екстремне тачке (подразумева се да је бар једна таква тачка позната, што у општем случају не мора да буде тривијално)

• Проверити да ли је могуће повећати *Z* преласком у другу суседну екстремну тачку

• Завршити алгоритам уколико није могуће повећати *Z* (решење је пронађено)

Пивотизација (основни корак алгоритма)

• Пронаћи зависну променљиву са позитивним коефицијентом у *Z* (повећање те променљиве повећава *Z*)

• Повећати променљиву максимално могуће на основу услова који постоје

• Пронаћи најстрожи услов (најмање повећање)

• Заменити основну и зависну променљиву

Анализа наредног корака

• *Z* = 50 – 2*s*1 – *s*2

• Нема више променљивих које имају позитивни коефицијент у *Z*

• Нема могућности за даље повећање *Z*

• Решење је пронађено

• Крај алгоритма

Генерализација

• Уколико су чланови са десне стране услова *bi* ненегативни, постоји решење

• Како организовати алгоритам уколико су *bi* негативни?

• Описани поступак назива се “фаза 2” симплекс алгоритма

• Претпроцесирање (“фаза 1”) своди све друге LP проблеме у облик који се може решити “фазом 2”

• Потребно је пронаћи базисни вектор

– додају се “вештачке променљиве”

– пивотизацијом се решава “додатни проблем” који даје формулацију полазног проблема који се може решити “фазом 1”

• Може се догодити да полазни проблем

– нема решење

– оптимизациона функција (маx) није ограничена са горње стране

• Претпроцесирање је сложено у општем случају

Формулација

• Трошкови испоруке су где је *xij* број пакета који иде из фабрике *i* у објекат *j* (*i* є{1,2} и *j* є {1,2,3})

• Потребно је поронаћи све *xij* тако да функција *z* има минималну вредност, при задатим условима

• *xij* мора бити цео број и ненегативан (*xij ≥* 0)

Основне чињенице о Dantzig simplex алгоритму

• Најчешће коришћен алгоритам у пракси за LP оптимизацију

• У зависности од полазног решења, могуће је обићи свe екстреме да би се стигло до најбољег (што је еквивалентно систематском претраживању)

• Време потребно за претраживање је експоненцијално (у најнеповољнијим случајевима, постоји строг доказ)

• Без обзира на то, овај алгоритам добро ради у пракси

• Сложеност

– у пракси је типично *O*(*N*3),

– теоријски експоненцијална (обилазак свих екстремних тачака,могу се конструисати примери где алгоритам мора да прође све екстреме пре него што пронађе решење)

Други алгоритми за LP

• Criss-cross algorithm нешто једноставнији од Dantzig simplex

• Fourier–Motzkin elimination

• Karmarkar's algorithm један од ретких алгоритама који је био патентиран, прилично компликован (перформансе у пракси нису боље од Dantzig симплекса)

• Методи унутрашње тачке (interior-point methods) проналазе решење крећући се по унутрашњости симплекса

(за разлику од Dantzig алгоритма који иде по ивицама!)

• Сви алгоритми који решавају NLP проблеме могу да се примене (при томе, некада је потребно променити оптимизациону функцију *f*0= – || *f ||p +*const.)

P7

Динамичко програмирање је техника за проналажење оптималне комбинације догађаја који су међусобно повезани

• За разлику од линеарног програмирања не постоји стандардна математичка формулација динамичког програмирања

• Постоји принцип који се модификује према конкретном оптимизационом проблему

Нема ограничења у погледу линеарности/нелинеарности оптимизационе функције

Једноставан приступ – у сваком кораку изабрати прелаз са најмањом вредношћу

(енг: greedy тј. локални алгоритам)

Најбоље решење можемо да нађемо и потпуном претрагом

Основна идеја је поделити проблем на мање потпроблеме

• Потпроблеми:

– су међусобно повезани

– решавају се одвојено

– крајње решење се добија на основу решења појединачних потпроблема

• Сваки потпроблем одговара једној оптимизационој променљивој (текућа променљива)

• Проблем се решава у корацима

• Кораци могу бити и дискретни одбирци времена (одатле “динамичко” програмирање)

Оптимални избор корака за читав проблем има особину да без обзира на почетан избор избор преосталих корака мора бити оптималан у односу на текуће стање

