Izdelava urnikov

Nejc Širovnik

14. 10. 2019

Povzetek

V članku so predstavljeni nekateri postopki za avtomatsko izdelavo urnikov. Opisani so potrebni vhodni podatki, naštete so omejitve, ki jih je potrebno upoštevati pri izdelavi urnikov, predstavljena je tudi metoda merjenja kakovosti izdelanega urnika.

1 Program za izdelavo urnika

Spominjam se osnovnošolskih in srednješolskih dni, ko smo vsi z nestrpnostjo pričakovali začetek novega šolskega leta, ki pa je nemalokrat prinesel zelo hladen tuš ob pogledu na natrpan urnik. Zakaj moramo v ponedeljek tako zgodaj vstati, ko pa se teden še ni niti prav začel? Zakaj nam vsaj v petek niso malo popustili? Kdo je sploh lahko izdelal takšen urnik?

To so vprašanja, na katera seveda nikoli nismo dobili odgovora, čeprav je le-ta na zadnje vprašanje več kot očiten – izdelala ga je oseba, ki so ji to delo naprtili. Naprtili je zelo primerna beseda, saj gre za zahtevno opravilo, ki izdelovalcu urnikov nemalokrat pusti ogromno sivih las. Ne bi se rad postavil v kožo ubožca, ki konča z izdelavo urnika, nakar ves zgrožen ugotovi, da je spregledal prekrivanje dveh šolskih ur, kar pomeni, da bi učenci morali biti pri dvoje urah hkrati. Tako mu seveda ne preostane drugega, kot da poskusi znova ter po nekaj neuspelih poskusih vendarle izdela urnik, na katerega pa vseeno leti kopica pripomb.

Naprednejše ročne metode izdelovanja urnikov potekajo s pomočjo modelov in tabel, vendar tudi te pogosto ne vodijo do kakovostnih urnikov. Pri vseh je namreč prisoten faktor omejenosti človeških možganov, ki se po hitrosti obdelave podatkov niti približno ne morejo kosati z današnjimi procesorji.

Prav zmogljivost procesorjev je tista, ki že od izuma računalnika žene programerje k ustvarjanju optimizacijskih algoritmov za iskanje kakovostnejšega urnika. Prav kakovost pa je lahko dvorezen meč, saj nekateri urniki zadovoljijo potrebe učencev, a se premalo osredotočajo na potrebe učiteljev, in obratno. Kateri urnik je potemtakem najkakovostnejši? Ali tak urnik sploh obstaja?

Znano je, da so za izdelavo najzahtevnejši fakultetni urniki, zato od sedaj naprej obravnavamo le slednje. Seznam omejitev, ki jih moramo upoštevati pri teh urnikih, je namreč obsežnejši kot pri urnikih osnovnih in srednjih šol.

1.1 Vhodni podatki

Če želimo izdelati urnik, v prvi vrsti potrebujemo podatke, ki jih moramo upoštevati in kasneje tudi obdelati. Te podatke preberemo iz podatkovnih baz ali drugih vrst datotek, med katerimi vrh prestola zasedajo XML datoteke. Naslednji primer ponazarja, kako je v XML datoteki predstavljen profesor Janez Novak, ki trikrat na teden predava predmet matematika.

```
<profesor>
 <ime>Janez</ime>
 <priimek>Novak</priimek>
 <predmet>
    <imePredmeta>matematika</imePredmeta>
    <steviloUrNaTeden>3</steviloUrNaTeden>
 </predmet>
</profesor>
```

Hierarhija značk omogoča enostavno vnašanje podatkov, hkrati pa poskrbi za preglednost. Še enostavnejše je dostopanje do posameznih podatkov, saj večina programskih jezikov že vsebuje metode za delo z XML datotekami.

Pri objektno orientiranih programskih jezikih (Delphi, C++, Java) vse vhodne podatke opišemo s posameznimi razredi. Poglejmo, katere razrede je smiselno implementirati ter katere podatke naj vsebujejo.

