



Heuristické optimalizačné procesy

Únik z lokálneho optima, hybridné lokálne prehľadávanie

prednáška 5
Ing. Ján Magyar, PhD.
ak. rok. 2023/2024 ZS

Prístupy

- opakované reštarty
- zväčšenie okolia
 - prepínanie okolí rôznych veľkostí
- dlhý krok mimo okolia
- akceptácia horšieho riešenia
- pridanie pamäte
- dynamické lokálne prehľadávanie

Opakované reštarty

po dosiahnutí lokálneho optima je algoritmus reinicializovaný
vhodná stratégia ak:

- počet lokálnych optím nie je príliš veľký
pri optimalizačnom probléme vhodné aj pri väčšom počte
lokálnych optím
- cena reštartu nie je príliš nákladná

Štruktúra s reštartom

input: π , max

output: $r \in S$

$r \leftarrow \square$, $g(r) = -\infty$

$i = 1$

repeat

$s = urp()$

while ($\#I(s) > 0$)

$s = select(I(s))$

endwhile

if($g(s) > g(r)$) **then**

$r = s$

endif

$i = i + 1$

until $i > \text{max}$

if($valid(r)$) **then**

return r

else

return \square

endif

Väčšie okolie

viac kandidátov v okolí (väčšia šanca na lepšieho kandidáta)

graf okolia s menším priemerom

lokálne optimum malého okolia nemusí byť optimom väčšieho okolia

rastú nároky na čas potrebný pre preskúmanie okolia

modifikácie

- orezávanie okolia
- pivotné pravidlo (funkcia *step*)

Pivotné pravidlo

$$I(s) = \{ x \in N(s) \mid g(x) > g(s) \}$$

$$I^*(s) = \{ x \in N(s) \mid g(x) = \max \{ g(y) \mid y \in N(s) \} \}$$

najlepšie zlepšenie: $p(x) = 1 / \#I^*(s)$ ak $x \in I^*(s)$, inak $p(x) = 0$

prvé zlepšenie $p(x_i) = 1$ ak $x_i \in I(s)$ a $\neg(x_{i-j} \in I(s))$

náhodné prvé zlepšenie

náhodné zlepšenie $p(x) = 1 / \#I(s)$ ak $x \in I(s)$, inak $p(x) = 0$

Prepínanie okolí

používanie okolia

- malé okolie (zlepšovanie aktuálneho kandidáta)
- veľké okolie (zabránenie uviaznutiu)

prepínanie okolí - VNS algoritmy, napríklad VND

- k okolí $N_1 < N_2 < \dots < N_k$
- zväčšovanie okolia: $N_i \rightarrow N_{i+1}$
- zmenšovanie okolia: $N_i \rightarrow N_1$
- začiatok: N_1
- koniec: uviaznutie v N_k

Štruktúra VND

input: π

output: $r \in S$

$r = urp()$

$i = 1$

repeat

$s = best(N_i(r))$

if($g(s) > g(r)$) **then**

$r = s$

$i = i + 1$

else

$i = i + 1$

endif

until $i > k$

if(*valid*(r)) **then**

return r

else

return

endif

Dlhý krok

kompozícia dlhého kroku z viacerých krokov v malom jednoduchom okolí

- kroky pozostávajú z rôzne dlhých sekvencií jednoduchých krokov
- sekvencie spĺňajú ohraničenia na prijateľnosť (napríklad reštrikcia ceny, tabu reštrikcia)

algoritmy VDS - príkladom je Lin-Kernighan algoritmus (TSP)

- sekvencia 2-exchange krokov

Štruktúra VDS

input: π

output: $r \in S$ \square

$r = urp()$

repeat

$t = r$

repeat

$s = t$

$t = bestfeasible(N(s))$

until $termconstruct(t, s)$

$impr = not$

if($g(r) < g(s)$) **then**

$r = s$

$impr = yes$

endif

until not $impr$

if($valid(r)$) **then**

return r

else

return \square

endif

Akceptácia horšieho riešenia

horší kandidát

- jediná možnosť (uviaznutie v lokálnom optime)
- jedna z možností (existuje aj lepší kandidát)

stratégia výberu zhoršujúceho kroku

- iná možnosť nie je
- pravidelná alternácia
- pravdepodobnostný výber

pravdepodobnosť zhoršujúceho kroku

- pevná (RII)
- adaptívna (PII)