Проблем се може поделити на потпроблеме који се решавају сукцесивно

• Сваки потпроблем има коначан број избора који се могу направити

Тражимо оптималну путању кроз граф

• У текућем стању оптимални избор наредног корака зависи само од преосталих стања а не зависи од претходних стања (Белманов принцип оптималности)

• Први корак је одређивање оптималног последњег (могућег) избора

• Постоји рекурзивна релација којом се одређује оптимални избор у текућем кораку уколико је познат оптимални избор за све наредне кораке

Примери алгоритама који користе динамичко програмирање

• Dijkstra's algorithm (проналажење најкраћег пута у дискретном простору)

• Viterbi algorithm (проналажење највероватније секвенце скривеног процеса)

• Bellman–Held–Karp algorithm (проналажење оптималне путање TSP проблема динамичким програмирањем)

Број провера и рачунских операција се смањује на рачун памћења претходно решених потпроблема (табела)

• Код великих проблема меморијски ресурси постају критични!

• Решавање се може организовати генерално на два начина

– од краја ка почетку (енглески: top-down ) или

– од почетка ка крају (енглески: bottom-up)

• Динамичко програмирање = рекурзија + “здрава памет”

– рекурзија: изражавамо текућу вредност преко претходних

– “здрава памет”: организујемо поступак тако да се памте све вредности које су већ једном одређене

Алгоритми које смо до сада радили или

– захтевају недопустиво много времена да пронађу најбоље решење

– заглаве се у локалном оптимуму

• Немогуће је направити алгоритам који у кратком времену проналази глобални оптимум у општем случају

Основни кораци симулираног каљења

• Генерисање наредне тачке (у околини која се претражује)

• Прелазак у наредну тачку (прелазак се одиграва стохастички)

• Формирање следеће температуре (хлађење) и промена околине у којој

се генерише наредна тачка

Генерисање наредне тачке

• Наредна тачка у оптимизационом простору се бира на случајан начин из околине текућег решења

• Најчешће се користе генератори

– униформне или

– Гаусове расподеле случајних бројева

• Током хлађења запремина околине у којој се генеришу наредне тачке се сужава и теоријски на крају оптимизације постаје нула

– NLP околина сфера полупречника *R*

– SAT околина Хамингово растојање

– TSP околина број градова којима заменимо места

Прелазак у наредну тачку

• Наредна тачка се прихвата тј. постаје текућа тачка оптимизације у зависности од прираштаја функције грешке

• Уколико је прираштај опт. Функције Δ*E* негативан тј. опт. функција је мања у наредној тачки тачка нужно постаје текућа тачка оптимизације

• Међутим дозвољени су и преласци у тачке у којима је Δ*E* позитивно!

Вероватноћа преласка одговара Болцмановој расподели те се овај алгоритам назива и Болцманово хлађење

• Уколико је *T* >> Δ*E* тада је *p* ≈ 1 сви преласци су дозвољени алгоритам се понаша као случајно претраживање

• Уколико је *T* << Δ*E* тада је *p* ≈ 0 само преласци у боље решења су дозвољени тј. понаша се као локални оптимизациони алгоритам

Формирање следеће температуре (хлађење)

• Температура у следећем кораку је увек мања или једнака од температуре у текућем кораку

• Следећа температура се формира по шеми хлађења

• Постоји неколико прихваћених шема хлађења од којих су две најчешћереференциране

Смањивање околине у којој се проверава наредно решење

• Током оптимизације “запремина” околине у којој се генерише и проверава наредно решење би требало да буде монотоно опадајућа функција

• Обично се користи линеарно смањивање са бројем итерација али може и било која руга (нерастућа) функција

Варијације симулираног каљења

• На једној температури је могуће урадити више итерација

• Понављање алгоритма из претходно добијеног минимума је још једна од честих модификација алгоритма

• Понављање се у литератури назива поновљено каљење (reannealing)

Основна идеја Табу претраживања: меморија

• Идеја: памте се претходна стања и њихове околине се проглашавају за забрањене

(табу) зоне

• Уколико се алгоритам дуже времена задржи у локалном минимуму тај део простора се прогласи за табу

• Дефинисање меморије?

Различите меморије

• Краткорочна меморија – памти се неколико последњих испитаних решења и њихове околине

• Дугорочна меморија – памте се (скоро) сва решења која су испитана

• Средњорочна меморија – памти се већи број решења која су пробана

• Избор меморије и дефинисање шта меморија представља зависе од проблема!