- Razred *Profesor* vsebuje osebne podatke profesorja (ime in priimek) ter podatke o predmetih, ki jih predava. Hkrati moramo za vsako uro v tednu vedeti, če je profesor takrat že zaseden, da mu slučajno ne bi dodelili dveh predavanj v istem časovnem terminu. To je najlažje preverjati s pomočjo tabele zasedenosti, ki je podrobneje opisana na koncu poglavja.

- Razred *Skupina* predstavlja skupino študentov. Kot podatke torej potrebujemo smer skupine (npr. pedagoška matematika) ter število študentov v njej. Podobno kot pri profesorjih, tudi tukaj potrebujemo tabelo zasedenosti.

- Razred *Predavalnica* nosi podatke o kapaciteti predavalnice, njeno oznako ter podatke o zasedenosti, ki so predstavljeni s tabelo.

- Razred *Predmet* vsebuje podatke o posameznih predmetih. Kot podatke tega objekta je smiselno navesti ime predmeta ter tedensko število ur njegove izvedbe. Zanima nas tudi, katerim skupinam je predmet namenjen, ter v katerem semestru se bo predaval.

Kar pri treh razredih je kot podatek navedena tabela zasedenosti, zato si jo podrobneje oglejmo. Vsako tabelo določata njena velikost ter tip vrednosti. Naslednji izračun nam poda velikost tabele:

$$textVelikostTabele = \text{ŠteviloUrNaDan} \cdot \text{ŠteviloDelovnihDni}$$

Če bi predpostavili, da študijski dan traja od sedme ure zjutraj do osmih zvečer in da imamo pet delovnih dni, je velikost naše tabele 65 polj. Velikost te tabele torej pove, koliko ur na teden obravnavamo. Ker se šolski tedni periodično ponavljajo, je seveda dovolj, če podamo zasedenost le za en teden.

Tip vrednosti tabele ni striktno določen. Tako bi nekateri programerji uporabili celoštevilsko tabelo, kjer bi vrednost 1 pomenila, da je oseba zasedena, vrednost 0 pa bi predstavljala njeno dosegljivost. Podobno bi postopali, če bi izbrali logični tip (npr. bool v C++), kjer bi vrednost true predstavljala dosegljivost, false pa zasedenost.

1.2 Omejitve

Programa za izdelavo (nekakovostnega) urnika načeloma ni težko narediti. Potrebujemo le metodo, ki naključno prireja termine posameznim predmetom. Težava pri takšnem postopanju je, da pri naključnem vstavljanju nismo upoštevali nobenih omejitev, prav zadoščanje omejitvam pa je tisto, kar odlikuje dober urnik.

1.2.1 Stroge omejitve

Verjetno še niste videli profesorja, ki vsakih pet minut teče iz ene predavalnice v drugo, ker so mu izdelalovalci urnika dodelili dve predavanji v istem časovnem terminu. Razlog je preprost – vsak urnik mora biti izdelan tako, da zadošča vsem strogim omejitvam. Poglejmo si nekaj omejitev, ki jih urnik ne sme kršiti:

- V urnik moramo razdeliti vse ure vseh predmetov.
- Noben profesor ne sme imeti več predavanj hkrati.
- Nobena skupina ne sme imeti več predavanj hkrati.
- Nobena predavalnica ne sme biti dodeljena večim predavanjem hkrati.
- Predavalnica mora imeti zadostno kapaciteto.
- Predavanja posebne vrste (npr. vaje iz računalništva) se morajo izvajati v ustrezni predavalnici.
- Urnik mora zadoščati zastavljenemu časovnemu okvirju (npr. od sedmih zjutraj do osmih zvečer).

Urnik, ki zadošča vsem strogim omejitvam, imenujemo zadosten urnik. Zadostni urnik tako ne krši nobenega izmed zgoraj naštetih načel, a še zdaleč ne ustreza merilom kakovostnega urnika.

Algoritem je najbolje implementirati tako, da do kršenja strogih omejitev sploh ne more priti. Tabela zasedenosti je dober primer pametne implementacije, saj tako že v osnovi preprečujemo dodeljevanje t.i. prekrivanj. Če želi algoritem dodeliti predavanje določeni predavalnici, mora najprej v tabeli zasedenosti preveriti, če je v tistem terminu le-ta še na voljo, sicer se vstavljanje ne izvede.