Randomizované iteračné zlepšovanie

funkcia *step*

- zlepšujúci krok - $step_{ii}$
- zhoršujúci krok - $step_{urw}$
- parameter wp : $step = wp * step_{urw} + (1 - wp) * step_{ii}$

zmena terminačného kritéria

- dosiahnutie limitu
- absencia zlepšenia

únik z lokálneho extrému

dosiahnutie globálneho optima

Štruktúra RII

input: π

output: $r \in S$

$s = urp()$

$r = s$

repeat

if($rand(0-1) < wp$) **then**

$s = step_{urw}(s)$

else

$s = step_{ii}(s)$

endif

if($g(s) > g(r)$) **then**

$r = s$

endif

until $term(s)$

if($valid(r)$) **then**

return r

else

return

endif

Pravdepodobnostné iteračné zlepšovanie

funkcia *step* je dvojkrokový proces

1. výber kandidáta: $p(x) = 1 / \#N(s)$, $x \in N(s)$
2. akceptovanie kandidáta: $p_a = (1 + e^{\Delta/k})^{-1}$

príklad: simulované žíhanie

Štruktúra PII

input: π

output: $r \in S$

$s = \text{urp}()$

$r = s$

repeat

$s' = \text{urw}(N(s))$

$p_a = f_{akc}(s')$

if($\text{rand}(0-1) < p_a$) **then**

$s = s'$

if($g(s) > g(r)$) **then**

$r = s$

endif

endif

until $\text{term}(s)$

if($\text{valid}(r)$) **then**

return r

else

return \square

endif

Zakázané lokálne prehl'adávanie

funkcia *step* - najlepšie zlepšenie

$$p(x) = 1 / \#I^*(s) \text{ ak } x \in I(s)\text{-}s, \text{ inak } p(x) = 0$$

krátkodobá pamäť znemožňuje návrat

zabránenie (posledne) skúmaných kandidátov

dynamická reštrikcia okolia

obsah pamäti

reverzie predchádzajúcich transformácií

iba dočasné uchovávanie obsahu

reaktívne zakázané prehl'adávanie

realizácia krátkodobej pamäte

Rozšírenia zakázaného prehl'adávaní

ašpiračné kritérium

podmienka ignorovania krátkodobej pamäte

dlhodobá pamäť

cieľom je rovnomerné používanie transformácií

frekvencia použitia transformácií

výber transformácie $g(t_i(s))$ vs $g(t_i(s)) + k * DP(t_i)$

použitie dlhodobej pamäte

Štruktúra TS

input: π

output: $r \in S[\square]$

$s = urp()$

$r = s$

repeat

$s' = ac(N(s))$

if(s') **then**

$s = s'$

else

$s = step(restr(N(s), M))$

endif

$update-memory(M, s)$

if ($g(s) > g(r)$) **then**

$r = s$

endif

until $term(s)$

if($valid(r)$) **then**

return r

else

return \square

endif

Dynamické lokálne prehľadávanie

penalizácia lokálneho optima vedúca na degradáciu ohodnocovacej funkcie g

opakovanie až kým kandidát prestane byť lokálnym optimom

penalizácia

- stratégia penalizácie: additívna/multiplikačná schéma, pamäť, parametrizácia

- voľba komponentov lokálneho optima

$$g'(s) = g(s) + \sum_{i \in SC} pen(i)$$

- voľba komponentu s najväčšou užitočnosťou

$$util(i,s) = f_i(s) / (1 + pen(i))$$

Štruktúra DLS

input: π

output: $r \in \mathcal{S}$

$s = urp()$

$s = LS_{II}(s)$

$r = s$

repeat

$g = update-penalties(g, s)$

$s = LS_{II}(s)$

if($g_{orig}(s) > g_{orig}(r)$) **then**

$r = s$

endif

until $term(s)$

if($valid(r)$) **then**

return r

else

return \square

endif

Hybridné lokálne prehľadávanie

Hybridné algoritmy

kombinovanie jednoduchších stratégií do komplexnejších schém

pridanie triviálnych stratégií

- RII (II + URW)
- II s reštartmi (II + URP)

kombinovanie dvoch stratégií

- ILS
- GRASP
- AILS

Iteračné lokálne prehl'adávanie

nevyhýba sa lokálnemu optimu

dva typy krokov

- lokálne prehl'adávanie - dosiahnutie lokálneho optima
- perturbácia - štartovací bod pre nové lokálne prehl'adávanie

pohyb v priestore lokálnych optím - voľba optima pre pokračovanie

príklad: iteračný Lin-Kernighan (pre TSP)