Закључци о Табу претраживању

• Табу претраживање се једноставније примењује на дискретне проблеме (ТSP, SAT)

• За дискретне проблеме се меморија и табу зоне једноставније дефинишу

• Табу претраживање се релативно ретко примењује на генералне NLP оптимизационе проблеме

Свако покретање стохастичког алгоритма даје различите резултате

• Формално решење је cлучајна променљива

P8

Генетички алгоритам: основне референце

У природи, еволуција се заснива на

– размножавању

– мутацији

– такмичењу

– селекцији

• Овај процес се примењује на све живе организме, генерацију за генерацијом

• Немогуће је за једну врсту (живог света) да буде потпуно прилагођена на околину, јер се околина стално мења

Генетички алгоритам: симулација опстанка

• Основна идеја: имплементације алгоритама опстанка из природе

• Генетички алгоритам (ГА) је један од генералних приступа оптимизацији

• Заснива се на принципу природне селекције и опстанка најприлагођенијих јединки околини

• ГА спада у стохастичке алгоритме за глобалну оптимизацију

• Алгоритам се најчешће описује

– строгe математичкe дефиницијe захтевају апстрактне дефиниције оператора

Једно могуће решење оптимизационог проблема (једна тачка оптимизационог простора) назива се индивидуа и састоји се од гена

Током оптимизације, текућа популација замењује се наредном популацијом решења

• Популација у једном кораку алгоритма назива се генерација

Основна идеја ГА и формирање почетне генерације

• Идеја: од најбољих решења из претходне генерације формира се наредна генерација решења која (би требало да) су боља

• Формирањем сукцесивних генерација долази се до све бољих решења оптимизационог проблема

• ГА почиње формирањем почетне популације (полазног скупа решења) које се обично назива полазна или нулта генерација

– најчешће је то скуп случајних вектора у опт. простору

– може се формирати и на неки други начин (детерминистички, униформно, полазећи од процене, итд.)

– ГА не залази у начин формирања нулте генерације

Један корак ГА се састоји од 3 операције

– селекција

– укрштање

– мутација

• Коришћењем ових операција од претходне генерације добија се наредна генерација

Бинарни и континуални ГА

• У зависности од представе гена, разликујемо две класе ГА:

– бинарни ГА и

– континуални ГА

• У литератури се под називом ГА најчешће подразумева бинарни ГА

• Од представе гена зависи реализација оператора укрштања и оператора мутације

• У литератури постоје опречна мишљења о томе које је представљање боље

(зависи од конкретног проблема)

Бинарна представа је врло погодна за дискретне комбинаторно-оптимизационе проблеме (SAT, TSP)

Оператор селекције

• У оквиру текуће генерације селектује се скуп родитељских решења

– У природи су то они чланови популације који су оставили потомство

• Тај скуп служи за стварање наредне генерације, док остали чланови текуће генерације не учествују у формирању наредне генерације

• На овај начин се фаворизује простирање добрих решења и омогућава се њихово даље побољшавање

• Селекција се може реализовати

– стохастички или

– детерминистички

Стратегије селекције

• Најчешће коришћене стратегије селекције

– Децимација

– Пропорционална (рулет) селекција

– Турнир селекција

• Постоји варијација за сваку од ових стратегија

• Постоје и многе друге стратегије које су предложене у литератури и коришћене у пракси

• Не постоји закључак која стратегија је (нај)боља

Стратегије селекције: Децимација

• Децимација популације

– сортирање популације према оптимизационој функцији

– затим се из тог скупа изабере подскуп првих k (најбољих) решења

• Предност овог приступа је једноставност имплементације

• Мана je сувише брз губитак разноликости генетског материјала унутар популације

Елитизам и децимација

• При формирању следеће генерације могу се

– задржати најбоља изабрана решења (елитистички приступ)

– потпуно избацити решења из претходне генерације

• Иако се на први поглед може чинити да је добро оставити бар најбоље решење из претходне генерације, нумерички експерименти (статистички) показују супротно

• Елитистичка стратегија

– доприноси бржој конвергенцији алгоритма

– повећава вероватноћу конвергенције читаве генерације ка локалном минимуму из ког је алгоритму потребно много времена да изађе