1.2.2 Šibke omejitve

V splošnem ne zahtevamo, da urnik zadošča vsem šibkim omejitvam, ker je to praktično neizvedljivo. Resda pa so prav šibke omejitve tiste, ki določajo kakovost urnika, in dober algoritem bo poskrbel, da se bo zadostilo večini leteh. Naštejmo nekaj izmed njih:

- Urnik naj vsebuje čim manj prostih ur za profesorje in študente.
- Študentje naj se čim manj selijo iz predavalnic.
- Predmet z veliko urami naj bo enakomerno porazdeljen čez celotno periodo (teden).
- Predavanja naj se izvajajo ob urah, ki so najugodnejše za poučevanje.
- V petek naj se študentom dodeli čim manj predavanj.
- Študentom se naj v času kosila dodeli prosta ura.

Obstaja še mnogo drugih šibkih omejitev, ki se jim je smiselno posvetiti. Nikomur najbrž ne bi bilo preveč všeč, če bi uram športne vzgoje sledila še vrsta predavanj, saj bi tako prepoteni in izčrpani študentje težko sledili tempu pouka.

Pri ročnem sestavljanju urnika se ponavadi sestavi zadostni urnik, ki zadošča le najpomembnejšim šibkim omejitvam. Dobro zasnovan algoritem pa lahko preverja veliko število šibkih omejitev, zato snovalci programov prav temu posvečajo ogromno pozornosti, saj bo njihov končni produkt le tako zadovoljil profesorje in študente.

1.3 Merjenje kakovosti

Kakovost urnika se meri z zadoščanjem šibkim omejitvam, pri čemer že predpostavljamo zadostnost urnika (zadoščanje strogim omejitvam). Ta način je smiseln predvsem zato, ker si urnik, ki ne zadošča strogim omejitvam, tega naziva sploh ne zasluži.

Človek, ki najprej ustreli in šele nato razmisli, bi verjetno dejal, da število zadoščenih šibkih omejitev določa kakovost urnika, kar pa je daleč od resnice. Medtem ko so stroge omejitve po pomembnosti (skoraj) enakovredne, se šibke omejitve med seboj krepko razlikujejo. Tako je bolj pomembno, da urnik vsebuje manj prostih ur za študente in profesorje, kot pa je dodeljena prosta ura v času kosila. Kako bi torej ovrednotili posamezne šibke omejitve?

V splošnem uporabljamo standardni postopek uteževanja omejitev, pri čemer vsaki šibki omejitvi dodelimo celoštevilsko utež glede na njeno pomembnost. Vprašanje je tudi, kako določiti to pomembnost. Če programer določa uteži po lastni presoji, so zagotovo zelo subjektivno dodeljene. Dosti bolj smiselno je anketiranje, kjer bi študentje in profesorji različnih izobraževalni ustanov ovrednotili posamezne šibke omejitve z ocenami od 1 do 10. S tem bi nastavljene uteži pridobile na objektivnosti, verjetnost, da bo urnik dobro sprejet s strani večine, pa bi se povečala.

Kot zanimivost naj omenimo, da se nekateri programerji odločajo tudi za utežitev strogih omejitev s to razliko, da so vrednosti teh uteži izjemno velike, zato se bo algoritem (skoraj) vedno "odločil", da jih ne bo prekršil. Tak način uteževanja pride v poštev takrat, ko algoritem v osnovi ne zadosti strogim omejitvam urnika.

Ko enkrat ustrezno nastavimo vrednosti uteži, je treba ugotoviti, kako bomo postopali v nadaljevanju. Algoritem za izdelavo urnikov je najbolje zasnovati tako, da po vsaki spremembi izmeri njegovo kakovost. Tako moramo na vsakem koraku ugotavljati, katere omejitve so bile kršene ter kolikokrat se kršitev pojavi.