Štruktúra ILS

input: π

output: $r \in S$ ☐

$s = \text{init}()$

$s = \text{localsearch}(s)$

$r = s$

while not $\text{term}(s)$

$s' = \text{perturb}(s)$

$s'' = \text{localsearch}(s')$

if($g(s'') > g(r)$) **then**

$r = s''$

endif

$s = \text{accept}(s, s'')$

endwhile

if($\text{valid}(r)$) **then**

return r

else

return ☐

endif

Lačné randomizované adaptívne hľadanie (GRASP)

prehl'adávanie začína z čo najlepšieho kandidáta

dva typy krokov

- tvorba štartovacieho kandidáta (konštrukčné prehl'adávanie)
- lokálne zlepšovanie kandidáta (perturbačné prehl'adávanie)

SGH - vynechanie perturbačnej fázy

Štruktúra GRASP

input: π

output: $r \in S$ \square

$r = \square$, $g(r) = -\infty$

while not *term*(s)

$s = \text{construct}()$

$s' = \text{localsearch}(s)$

if($g(s') > g(r)$) **then**

$r = s'$

endif

endwhile

if(*valid*(r)) **then**

return r

else

return \square

endif

GRASP - konštrukčná fáza

štartuje z prázdneho kandidáta

prechádza parciálnymi kandidátmi

- dopĺňa jednu zložku v každej iterácii

končí vytvorením úplného kandidáta

generuje rôznych úplných kandidátov

Obmedzený zoznam kandidátov

založený na počte

- pevne daný počet k
- náhodný alebo lačný výber

založený na ohodnotení (prah $k \in <0, 1>$)

$$g(z) \leq g(z_{\perp}) + k * (g(z_{\top}) - g(z_{\perp}))$$

GRASP *construct()* - greedy-random

construct()

$x = \square$

$V = \{ v_1, v_2, \dots, v_n \}$

while not *complete*(x)

$SC = \text{sort}(V, g)$

$RCL = \text{form}(SC, k)$

$v = \text{select}(RCL)$

$x = x + v$

$V = V - v$

endwhile

return x

endconstruct

GRASP *construct()* - random-greedy

construct()

$x = \square$

$V = \{ v_1, v_2, \dots, v_n \}$

while not *complete*(x)

$RCL = \text{sample}(V, k)$

$SRCL = \text{sort}(RCL, g)$

$v = \text{select-best}(SRCL)$

$x = x + v$


$V = V - v$

endwhile

return x

endconstruct

GRASP *construct()* - random+greedy

```
construct()  
  x =   
  V = { v1, v2, ..., vn }  
  for i=1, ..., k  
    v = select-random( V )  
    x = x + v  
    V = V - v  
  endfor  
  while not complete( x )  
    SV = sort( V, g )  
    v = select-best( SV )  
    x = x + v  
    V = V - v  
  endwhile  
  return x  
endconstruct
```

Adaptívne iteračné konštrukčné hľadanie (AICS)

spätná väzba

- forma súboru váh
- váhy sú adaptované v každej iterácii
 - podľa komponentov tvoriacich dosiahnutého kandidáta
 - podľa kvality dosiahnutého kandidáta

použiteľné aj bez perturbačného hľadania

príklad: SWO (squeaky wheel optimization)

Štruktúra ALCS

input: π

output: $r \in S \mid \square$

$r = \square$, $g(r) = -\infty$

$w = \text{initweights}()$

while not $\text{term}(s)$

$s = \text{construct}()$

$s' = \text{localsearch}(s)$

if($g(s') > g(r)$) **then**

$r = s'$

endif

$w = \text{adaptweights}(s', w)$

endwhile

if($\text{valid}(r)$) **then**

return r

else

return \square

endif

otázky?