• Елитистички приступ није погодан у општем случају, али се може користи као опција за убрзање конвергенције алгоритма

Вишеструким понављањем избора добија се група индивидуа за укрштање

• Фаворизују се најбоља решења, али постоји коначна вероватноћа да

понеко лоше (али “срећно”) решење учествује у укрштању

Селекција се врши на основу оптимизационе функције (способности)

• Избором стратегије селекције се прави компромис између брзине конвергенције алгоритма и вероватноће проналажења локалног уместо глобалног минимума

• Избор стратегије селекције не зависи од записа оптимизационог проблема

• Све описане стратегије селекције могу се применити на било који оптимизациони проблем (SAT, TSP, NLP)

• Формално, било која операција која издваја подскуп решења из текуће генерације је оператор селекције

– Селекција на случајан начин: ГА = случајно претраживање

2. Оператор укрштања

• Избором два или више решења из скупа најбољих формира се група решења за укрштање

• Њиховим укрштањем формирају се нова решења

• Укрштање се може реализовати

– стохастички или

• Вероватноћа укрштања је вероватноћа са којом се изабрана група индивидуа укршта и типично износи од 0,6 до 0,9

– детерминистички

• Конкретна реализација оператора укрштања зависи од записа проблема

Бинарно укрштање (SAT проблеми)

• Гени су у овом случају бити xk = {0,1}

• Индивидуе (решења) су секвенце бита (низ карактера поређаних један за другим)

x = (0,1,1,0,1,0,0,1)

• Нека постоје два селектована решења за укрштање

• На случајан начин се изабере тачка (бит) укрштања у којој се пресеку низови оба родитељска решења

• Надовезивањем добијених поднизова добијају се два нова решења “потомка”

• На први подниз првог родитеља надовеже се други подниз другог родитеља и обрнуто

Континуално укрштање (NLP проблеми)

• Континуална представа гена (реалне променљиве, xk є ℝ)

• Укрштање се најчешће врши линеарном комбинацијом вектора који представљају

родитеље (r1 и r2), параметар 0 < α < 1

Укрштање за TSP проблеме

• Запис TSP проблема је низ пермутованих елемената

• Два решења x1={1,3,2,4,5} x2={3,2,5,4,1}

• Потребна је креативност за укрштање две пермутације

• Најједноставније је користити једног родитеља и заменити места неколико гена

• Замена места гена k парова

• Може се увести вероватноћа са којом се замена места извршава

Оператор укрштања: закључци

• Укрштањем се формирајунова решења оптимизационог проблема (нове индивидуе)полазећи од селектованих решења

• Укрштање се понавља онолико пута колико је потребно да се формирају сва решења (индивидуе) за наредну генерацију

• Избор начина укрштања зависи од записа оптимизационог проблема

• За сваки тип записа (SAP, TSP, NLP) постоје посебне реализације укрштања

• Формално, било која операција која, на основу једног или више селектованих решења, формира нова решења је оператор укрштања

Мутација

• Реализује се искључиво стохастички

• Са унапред задатом вероватноћом изаберу се гени унутар новоформиране генерације који се на случајан начин промене

• Мутацијом се додаје нови генетски материјал у следећу генерацију (уноси се нова информација о оптимизационом простору)

• Мутација представља могућност да се истраже неистражени делови оптимизационог простора

• Вредност за вероватноћу мутације је обично мала, типично од 0,01 до 0,15

• Граничне вредности вероватноће мутације:

→1: ГА = случајно претраживање

→0: ГА = локално претраживање

Критеријуми за завршетак

• Алгоритам се прекида онда када је решење пронађено или када се гени читаве генерације разликују за мање од неког унапред задатог прага

• Уколико се оптимизација врши довољно дуго, пре или касније ће се десити мутације које ће довести до проналажења глобалног минимума

• У пракси је готово увек недопустиво чекати толико дуго и алгоритам се прекида после одређеног броја генерација уколико критеријуми за завршетак нису пре тога достигнути

• Експериментално је утврђено да после ~50 генерација алгоритам конвергира за популације до 200 000 индивидуа

Величина популације (*N*pop)

• Број генерација (*N*gen)

– Укупан број итерација (израчунавања опт.функције) је *N*pop \**N*gen

• Запис променљивих

• Начин реализације сваког оператора (селекција, укрштање и мутација)

• Избор вероватноћа код стохастичких оператора

ГА са више популација које коегзистирају заједно са разменом од неколико индивидуа у свакој генерацији (енг: multipopulation GA)

• Микро ГА: вишеструко понављање ГА са релативно малим популацијама

• Коришћење локалних оптимизационих алгоритама у спрези са ГА (вишестепене оптимизације)

• ГА код кога су параметри алгоритма придружени оптимизационим променљивима xtot = (xopt, xGA)

– параметри ГА су променљиве које се мењају током оптимизације

– ГА сам подешава своје параметре (могуће осцилације и дивергенција!)