S k_n označimo število kršitev omejitve n, z u_n pa njeno utež. Z naslednjo formulo izračunamo kakovost urnika:

$$k_1 \cdot u_1 + k_2 \cdot u_2 + \ldots + k_{n-1} \cdot u_{n-1} + k_n \cdot u_n$$

Nižja vrednost formule pomeni kakovostnejši urnik, izračunamo pa jo na vsakem koraku. Korak pri izdelavi urnika predstavlja eno izmed operacij nad njim (označuje njegovo spremembo).

V 2. poglavju bomo opisali postopek gradnje urnika, saj bomo podrobneje spoznali operacije nad urnikom in več vrst optimizacijskih metod, ki jih lahko brez oklevanja poimenujemo kar osrčje programa za avtomatizirano izdelavo urnika.

2 Avtomatizirana izdelava urnika

V 1. poglavju smo izvedeli, da ročna izdela urnikov ni tako preprosta, kot bi si na prvi pogled predstavljali. Še večji izziv predstavlja implementacija algoritma, ki da (fakultetni) urnik. Obstaja pa velika razlika med prvim in drugim pristopom, saj ko je algoritem enkrat dobro napisan, smo se težav z izdelovanjem urnika za vselej rešili. Tako nas vsako leto čaka le spreminjanje vhodnih podatkov, ki jih algoritem sprejme in nato iz njih izdela celovit urnik. Te vhodne podatke smo si skupaj s strogimi in šibkimi omejitvami že podrobneje ogledali v prvem delu, tako da se lahko tokrat osredotočimo na osrčje programa.

Za lažjo ilustracijo bi lahko potegnili kar nekaj vzporednic med programiranjem in – ne boste verjeli – kuhanjem. Za kuhanje potrebujemo recept, ki je sestavljen iz dveh delov. Prvi del predstavljajo sestavine, ki jih potrebujemo za kuhanje, glavni del pa predstavlja postopek, kako iz teh sestavin pripraviti okusno kosilo. Postopki kuhe se med seboj razlikujejo in nekateri pripeljejo do nekoliko slastnejšega obroka kot drugi.

Če se ob upoštevanju zgornjih opomb vrnemo k programiranju, sestavine v našem programu že imamo, to so namreč podatki o profesorjih, predavalnicah, skupinah ter predmetih. Prav tako imamo izbrane uteži za posamezne omejitve. Manjka le še glavni del recepta, ki podatke ustrezno obdela ter iz njih sestavi urnik. Obstaja več postopkov oziroma metod, ki vodijo do končnega urnika, podobno kot pri kuhi pa je tudi tukaj kakovost končnega izdelka odvisna od izbire najustreznejše metode.

Končni produkt, ki ga bomo ustvarili s pomočjo kasneje omenjenih metod, bo urnik, ki je predstavljen z množico *srečanj*. Posamezno srečanje je sestavljeno iz predmeta, profesorja, ki predmet predava, ter skupin, ki jim je predmet namenjen. Srečanja so nato vstavljena v termine predavalnic, tako da se za vsako srečanje ve, ob kateri uri in v kateri predavalnici se bo izvajalo.

2.1 Požrešna metoda

Požrešna metoda je ena izmed strategij, s katerimi lahko rešujemo optimizacijske probleme - to pomeni, da za dani problem iščemo najboljšo (optimalno) rešitev. Metoda na vsakem koraku izmed vseh možnih rešitev izbere tisto, ki daje (trenutno) največji dobiček oziroma najbolj poveča vrednost kriterijske funkcije.

Podobno kot požrešna metoda nemalokrat postopamo tudi sami. Božični nakupi v trgovine prinašajo dolge čakalne vrste in ponavadi se odločimo, da bomo z vozičkom stopili do blagajne, na kateri čaka najmanj ljudi. Šele po polurnem polžjem približevanju blagajni ugotovimo, da bi pri večini drugih blagajn prišli na vrsto prej, saj so tam ljudje opravljali le manjše nakupe.