EA = Genetic algorithm, Genetic programming, evolution strategies, evolutionary programming…

• Марковљев ланац (енг: Markov chain) будуће стање (стохастичког) система зависи само од текућег (садашњег) стања, а не зависи од претходних стања

• Наредна генерација решења добија се само на основу претходне, стога ГА/ЕА представља Марковљев ланац

P9

У суштини је једна имплементација ГА/EA

– ГА: селекција и укрштање (нема мутације!)

– ЕА: селекција и варијација

• Алгоритам ради са скупом решења (популацијом)

• Почетни корак је иницијализација полазне популације

– типично на случајан начин

– израчунавање описне функције за полазну популацију

Селекција

• За свако решење x из текуће популације изаберу се три различита вектора (xa, xb и xc)

– популација мора да има 4 или више елемента

• xa, xb и xc морају бити различити међусобно и различити од x

• Дефинишу се параметри

– Диференцијални тежински фактор F из интервала од 0 до 2

– Вероватноћа укрштања CR из интервала од 0 до 1

Полазећи од изабраних xa, xb и xc

• Формира се међурешење z = xa + F\*(xb - xc)

• Како је xb ≠ xc нови вектор z је сигурно различит од xa, xb и xc

Укрштање (или оператор варијација)

• Изабере се случајан цео број R између 1 и D, где је D број димензија оптимизационог простора

– Смисао случајног броја R је да ће променљива под редним бројем R сигурно бити замењена

– Тиме се обезбеђује да резултат укрштања увек буде различит од полазног решења!

• За сваки елемент вектора решења (променљиву) изабере се случајан број ri са униформном расподелом од 0 до 1

Услов за прихватање новог решења

• За свако решење xk из популације формира се ново решење yk ,k = 1,2,…Npop

• У меморији се чувају две популације

– текућа популација решења x1, x2,… xNpop и

– популација са кандидатима y1, y2,… yNpop

• Уколико је f(yk) < f(xk) xk се замењује са yk у текућој популацији

• Уколико је f(yk) ≥ f(xk) xk остаје у текућој популацији

Примена за SAT и TSP проблеме?

• SAT проблеми:

– најједноставнија имплементација је тумачити бите као реалне бројеве

– извршити израчунавње и

– заокружити резултат за сваки бит на 0 или 1

• TSP проблеми:

– потребно је дефинисати разлику два пермутована низа

– добијени алгоритам више личи на ГА него на диферецијалну еволуцију

– отворено је питање ефикасности за TSP проблеме

Настајање норми и култура у оквиру једног друштва је оптимизациони процес!

• Симулације друштвених процеса су истовремено и алгоритми за решавање оптимизационих проблема

• Мисли појединца могу се схватити као један елемент друштва (скупа појединаца)

Дељење информација појединих јединки је корисно за опстанак и напредак групе тих јединки

– Јединка има своје знање/информације

– Знање се преноси кроз групу (кроз веровања, предрасуде, понашање, социјалне мреже итд. )

– Култура настаје као резултат тог процеса (култура оптимизира спознају – резултати се преносе на јединке које су далеко у групи)

• Имитирање других јединки, које раде неку операцију боље, је један (могући) алгоритам за оптимизацију

Одлука се доноси на основу два скупа информација

– Информације које појединац сам прикупи (градиво, наставник, начин полагања итд.)

– Информације које појединац добије од других чланова групе (познаника)

PSO терминологија

• Агент (елемент, честица) је једно могуће решење у оптимизационом простору

– вектор координата решења, x = (x1,x2,…xD)

• Јато (група) је скуп агената помоћу којих се врши оптимизација (исто као популација код ГА)

• PSO је оптимизациони алгоритам који врши операције над скупом решења

Оптимизација и кретање јата

• Агент води рачуна о најбољем решењу које је пронашао током оптимизације (pbest)

• Јато води рачуна о најбољем решењу које су пронашли сви агнети јата (gbest)

• Агенти мењају своју позицију у (оптимизационом) простору на основу ова два решења (pbest и gbest)

Промена брзине

• Промена брзине се рачуна према формули

• vn-1 је брзина агента у n-1 кораку

• rand() је функција која генерише случајан број у интервалу [0,1]

• w је коефицијент инерције

• c1 је когнитивни коефицијент

• c2 је социјални коефицијент

PSO параметри и стабилност: закључци

• PSO није апсолутно стабилан алгоритам

• “Експлозија” јата дешава се уколико параметри нису добро подешени

• Посебно, коефицијенти c1 и c2 морају бити мањи од 4

– Ова граница је теоретски показана

• Смањивање w води ка брзој конвергенцији повећавање w води ка расипању јата

Тополошки PSO: варијације

• Уместо читавог јата агент има информације само о околини

• “Ring” топологија је најчешћа, са 2 суседа

• Може се генерализовати на “m-околину

PSO закључци

• Једноставан оптимизациони алгоритам

• Стохастички оптимизациони алгоритам

– случајно генерисане полазне позиције

– rand() у сваком кораку алгоритма

• Може да се заустави у локалном оптимуму, ако сви агенти исконвергирају ка њему (нема даљег начина за излазак!)

• На граници између локалних и глобалних

Основна идеја и терминологија

• Колонија мрава = популација

• Позиција мрава = једно могуће решење оптимизационог проблема

• Координате позиције одговарају оптимизационим променљивима

• Идеја: што више мрава прође неком путањом то је та путања боља

Наредна решења

• Вероватноћа испитивања наредних решења (позиција) зависи од два параметра

– Атрактивности решења

– Нивоа пута (колико је мрава прошло туда)

• Ови параметри се мењају током оптимизације

• Релативно компликовано рачунање Параметара

P10

Методе више минимума засноване на понављању

• У пракси, оптимизациони алгоритми се покрећу више пута са различитим почетним решењима

• Такав приступ се примењује уколико је:

– потребно пронаћи више различитих решења или

– доказати да бољих решења (највероватније) нема

• Три основна корака

Блок дијаграм

• Генерисање скупа почетних решења

• Селекција (нај)бољих решења

• Оптимизација са полазним решењем добијеним у претходном кораку

Понављање основних оптимизационих алгоритама са случајно изабраном почетном тачком

– градијентна метода или симплекс са случајно изабраним полазним решењем

– симулирано каљење са случајном полазном тачком

– генетички алгоритам са случајно изабраном првом генерацијом

– PSO или диференцијална еволуција са случајно изабраним полазним популацијама

Вишеструка понављања: литература и примена

• Вишеструко понављање ГА се у литератури често назива микро-генетички алгоритам (microgenetic algorithm)

• Вишеструко понављање симулираног каљења се назива прекаљивање (reannealing)

• Ови алгоритми се могу користити за:

– проналажење једног (глобалног) оптимума и

– проналажење више решења

Особине поновљених оптимизација

• Добра страна метода заснованих на понављању је лака имплементација

• Основна мана алгоритама ове класе је недостатак корелације између појединачних покретања

• Ово има за последицу то да су крајња решења која се добијају после појединачних покретања алгоритама међусобно независна

• Вероватноћа проналажења истог решења је релативно велика, што је неефикасан начин оптимизације

Претпоставимо да оптимизациона функција има N минимума

– униформно распоређених у простору и

– једнаких дубина

• Вероватноћа проналажења сваког минимума је стога једнака и износи 1/N

• Колика је вероватноћа проналажења свих N минимума после k независних покушаја?