Znano je, da požrešna metoda pri nekaterih problemih odpove, je pa vseeno zelo uporabna tehnika za iskanje rešitev, ko ne zahtevamo njihove optimalnosti. Urnik, ki ga dobimo s pomočjo požrešne metode, je lahko namreč še vedno za nekaj nivojev boljši od ročno izdelanega. Naslednje vrstice bodo pojasnile, kako implementirati postopek požrešne metode za primer urnika.

```
 (1) Sprehod po vseh predmetih.
{
   (2) Sprehod po vseh predavalnicah.
   {
      (3) Računanje cene za proste termine.
      (4) Primerjava cene z najboljšo.
   }
   (5) Vstavljanje srečanja v urnik.
}
```

Zgoraj predstavljena shema je le grob opis postopka, ki ga nad urnikom izvaja požrešna metoda, zato si je smiselno podrobneje ogledati posamezne korake.

1. Z zanko se sprehodimo po vektorju vseh predmetov. Predmet na i-tem koraku označimo s predmet_i.

2. Naslednja vgnezdena zanka nas popelje po vseh predavalnicah. Predavalnico na j-tem koraku označimo s predavalnica_i.

3. Za vse proste termine predavalnica_j izračunamo ceno vstavljanja srečanja_i, ki ga sestavlja predmet_i, profesor, ki ta predmet predava, ter skupina, kateri je predmet namenjen.

4. Če je trenutna cena boljša od najboljše dosedanje, potem najboljša dosedanja dobi vrednost trenutne. Hkrati si zapomnimo, za kateri termin smo izračunali ceno vstavljanja, da bomo predmet_i na koncu vstavili v termin z najboljšo ceno vstavljanja.

5. Ko preverimo vse proste termine vseh predavalnic, vstavimo srečanja_i v termin z najboljšo ceno.

6. Sedaj se ponovno vrnemo na začetek in pogledamo najugodnejši termin za naslednji predmet. Postopek ponavljamo, dokler se zunanja zanka, ki se sprehaja po vseh predmetih, ne zaključi. S tem smo namreč v urnik vstavili vse predmete.

2.2 Optimizacijske metode

Optimizacijske metode se že desetletja uporabljajo za reševanje optimizacijskih problemov. Nekatere manj, druge bolj uspešno rešujejo zapletene probleme, pri čemer je eksponentno naraščajoča zmogljivost procesorjev le voda na njihov

mlin, saj tako lahko v krajšem času izvedejo več operacij. Končne rešitve, ki jih dobimo z optimizacijskimi metodami, so lokalni ali globalni optimumi.

Če želimo spoznati optimizacijske metode, se moramo najprej seznaniti z dvema pojmoma. *Operacija* je vsaka sprememba oziroma poseg v urnik med izvajanjem optimizacije. Na primeru urnika si oglejmo, katere so smiselne operacije:

- *Vstavljanje* srečanja v urnik.
- *Brisanje* srečanja iz urnika.
- *Zamenjava* dveh srečanj v urnik.

Okolica operacije je množica vseh možnih sprememb, ki jih lahko s to operacijo opravimo na nekem koraku optimizacijskega postopka. Okolica zamenjave dveh srečanj v urniku je torej množica vseh možnih zamenjav srečanj v trenutnem urniku.

Sistem okolic je množica vseh možnih sprememb, ki jih lahko z vsemi operacijami iz okolic sistema opravimo na nekem koraku optimizacijskega postopka. Oboroženi z novimi pojmi se podajmo v naslednje vrstice.

2.2.1 Lokalno vzpenjanje

Lokalno vzpenjanje temelji na opazkah, da izbira širšega sistema okolic prinaša kakovostnejše rešitve. Sistem okolic v primeru urnika predstavljajo prej omenjene operacije vstavljanja, brisanja ter zamenjave srečanj. Osnovno lokalno vzpenjanje se teoretično sploh ne razlikuje od požrešne metode, saj tudi pri tem postopku iz okolice vzamemo najboljšo rešitev. Kaj pa, če bi lokalno vzpenjanje nadgradili?