Процена локалних минимума

• Генерисање полазног скупа решења која се налазе у целом оптимизационом простору

• Процењивање положаја локалних минимума

• Покретање “основног” оптимизационог алгоритма

Процена локалних минимума

• Проценом локалних минимума пожељно је пронаћи околине свих минимума са најмањим могућим бројем рачунских операција

• Процена се може извршити на много различитих начина

• Један једноставан начин да се то уради јесте да се за сваку тачку из основног скупа пронађе М најближих тачака и уколико је вредност функције грешке у тој тачки мања од свих М суседних, тачка се прогласи локалним минимумом

Поређење метода на аналитички задатим функцијама

• Циљ: сагледати особине и утицај параметара алгоритама

• Стохастички алгоритми: резултат је случајна променљива

– посматрамо средњу вредност најбољег решења после одговарајућег броја итерација

– може се посматрати и девијација средњег најбољег решења

P11

Оптимизациона функција

• До сада смо сматрали да је позната или дефинисана једна оптимизациона функција

• То је тачно уколико постоји само један критеријум оптимизације

Посматраћемо пример куповине

• Купац, у принципу, увек води рачуна о два критеријума који су супротстављени

– Критеријум #1: максимизирати добит (купити што квалитетније или што више)

– Критеријум #2: минимизирати количину новца

• Да ли у таквим случајевима постоји једно најбоље (оптимално) решење?

Ако фиксирамо други критеријум (количину новца)

– тада је оптимално решење оно које максимизира добит (први критеријум)

• Ако фиксирамо први критеријум (добит)

– тада је оптимално решење оно које минимизира дату количину новца (други крит.)

• Уколико дозволимо постојање опсега вредности по оба критеријума, нема једног оптималног решења!

Геометријско место тачака у простору оптимизационих критеријума чији је број димензија највише *G*−1, где је *G* број критеријума

– уколико постоји 2 критеријума парето фронт је крива,

– уколико постоји 3 критеријума парето фронт је површ,

– уколико постоји 4 критеријума парето фронт је запремина, итд.

• Уколико постоје несупротстављени критеријуми, број димензија парето фронта се (у принципу) смањује

Потпуна претрага и провера да ли решење припада парето фронту

• Најједноставнији начин за проналажење парето фронта је провера свих тачака у оптимизационом простору да ли припадају парето фронту или не

• Тачке се могу генерисати

– потпуним претраживањем (за опт. просторе са коначним бројем решења)

– систематским претраживањем

– случајним претраживањем

• Ово је изузетно неефикасан начин, јер у пракси најчешће није могуће потпуно претражити оптмизациони простор

• Ефикаснија решења: коришћење оптмизационих алгоритама

Теорија и инжењерска пракса

• Теорија: парето фронт се добија ако узмемо све могуће вредности за тежинске факторе и урадимо оптимизацију за сваку комбинацију

• Бесконачно вредности коефицијената+ бесконачно комбинација тежинских фактора

= непрактичан приступ

• Које тежинске коефицијенте узети у пракси?

• Више покушаја или морамо имати предзнање о појединачним критеријумима

• Генерално, неефикасан начин и зависан од предзнања корисника

Nondominated sorting GA (NSGA)

• Заснива се на стандардном ГА са изменама у делу за рачунање описне функције

• Израчунавају се описне функције за сваки критеријум, за свако решење у генерацији

• Проналазе се сва (nondominated solutions) решења која одговарају процењеном парето фронту у текућој генерацији

• Додељује се rank#1 и понавља се док сва решења нису рангирана

NSGA: одређивање парето фронта

• Описна функција на основу које се бирају решења за укрштање се рачуна на основу ранга (први ранг је бољи од осталих)

• Укрштање и мутација се раде на стандардан начин примерен ГА

• Парето фронт се памти и побољшава током генерација (тј. током оптимизације)

Предности и мане NSGA и ПЛМ алгоритама

• NSGA:

– нема тежинских фактора

– парето фронт се проналази у једном покретању

– спора конвергенција (дуготрајна оптимизација)

– потенцијални проблеми са оштрим минимумима

• ПЛМ:

– бржа конвергенција од NSGA

– мање зависан од тежинских коефицијената од вишеструког покретања основних опт. алгоритама

– избор тежинских фактора зависи од проблема

Алгоритми за проналажење парето фронта

• MOGA: Multi-Objective Genetic Algorithm

• NPGA: Niched Pareto Genetic Algorithm

• Predator-Prey Evolution Strategy

• Distributed Reinforced Learning Approach– агентима додељене различите опт. функције

• NCGA: Neighborhood Constrained GA – критеријуми се претварају у ограничења

• Nesh GA – поправљање једног по једног критеријума (док се остали не дирају)

• MOEA: Multi-Objective Evolutionary Algorithms