Do sedaj smo vedno dopuščali le izbiro najboljše možne rešitve iz določene okolice, kar nas je hitro pripeljalo do lokalnega maksimuma, iz katerega pa se je bilo nemogoče izviti, saj je najboljša rešitev iz okolice lokalnega maksimuma ponavadi kar lokalni maksimum sam. Nadgradnja lokalnega vzpenjanja je jasno razvidna iz naslednje sheme.

```
 (1) Izvajaj dokler...
{
   (2) Iz sistema okolic izberi okolico.
   (3) Iz okolice izberi spremembo.
   (4) Izvedi spremembo.
}
```

Nadobudnežem priporočamo, da najprej sami razmislijo, kaj je globja vsebina posameznih korakov, in si šele nato ogledajo njihov natančen opis.

1. Zunanjo zanko izvajamo, dokler ne dosežemo enega izmed ustavitvenih pogojev. Če nekaj časa kljub spreminjanju operacij nad urnikom vedno znova obstojimo na isti rešitvi, potem to najverjetneje pomeni, da smo prišli do lokalnega ekstrema. Drugi ustavitveni pogoj je zadovoljivo velika vrednost kriterijske funkcije.

2. Naključno izberemo okolico trenutne rešitve. Z okolico izberemo tudi operacijo, ki jo izvajamo na tem koraku (npr. zamenjava dveh srečanj).

3. Če v tem koraku izberemo tisto zamenjavo srečanj, ki izmed vseh možnih najbolj poveča vrednost kriterijske funkcije, potem postopamo enako, kot požrešna metoda. Če želimo nadgraditi postopek lokalnega vzpenjanja, ne potrebujemo nujno najboljše možne zamenjave, ampak prvo, ki poveča vrednost kriterijske funkcije. Izjemoma dovolimo tudi tisto zamenjavo, ki vrednost kriterijske funkcije nekoliko zmanjša, kar se na prvi pogled zdi nesmiselno, nas pa dolgoročno pripelje do boljših rešitev.

4. Izbrano zamenjavo dveh srečanj izvedemo na trenutnem urniku. Vidimo, da en korak lokalnega vzpenjanja res predstavlja eno spremembo nad urnikom, v konkretnem primeru smo izvedli (eno) zamenjavo dveh srečanj.

Na tak način izboljšana metoda lokalnega vzpenjanja je časovno veliko zahtevnejša od požrešne metode, saj se vrednost kriterijske funkcije ne povečuje tako drastično. Vseeno pa vidimo, da se počasi približujemo lokalnemu maksimumu in sčasoma nas bo postopek pripeljal do njega. Naključna izbira okolic ter sprejemanje ne nujno najboljše rešitve nas varujeta, da ne bomo obstali v prvem lokalnem maksimumu, ampak bomo vedno znova iskali boljše rešitve.

Slabost do sedaj obravnavanih tehnik še vedno ostaja – ko enkrat obtičimo v lokalnem maksimumu, se iz njega ne moremo rešiti. Naslednja metoda poskuša na svojevrsten, a zelo sistematičen način odpraviti to težavo.

2.2.2 Iskanje s tabuji

Že samo ime spominja na prepovedi, ki jih le-ta skriva v svojem naročju. Kaj sploh prepovedujemo? Glede na to, da bi naj iskanje s tabuji odpravilo problem vračanja v isti lokalni maksimum, prepovedujemo izvajanje sprememb, s katerimi bi se vrnili v že najden lokalni maksimum.

Rešitev se ponuja kar sama – za naslednjih n korakov prepovemo vsako spremembo, ki jo opravimo v koraku optimizacije. Izbira konstante n je odvisna od samega problema, saj ne obstaja neki splošen n, ki bi nas v vsakem primeru pripeljal iz lokalnega ekstrema. Imamo torej seznam prepovedanih sprememb – *tabujev* in na vsakem koraku za izbrano spremembo preverimo, če je element seznama tabujev. Če spremembe ni v seznamu tabujev, jo preprosto izvedemo, v primeru, da je, pa postopek vrnemo na iskanje druge spremembe. Algoritem iskanja s tabuji se od lokalnega vzpenjanja torej razlikuje le v tem, da pred izbiro spremembe preveri, če je vsebovana v seznamu prepovedanih sprememb. Poglejmo, zakaj je takšno postopanje smiselno.

Naj bo $n = 100$ ter s_1 sprememba urnika, ki nas je pripeljala v lokalni maksimum. To spremembo (npr. zamenjava srečanja$_i$ in srečanja$_i$) sedaj shranimo v seznam tabujev. V naslednjem koraku izvedemo spremembo s_2, ki se prav tako shrani v seznam prepovedanih sprememb. Podobno naredimo za vse spremembe v nadaljnih korakih. S tem poskrbimo, da se lahko v isti lokalni maksimum vrnemo šele čez $n = 100$ korakov, saj se sprememba s_1, ki vodi do lokalnega maksimuma, šele tedaj izbriše iz seznama prepovedanih sprememb.

Če smo dosegli lokalni maksimum in vrednost kriterijske funkcije v n korakih uspemo rešiti iz okolice tega lokalnega maksimuma, potem se je metoda iskanja s tabuji izkazala kot uspešna.

2.3 Končni algoritem

Za konec si oglejmo, kako je videti optimizacijski del algoritma za iskanje najboljšega urnika, pri čemer je najboljši urnik tisti, ki s kršenjem omejitev pridobi najmanj negativnih točk, oziroma ki ima največjo vrednost kriterijske funkcije.

```
% začetek okolja verbatim
(1) r = rešitev, dobljena s požrešno metodo.
(2) Izvajaj, dokler...
{
   (3) Z optimizacijo izboljšaj r.
   (4) Oceni r.
}
```

1. V prvem koraku izvedemo požrešno metodo, ki rešitev urnika pripelje v lokalni maksimum.

2. Zanko izvajamo, dokler ne dosežemo enega izmed ustavitvenih pogojev.

3. Rešitev r, ki smo jo dobili s pomočjo požrešne metode, izboljšamo z lokalnim vzpenjanjem ali iskanjem s tabuji. Kot je že bilo povedano, je iskanje s tabuji nekoliko primernejša metoda, saj nas bo lažje pripeljala iz lokalnega maksimuma, do katerega smo prišli s požrešno metodo. Vsekakor pa ni nič narobe, če srečo poskusimo tudi z lokalnim vzpenjanjem.

4. Na vsakem koraku ocenimo izboljšamo rešitev r. Če je vrednost kriterijske funkcije za trenutno rešitev prešla zastavljeno zadovoljivo mejo, potem se program zaključi. Prav tako se zaključi, če po n korakih še vedno tičimo v istem lokalnem maksimumu.

To je le ena izmed možnosti, kako zastaviti optimizacijski potek, s katero smo želeli predstaviti simbiozo opisanih metod. Nič ne bi bilo narobe, če bi za iskanje najboljšega urnika posamično uporabili katero izmed tehnik – mogoče bi katero izmed teh postopanj dalo celo boljši rezultat.

2.4 Zaključek

Sledili smo receptu in s tem izdelali "okusen obrok", treba ga je le še ustrezno postreči gostom za mizo. S tem imamo v mislih grafični uporabniški vmesnik (Graphical User Interface), ki mora urnik uporabnikom predstaviti na razumljiv in pregleden način. Končnega izgleda izdelka nikakor ne smemo zanemariti, saj je ta med uporabniki vreden skoraj toliko kot kakovosten urnik. Odločili smo se, da za razmišljanje o tej temi vaši domišljiji pustimo prosto pot.

Spoznali smo torej tri metode za reševanje optimizacijskih problemov, pri čemer pa niti za eno ne moremo z gotovostjo trditi, da nas bo pripeljala do globalnega maksimuma. To nekako opozarja na kompleksnost problemov, hkrati pa tudi na neraziskanost tega področja. Verjamemo, da bo prihodnost prinesla postopke, ki bodo s pomočjo tedanjih več tisoč-jedrnih procesorjev znali poiskati globalne rešitve za nekatere današnje optimizacijske probleme. Prihodnost pa bo hkrati prinesla tudi zahtevnejše probleme in s tem nove izzive za velike ume tistega časa.